(естественная адресация), а вспомогательный для очереди команд, если условие выполняется или нарушается естественный порядок следования команд. В ряде моделей ЭВМ используются два и более вспомогательных буфера, т.е. для каждого условия или цикла свой буфер;

♦ использование таблицы с предисторией переходов. Метод основан на использовании ассоциативной памяти, в которой в ходе выполнения программы запоминаются наиболее часто используемые переходы в командах условного перехода, циклов, вызова подпрограмм и т.д., например, по стратегии псевдо LRU-стека. Тогда при повторном выполнении некоторого участка программы осуществляется быстрый доступ к АЗУ и выполняется предвыборка команд и помещение их в очередь команд.

Необходимо также учитывать, что для ликвидации конфликтов по доступу к устройствам, располагаемых на различных уровнях конвейера, вместо магистрального принципа обмена данными между уровнями конвейеризации вводятся непосредственные связи "каждый с каждым". Для ликвидации конфликтов по доступу к устройствам, используемых на нескольких стадиях конвейеризации, вводятся многопортовые кэш-памяти или разделение ресурсов (например, отдельная кэш-память команд и двухпортовая кэш-память данных для чтения операнда и записи результата).

3 Средства управления памятью

3.1 Логические адреса и сегментная организация памяти

В современных компьютерах адресная часть команды представлена в виде логического адреса (ЛА), а не физического (ФА), что требует дополнительного времени и средств для выполнения преобразования ЛА в ФА. Необходимость представления адресной части команды в виде ЛА обусловлена следующими причинами:

- ♦ обеспечение программной совместимости моделей центрального процессора снизу доверху за счет преемственности (совместимости) форматов команд;
- ♦ возможности наращивания емкости оперативной памяти (ОП) без изменения разрядности ЛА (формата команды);
- ♦ обеспечивает возможность написания программ без привязки к ФА памяти;
- ♦ возможность динамического распределения памяти программ и данных и эффективное использование емкости ОП;
- ♦ возможность обеспечения защиты сегментов памяти программ и данных как в однозадачных, так и в многозадачных режимах работы.

Для программы адресное пространство разделено на блоки смежных адресов, называемых сегментами, а программа может обращаться к данным, находящимся только в этих сегментах. Внутри сегментов используется линейная адресация. Такая адресация осуществляется относительно начала сегмента (базового адреса), а физический адрес скрыт от программиста, так как программист пишет программу в ЛА. ОС выполняет функции распределения сегментов по ФА, что требует перед обращением к ОП выполнения процедуры преобразования ЛА в ФА.

Для представления адресной части команды используют два способа адресации: **линейную и сегментную адресации.**

При линейной адресации разрядность логического адреса совпадает с разрядностью физического адреса, а программа и данные размещаются в линейном адресном пространстве в соответствии с требуемым объемом памяти для хранения всей программы и/или данных.

При сегментной адресации все пространство адресов делится на множество сегментов различной длины, определяемой необходимым размером данного сегмента. Начальный адрес сегмента называется

базовым, а за каждым сегментом закреплен соответствующий номер. Порядок разбиения на сегменты может быть произвольным, а исполнительный адрес определяется номером сегмента (базовым адресом) и смещением внутри сегмента.

Отсюда ЛА можно представить в виде двух целочисленных величин (полей): номера сегмента и смещения. При этом разрядность смещения определяет максимальный размер сегмента в байтах. В ряде 16-разрядных процессоров для удобства преобразования ЛА в ФА сегмент разбивается на блоки, кратные 2^{k-1} байт, т.е. смещение разбивается еще на два поля: номера блока в сегменте и адреса байта в блоке.

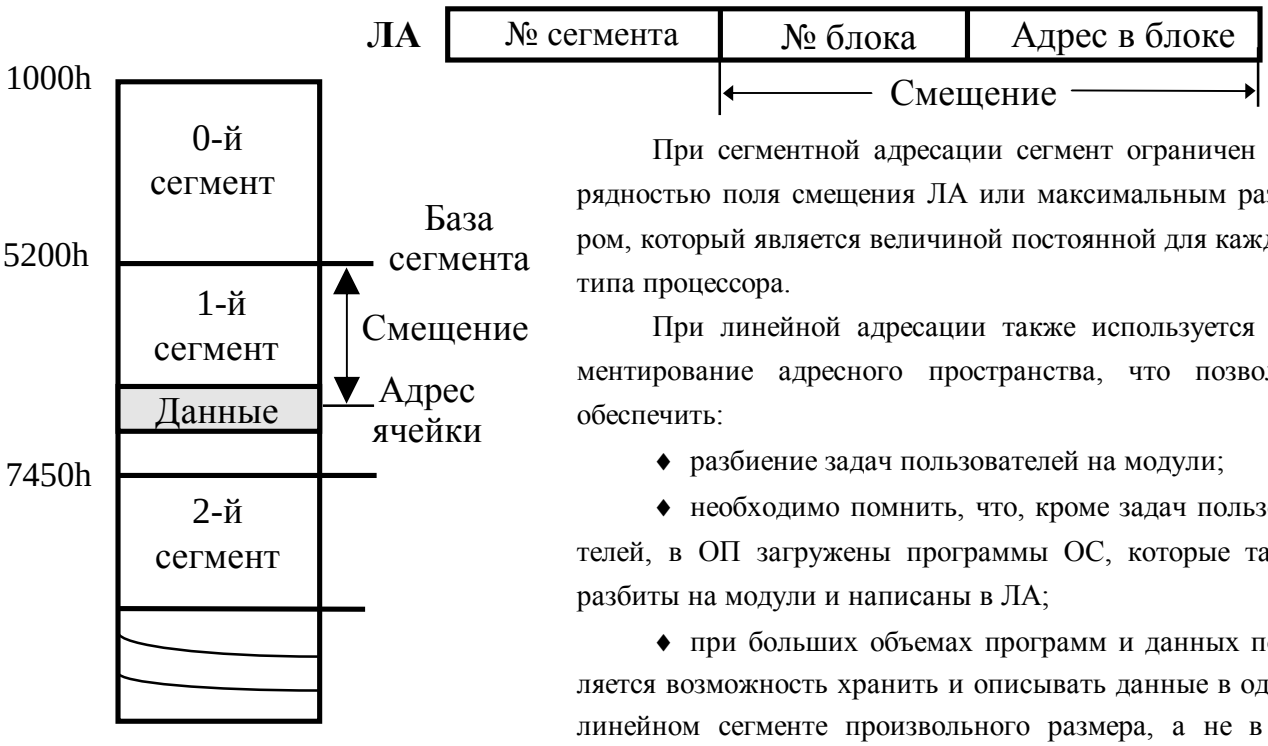


Рисунок 1.1 - Сегментированное адресное пространство

При сегментной адресации сегмент ограничен разрядностью поля смещения ЛА или максимальным размером, который является величиной постоянной для каждого типа процессора.

При линейной адресации также используется сегментирование адресного пространства, что позволяет обеспечить:

- ♦ разбиение задач пользователей на модули;
- ♦ необходимо помнить, что, кроме задач пользователей, в ОП загружены программы ОС, которые также разбиты на модули и написаны в ЛА;
- ♦ при больших объемах программ и данных появляется возможность хранить и описывать данные в одном линейном сегменте произвольного размера, а не в нескольких, как при сегментной адресации;
- ♦ динамическое распределение памяти и перемещаемость сегментов;
- ♦ защиту сегментов пользователей и ОС от взаимного преднамеренного или случайного воздействия друг на друга.

В некоторых ЭВМ программы логически разделяются на области (сегменты) кода (программ) и данных, которые, в свою очередь, подразделяются на сегменты непосредственно данных и стека, что позволяет упростить изолирование составляющих программ и задач друг от друга в мультизадачной среде и обеспечить эффективные средства защиты сегментов при возникновении ошибок по доступу к сегментам.

В целях рационального процесса программирования для ряда задач программы и данные разбивают на модули, что обеспечивает удобство отдельной отладки модулей программ и использования модулей данных по их назначению несколькими программами. Для этих целей каждый программный модуль (кода) или модуль данных можно разместить в отдельном сегменте с определенным номером, а сегменты размещать в произвольной области памяти. Местонахождение сегмента будет осуществляться по номеру сегмента, а адресация к ячейкам внутри сегмента по значению поля смещения. Таким образом, вычисление ФА памяти заключается в преобразовании номера сегмента ЛА в базовый адрес ОП, с которого начинается сегмент, и суммированием его со смещением ЛА внутри сегмента.

При линейной адресации число разрядов, определяющих номер сегмента, можно установить произвольным или, другими словами, устанавливать произвольным размер сегмента (или разрядность смещения для каждого сегмента). Например, в таблице 3.1 приведен пример такого разбиения 16-ти мегабайтного 24-разрядного адресного пространства логического адреса процессора MC68000 на два сегмента по 4 Мбайта, два сегмента по 2 Мбайта и 64 сегмента по 64 Кбайт памяти:

2 x 4 Мбайт + 2 x 2 Мбайт + 64 x 64 Кбайт = 16 Мбайт

Здесь старшие k разрядов ЛА определяют номер сегмента, а младшие n-k-1 бит разрядность смещения. Естественно, для реализации такого принципа распределения для каждого сегмента необходимо дополнительно указывать его максимальный разрешенный размер.

На рисунке 3.2 представлены примеры форматов ЛА процессоров MC68000, Z8001, Intel 8086 и фирмы DEC.

Процессор MC68000 имеет 24-разрядный линейный ЛА. В остальных процессорах используется сегментная адресация. В Z8001 и DEC логический адрес делится на 7-ми и 3-х разрядные поля номера сегмента соответственно, которые определяют количество регистров сегментов процессора для хранения базовых адресов этих сегментов, т.е. преобразование номера сегмента в базовый (начальный) номер сегмента сводится к его табличному преобразованию путем обращения к дополнительной памяти регистров сегментов. Смещение занимает 16-ти и 13-разрядные поля и определяет максимальный размер сегмента по 64 и 8 Кбайт соответственно. Поле смещения разделено на две части, разбивающие сегменты на блоки емкостью по 256 и 64 байта. Максимальное число блоков, входящих в сегмент, равно 256 (Z8001) и 128 (DEC).

Таблица 3.1 - Пример смешанного использования сегментов разного размера для линейной адресации

Номер сегмента	Разряды базы логического адреса A23	A22	A21	A20	A19	A18	A17	A16	Размер сегмента
0-й сегмент	0	0	X	X	X	X	X	X	2^{22} = 4 Мб
1-й сегмент	0	1	X	X	X	X	X	X	2^{22} = 4 Мб
2-й сегмент	1	0	0	X	X	X	X	X	2^{21} = 2 Мб
3-й сегмент	1	0	1	X	X	X	X	X	2^{21} = 2 Мб
4-й сегмент	1	1	0	0	0	0	0	0	2^{16} = 64 Кб
5-й сегмент	1	1	0	0	0	0	0	1	2^{16} = 64 Кб
6-й сегмент	1	1	0	0	0	0	1	0	2^{16} = 64 Кб
7-й сегмент	1	1	0	0	0	0	1	1	2^{16} = 64 Кб
. . . . .	.	.	.	.	.	.	.	.	. . . . .
67-й сегмент	1	1	1	1	1	1	1	1	2^{16} = 64Кб

X совместно с разрядами A15-A0 - смешение внутри сегмента

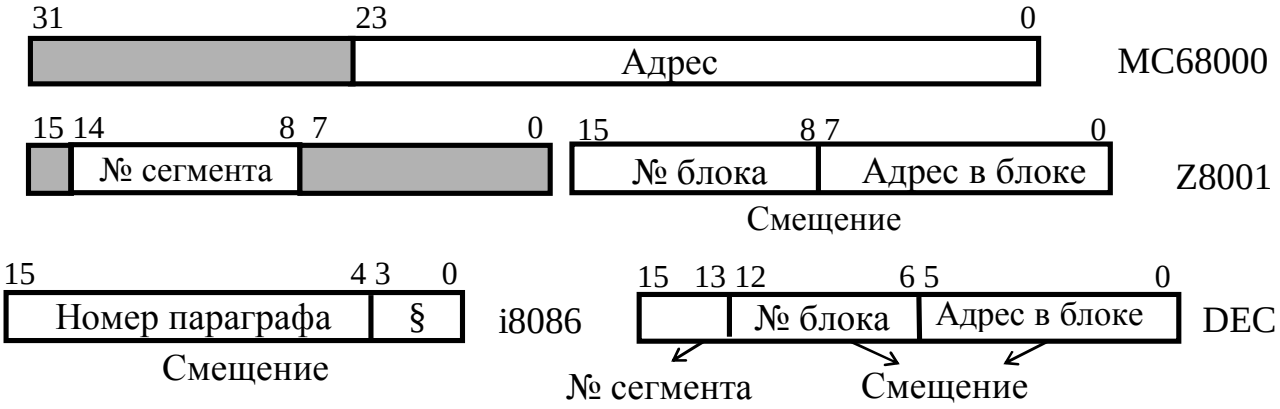


Рисунок 3.2 - Форматы логического адреса процессоров

В процессоре Intel 8086 в формат ЛА не входит поле номера сегмента, что напрямую связано с архитектурой процессора и регистровыми структурами. Поэтому ЛА включает только текущее смещение, состоящее из 4-разрядного адреса байта в блоке (параграф) и 12-разрядного номера блока (параграфа) в сегменте.

Отметим также, что для процессоров MC68000 и Z8001 ЛА хранится в 32-разрядном двойном слове, а для ЦП i8086 и DEC в 16-разрядном слове.

3.2 Регистровые структуры

Во всех 16-разрядных процессорах используются программно доступные регистры общего (РОН) и специального назначения. Структура РОН зависит от структуры ЛА (рисунок 3.2). РОН в составе ЦП используются для хранения операндов, наиболее часто используемых при вычислениях, что сокращает количество обращений к ОП за операндами, а следовательно, позволяет повысить и производительность процессора. Чем больше емкость программно доступных РОН, тем больше промежуточных данных можно в них хранить без обращения к ОП. Также РОН можно использовать для хранения адресов при выполнении некоторых видов адресации (регистровая косвенная, базовая и т.п.) и для других целей.

Процессор MC68000. В процессоре MC68000 логический адрес размещается в двух 16-разрядных ячейках памяти. Поэтому РОН выполнены в виде 32-разрядных регистров, которые по назначению делятся на две группы по 8 регистров в каждой:

* регистры данных D7-D0, позволяющие хранить как слова двойной длины, так и словные операнды, а в младшем слове разрешена обработка и байтов (в формате команды два бита указывают на размер операнда). Регистры данных используются при выполнении арифметико-логических преобразований, операций сдвига и т.п.;
* регистры адреса A7-A0 используются для хранения ЛА для некоторых видов адресации, при этом регистр A7 выполняет функции указателя стека, а так как процессор работает в двух режимах - системном и пользовательском, то таких регистров в составе РОН два: A7 и A7', которые выбираются автоматически в зависимости от режима работы.

Кроме этих регистров, в состав процессора входит 32-разрядный программный счетчик PC и 16-разрядный регистр слова состояния SR.

В состав процессора Z8001 входит шестнадцать 16-разрядных РОН для адресации к словам, из которых R7-R0 можно использовать для выполнения байтных команд. Кроме того, все регистры образуют восемь регистровых пар RR0-RR14, которые служат для хранения адресов, так как формат ЛА 32-разрядный. Выбор регистровой пары осуществляется автоматически при обработке соответствующего вида адресации.

Z8001 также работает в двух режимах работы: системном и пользовательском, поэтому в состав РОН входят два указателя стека RR14 и RR14', выбираемых автоматически в зависимости от режима работы процессора.

Программный счетчик PC 32-разрядный занимает два регистра, один регистр используется в качестве слова состояния процессора SR и два регистра PSAP используются в качестве указателя области состояния программ.

Необходимость расширения РОН до шестнадцати обусловлена:

* хранением в РОН 32-разрядных логических адресов;
* при выполнении арифметико-логических преобразований результат операции помещается

только в РОН, что требует увеличения их количества для сокращения пересылок между РОН и ОП (в формате команды нет бита направления записи результата).

В архитектуре **процессора фирмы DEC** предусмотрено 8 программно доступных регистров, из которых R6 и R6' используются как указатели стеков системного и пользовательского режимов, а регистр R7 - в качестве программного счетчика PC, что позволяет реализовать 4 дополнительных вида адресации (непосредственную, абсолютную, относительную по PC, косвенную относительную по PC) на основе прямой и косвенной автоинкрементной и индексной адресаций.

Программно доступные регистры R0-R5 используются для хранения операндов, адресов, индексов, счетчиков циклов и не имеют жесткой привязки по назначению. В процессоре выполнение байтовых операций возможно только с младшим байтом РОН.

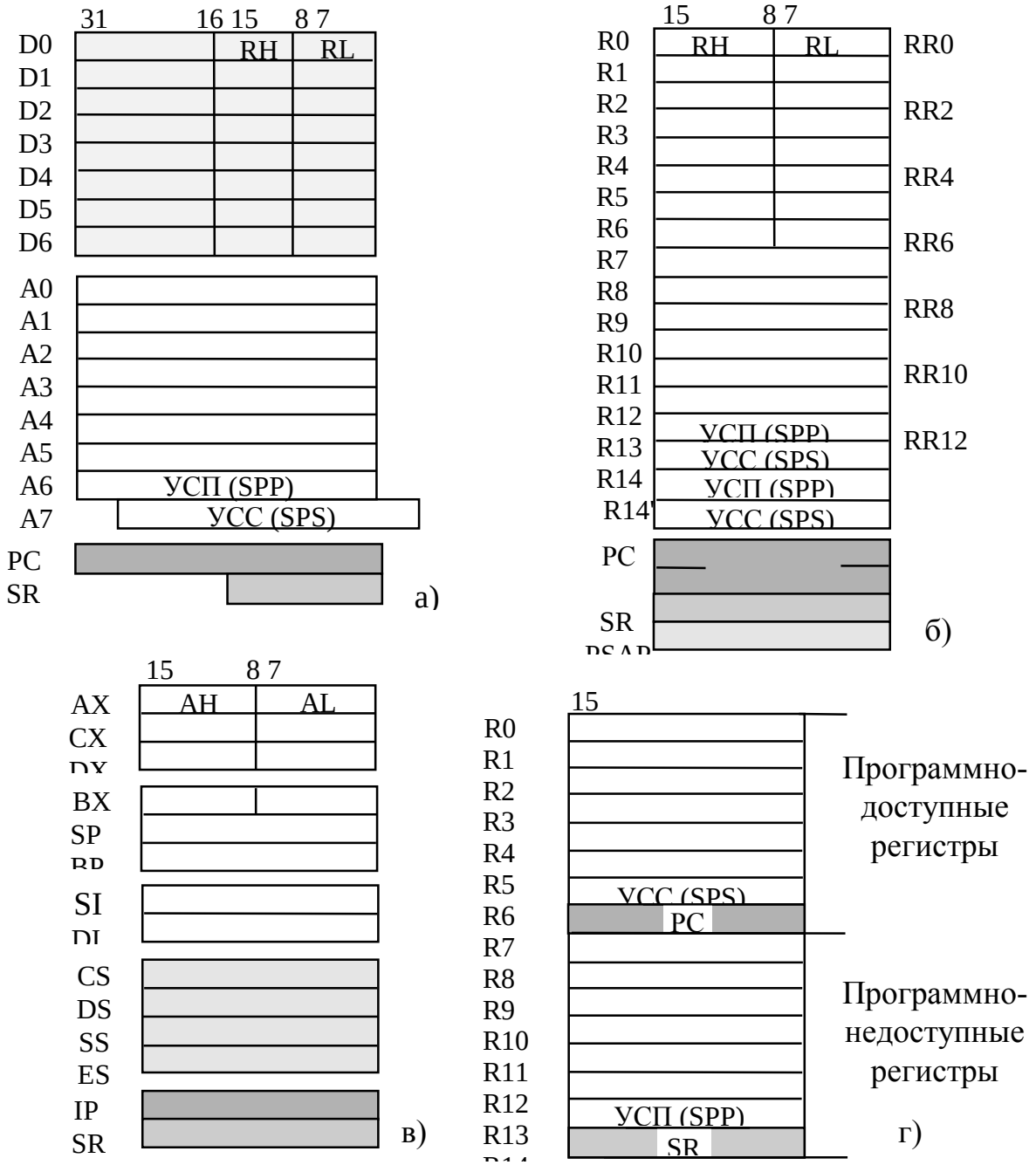


Рисунок 3.2 - Структура регистров процессоров MC6800 (а), Z800 (б), Intel 8086 (в), DEC (г)

Программно недоступные регистры R8-R15 предназначены для выполнения определенных функций для временного хранения исполнительного ЛА, входных операндов, промежуточных результатов и других целей.

Процессор **Intel 8086** имеет 14 16-разрядных регистров, которые можно разделить на 4 набора: 3 набора по четыре программно доступных регистра и два программно недоступных: указатель команды IP и слово состояния процессора SR.

РОНы в процессоре i8086 представлены 16-разрядными регистрами AX, BX, CX и DX, которые можно использовать как словные, так и как байтные (AL, BL, CL, DL - младший байт, AH, BH, CH, DH - старший байт). Адресация к регистрам показана ниже.

100	AH	000	AL	AX	000	Аккумулятор
101	CH	001	CL	CX	001	Счетчик
110	DH	010	DL	DX	010	Данные
111	BH	011	BL	BX	011	База
				SP	100	Указатель стека
				BP	101	Указатель базы
				SI	110	Индекс источника
				DI	111	Индекс приемника

В некоторых командах функции РОН специализированы, например, регистр CX является счетчиком в цепочечных командах.

Вторую группу составляют указательные и индексные регистры (SP, BP, SI, DI), которые обычно содержат внутрисегментные смещения.

Третью группу регистров составляют сегментные регистры CS, DS, SS и ES, каждый из которых идентифицирует конкретный текущий сегмент и функции их совершенно различны: CS - идентифицирует текущий сегмент кода (программы), DS - текущий сегмент данных, SS - текущий сегмент стека, ES - текущий дополнительный сегмент (данных). Таким образом, логический адрес в команде задает только смещение в сегменте, а требуемый текущий сегментный регистр определяется командой по умолчанию или задается в команде. Например, выборка всех команд осуществляется из текущего сегмента кода CS, а смещение задается в указателе команды IP. Базовые адреса операндов хранятся в текущих сегментах данных DS или ES, при стековых командах в сегменте стека SS, а смещение задается в команде или хранится в одном из РОН.

Такое разделение на специализированные сегментные регистры позволяет программе использовать до четырех сегментов емкостью до 64 Кбайт каждый, что в первых моделях процессоров было вполне достаточным, и перезагрузка сегментных регистров новыми базовыми адресами выполнялась редко.

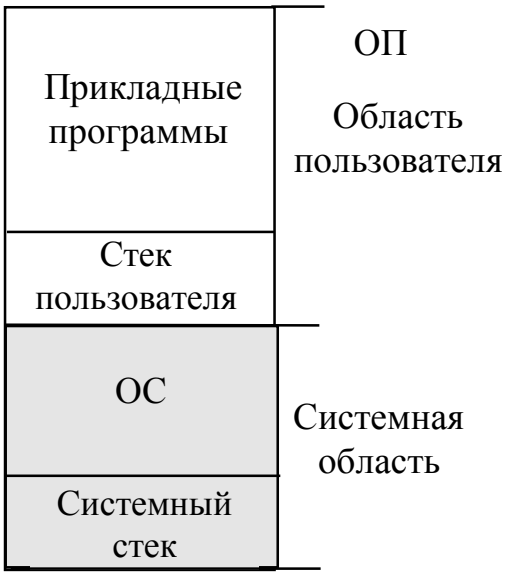
3.3 Режимы работы процессора и виды прерываний

Управление работой ЭВМ осуществляется с помощью операционной системы (ОС), которая включает набор средств для управления выполнением последовательности команд, файлами, трансляцией и всеми периферийными устройствами (ПУ) с помощью драйверов.

В памяти ЭВМ хранятся программы ОС и прикладные программы. ОС инициирует выполнение прикладной программы, при этом в том месте, где требуется обращение к ПУ, управление должно передаваться ОС. То же происходит при возникновении прерываний от ПУ. По окончании выполнения программы, принадлежащей ОС, возобновляется выполнение прикладной программы.

Во всех компьютерных системах для целей защиты предусматриваются, как минимум, два режима работы – системный режим, называемый также режимом супервизора, и пользовательский режим.

Основное различие между ними состоит в том, что программам, работающим в системном режиме (а ими обычно являются программы ОС), доступны все ресурсы системы. В пользовательском режиме программам запрещается выполнение некоторых (привилегированных) команд, влияющих на общесистемные ресурсы, а также программ, выполняющих функции управления системными устройствами (ПУ). Обычно один бит регистра слова состояния процессора используется для указания текущего режима работы. Основная память разделяется на системную область и область пользователя.



Для каждой области имеется свой стек, что предотвращает взаимное вмешательство программ при вложенных вызовах. Из системного режима можно перейти в пользовательский, а возможность обратного перехода отсутствует, так как регистр слова состояния SR процессора относится к программно недоступным со стороны пользовательских программ.

Поэтому должны быть средства, обеспечивающие обращение к подпрограммам, входящим в состав ОС из пользовательского режима. В качестве таких средств в состав процессора входит так называемая подсистема прерываний.

Таким образом, для повышения эффективности работы средств поддержки обмена необходимо иметь средства, обеспечивающие прерывание текущего процесса (программы) и переход к обслуживанию запросов на обмен или обработку специфических ситуаций, называемые подсистемой прерывания. Все прерывания можно разделить на:

- **внутренние ;**
- **программные ;**
- **внешние.**

Прерывания, происходящие в результате событий, локализованных внутри модуля системы, например, в процессоре при делении на ноль, обрабатываются с помощью средств **внутренних прерываний**, в функции которых обычно входит распознавание причины прерывания (формирование вектора прерывания), сохранение текущего состояния процессора (SR, PC и т.п.) и загрузка в программный счетчик начального адреса подпрограммы, обрабатывающей данную ситуацию.

Некоторые авторы (программисты) внутренние прерывания относят к программным (так как они не требуют наличия аппаратуры прерывания) и называют их особыми случаями. Особые случаи возникают, например, при нарушении защиты по привилегии, превышении размера сегмента, выходе за границы массива, делении на ноль и т.д. Возникновение данных ситуаций носит случайный характер и особые случаи невозможно предсказать.

Все особые случаи (внутренние прерывания) можно разделить на три группы:

- **нарушение** - это такой особый случай, который процессор может обнаружить **до возникновения фактической ошибки** (например, нарушение правил привилегий, превышение размера сегмента, нарушение атрибутов доступа к сегменту, недействительного кода операции и т.д.). После обработки нарушения (выполнения ППОП) можно продолжить программу **путем повторного выполнения (рестарта) виноватой команды;**

♦ **ловушка** - это такой особый случай, который процессор обнаруживает **после окончания выполнения виноватой команды** (например, прерывание при переполнении, большинство отладочных команд INT n). После обработки прерывания особого случая процессор возобновляет действия с той команды, которая находится после "захваченной" команды;

♦ **авария** - это ситуация, когда ошибка настолько серьезна, что ее невозможно устранить и продолжить выполнение программы. При аварии вычислительный процесс прекращается. К таким видам ошибок относятся аппаратные ошибки, обнаруживаемые ОС, или недопустимые значения в системных таблицах.

Также заметим, что **маскирование особых случаев** (внутренних прерываний) **невозможно даже запрещением прерываний в процессоре.**

Обработка при нарушении правил привилегии необходима для защиты ОС. Привилегированной называется команда, выполнение которой разрешается только в системном режиме. К ним относятся: команды изменения содержимого регистра состояния и системных регистров процессора, команда возврата RTI из системного режима, команды останова HALT и сброса RESET, команды ввода-вывода IN и OUT и другие.

С помощью **внешних** (аппаратных) **прерываний** осуществляется взаимодействие процессора с ПУ (клавиатурой, дисками, таймером и т.д.), сообщается о возникновении ошибок в устройствах от схем контроля (ошибки в памяти, на шине, аварийное выключение питания и т.д.).

Функциями средств, обслуживающих **внешние прерывания,** поступающих от ПУ (аппаратуры прерывания), являются:

- ♦ фиксация запросов на прерывание от внешних источников;
- ♦ определение номера приоритетного незамаскированного запроса для обслуживания;
- ♦ запоминание в стеке состояния текущего процесса (SR, PC и т.д.);
- ♦ передача управления подпрограмме обслуживания (обработки) данного запроса (ППОП) (поместить символ с клавиатуры в буфер, считать сектор с диска и т.п.);
- ♦ возврат - восстановление состояния прерванного процесса (программы) и передача ему управления.

В зависимости от способа реализации каждой из перечисленных функций подсистемы внешних прерываний могут классифицироваться:

- ♦ **с маскированием входов** запросов на прерывание;
- ♦ **без маскирования;**
- ♦ **бесприоритетные** (запросы на прерывание обслуживаются в порядке поступления);
- ♦ **приоритетные** (обслуживание запросов происходит в соответствии с назначенными приоритетами, которые могут быть фиксированными или циклически изменяемыми);
- ♦ **одноуровневые** (без вложенности);
- ♦ **многоуровневые**, допускающие вложение ППОП в соответствии с назначенным приоритетом;
- ♦ **динамически маскируемые**, при которых допускается обслуживание запроса на прерывание от источника с меньшим приоритетом (специального маскирования);
- ♦ **безвекторные**, при которых передача управления осуществляется по фиксированному адресу независимо от источника прерывания;
- ♦ **векторные** (запрос от каждого устройства обслуживается своей ППОП, для которой служит вектор точек входа (начальный адрес ППОП)).

В большинстве процессоров реализована система векторных прерываний, включающая 256 векторов (от 0 до 255), которые закреплены за всеми возможными причинами прерываний: внутренние, внешние, немаскируемые, программные и часть векторов зарезервирована для расширения.

Немаскируемые прерывания, как правило, поступают на отдельный вход немаскируемых прерываний процессора NMI. Процедура обработки запроса отличается от описанной только тем, что немаскируемое прерывание имеет фиксированный номер (вектор) с наивысшим приоритетом и не требуется процедура фиксации запроса и его распознавания, что ускоряет реакцию процессора на ситуацию.

В микропроцессорном комплекте (МПК) серии К1810 имеется специальная БИС контроллера прерываний ВН59А, с помощью которой можно путем программной настройки создавать подсистемы прерываний следующих видов: с маскированием входов; приоритетные (с фиксированными и циклически изменяемыми приоритетами); многоуровневые (до 64 уровней); векторные (до 64 векторов) и динамически маскируемые.

Программные прерывания.

Подпрограмму, находящуюся в системной области памяти, нельзя выполнить в пользовательском режиме непосредственно с помощью команд вызова CALL. Для этой цели в систему команд вводится специальная команда программного прерывания INT. В некоторых процессорах, работающих в двух режимах работы, в систему команд входят две команды программного прерывания: EMT - вызова системного прерывания (специального прерывания) и TRAP - вызова пользовательского прерывания (вектора захвата). При выполнении команды EMT информация сохраняется в стеке системной области памяти и в слове состояния процессора устанавливается системный режим работы, а по команде TRAP информация сохраняется в стеке пользователя без изменения режима работы процессора.

Таким образом, программные прерывания инициируются специальными командами, при выполнении которых в системном стеке запоминается состояние текущего процесса:

- содержимое специальных регистров, включая слово состояния процессора и программного счетчика, а также регистров общего назначения (в большинстве систем эти функции возлагаются на программиста);
- в слове состояния процессора изменяется разряд режима работы (если процессор работает в двух режимах, но не по привилегиям, которые обрабатываются по своим алгоритмам);
- обеспечивается загрузка в программный счетчик номера команды подпрограммы обработки прерывания, который содержится либо в формате команды программного прерывания в виде адреса или номера вектора, который предварительно необходимо преобразовать в начальный адрес подпрограммы обработки системного прерывания.

Возврат из программного прерывания выполняется с помощью специальных команд или программных средств: для команды INT по команде IRET возврата из прерывания, а для команд EMT и TRAP по командам RTI и RTT соответственно, которые определяют, из какого стека, системного или пользовательского, выполнять восстановление состояния процессора для возврата из подпрограммы обработки прерывания. Прежний режим работы процессора восстанавливается автоматически из слова состояния процессора SR, так как изменение режима работы в SR осуществляется после сохранения слова состояния в стеке.

3.4 Преобразование логического адреса в физический и средства защиты памяти

Как было показано выше, при сегментной организации памяти программирование ведется в логических адресах, которые при выполнении команды должны быть преобразованы в физические адреса ОП аппаратными средствами. Для этой цели между процессором и ОП помещают устройство, преобразующее ЛА в ФА (рисунок 3.3), которое называется устройством управления памятью (УУП) или диспетчером памяти (ДП).

Принцип преобразования ЛА в ФА заключается в следующем: в УУП помещаются и хранятся базовые адреса каждого сегмента (или текущего сегмента), которые назначает ОС при распределении адресного пространства ОП при загрузке сегментов программ и данных. Логический адрес представлен номером сегмента (рисунок 3.2) (или определяется по умолчанию значением текущего сегмента, как в процессоре i8086) и смещением, поэтому, используя адрес базы сегмента, по его номеру или типу получим:

ФА = Базовый адрес сегмента + Смещение.

Кроме того, к современным компьютерам предъявляются высокие требования по надежности, живучести и защищенности системы, особенно при работе с базами данных и в сети ЭВМ. Так как большинство ЭВМ работают в мультипрограммном режиме работы (одновременно выполняется несколько программ), то ошибки в одной из них могут влиять на выполнение других программ. Поэтому при обращении к неверному пространству памяти (к запрещенным сегментам данных и программ) механизм защиты должен блокировать обращение к ОП и сообщать о возникновении и причине ошибки.

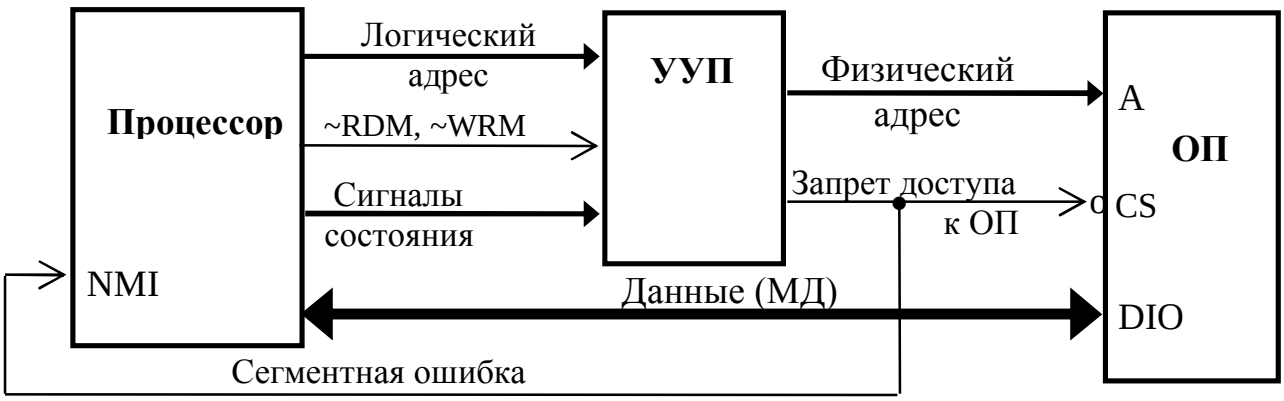


Рисунок 3.3 - Преобразование логического адреса в физический с помощью устройства управления памятью

Для реализации системы защиты памяти в УУП к регистру, хранящему базовый адрес сегмента, придается дополнительная информация, обеспечивающая защиту сегментов по атрибутам, размеру сегмента и ряду других параметров в зависимости от типа процессора, и она называется дескриптором сегмента. При этом преобразование ЛА в ФА в УУП выполняется параллельно с контролем реализованных средств защиты памяти. Так при нарушении хотя бы одного из параметров прав доступа к сегменту обращение к ОП должно блокироваться с выработкой вектора прерывания от системы защиты памяти для оповещения пользователя о причине прерывания вычислительного процесса по одной из ошибок программы или в ППОП делается попытка восстановления вычислительного процесса (устранения ошибки) в зависимости от причины (вектора) прерывания.

Рассмотрим работу УУП поэтапно, начиная с преобразования ЛА в ФА, так как при преобразовании ЛА в ФА и при защите памяти используется различная аппаратура, которая работает параллельно.

3.4.1 Преобразование логического адреса в физический

Для реализации процедуры преобразования ЛА в ФА и защиты памяти в УУП имеется дескриптор сегмента, формат которого для каждого типа процессора является индивидуальным, однако можно выделить общие части: поле базового адреса сегмента, поле размера сегмента (предел), поле атрибутов защиты. Число дескрипторов определяется количеством используемых сегментов, а на практике применяются три способа выбора дескрипторов:

* текущего номера дескриптора сегмента по умолчанию, определяемого типом сегмента (процессор i8086), когда изменение содержимого текущего дескриптора выполняется программно;

* адресом дескриптора служит номер сегмента (прямая адресация к ячейкам дескриптора) (рисунок 3.4 а). Таким образом, если номер сегмента является k-разрядным, то используется 2^{k-1} дескрипторов. При этом дескрипторы могут храниться:

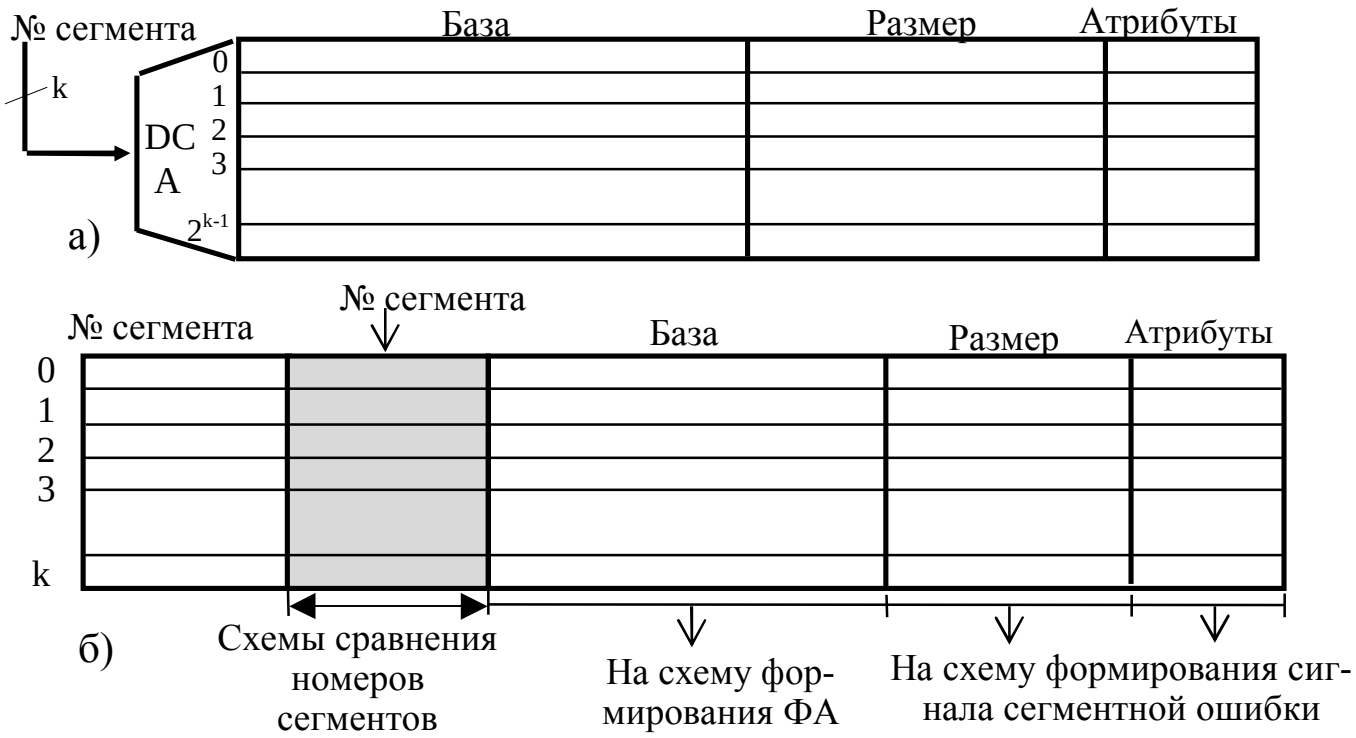


Рисунок 3.4 - Способы выбора регистров-дескрипторов сегментов

* в оперативной памяти (процессоры Intel 286 и старше в защищенном режиме работы);

* в быстродействующей регистровой памяти (ЦП Z8001 и DEC);

♦ в ассоциативном запоминающем устройстве (АЗУ) (рисунок 3.4 б) (ЦП MC68000). Адрес ячейки АЗУ определяется совпадением выделенного номера сегмента из ЛА с номером сегмента, дескриптор которого хранится в АЗУ. Число дескрипторов, хранимых в АЗУ, может быть меньше 2^{k-1}, а в АЗУ хранятся дескрипторы только активных сегментов, используемых в текущий момент времени. Полная таблица дескрипторов хранится в ОП.

Форматы дескрипторов рассматриваемых процессоров приведены на рисунке 3.5. Рассмотрим примеры реализации УУП.

Процессор MC68000. УУП содержит 32 регистра дескриптора сегментов с организацией по принципу ассоциативной памяти. В процессоре используется линейная адресация, поэтому в ЛА явно не указано поле номера сегмента. В каждом дескрипторе сегмента (рисунок 3.5) можно выделить 16-разрядные поля базы логического адреса (БЛА), маски логического адреса (МЛА), базы физического адреса (БФА) и 8-разрядные поля атрибутов защиты и других данных.

Из 24 бит логического адреса (рисунок 3.1) младшие 8 бит непосредственно используются в качестве ФА, так как представляют собой адрес байта в блоке из 256 байт. Старшие 16 бит ФА формируются в УУП при помощи базы ЛА, маски ЛА и базы ФА. БЛА показывает начало области сегмента, а конец области определяется по МЛА (поэтому указание размера сегмента в дескрипторе не используется).

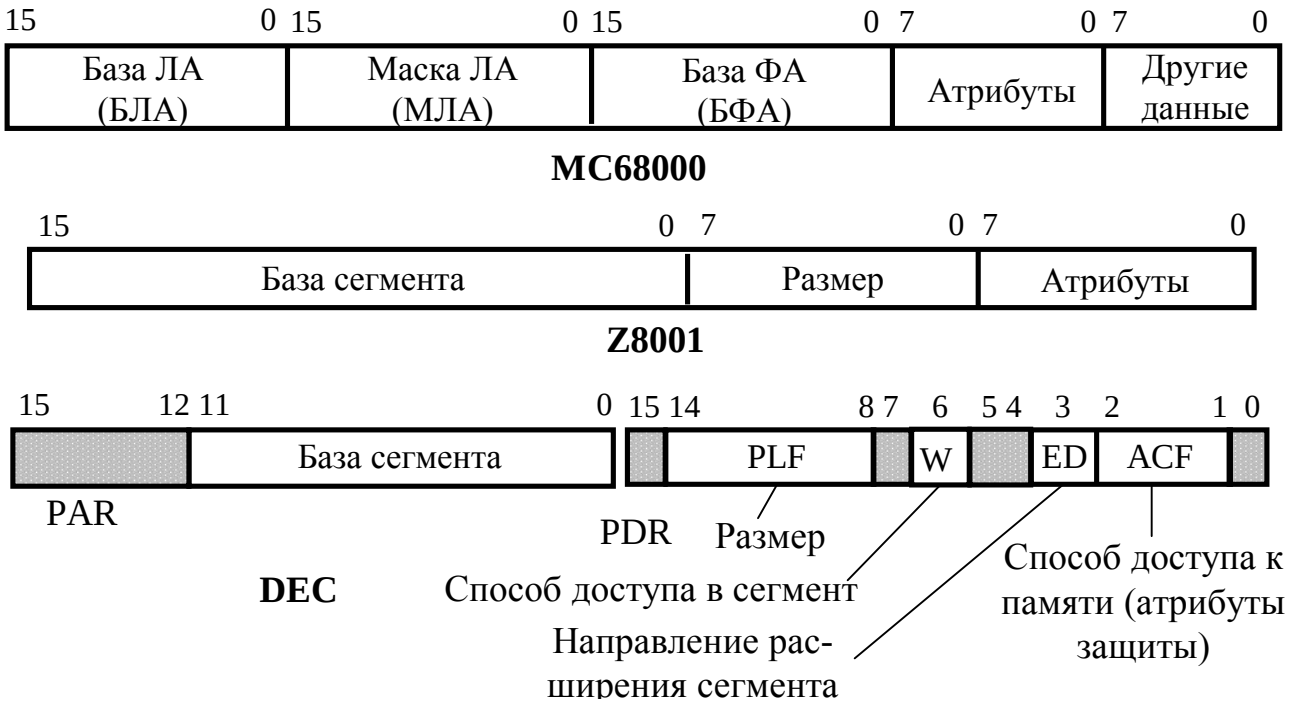


Рисунок 3.5 - Форматы дескрипторов процессоров MC68000, Z800 и DEC

Например: БЛА = 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
МЛА= 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0, то
Начало области - 1 1 0 0 0 1 0 **0 0 0 0 0 0 0 0**
Конец области - 1 1 0 0 0 1 0 **1 1 1 1 1 1 1 1.**
Номер сегмента

Так как в данном примере старшие 7 бит МЛА равны 1, то номер сегмента определяется семью старшими разрядами БЛА. Для распознавания, принадлежит ли выданный процессором ЛА данной области и находится ли дескриптор сегмента в АЗУ, необходимо сравнить логическое произведение БЛА и МЛА с логическим произведением ЛА и МЛА.

Для реализации такой процедуры в ассоциативной части УУП в АЗУ необходимо хранить БЛА сегмента и соответствующую ему МЛА (размер сегмента), а при ассоциативном поиске на входы АЗУ подаются старшие 16 разрядов ЛА и параллельно во всех 32 ячейках выполняется сравнение очищенных по МЛА старших разрядов БЛА и ЛА (рисунок 3.6). Если на выходе АЗУ формируется сигнал ЛА $\in$ АЗУ, то из одноименной ячейки поля данных АЗУ выбирается база ФА. Выбранные МЛА, БФА и ЛА поступают на комбинационную схему формирования ФА (рисунок 3.7). ФА образуется операцией конкатенации трех составляющих:

- путем выделения старших разрядов БФА по МЛА;

- выделения части разрядов ЛА по инверсии МЛА (в операции принимают участие биты [23-8] логического адреса;
- младших 8 разрядов логического адреса.

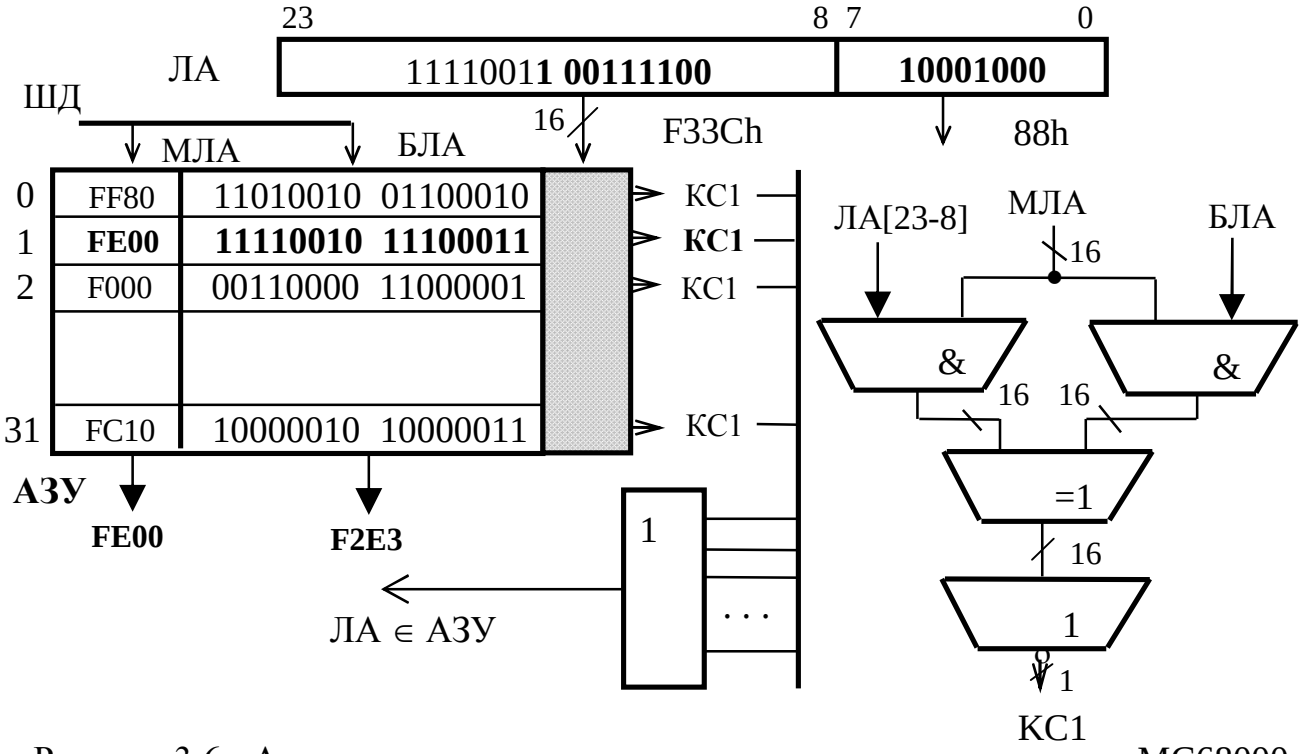


Рисунок 3.6 - Ассоциативная часть памяти дескрипторов процессора MC68000

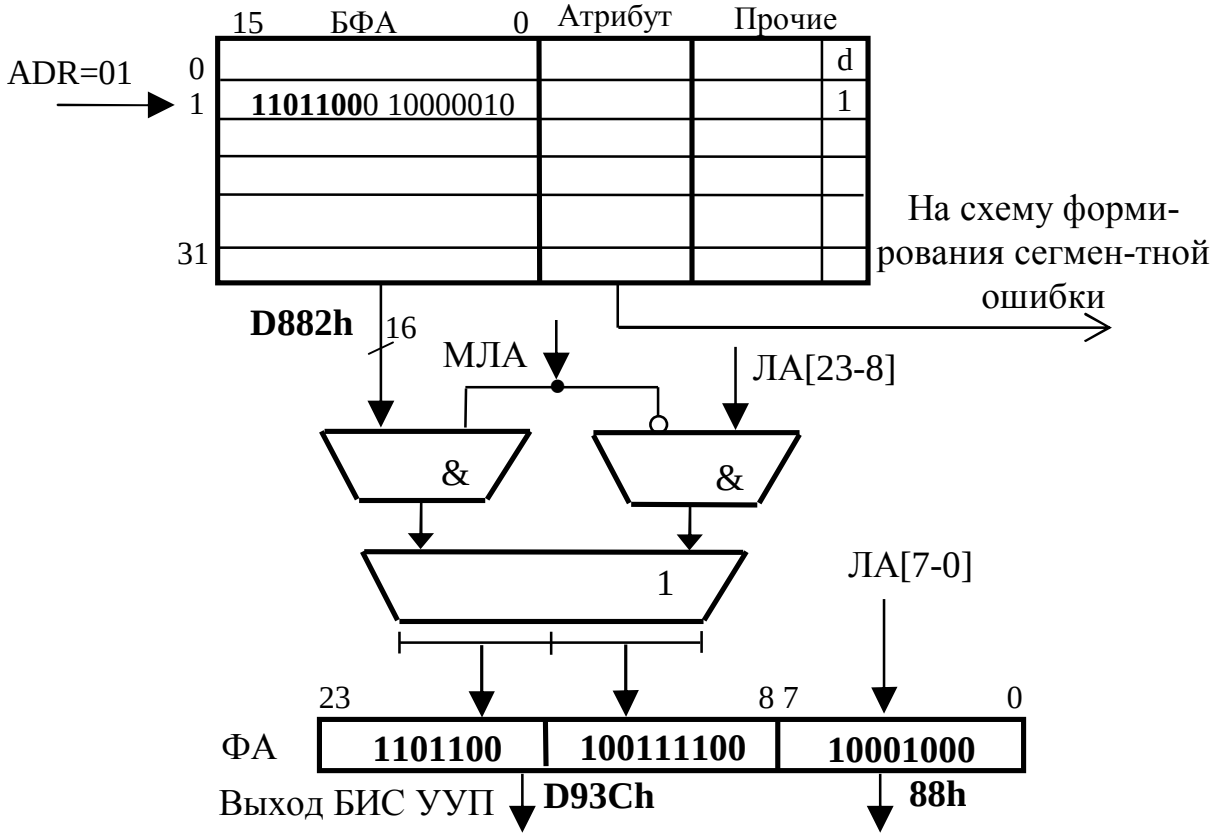


Рисунок 3.7 - Поле данных БФА дескрипторов процессора MC68000

Таким образом, размер сегмента определяется маской логического адреса по формуле:

$$\text{Размер} = 2^{24-k},\ k=1, 2, 3, \ldots, 16,$$

где k - старшие биты МЛА, равные 1.

При k=16 размер сегмента равен 256 байт, и можно адресоваться до 64 К сегментов с таким размером. При k=4 можно адресовать 16 сегментов емкостью по 1 Мбайт.

Кроме того, допускается одновременное использование сегментов разных размеров, так как в процессоре MC68000 при линейной адресации сегмента МЛА можно задавать для каждого сегмента индивидуально (таблица 3.1: 2 сегмента по 4 Мбайт, 2 сегмента по 2 Мбайт и 64 сегмента по 64 Кбайт). Таким образом, в данном УУП для формирования ФА не требуется сумматор.

В процессоре Z8001 дескриптор (рисунок 3.4) состоит из трех полей: 16-разрядного поля базы и 8-разрядных полей размера сегмента (в блоках) и атрибутов защиты. Формат ЛА (рисунок 3.3) состоит из 7-разрядного поля номера сегмента и 16-разрядного смещения, которое разделяется на 8-разрядное поле номера блока в сегменте и 8-разрядное поле адреса байта в блоке. Следовательно, УУП должно содержать 128 дескрипторов сегментов, которые хранятся в двух модулях по 64 регистра в каждом.

Преобразование ЛА в ФА выполняется путем суммирования базового адреса сегмента, выбранного из дескриптора по номеру сегмента, указанного в разрядах [14-8] старшего слова ЛА, и смещения, указанного в разрядах [15-8] младшего слова ЛА, и присоединения восьми младших разрядов ЛА [7-0] (рисунок 3.8).

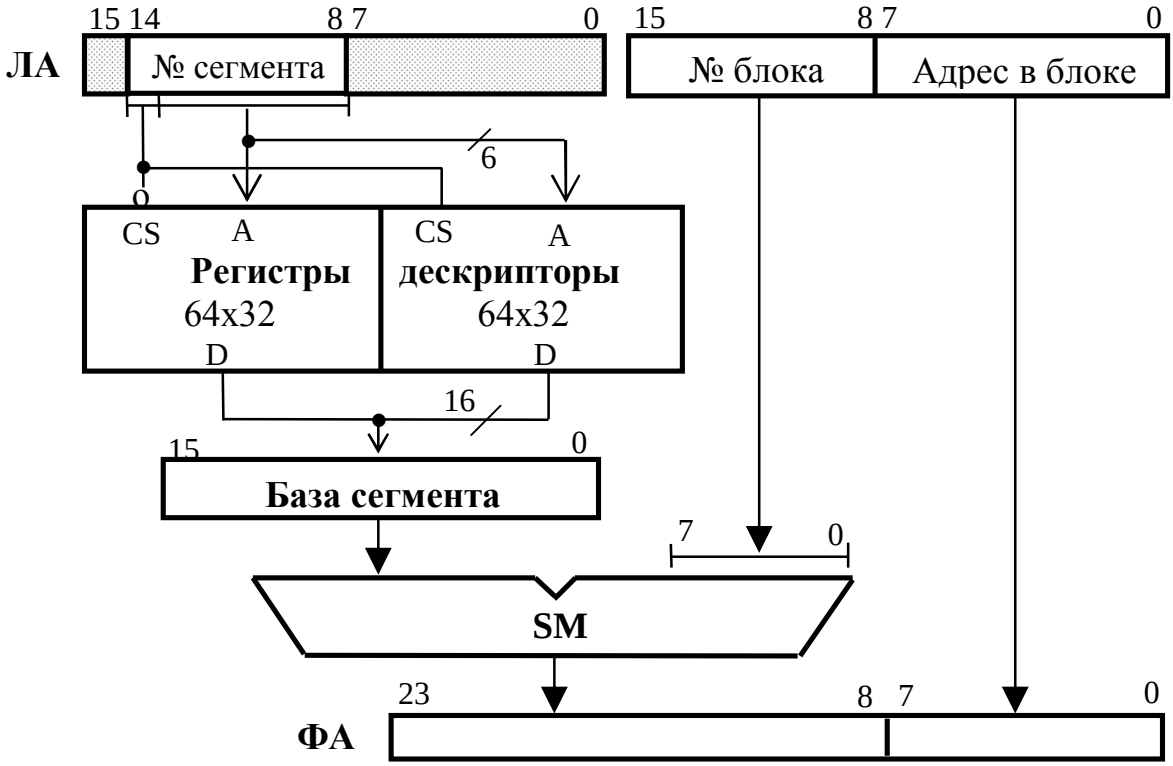


Рисунок 3.8 - Схема преобразования ЛА в ФА Z8001

Исходя из формата ЛА можно отметить, что базовый адрес и размер сегмента должны быть кратны 256, минимальный размер сегмента 256 байт (1 блок), максимальный - 64 Кбайт (256 блоков).

Для процессора i8086 ФА вычисляется в соответствии со схемой, показанной на рисунке 3.9. В качестве базы сегмента выбирается содержимое одного из текущих сегментных регистров CS, DS, SS или ES, выбираемого УУП автоматически в зависимости от типа информации (кода (программы) или

данных). Тип сегмента определяется кодом состояния процессора, типом выполняемой команды (стековая или данные) или источником данных, задаваемым в команде (сегмент DS или ES). Младшие четыре разряда ЛА присоединяются к ФА операцией конкатенации, а старшие получаются сложением 16-разрядного базового адреса одного из сегментных регистров и старших разрядов ЛА [15-4].

Емкость сегмента может изменяться от 16 байт до 64 Кбайт. Сегменты могут следовать друг за другом непрерывно, с интервалом или могут перекрываться.

УУП процессора фирмы DEC выполняет следующие функции:

- расширение емкости адресуемой памяти с 64 Кбайт до 256 Кбайт или 4 Мбайт;
- защиту пользовательских и системных программ друг от друга;
- использование различных областей адресного пространства для режима пользователя и ОС (системный режим).

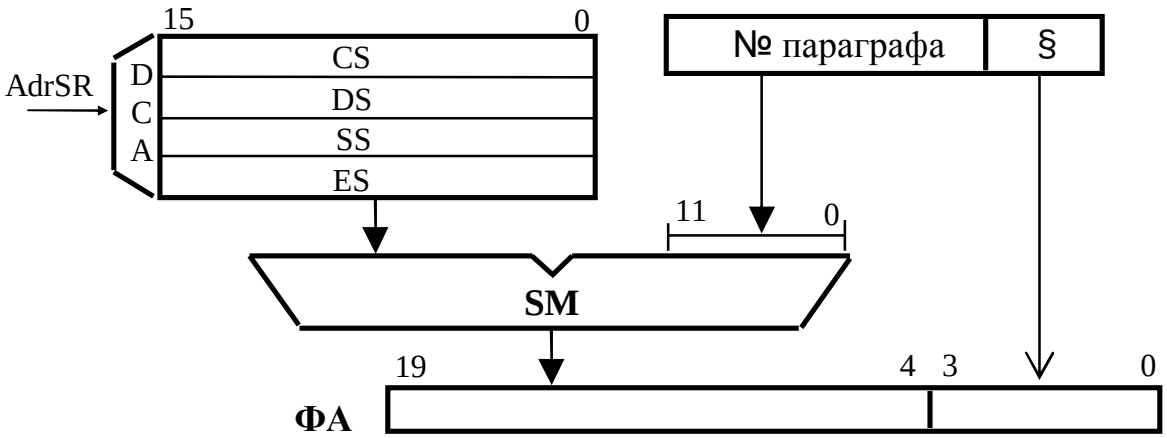


Рисунок 3.9 - Схема преобразования ЛА в ФА процессора i8086 (реальный режим)

Преобразование 16-разрядного ЛА (рисунок 3.3) в 18-разрядный (или 22-разрядный) ФА выполняется только при включенном УУП (ДП). Содержимое ЛА рассматривается УУП в виде трех полей: биты [15-13] определяют номер сегмента (регистра PAR), в котором хранится базовый адрес сегмента, в битах [12-6] размещается адрес блока в сегменте относительно базового адреса, а в битах [5-0] указан адрес байта внутри блока. Как видно, структура ЛА процессора DEC отличается от ЛА процессора Z8001 только разрядностью полей.

Схема формирования ФА приведена на рисунке 3.10. УУП может работать в двух режимах: с выключенным преобразователем ЛА в ФА и системой защиты памяти и с включенным. При выключенном УУП ЛА совпадает с ФА и поступает в RgФА в обход сумматора. При этом, если по команде осуществляется обращение к сегменту ввода-вывода (ЛА=160000_8 - 177776_8), то аппаратные средства УУП формируют два (при 18-разрядном или шесть при 22-разрядном ФА) старших бита ФА равными единице, так как сегмент ввода-вывода содержит фиктивные адреса в адресном пространстве ЭВМ и всегда занимает последний номер в регистрах дескрипторах (PAR7) и последний сегмент (ввода-вывода) в адресном пространстве ОП (базовый адрес данного сегмента фиксирован и равен 7600_8 (или 177600_8)). При выключенном УУП процессор адресует только 64 Кбайт памяти, а остальное адресное пространство используется в качестве электронного диска.

При включенном УУП базовый адрес размещается в дескрипторе сегмента в одном из восьми регистров PAR, а в регистр PDR загружается информация, описывающая полномочия сегмента для организации защиты памяти.

УУП включает две группы регистров для системных и пользовательских сегментов данных, что с одной стороны обеспечивает их защиту при неверном обращении к ним со стороны пользователя, а с другой стороны вызвана малым размером сегментов (до 8 Кбайт) и небольшим числом дескрипторов (максимум 8). Введение дополнительной группы регистров не требует частой перезагрузки базовых адресов сегментов, если задача использует более 7 сегментов данных, а также при переходе из одного режима работы в другой.

По значению битов PSW[15-14], в которых указывается режим работы процессора - системный или пользовательский, определяется, какая из групп регистров (системная или пользовательская) в данный момент используется. По номеру сегмента (ЛА[15-13]) выбирается соответствующий ему дескриптор, из которого 12-разрядный базовый адрес сегмента суммируется с номером блока в сегменте (ЛА[12-6]) и к полученной сумме операцией конкатенации присоединяются младшие 6 разрядов ЛА[5-0]. В результате получается 18-разрядный ФА. Для формирования 22-разрядного ФА в дескрипторе сегмента хранится 16-разрядный базовый адрес.

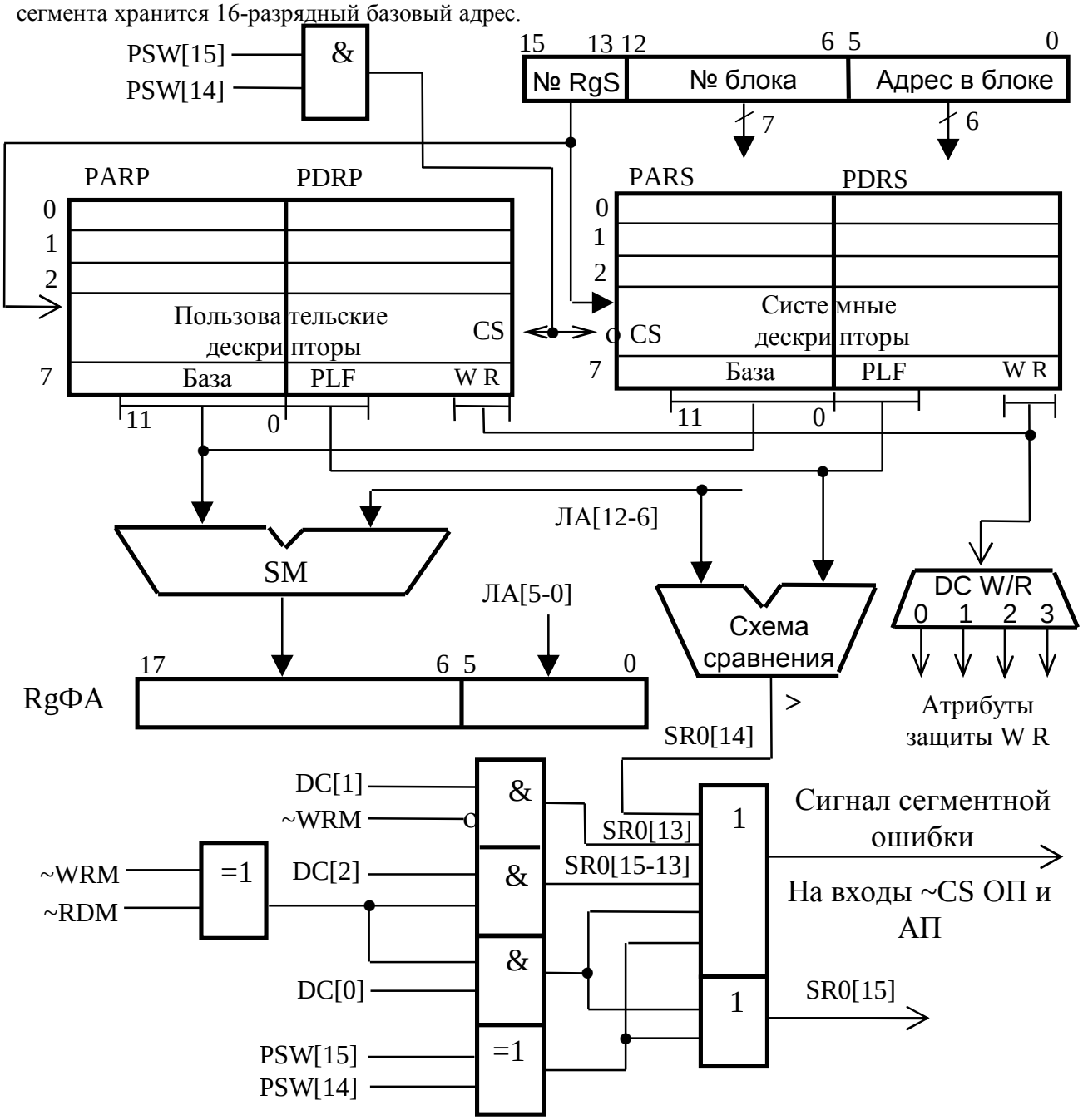


Рисунок 3.10 - Схема преобразования ЛА в ФА и формирования сегментной ошибки процессора фирмы DEC

3.4.2 Организация системы защиты памяти

Средства защиты памяти должны предотвращать:

- неразрешенные взаимодействия пользователей друг с другом;
- несанкционированный доступ пользователей к данным;
- повреждения программ и данных из-за ошибок в программах;
- намеренные попытки разрушить целостность системы (ОС);
- случайные искажения данных.

Основной целью введения системы защиты памяти является не допустить искажения достоверности информации, хранимой в ОП, от преднамеренных или случайных ошибок с целью быстрого восстановления вычислительного процесса. Особенно данная задача актуальна в многозадачных режимах работы.

Обычно механизм защиты подразделяется на две части:

1. Управление памятью;
2. Защита по привилегиям (режимам работы).

Схемы управления памятью обнаруживают большинство программных ошибок, например, формирование неверных адресов, нахождение индекса (номера блока или дескриптора) за пределами сегмента, искажение стека вызова/возврата и т.д.

Защита по привилегиям фиксирует более тонкие ошибки и намеренные попытки нарушить функционирование системы.

Также различают защиту памяти в зависимости от ее организации: на уровне защиты сегментов и на уровне защиты страниц.

Как было отмечено ранее, для реализации системы защиты памяти в УУП к регистру, хранящему базовый адрес сегмента, придается дополнительная информация, обеспечивающая защиту сегментов по атрибутам, размеру сегмента и ряду других параметров в зависимости от типа процессора; она называется дескриптором сегмента, форматы которого приведены на рисунке 3.3. При этом преобразование ЛА в ФА в УУП выполняется параллельно с контролем реализованных средств защиты памяти. Так при нарушении хотя бы одного из параметров прав доступа к сегменту **обращение к ОП должно блокироваться с формированием вектора прерывания от системы защиты памяти для оповещения пользователя о прерывании вычислительного процесса по одной из ошибок программы, в ряде случаев может выполняться попытка исправления ошибки, и вычислительный процесс продолжается с прерванной или следующей команды (в зависимости от причины прерывания, см. выше).**

Выделим следующие основные методы защиты памяти, применяемые в процессорах в различных сочетаниях в зависимости от архитектуры и степени (надежности) обеспечения защиты сегментов, хранимых в ОП:

- разделение ОП на системную область и область пользователя, что исключает прямое использование пользователем программ и данных ОС или внесение искажений в ОС (MC68000, Z8001, DEC);
  - ◊ с использованием ***раздельных*** (DEC) или ***общих*** (Z8001) регистров-дескрипторов для системных и пользовательских программ;
- защиту по привилегиям (Intel 80286 и выше), которая в общем случае позволяет контролировать:
  - ◊ возможность выполнения текущей команды (для привилегированных команд);

- ◊ возможность обращения к сегментам данных других программ;
- ◊ возможность передачи управления (перехода) в другой сегмент кода (программ), имеющему другой уровень привилегии по отношению к текущему кодовому сегменту;

♦ разделение сегментов по выполняемым функциям и типам хранимых данных (сегменты кода, стека, данных и т.д.) (Intel), что исключает возможность записи данных в область программ (кода);

♦ защиту сегментов по размеру, т.е. контроль значения ЛА (смещения) по выходу за границы сегмента (обращение к другому сегменту);

♦ защиту сегментов по правам доступа к сегменту (совокупности правил, запрещающих/разрешающих выполнять те или иные преобразования информации в сегменте, называемые также атрибутами защиты). При этом права доступа к различным типам сегментов, как правило, являются различными:

- ◊ в случае работы с сегментом кода (программы) атрибуты защиты допускают либо только выполнение программы, либо также и ее считывание (по умолчанию обычно запись в сегмент кода запрещена и отдельный атрибут защиты по записи в сегмент не требуется);
- ◊ в случае работы с сегментом данных в различных процессорах используются различные атрибуты защиты, например:
  - ⇒ запрет доступа к сегменту как по чтению, так и по записи;
  - ⇒ запрет доступа к сегменту только по записи;
  - ⇒ разрешение доступа по чтению и по записи;
- ◊ ограничение использования атрибутов защиты для сегмента стека, так как запрет обращения к сегменту стека по записи и по чтению, или по записи делает его бессмысленным при использовании (любое прерывание вызовет выработку сигнала сегментной ошибки, а также все машинные команды, использующие стек: CALL, INT, PUSH, POP).

В таблице 3.2 приведен пример одного из вариантов кодирования атрибутов защиты сегментов кода и данных для процессора, работающего в двух режимах работы с общими регистрами-дескрипторами для системного и пользовательского режимов (процессор Z8001).

Таблица 3.2

Режим	Содержание ОП	Атрибут	Режим	Код защиты
Системный	Программа	Только выполнение Допускается считывание	0 0	00* 01
	Данные	Только считывание Допускается запись	0 0	10 11
Пользовательский	Программа	Только выполнение Допускается считывание	1 1	00 01
	Данные	Только считывание Допускается запись	1 1	10 11

* Первая цифра указывает на тип сегмента (кода или данных), вторая на атрибут защиты.

В каждом цикле шины (обращении к ОП) процессор вырабатывает серию сигналов состояния (тип цикла шины процессора) и один из управляющих сигналов чтения ~RD и записи ~WR для обращения к ОП или УВВ, а УУП сравнивает атрибуты защиты из дескриптора сегмента с этими сигналами и размер сегмента (рисунок 3.11)

Если при сравнении размер сегмента из дескриптора и из ЛА или атрибуты защиты и сигналы состояния и управления памятью из процессора не "совпадают", то схема защиты памяти формирует сигнал сегментной ошибки, который выполняет следующие функции:

◊ блокирует доступ к памяти по чтению или по записи для предотвращения искажения информации в ОП (непосредственно защита памяти);

◊ фиксируется состояние процессора (причина (код) ошибки) в специальных регистрах ошибок, состав которых зависит от типа процессора;

◊ этот же сигнал используются для формирования номера вектора прерывания (прерывание немаскируемое), по которому выполняются те же действия, что и по команде программного прерывания INT. Процессор прерывает выполнение текущей программы и приступает к обработке особого случая прерывания (подпрограмму обработки прерывания (ППОП));

◊ в ППОП выполняется считывание состояния регистров ошибок и процессора, выполняется распознавание причины прерывания и на экран выводится сообщение для пользователя о типе (причине) ошибки в виде вектора прерывания и кода ошибки, номер команды (или сама команда, вызвавшая прерывание) и другая вспомогательная информация для быстрой локализации причины прерывания;

◊ процессор пытается выполнить исправление ошибки в зависимости от ее серьезности и восстановить вычислительный процесс при возврате из прерывания в основную программу.

Таким образом, процедура выработки сигнала сегментной ошибки системы защиты памяти для всех типов процессоров строится по общему алгоритму (рисунок 3.11) и для каждого типа процессора зависит только от формата дескриптора сегмента, архитектуры процессора, кодирования и состава атрибутов защиты, сигналов состояния конкретного процессора из всего множества рассмотренных и реализованных методов защиты памяти. То есть разработка конкретной схемы защиты памяти сводится к синтезу схем сравнения по размеру сегментов, атрибутам защиты и привилегиям.

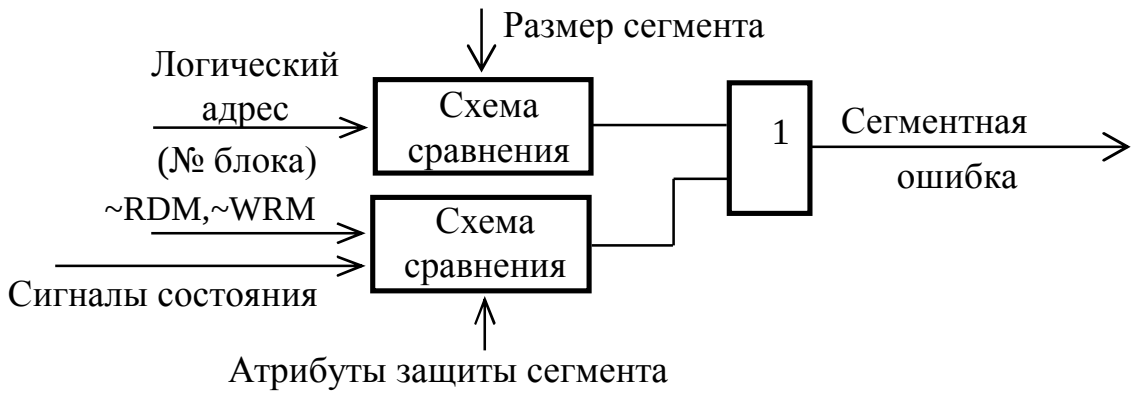


Рисунок 3.11 - Схема формирования сигнала сегментной ошибки при нарушении размера сегмента и атрибутов защиты по доступу к памяти

Рассмотрим организацию системы защиты памяти для процессора фирмы DEC. Напомним сразу, что в процессоре нет разделения сегментов по типам хранимой в них информации.

На рисунке 3.5 показан формат дескриптора сегмента. Регистр PDR содержит информацию для организации системы защиты памяти.

Поле ACF управляет доступом в память, т.е. устанавливает права доступа или атрибуты защиты: 00 - сегмент не доступен ни для чтения, ни для записи; 01 - сегмент доступен только для чтения; 10 - не используется, так как является бессмысленным (запрет чтения с разрешением записи в сегмент) и 11 - сегмент доступен для чтения и для записи. Любой из этих атрибутов можно использовать как для сег-

ментов кода, так и для сегментов данных и стека. Значение поля ACF загружается при инициализации системы и может изменяться ОС в процессе загрузки дескрипторов сегментов. Нарушение прав доступа хотя бы по одному из установленных атрибутов защиты вызывает выработку вектора прерывания с номером 250.

Поле PLF указывает разрешенную длину сегмента в блоках. Максимальная разрешенная длина сегмента 2^7=128 блоков.

Поле ED указывает, в каком направлении расширяется сегмент. При ED=0 расширение сегмента происходит в сторону увеличения, а при ED=1 - в сторону уменьшения адресов, при этом содержимое поля PLF записывается в дополнительном коде (для ED=0 в прямом коде), т.е. бит ED выполняет функцию знакового разряда поля размера PLF сегмента и может служить указателем сегмента стека. Для стекового сегмента (ED=1) PLF=0 соответствует максимальному размеру сегмента, а для сегмента данных (ED=0) минимальный размер в один блок (64 байта).

Поле W выполняет функцию контроля записи в сегмент и устанавливается в 1 только аппаратно, если в сегмент была произведена хотя бы одна запись. В дальнейшем значение данного бита используется при дозагрузке дескрипторов сегментов, находящихся на диске в режиме виртуальной адресации при выполнении процедуры свопинга сегментов.

На рисунке 3.10 приведена функциональная схема системы защиты памяти процессора. Параллельно с преобразованием ЛА в ФА выполняется сравнение размера сегмента из поля PLF дескриптора с номером текущего блока | ЛА[12-6] | > | PLF[14-8] |, а также атрибутов защиты, декодируемых на дешифраторе для простоты понимания, с сигналами ~RD и ~WR и сигналами состояния процессора, определяющими тип цикла шины и состояние процессора в текущий момент времени (на схеме сигналы состояния не показаны).

При нарушении размера сегмента или хотя бы одного атрибута защиты вырабатывается сигнал сегментной ошибки, доступ к памяти блокируется через вход ~CS выборки кристалла ОП, формируется вектор прерывания с номером 250 для обращения к таблице векторов и переход на ППОП.

В состав УУП также входят два регистра ошибок SR0 и SR2 (на схеме рисунка 3.10 не показаны), которые используются ОС для анализа кода ошибки и причины прерывания по защите памяти (в отличие от процессора Intel используется один вектор прерывания для всех причин нарушения доступа к памяти).

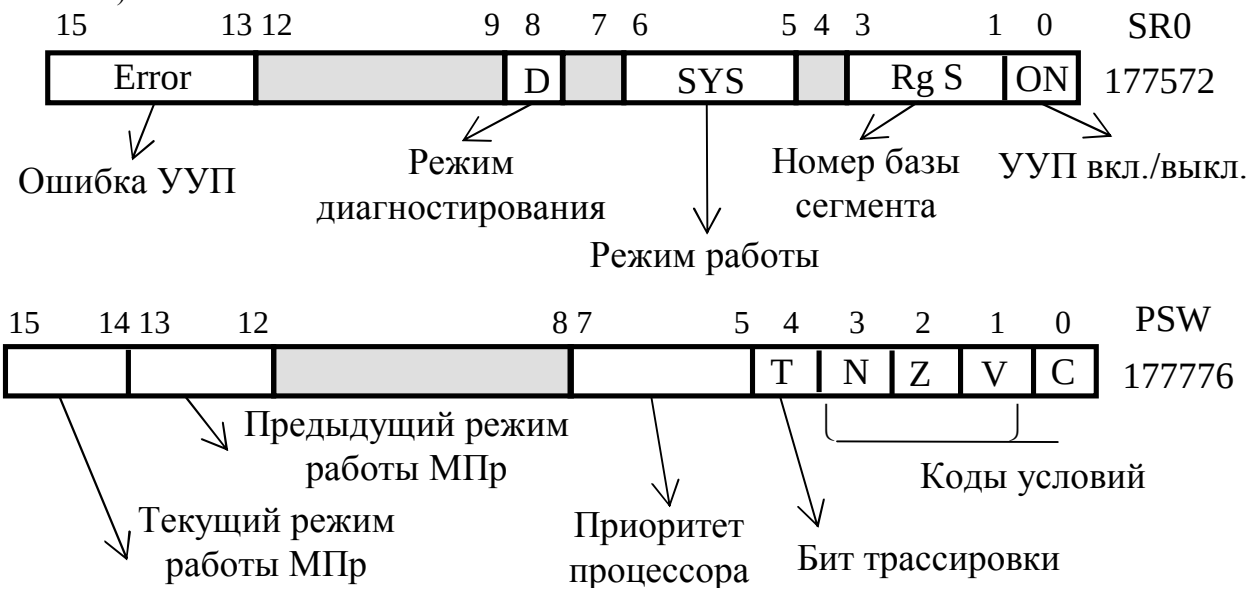


Рисунок 3.12 - Форматы регистра ошибок SR0 и слова состояния PSW процессора фирмы DEC

Регистр SR0 имеет ЛА 177572 на общей шине и указывает характер ошибки, состояние УУП и другую информацию, используемую при выполнении ППОП (рисунок 3.12).

Биты [15-13] SR0 указывают причину прерывания только для вектора 250, которые устанавливаются аппаратно:

- бит [15] устанавливается при обращении к неактивному сегменту (поле ACF=00 (запрет по чтению и записи) или в случае установки в PSW[15-14] недопустимого режима работы (PSW[15-14]=01 или 10);
- бит [14] устанавливается при нарушении длины (размера) сегмента;
- бит [13] устанавливается при попытке выполнить запись в сегмент, доступный только для чтения (ACF=01);
- биты [15-13] устанавливаются, если в поле ACF дескриптора установлен неиспользуемый атрибут защиты (ACF=10).

При обнаружении любой из перечисленных ошибок происходит прерывание по вектору 250. В случае обнаружения нескольких одновременных ошибок ППОП обрабатывает их в порядке их приоритета: 15, 14, 13 биты SR0. Также в регистре SR0 фиксируется режим работы процессора, при котором обнаружена ошибка (биты [6-5]), номер сегмента (дескриптора) (биты [3-1]), устанавливается режим диагностирования (бит [8]). В бите [0] SR0 программно устанавливается бит включения/выключения УУП.

В регистре SR2 фиксируется ЛА, при обработке которого обнаружена ошибка, поэтому запись в регистры SR0 и SR2 производится только по сигналу сегментной ошибки.

На рисунке 3.12 также показан формат слова состояния процессора, в котором указывается текущий и предыдущий режим работы процессора при переходе из одного режима в другой, приоритет процессора по отношению к внешним устройствам, значение бита трассировки Т и поле признаков слова состояния программы.

Процессор также контролирует диапазон базовых адресов сегментов и работу преобразователя ЛА в ФА путем формирования вектора прерывания ошибки 160 при возникновении единицы переноса из сумматора. Выработка такого вектора возможна только при искажении или неверной загрузке начального адреса сегмента, превышающего значение 7600 (или 177600) в дескрипторе.

Защита ОС от обращений из пользовательского режима к системным устройствам ввода-вывода реализована следующим образом. Все адресное пространство логических адресов ввода-вывода (160000-177776) разделено на две равные части, из которых ЛА 160000-167776 можно использовать для подключения дополнительных периферийных и других устройств пользователя, а ЛА 170000-177776 принадлежат системным устройствам, к которым может обращаться только ОС, а пользователь только по команде системного программного прерывания EMT (только к разрешенным подпрограммам). Тогда при начальной инициализации ОС в регистры дескрипторы номера сегмента 7 и системного, и пользовательского режимов, принадлежащих сегменту ввода-вывода, загружает базовые адреса 7600. В поле размера пользовательского дескриптора загружается максимальный разрешенный размер сегмента равный 63 (0111111), а системного дескриптора - 127 (1111111). Тогда при попытке пользователя из своей программы напрямую обратиться к устройству (порту) с адресом, начиная со 170000, вызовет выработку вектора прерывания 250, а в SR0 будет установлен бит [14] - нарушение зашиты памяти по допустимому (разрешенному) размеру.

4 Организация виртуальной памяти

При увеличении размера выполняемых программ и объема обрабатываемых данных возникает проблема размещения сегментов в ОП. Если размер одной или нескольких программ в мультипрограммном режиме работы превышает емкость физической ОП, то программисту приходится решать задачу обмена модулями (сегментами) программ и данных между ОП и вторичной (дисковой) памятью, т.е. программа должна быть оверлейной. При этом возникает другая проблема - фрагментации памяти, так как удаляемые из ОП сегменты программ и загружаемые на их место новые сегменты могут отличаться по размеру.

Для упрощения и автоматизации процедуры дозагрузки сегментов программ и освобождения памяти для них был предложен метод, называемый виртуальной памятью.

Виртуальной памятью называется видимое (доступное) программисту адресное пространство, представляющее собой адресное пространство оперативной (основной) памяти и быстродействующей дисковой памяти (обычно жесткого магнитного диска).

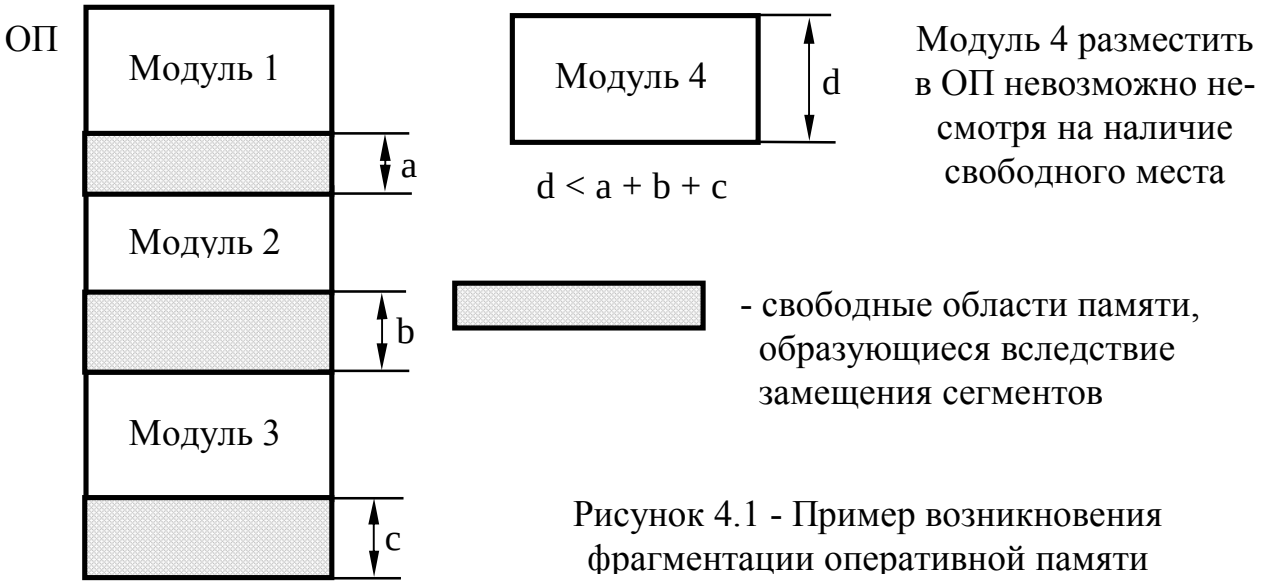


Рисунок 4.1 - Пример возникновения фрагментации оперативной памяти

Адреса виртуального адресного пространства называют виртуальными адресами, а адреса ОП - физическими адресами.

Как и при использовании логических адресов, программист пишет программы в виртуальных адресах (ВА), поэтому также требуется процедура преобразования ВА в ФА.

4.1 Страничная организация памяти

Организация виртуальной памяти (ВП) имеет цель:

Предоставить программисту достаточно большое (кажущееся) адресное пространство памяти.

Предоставить программисту область памяти, физически не используемую из-за фрагментации, в виде логически непрерывного пространства.

В такой системе ОП, внешняя дисковая память и программа разбиваются на части равной величины, называемые страницами. В начальный момент времени все страницы, представляющие программу, хранятся на дисковой памяти и по мере необходимости затребованные страницы помещаются в ОП. Размер страницы обычно выбирают в пределах 2-4 Кбайт.

Таким образом, виртуальное пространство является линейным, разбитым на страницы одинакового размера, что исключает фрагментацию ОП при дозагрузке страниц из дисковой памяти в ОП. Если при запросе доступа к странице она отсутствует в ОП, то страница загружается из дисковой памяти

в ОП. Такая ситуация называется страничным сбоем. Если в ОП нет свободной области памяти для дозагрузки страниц, то необходимо предварительно освободить область ОП путем удаления одной из страниц из ОП на дисковую память. Удаляемая страница определяется алгоритмом замещения страниц.

Для определения момента загрузки страницы в ОП используются в основном два метода:

- выборка (замещение) по запросу, когда загрузка выполняется после страничного сбоя;
- предвыборка, когда загрузка выполняется заранее по предположению о возникновении страничного сбоя.

Для преобразования ВА в ФА можно выделить два метода:

- с использованием таблицы прямого преобразования:
- с использованием буфера ассоциативного преобразования.

Первый способ реализуется аппаратно путем преобразования номера страницы в адрес ОП (страничный кадр) через таблицу страниц, хранимую в ОП, объем которой равен числу страниц всего виртуального пространства (может достигать до 4 - 8 М страниц).

В таблице страниц для каждой страницы хранится ее дескриптор, т.е. указатель, описывающий местоположение страницы, разрешенные операции с ней (права доступа к странице) и другая вспомогательная информация. Например, пусть дескриптор страницы содержит следующие поля: бит d достоверности страницы, показывающий, существует или нет данная страница; P - бит присутствия, показывающий местонахождение страницы: 1 - в ОП, 0 - на диске; бит обновления страницы D, показывающий, было или нет обновление страницы (запись в страницу) в ОП, и используемый для предотвращения ненужной процедуры перезаписи страницы на диск в случае удаления ее из ОП; поле RWX, характеризующее, какие права доступа к странице (считывание/запись или и то, и другое) разрешены при обращении к странице; поле PAR - адрес страничного кадра (базовый начальный адрес страницы) и другая информация. На рисунке 4.2 показана схема преобразования ВА в ФА через одноуровневую таблицу страниц.

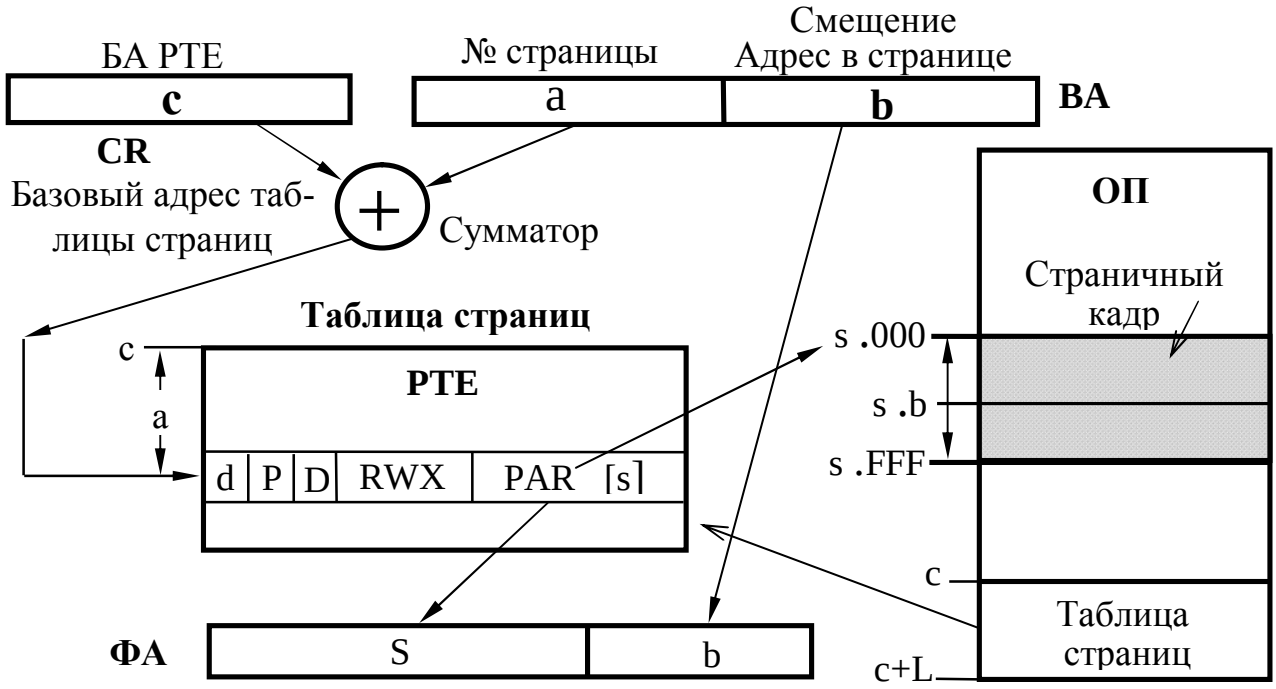


Рисунок 4.2 - Одноуровневая система преобразования виртуального страничного адреса в физический

Так как при одноуровневом преобразовании страничного ВА в ФА объем таблицы страниц может быть значительным (равен максимальному адресуемому числу страниц), в структуру введем регистр CR, в который при инициализации загружается базовый начальный адрес таблицы страниц (БА PTE), что делает ее перемещаемой в адресном пространстве ОП. Виртуальный адрес состоит из двух полей: номера страницы и адреса внутри страницы. По адресу, сформированному как сумма БА PTE и номер страницы, выполняется обращение к таблице страниц и выбирается дескриптор страницы. Если страница достоверна (d=1) и находится в ОП (P=1), то ФА формируется операцией конкатенации базового адреса страницы PAR [s] (адреса страничного кадра) и смещения [b] из ВА.

Основными недостатками данного метода являются:

♦ **низкое быстродействие,** так как для доступа к операнду необходимо два обращения к памяти: к таблице страниц за дескриптором и к ОП за операндом;

♦ **таблица страниц может занимать до нескольких Мбайт памяти**, так как число страниц достигает до 1 М и более дескрипторов, а каждый дескриптор требует до 4 байт памяти;

♦ в мультипрограммном режиме работы возникает **проблема перераспределения номеров страниц между задачами**, так как внутри каждой задачи страницы имеют номера с нулевого, а в общей таблице страниц номера могут совпадать, т.е. номер задачи должен входить в формат виртуального адреса, что увеличивает размер таблицы страниц, либо для каждой задачи необходимо хранить их начальные виртуальные адреса, загружаемые при переключении задач в специальный регистр настройки, а номер страницы для текущей задачи будет определяться как сумма номера страницы виртуального адреса и содержимого регистра настройки. Например, на диске все страницы имеют номера с нулевого по L. Для каждой задачи виртуальные адреса операндов также начинаются с нулевого.

Задача	Номера страниц задачи	Номера страниц на диске	Регистр настроек
0	0-23	0-23	0
1	0-36	24-60	24
2	0-21	61-82	61
3	0-47	83-130	83

Данная проблема решается путем организации мультипрограммной страничной виртуальной памяти.

Решение второй проблемы будет рассмотрено в дальнейшем на примере организации преобразования страничного линейного адреса в ФА для ЭВМ семейства IBM.

Метод ассоциативного преобразования позволяет существенно сократить время преобразования ВА в ФА. В структуру УУП вводится быстродействующая ассоциативная память небольшой емкости (8-128 дескрипторов), в которой хранятся дескрипторы наиболее часто используемых страниц. Тогда при первой попытке обращения к странице преобразование выполняется с помощью таблицы страниц из ОП, а все остальные через АЗУ. Замещение дескрипторов в АЗУ обычно осуществляется по алгоритму LRU, а сам ассоциативный буфер строится на основе частично-ассоциативного распределения. Ассоциативная память называется буфером предистории трансляции или ассоциативным буфером преобразования TLB.

Параллельно с формированием ФА в УУП выполняется контроль по полям прав доступа к странице, которые будут рассмотрены ниже.

Если бит присутствия в дескрипторе страницы P=0, то формируется прерывание особого случая неприсутствия страницы в ОП и выполняется процедура замещения страницы с диска в ОП (свопинг страницы).

Если бит обновления страницы D=1, то это означает, что в данную страницу была выполнена хотя бы одна запись (бит D устанавливается в "1" каждый раз, когда в страницу выполняется запись) и предварительно данная страница удаляется из ОП и переписывается на диск, а в ее дескрипторе изменяются соответствующие поля (P=0, D=0, в поле PAR помещается адрес местоположения страницы на диске). Далее выполняется замещение страницы, т.е. отыскивается ее местоположение на диске и она перемещается в ОП на место удаленной, а в дескрипторе устанавливаются соответствующие поля (d=1, P=1, в поле PAR загружается базовый адрес ОП страницы).

4.2 Мультипрограммная страничная виртуальная память

При использовании общей таблицы страниц в мультипрограммном режиме работы проблематично решить проблему защиты программ (задач) друг от друга и настраиваемости (использование регистра настройки требует дополнительного времени на формирование ФА). Поэтому операционная система должна выполнять функции перераспределения виртуального пространства задач (формирование таблицы настроек номеров страниц для каждой задачи).

В мультипрограммной страничной виртуальной памяти каждой задаче отводится свое линейное виртуальное пространство. Отличие от однозадачной виртуальной памяти заключается в том, что для каждой задачи отводится своя локальная таблица страниц. В структуру УУП вводится дополнительный регистр-указатель, в который при переключении задач из быстродействующей памяти загружается базовый (начальный) адрес таблицы страниц для данной задачи (рисунок 4.3).

Данный регистр выполняет функции регистра CR в однозадачной виртуальной памяти. Дальнейший процесс преобразования ВА в ФА аналогичен предыдущему алгоритму.

Так как каждая задача имеет свою таблицу страниц, то при такой организации невозможно ошибочное разрушение программ и данных со стороны других задач. Кроме того, поскольку программы любой задачи загружаются в независимое виртуальное пространство, начиная с нулевого номера страницы, необходимость в настройке отпадает.

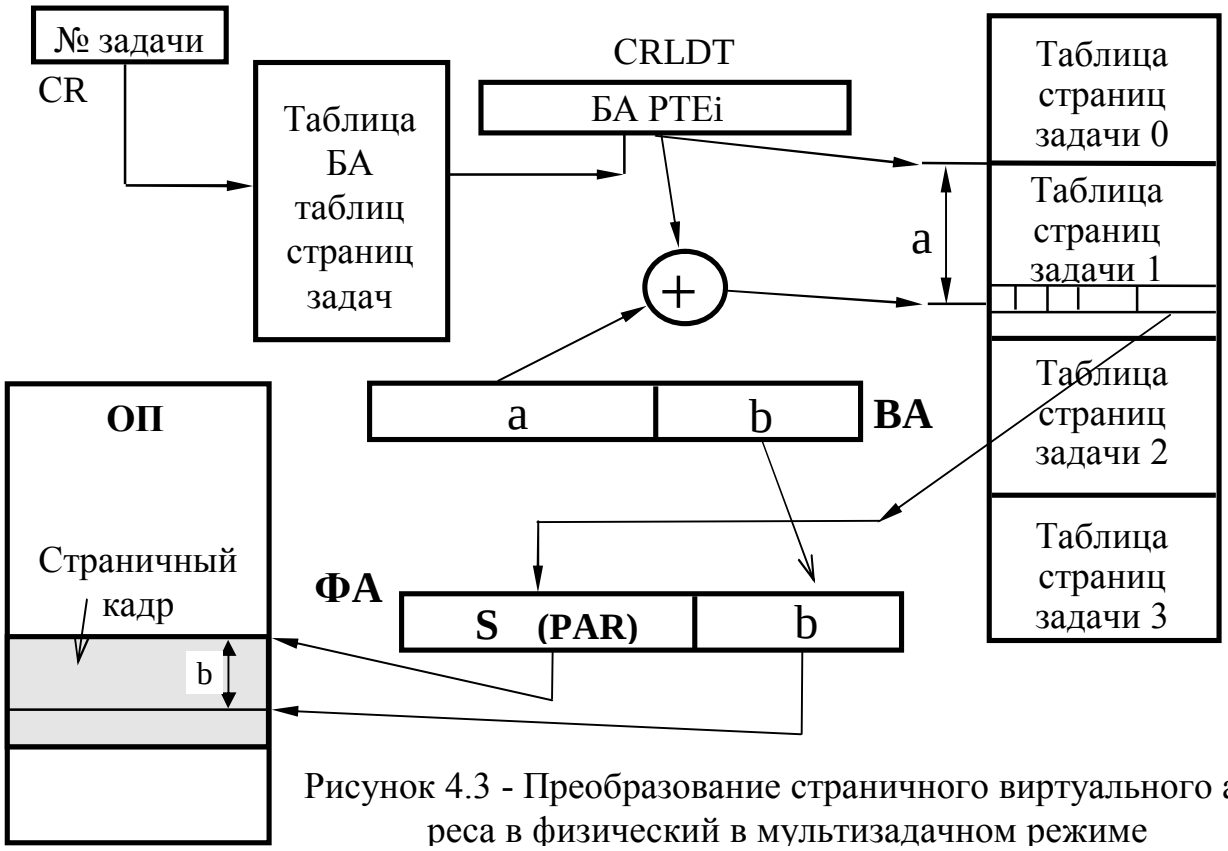


Рисунок 4.3 - Преобразование страничного виртуального адреса в физический в мультизадачном режиме

Однако такой подход имеет и один существенный недостаток: нет возможности использования несколькими задачами одних и тех же страниц (программ и данных), что требует дублирования ряда страниц, а следовательно, приводит к неэффективному использованию емкости ОП. Этот недостаток можно устранить при сегментной организации памяти.

4.3 Сегментная организация виртуальной памяти

Как было показано ранее, при сегментной организации памяти единицей загрузки в ОП задачи является сегмент. При этом задача может состоять из нескольких сегментов. В свою очередь сегмент задачи состоит из кодовой части (команд), данных и стековой области, которые разделяются по своим одноименным сегментам: сегменты кода, сегменты данных и сегмент стека. В отличие от страниц сегменты представляют собой совокупность линейных адресов переменной длины (рисунок 4.4).

Виртуальный адрес определяется номером сегмента и адресом внутри сегмента. Для преобразования ВА в ФА используется сегментная таблица (рисунок 4.5). Процедура преобразования виртуального сегментного адреса в ФА выполняется аналогично преобразованию виртуального страничного адреса в ФА (рисунок 4.2). Основные отличия заключаются в следующем:

♦ разрядность поля смещения в ВА при сегментной организации памяти имеет существенно большую разрядность, чем страница, и позволяет адресовать линейное пространство от 1 байта (блока) до 64 Кбайт (при сегментной адресации) или до емкости ОП (при линейной адресации от одного блока до k блоков), т.е. размер сегмента не является фиксированным;

♦ формирование ФА осуществляется не путем операции конкатенации смещения и базового адреса сегмента, а их сложением, так как базовый адрес сегмента не является кратным смещению.

Задача состоит из n сегментов различного типа: сегменты кода, данных и сегмент стека. Нумерация сегментов внутри каждой задачи начинается с нулевого номера.

Сегменты имеют разный размер и атрибуты защиты (права доступа).

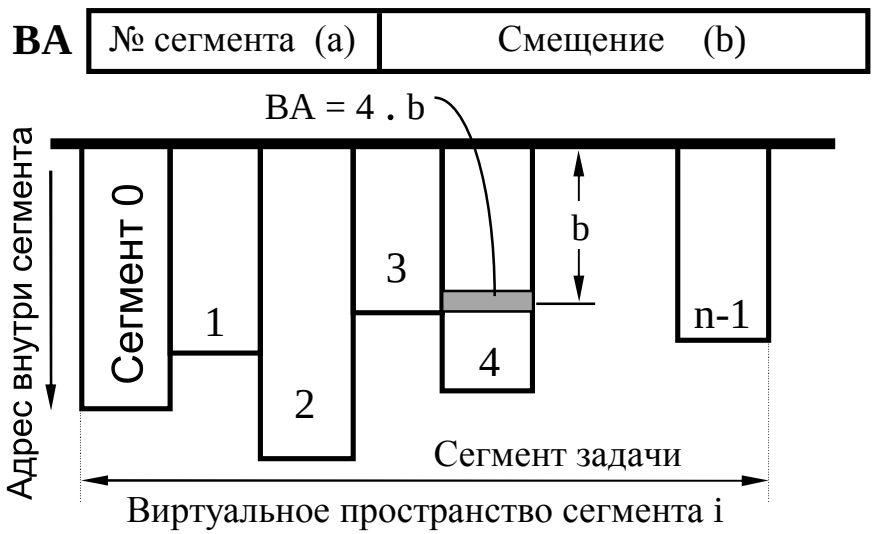


Рисунок 4.4 - Сегментное виртуальное адресное пространство задачи

Замещение сегментов при их отсутствии в ОП выполняется аналогично страницам. При доступе к сегментам выполняется дополнительный контроль по размеру при обращении к таблице сегментов (размер таблицы сегментов значительно меньше таблицы страниц, определяется при инициализации

системы, может быть переменным и задается в поле L регистра базового адреса таблицы сегментов CR), а при обращении в дескрипторе сегмента в поле M задается его размер.

На рисунках 4.6 и 4.7 показаны примеры совместного использования сегментов несколькими задачами (процессами).

В первом варианте используется метод преобразования ВА в ФА через общую для всех задач таблицу дескрипторов сегментов (ее часто называют глобальной GDT). Данному методу присущи серьезные недостатки:

Проблема настраиваемости, т.е. перераспределения номеров сегментов между задачами, так как сегменты в таблице GDT имеют сквозную нумерацию от 0 до k для всех задач, а внутри задачи адресация выполняется также с нулевого сегмента (см. страничную организацию памяти), т.е. необходимо формировать базовые номера сегментов задач при распределении ресурсов между задачами, а перед переключением задач выполнить перезагрузку регистра настроек задач.

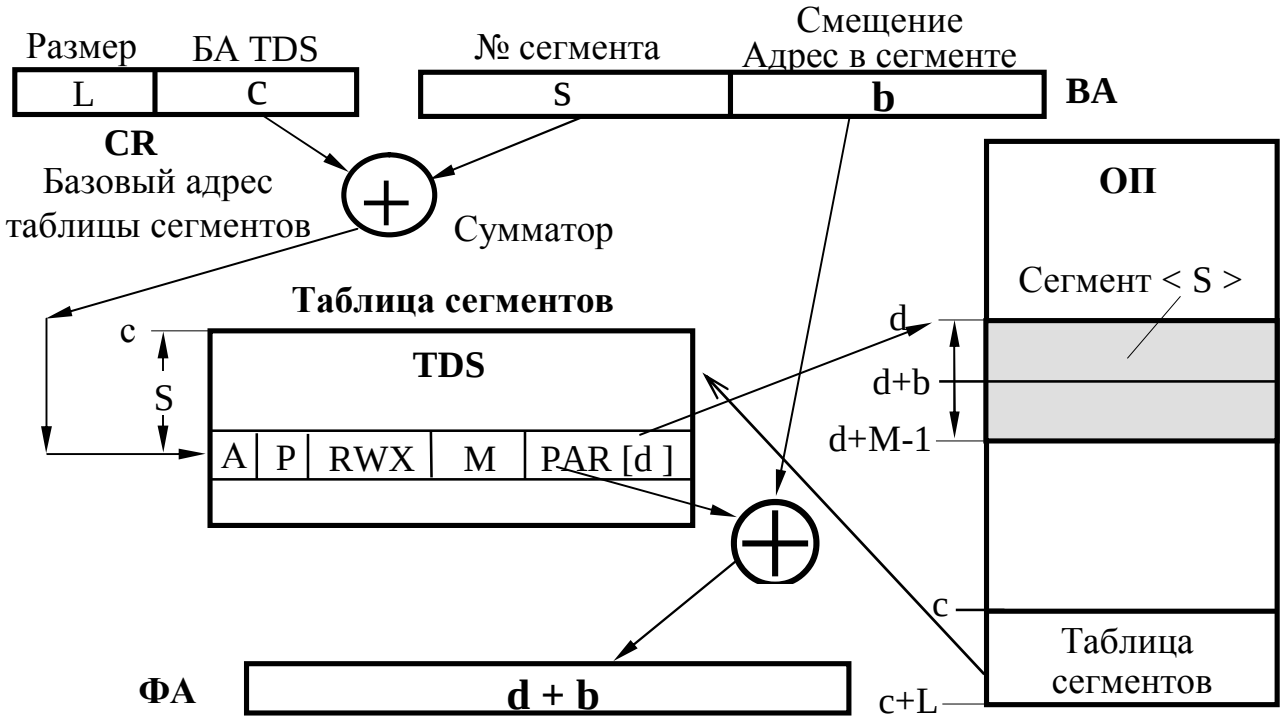


Рисунок 4.5 - Преобразования виртуального адреса в физический при сегментной организации памяти

2. В мультипрограммном режиме работы любая задача имеет доступ к любому сегменту таблицы GDT, даже к тем, которые могут использоваться только одной конкретной задачей. Это означает, что невозможно организовать эффективную защиту сегментов задач от случайного или преднамеренного доступа со стороны других пользователей.

3. После выполнения нескольких замещений сегментов с диска в ОП возникает проблема фрагментации, так как сегменты могут иметь различный размер, что приводит к неэффективному использованию адресного пространства ОП.

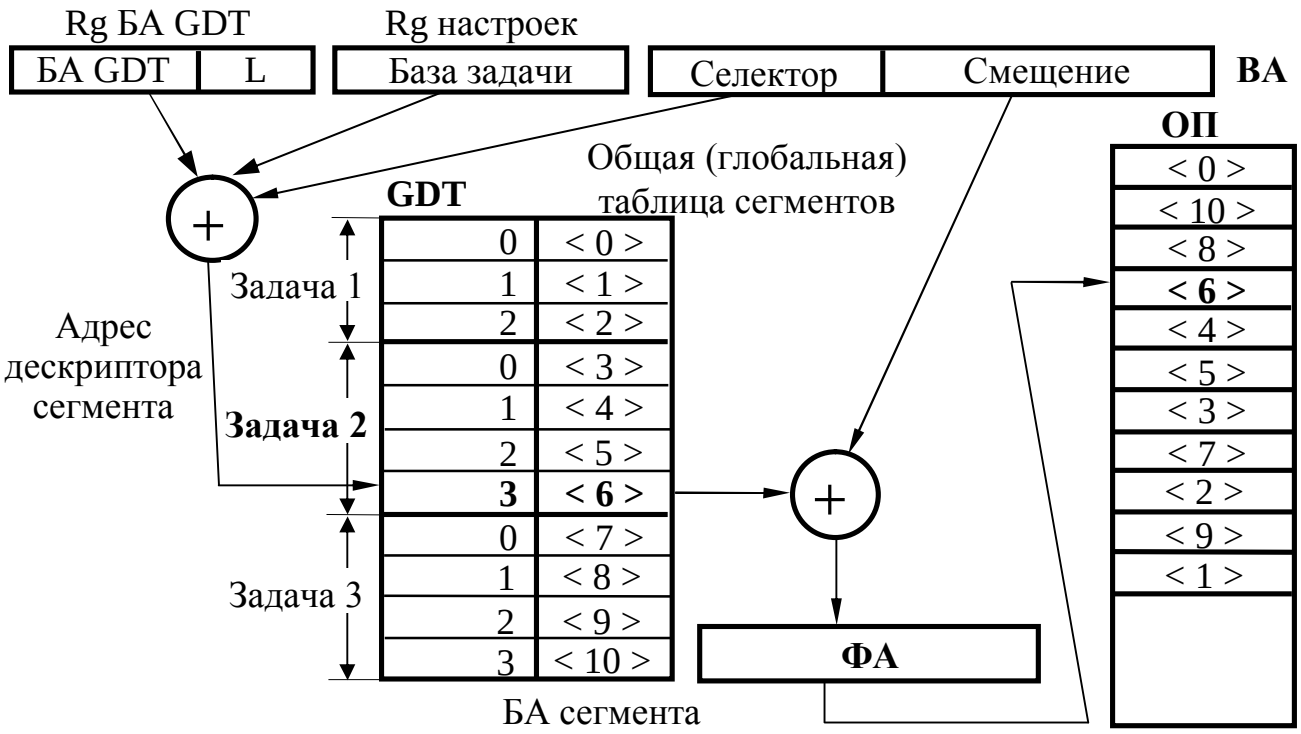


Рисунок 4.6 - Организация доступа к сегментам из разных задач через общую (глобальную) таблицу сегментов (дескрипторов)

При втором варианте используется метод, аналогичный страничной организации памяти через разделение таблицы сегментов между задачами (использование локальных таблиц сегментов задач LDT). С номером задачи однозначно связана своя таблица сегментов задач, в каждой из которых сегменты распределены с нулевого номера.

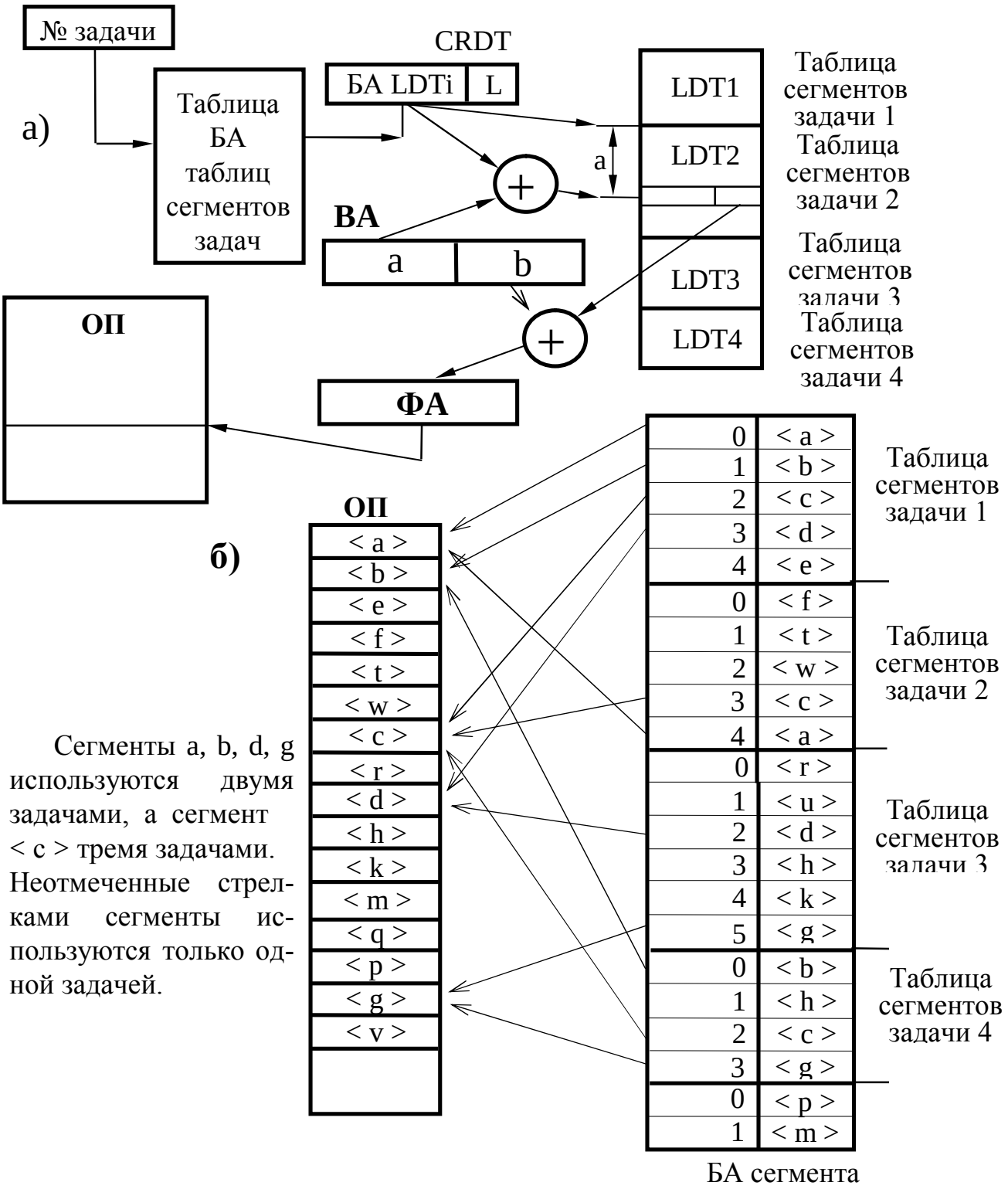


Рисунок 4.7 - Преобразование сегментного виртуального адреса в физический в мультипрограммном режиме (а); совместное использование сегмента несколькими задачами (б)

В данном методе устранены недостатки 1 и частично 2, но проблема фрагментации памяти остается.

Решением проблемы фрагментации является сегментно-страничная организация памяти (в дальнейшем будет рассмотрен вариант совместного использования двух типов таблиц сегментов: глобальной GDT и локальных LDT, используемых в ЭВМ семейства IBM).

4.4 Сегментно-страничная организация памяти

Простейший вариант сегментно-страничной организации памяти можно представить в виде следующей структуры: программа, как и при сегментной организации памяти, состоит из сегментов кода, данных и стека переменной длины, а те в свою очередь делятся на страницы фиксированной длины.

Тогда формат ВА можно представить в виде трех полей: поле номера сегмента S, поле номера страницы P и поле смещения (адрес внутри страницы).

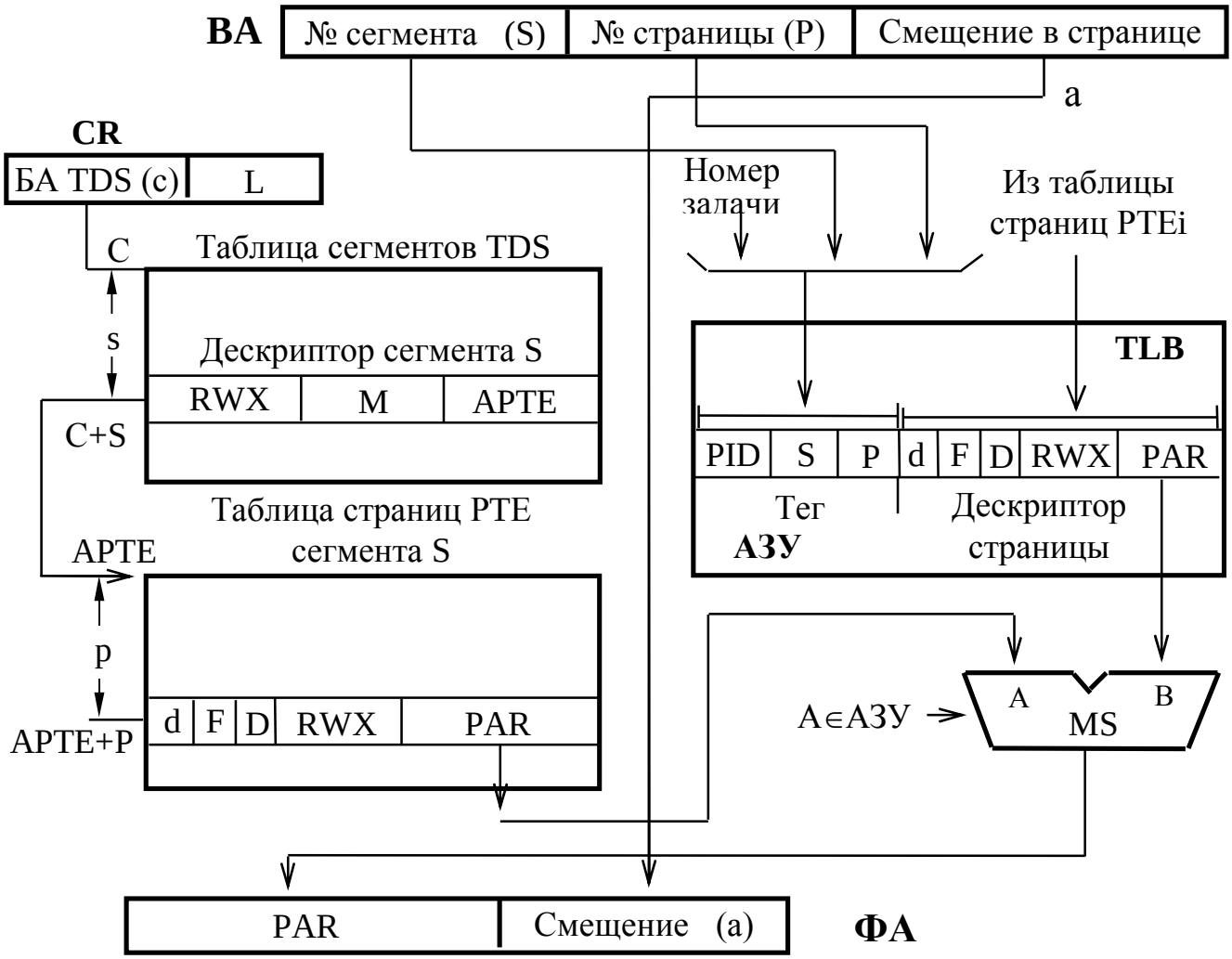


Рисунок 4.8 - Преобразование ВА в ФА при сегментно-страничной организации памяти с использованием ассоциативного буфера TLB

На рисунке 4.8 приведена схема преобразования ВА в ФА на основе одной глобальной таблицы сегментов с использованием быстродействующего ассоциативного буфера TLB для хранения дескрипторов активных страниц. Первоначально по полям номера задачи, сегмента и страницы осуществляется ассоциативный поиск дескриптора страницы в АЗУ. Если A∈АЗУ, то выполняется контроль по атрибутам защиты страницы, и если доступ к странице разрешен, то формируется физический адрес опера-

цией конкатенации поля PAR (базового адреса страничного кадра) из дескриптора страницы АЗУ и смещения из ВА и устанавливаются необходимые биты в дескрипторе страницы.

Если А∉АЗУ, то преобразование ВА в ФА выполняется через таблицы сегментов и страниц, хранимых в ОП. Если все биты достоверности АЗУ d=1, то по одному из алгоритмов замещения страниц определяется кандидат на удаление из АЗУ. В регистре CR хранится начальный (базовый адрес) таблицы сегментов и ее размер. По номеру сегмента S из таблицы сегментов выбирается его дескриптор и выполняется контроль по правам доступа к сегменту и размеру, и если доступ не запрещен, то в поле АРТЕ дескриптора сегмента указан базовый номер таблицы страниц данной задачи, а по полю P виртуального адреса из таблицы страниц выбирается ее дескриптор и также проверяются права доступа к странице и определяется ее местонахождение по биту присутствия F.

Если страница находится в ОП, то формируется физический адрес слова операцией конкатенации поля PAR дескриптора страницы и смещения [a], а дескриптор страницы и тег виртуального адреса загружаются в АЗУ на место назначенного на удаление дескриптора.

Если запрашиваемая страница находится на диске (F=0), то выполняется процедура свопинга страницы по алгоритму, описанному выше.

4.5 Алгоритмы замещения страниц в виртуальной памяти

При отсутствии страницы в ОП (бит присутствия P=0), и если в ОП нет свободного места, то она замещает одну из страниц, находящихся в ОП. При этом, если в замещаемую страницу во время ее пребывания в ОП производилась запись, она должна быть передана в дисковую память.

Эти процедуры передачи информации, называемые **свопингом страниц**, вызывают простои процессора, поэтому следует стремиться уменьшить число таких операций во время выполнения программы.

При отсутствии страницы в ОП возникает особый случай прерывания, называемый **страничным сбоем**. Процедура удаления страницы из ОП называется процессом замещения страниц, а правило, по которому при возникновении страничного сбоя выбирается страница для удаления из ОП, - **алгоритмом замещения**. Таким образом, для повышения производительности процессора алгоритм замещения должен свести число замещений к минимуму.

На практике используют эвристические методы, использующие информацию об обращениях к страницам в прошедшие моменты времени, так как информация о потоке обращений в будущем отсутствует.

Можно выделить следующие алгоритмы замещения:

Алгоритм случайного замещения. Из ОП удаляется любая из находящихся там страниц (например, по счетчику номеров страниц, загруженных в ОП).

Алгоритм "первый пришел - первый вышел" (FIFO-алгоритм). Отсылается страница, дольше других находящаяся в ОП.

Алгоритм "последний пришел - первый вышел" (LIFO-алгоритм). Отсылается страница, позже других поступившая в ОП.

Алгоритм по времени неиспользования. Из ОП удаляется страница, наиболее давно неиспользовавшаяся, Для каждой страницы необходимо вычислять значения T1, T2, T3, ... , Tm, характеризующие времена неиспользования страниц, размещенных в ОП. Для этого каждой странице ставится в соответствие бит обращения A (бит неиспользования), который устанавливается при каждом обращении к странице. Для наблюдения отводятся интервалы времени, в течение которых процессор выполняет R команд и вырабатывается прерывание для начисления времени неиспользования. Если бит обращения

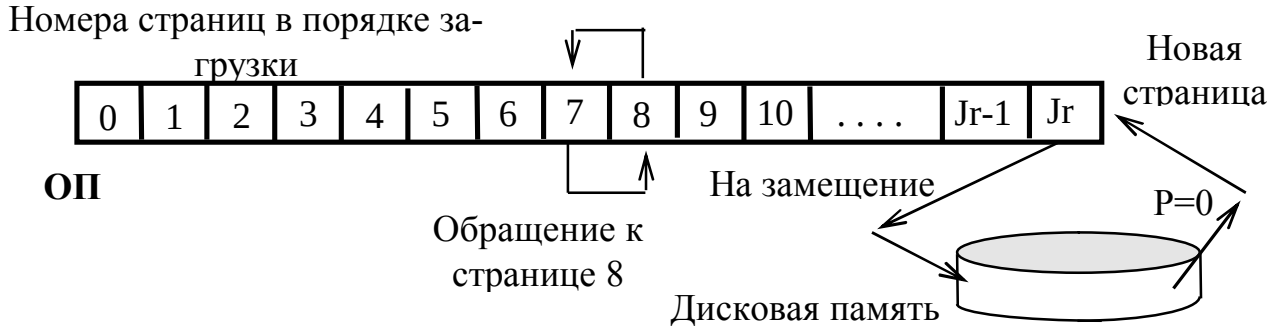
A=0 за это время (не было обращений к странице), то время неиспользования увеличивается на единицу Тн:=Тн + 1, записываемое в определенное поле дескриптора страницы. Удалению подлежит страница с максимальным временем неиспользования Тн.

5. Алгоритм "рабочий комплект". Страницы, находящиеся в ОП, использовавшиеся в течение заданного интервала времени по биту обращения A (A=1), образуют "рабочий комплект". Страницы, не вошедшие в "рабочий комплект" (с битом A=0), формируют две очереди страниц кандидатов на замещение:

1) Очередь страниц, в которые не вносились изменения (бит обновления D=0);
2) Очередь страниц, в которые вносились изменения (D=1);

Замещение производится по правилу «первый пришел из "рабочего комплекта" - первый ушел из ОП», при этом сначала подлежат замещению страницы из первой очереди.

6. Алгоритм "карабкающаяся страница". Страницы в ОП имеют последовательные номера Pi = 0, 1, 2, 3, . . . , Jr, которые при очередном обращении к странице меняются местами с соседней слева страницей или, другими словами, "карабкаются" к началу последовательности, подальше от ее конца, куда происходит замещение при страничном сбое.



7. **Алгоритм по вероятности использования** (псевдо LRU-стека). Аналогично алгоритмам 4 и 6 с помощью бита обращения A устанавливается активность страниц за время выполнения R команд. Далее все активные страницы перемещаются в начало списка страниц, хранимых в ОП. Удалению подлежит страница, замыкающая список. Данный метод является наиболее эффективным, но требует большого времени на процесс переупорядочивания списка страниц.

Многие авторы интерпретируют методы "рабочий комплект" и по времени неиспользования как алгоритм псевдо LRU-стека, так как они имеют приблизительно одинаковую эффективность определения кандидата на замещение.

4.6 Защищенный режим работы процессора фирмы Intel

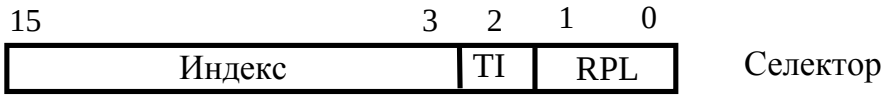
В реальном режиме максимальный объем адресуемой ОП составляет 1 Мбайт, а размеры сегментов не превышают 64 Кбайт. В защищенном режиме линейное адресное пространство увеличивается до 4 Гбайт (2^{32} байт). С точки зрения программистов, защищенный режим предоставляет большее адресное пространство и поддерживает новый механизм адресации, что позволяет выполнять более крупные программы и обеспечивает многозадачный режим работы.

В защищенном режиме логический адрес состоит из двух частей: 16-разрядного селектора и 32-разрядного исполнительного (эффективного) адреса или смещения. В результате преобразования получается 32-разрядный линейный адрес, который может быть использован в качестве физического адреса или дополнительно преобразован в физический адрес с применением механизма страничной организации памяти.

16-разрядный селектор находится в сегментном регистре, а смещение либо вычисляется в соответствии с режимом адресации операнда (прямая, база + смещение и другие), либо находится в регистре (IP, BP и других) и называется эффективным или исполнительным адресом. Селектор, также как и в реальном режиме, определяет базовый адрес сегмента, к которому для получения ФА прибавляется смещение, но базовый адрес получается посредством обращения (индексирования) к таблице памяти дескрипторов (в реальном режиме базовый адрес хранится непосредственно в регистре селектора CS, DS, SS или ES).

4.6.1 Структура регистров селектора и дескриптора

Селектор защищенного режима состоит из трех полей: запрашиваемого уровня привилегий RPL, индикатора таблицы TI и индекса.



Поле RPL не участвует в преобразовании адреса, а используется для защиты памяти и будет рассмотрено ниже. Поле индикатора таблицы дескрипторов TI показывает, какая из двух таблиц привлекается для поиска базового адреса. При TI=0 используется глобальная таблица дескрипторов GDT, которая является единственной и разделяется между всеми задачами. Если TI=1, то используется одна из локальных дескрипторных таблиц LDT, причем каждая задача имеет свою LDT. Следовательно, базовые адреса сегментов, разделяемых всеми задачами, хранятся в GDT, а базовые адреса «частных» сегментов каждой задачи находятся в своей LDT задачи. В GDT, кроме базовых адресов «общих» сегментов, также может храниться информация о состоянии задач и дескрипторах таблиц LDT (базовые адреса таблиц LDT задач, которые могут располагаться по любым адресам памяти) – отсюда и название «глобальные». В совокупности таблицы GDT и LGT обеспечивают защиту и изоляцию сегментов задач, а также разделение глобальных данных.

Рисунок 4.9 - Формат дескриптора сегмента

Поле индекса селектора служит индексом (смещением) дескриптора выбранной таблицы. Каждый элемент таблицы, описывающий сегмент или задачу, называется дескриптором и имеет длину 8 байт. Формат дескриптора показан на рисунке 4.9. В описание сегмента включается базовый адрес сег-

мента, размер сегмента, тип (определяющий целевое использование сегмента: программа, данные или служебный объект), уровень привилегий и дополнительную информацию о состоянии, т.е. все то, что необходимо знать о сегменте. Число дескрипторов в системе практически не ограничено, а размер таблицы дескрипторов указывается в специальном регистре описания местоположения таблицы дескрипторов, доступ к которой контролируется по размеру таблицы. Некоторая путаница полей дескриптора объясняется расширением разрядности полей базового адреса и предела для старших моделей микропроцессоров и необходимостью жесткой фиксации полей по разрядам дескриптора для обеспечения программной совместимости с младшими моделями МПр.

Базовый адрес – определяет любой 32-разрядный начальный адрес сегмента или таблицы LDT в линейном адресном пространстве 4 Гбайт.

Предел - 20-разрядное поле, определяющее размер сегмента или таблицы LDT в байтах или в страницах минус 1 и определяет границу сегмента относительно базового адреса. Таким образом, размер сегмента ограничен 1 М элементов, а размер элементов задается специальным битом гранулярности G или дробности (бит 55 дескриптора). Элементами сегмента могут быть байты (G=0, максимальный размер сегмента 1 Мбайт) или страницы по 4 Кбайт (G=1, максимальный размер сегмента 1 Мбайт х 4 Кбайт = 4 Гбайт, т.е. всему линейному адресному пространству памяти процессора).

Байт 5 дескриптора сегмента содержит **права доступа** (Access Rights – байт AR). Рассмотрим формат этого байта (рисунок 4.9).

Бит присутствия P установлен в 1, если описываемый дескриптором сегмент находится в физической памяти (ОП). В системе виртуальной памяти имеется возможность выполнять программы, большие, чем размер физической памяти, посредством автоматической пересылки частей программы (сегментов и страниц) между дисковым накопителем и ОП. Отсюда, во всех дескрипторах сегментов, находящихся в ОП, бит присутствия P=1, а для временно неприсутствующих в ОП P=0.

При обращении к дескриптору с битом P=0 возникает особый случай неприсутствия сегмента в ОП, и ОС либо отыскивает свободную область памяти, либо назначает один из сегментов кандидатом на удаление и копирует его на диск, а с диска в ОП загружается запрашиваемый сегмент и устанавливаются значения полей дескриптора для данного сегмента (размер, P=1 и т.д.). Такая процедура обычно называется свопингом или «подкачкой».

Бит доступа или обращения A автоматически устанавливается в 1, когда осуществляется обращение к данному сегменту по чтению или по записи, что позволяет ОС ассоциировать с каждым сегментом приблизительное время его последнего использования. ОС через определенные интервалы времени сбрасывает значение битов A во всех дескрипторах сегментов, предварительно выполнив инкремент времени последнего использования сегментов с битом A=1, что позволяет по значению бита A приблизительно оценить активность данного сегмента после того, как программа последний раз сбросила бит A в нуль (аналог бита неиспользования для кэш-памяти). Бит A (время последнего использования) совместно с другими битами применяется для назначения кандидата на удаление из ОП на диск неактивного сегмента с P=1.

Двухбитное поле уровня привилегий DPL определяет уровень привилегий сегмента. Привилегии являются составной частью защиты памяти и подробно рассматриваются ниже.

Бит системный S или сегмента описывает назначение сегмента. При S=1 сегмент является сегментом кода или данных (CS, DS, SS, ES), а при S=0 дескриптор описывает системный объект (например, для загрузки сегмента LDT) или нового селектора в регистры CS, DS, SS или ES.

Трехбитное поле типа TYPE определяет целевое назначение сегмента, задавая допустимые в сегменте операции, и служит для организации защиты памяти по доступу к сегменту.

Бит 43 поля прав доступа различает сегменты кода (1) и данных (0).

Для сегментов кода бит 42 называется **битом подчинения С** или согласования, который используется для организации колец защиты и позволяет намеренно лишить соответствующий сегмент кода защиты по уровням привилегий. Такое средство удобно для организации в системе библиотек, программы которых должны быть доступными всем выполняемым в системе задачам. Библиотечные программы оформляются как подчиненные сегменты программ, и они могут быть вызваны по команде CALL любой текущей задачей.

Для сегментов данных бит 42 называют **битом расширения вниз ED.** Он различает сегменты стека ED=1 и сегменты собственно данных ED=0 и управляет интерпретацией поля предела. Поэтому для сегментов данных предел указывает верхнюю границу сегмента относительно базового адреса. В сегментах стека с ED=1 предел определяет неадресуемую область сегмента, т.е. все смещения должны быть строго больше предела (в предыдущем случае меньше или равно пределу). При нарушении границ сегмента вырабатывается сигнал прерывания и доступ к ОП блокируется.

Таким образом, в сегменте данных минимальный адрес задается полем базового адреса дескриптора, а максимальный полем предела + базовый адрес. Для сегментов стека минимальный адрес определяется полем предела + базовый адрес, а максимальный адрес равен базовому адресу + дополнение предела (дополнительный код предела). Тогда максимальный размер стека получается, когда предел равен нулю.

Также необходимо учитывать диапазон изменения предела для сегментов стека. Если в дескрипторе стека бит D=0, то предел определяется в байтах (0FFFFh) (64 К – 1), а при D=1 (0FFFFFFFh) (4 Г – 1).

Бит 41 поля прав доступа для сегмента кода называется битом считываемого R сегмента, а для сегментов данных битом разрешения записи W.

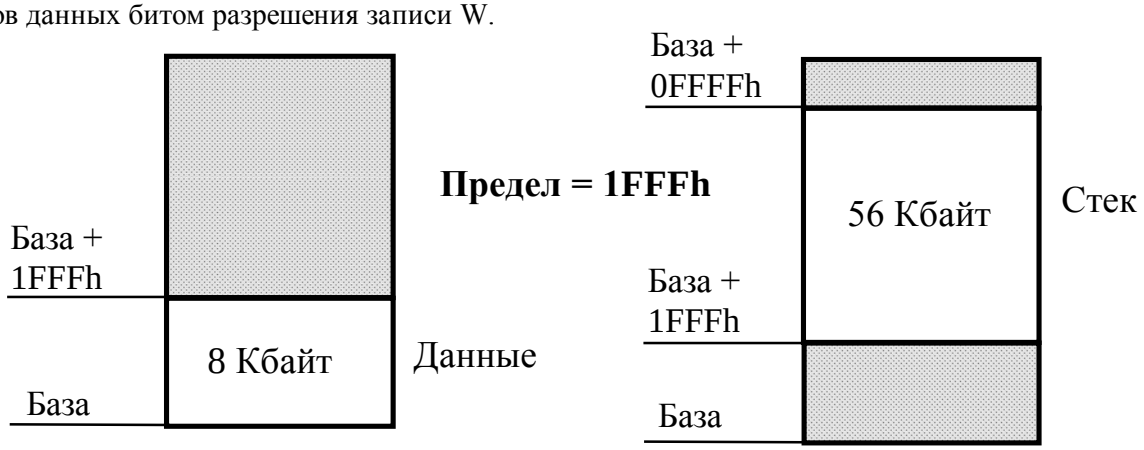


Рисунок 4.10 - Интерпретация поля предела для сегментов данных и стека

Если бит R=1 для сегмента кода, то наряду с обращением к такому сегменту для выполнения команд разрешается и обращение для их считывания. Если R=0, любая попытка считать из сегмента команду вызывает прерывание по защите памяти. Любая попытка записать информацию в исполняемый сегмент (кода) по умолчанию также приводит в выработке прерывания по защите памяти и доступ к памяти (запись в память) блокируется. При R=0 сегмент должен содержать «чистый» код (только команды), иначе данные невозможно будет прочитать.

Если бит W=1 для сегментов данных (DS, SS, ES), то помимо считывания из сегмента данных разрешена и запись в него. Если же W=0, то любая попытка записи в сегмент вызывает прерывание (особый случай защиты) и блокировку доступа к памяти по записи (такие сегменты могут содержать

таблицы констант, общесистемные справочные данные, информацию о состоянии, базы данных с запретом обновления записей и т.д.).

С системными объектами нельзя явно производить операции считывания, записи или выполнения, а они используются в процессе преобразования адреса и, как правило, принадлежат привилегированным командам.

4.6.2 Преобразование логического адреса в физический

Как было отмечено ранее, преобразование ЛА в ФА в защищенном режиме осуществляется через глобальные и локальные дескрипторные таблицы, в которых хранится базовый адрес сегмента и сопутствующая ему информация для обеспечения защиты сегментов.

Глобальная дескрипторная таблица GDT может использоваться всеми задачами для обращения к сегментам памяти в двух направлениях:

- для хранения дескрипторов базовых адресов сегментов, разделяемых всеми задачами (общие части программ и данных, используемых большинством задач);
- для хранения дескрипторов базовых адресов таблиц LDT, в которых хранятся дескрипторы базовых адресов сегментов для каждой задачи, т.е. нахождение базового адреса сегмента локальной таблицы осуществляется косвенно через глобальную таблицу GDT.

Таблица GDT может размещаться в ОП в произвольной области памяти, а ее местоположение и размер задается в специальном 48-разрядном программно доступном регистре RgGDT (рисунок 4.11), 32 бита которого определяют линейный базовый адрес местоположения таблицы GDT и 16 бит - предел (размер) таблицы с байтной грануляцией. Значение предела L связано с числом N дескрипторов в таблице соотношением L=8 x N - 1.

В мультизадачной системе для каждой задачи в дополнение к таблице GDT можно построить свою локальную дескрипторную таблицу LDT, описывающую сегменты, доступные только для этой конкретной задачи.

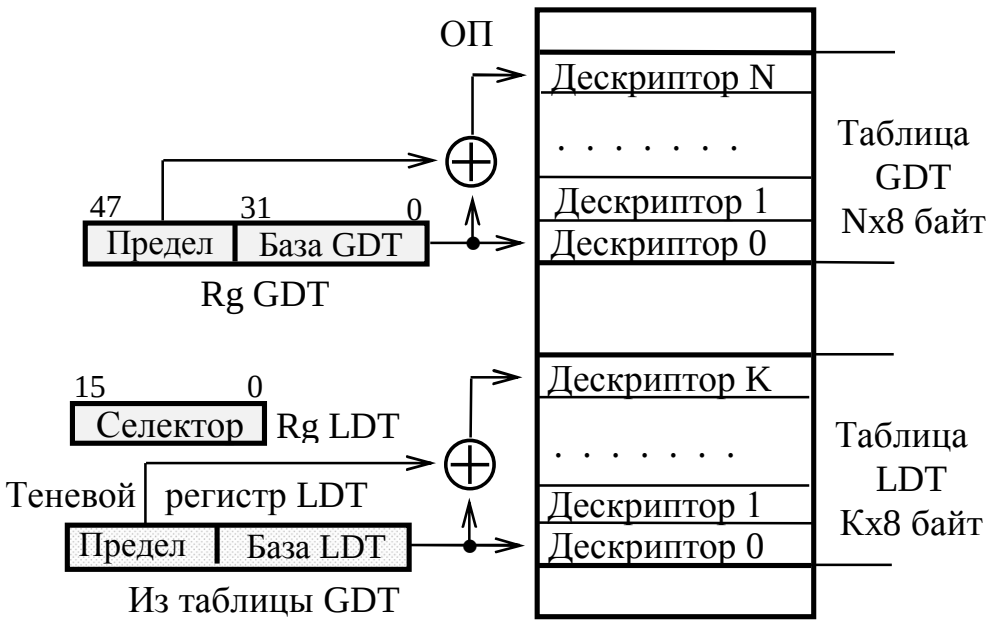


Рисунок 4.11 - Таблицы дескрипторов и системные регистры

Таблицы LDT также могут размещаться в произвольной области ОП, поэтому для определения их местоположения в состав УУП входит программно доступный 16-разрядный регистр LDT (RgLDT), являющийся селектором для каждой таблицы LGD и загружаемый ОС новым селектором при каждом переключении задач. То есть в каждый момент времени RgLDT хранит селектор сегмента, содержащего текущую таблицу LDT.

Таблицы LDT не являются обязательными и создаются по мере необходимости. Таблицы LDT хранятся в сегментах памяти, а дескрипторы этих сегментов загружаются при инициализации системы в таблицу GDT.

Внутри процессора для повышения быстродействия при вычислении ФА к регистру RgLDT придается теневой регистр, в котором и хранится дескриптор LDT текущей задачи, выбираемый из таблицы GDT. При этом в дескрипторе LDT, хранимом в таблице GDT, биты [55-52]=0000, бит S=0 (системный дескриптор), тип дескриптора [43-40]=0010, в поле базового адреса находится базовый адрес таблицы LDT, а в поле предел - размер таблицы LDT для данной задачи.

Рассмотрим процесс преобразования ЛА в ФА на примере выполнения команды MOV EAX,[ECX][ESI+20h]. Так как в команде нет специальных указаний об использовании сегмента данных, то значит она обращается к текущему сегменту, селектор которого находится в регистре DS. Пусть DS=00000000 00011**0**XXb. Так как бит TI=0, дескриптор сегмента находится в таблице GDT и имеет номер три. Без страничного преобразования алгоритм получения ФА будет следующий:

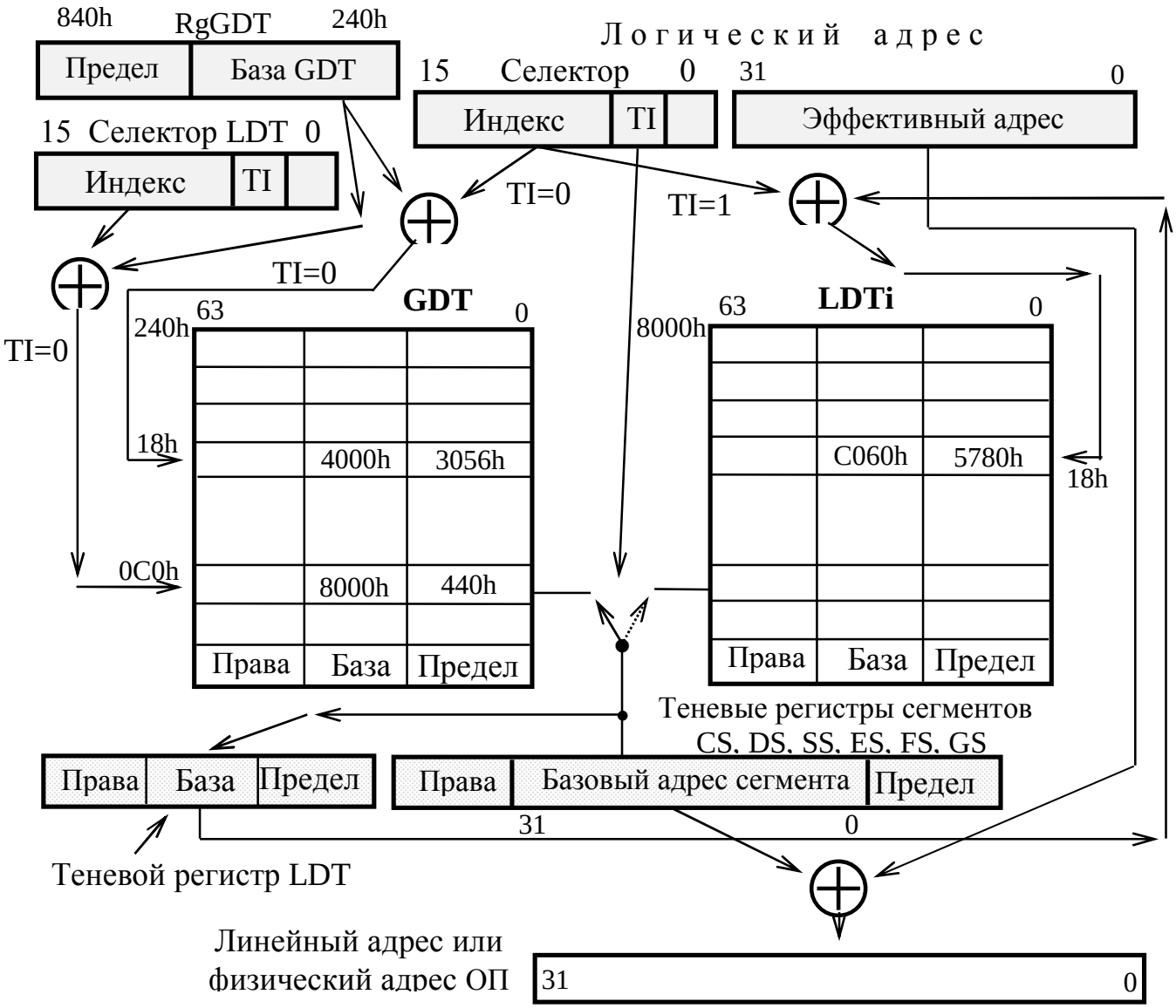


Рисунок 4.12 - Преобразование логического адреса в линейный (ФА)

1. Вычислить эффективный адрес EA= (ECX) + (ESI) + 20h.

2. Сложить содержимое поля базового адреса таблицы GDT из RgGDT (240h) с индексом из селектора DS, сдвинутым на три разряда влево (18h) (с контролем по размеру: предел RgGDT ≥ индекса селектора (840h ≥ 18h)), и по вычисленному адресу из ОП (таблицы GDT) выбрать дескриптор сегмента.

3. Просуммировать эффективный и базовый адреса сегмента из выбранного дескриптора (EA + 4000h). В результате получится линейный адрес операнда.

4. Если не используется страничное преобразование линейного адреса в ФА, то полученный адрес является физическим для ОП, и после выборки операнда из ОП по данному адресу следует поместить двойное слово в регистр EAX.

Если селектор DS=00000000 00011**1**XXb, т.е. TI=1, то дескриптор сегмента находится в таблице LDT. При этом предварительно перед переключением задач ОС выполняет загрузку в регистр LDT селектор текущей задачи (например, 00000000 11000**0**XX) (аналог номер текущей задачи). Тогда алгоритм вычисления ФА будет следующий:

1. Вычислить эффективный адрес EA= (ECX) + (ESI) + 20h.

2. Сложить содержимое поля базового адреса таблицы GDT из RgGDT (240h) с индексом из селектора LDT, сдвинутым на три разряда влево (0C0h) (с контролем по размеру: предел RgGDT ≥ индекса селектора (840h ≥0C0h)), и по вычисленному адресу из ОП (таблицы GDT) выбрать дескриптор, описывающий местоположение таблицы LDT текущей задачи.

3. Просуммировать базовый адрес из дескриптора таблицы LDT (8000h) с индексом из селектора DS, сдвинутым на три разряда влево (18h) (с контролем по размеру: предел дескриптора LDT ≥ индекса селектора (440h ≥ 18h)), и по вычисленному адресу из ОП (таблицы LDT) выбрать дескриптор сегмента.

4. Просуммировать эффективный адрес и базовый адрес сегмента из выбранного дескриптора таблицы LDT (EA + C060h). В результате получится линейный адрес операнда.

5. Если не используется страничное преобразование линейного адреса в ФА, то полученный адрес является физическим адресом ОП и после выборки операнда из ОП по данному адресу следует поместить двойное слово в регистр EAX.

На рис. 4.12 схематично показана процедура преобразования ЛА в ФА при TI=0 и TI=1 на численных примерах без контроля размера сегмента.

Описанные алгоритмы для обращения к сегменту требуют одного или двух обращений к ОП только для формирования ФА команды или данных, что существенно снижает производительность процессора по сравнению с реальным режимом. Так как обращения к ОП за командами и данными производятся чаще, чем изменения сегментов и переключения задач, то в защищенном режиме используют так называемое кэширование дескрипторов. Кэширование дескрипторов заключается в ассоциации с каждым сегментным регистром (CS, DS, SS, ES, FS, GS) и регистром LDT «теневого» регистра или кэш-регистра.

Рисунок 4.13 - Системные и теневые регистры процессора

Такие регистры невидимы и явно недоступны программам. При загрузке новых значений в регистры селекторов (CS, DS, SS, ES, FS, GS) и регистр LDT процессор автоматически считывает нужный дескриптор в соответствующий «теневой» регистр (пункт 2 первого алгоритма и пункты 2 и 3 второго). Поскольку теперь дескриптор находится внутри процессора, а не в ОП, то для получения линейного (физического) адреса памяти потребуется только сформировать эффективный адрес и просуммировать его с базовым адресом сегмента из нужного «теневого» регистра. В результате, если в программе редко модифицируются сегментные регистры и редко переключаются задачи, то преобразование ЛА в ФА будет выполняться приблизительно с такой же скоростью, как и в реальном режиме (пункты 1, 3, 4 первого алгоритма и пункты 1, 4, 5 второго).

При этом до загрузки селектора в сегментный регистр и кэширования дескрипторов осуществляется несколько предварительных контрольных проверок (защита памяти):

1. Контроль уровней привилегий в механизме защиты (на выполнение привилегированной команды и сравнение уровней привилегий).
2. Для предотвращения загрузки бессмысленных селекторов:
   - ♦ проверка поля индекса селектора на допустимость по размеру (пределу) доступа к таблице GDT или LDT, определяемой битом TI, так как пределы хранятся в дескрипторе или RgGDT вместе с базовыми адресами, иначе прерывание 13:

     RgGDT[47-32] ≥ ЛА Selector[15-3.000] для I=0;

     RgGDT[47-32] ≥ RgLDT Selector[15-3.000] для I=1,

где RgGDT[47-32] - поле предела глобальной таблицы дескрипторов;

ЛА Selector[15-3.000] - поле индекса селектора из адреса команды;

RgLDT Selector[15-3.000] - поле индекса селектора из регистра LDT;

   - ♦ для сегментных регистров данных (DS, ES, FS, GS) тип дескриптора должен разрешать выполнение/считывание из сегмента, т.е. не допускаются только выполняемые сегменты, когда данные в виде констант размещаются в сегменте кода CS, иначе прерывание 13:

     SOds = RgD[43] & ~RgD[41], TYPE = 1X0,

где RgD[43,42,41] - значение поля TYPE байта прав доступа (рис. 4.9);

SOds - сигнал сегментной ошибки при обращении к сегменту данных;

   - ♦ для селектора регистра SS в сегменте должны быть разрешены операции считывания и записи, иначе прерывание 12:

     SOss = ~RgD[43] & RgD[42] & ~RgD[41], TYPE = 010,

где SOss - сигнал сегментной ошибки при обращении к сегменту стека;

   - ♦ сегмент CS должен быть обязательно исполняемым (программой), иначе прерывание: SOcs = Acs & ~RgD[43], TYPE = 0XX,

где SOcs - сигнал сегментной ошибки при обращении к сегменту кода;

Acs - признак того, что загрузка селектора будет выполняться в регистр сегмента кода CS (сигнал состояния ЦП);

   - ♦ на последней стадии, если селектор прошел все эти проверки, процессор анализирует значение бита присутствия P дескриптора на присутствие сегмента в памяти, т.е. P=1, иначе прерывание 11:

     SOp = ~RgD[47],

где SOp - сигнал сегментной ошибки при обращении к сегменту, отсутствующему в оперативной памяти;

RgD[47] - значение бита присутствия P сегмента из поля байта прав доступа регистра дескриптора (рисунок 4.9).

Если хотя бы одна проверка дает отрицательный результат, формируется особый случай и загрузка селектора не производится. В противном случае селектор загружается в соответствующий регистр селекторов, а выбранный дескриптор записывается в "теневой" регистр и выполняется стандартная процедура контроля атрибутов защиты сегмента и его предела:

* контроль атрибутов защиты сегмента кода:

◊ запрет записи в сегмент кода, TYPE = 1XX;

◊ запрет считывания из сегмента кода, TYPE = 1X0:

SOcs =(RgD[43] & WRM) V (RgD[43] & ~RgD[41] & S[110] & RDM,

где S[110] - код состояния ЦП при выполнении цикла шины считывания данных из памяти (для считывания команд имеется специальный цикл шины предвыборки команд с состоянием S[100]);

♦ контроль атрибутов сегментов данных по записи (кроме сегмента стека, в который запись всегда разрешена):

SOds = WRM & ~RgD[43] & ~RgD[41] & ~RgD[42], TYPE = 000;

* контроль сегмента по размеру:

RgD[51-48.15-0] ≥ ЛА[19-0] - для сегментов кода и данных (ED=0);

RgD[51-48.15-0] < ЛА[19-0] - для сегмента стека (ED=1).

TYPE	Сегмент	Атрибут защиты	TYPE	Сегмент	Атрибут защиты
000	Данных	Защита по Зп	100	Кода #	Считывание запрещено
001	Данных		101	Кода #	
010	Стека	Запрещенный атрибут	110	Кода * #	Считывание запрещено
011	Стека		111	Кода* #	

* - для сегментов кода без защиты по привилегиям;

- защита по записи в сегмент кода по умолчанию.

Техническая реализация формирования сигналов сегментных ошибок (векторов прерываний) не представляет сложности.

4.6.3 Страничное преобразование адресов

Для организации виртуальной памяти, позволяющей программисту использовать адресное пространство большее, чем физическая оперативная память, применяется страничная организация памяти. При этом полученный линейный адрес при сегментном преобразовании рассматривается в качестве виртуального адреса. В отличие от сегмента, размер которого может быть практически любым, емкость памяти, отводимой под страницу, является фиксированной (4 Кбайта).

Таким образом, при страничном преобразовании все адресное пространство процессора 4 Гбайт разбивается на 1 М страниц по 4 Кбайт каждая. Физическая память (ОП) также разделяется на страницы (называемые страничными кадрами) с тем же размером 4 Кбайт. Так как физическое адресное пространство (1-32 Мбайт) значительно меньше пространства виртуальной памяти (4 Гбайт), то возникает задача перемещения затребованных страниц с диска в ОП, предварительно удалив из ОП страницы, потерявшие активность (к которым не выполняется обращение). При этом прикладные программы не касаются процесса страничного преобразования.

Начальные адреса страниц должны быть выровнены по границам кратным 4 Кбайт, а границы сегментов могут быть произвольными. Поэтому в принципе сегменты не обязательно выравнивать по границам страниц, так как при преобразовании адресов они работают с разными таблицами (сегменты определяются дескрипторными таблицами GDT и LDT), а страницы определяются набором таблиц страниц.

Также заметим, что при страничной организации памяти есть только два уровня привилегий (а не четыре, как при сегментации), которые называются уровнем пользователя и супервизора (уровень 3 - уровень пользователя, а уровни 0, 1 и 2 - уровень супервизора).

В процессе страничного преобразования необходимо 20 старших бит линейного адреса (номер виртуальной страницы) преобразовать в 20-разрядный физический адрес этой страницы, а младшие 12 бит линейного адреса (смещение в странице) остаются неизменными и указывают на номер байта в странице. При одноэтапном преобразовании потребуется линейная таблица, содержащая 1 М элементов (4 Г/4К=1М элементов) по 4 байта каждый (20 бит номера физической страницы плюс 12 бит дополнительной информации, описывающей страницу), т.е. необходимо в ОП выделить блок памяти из 4 Мбайт, а в мультизадачной среде такая таблица может потребоваться для каждой задачи.

Кроме того, если таблицу страниц хранить в ОП, время преобразования линейного адреса в физический будет несоизмеримо с преобразованием логического адреса в линейный при сегментации. Для ликвидации этого недостатка в структуру процессора входит устройство страничного преобразования, составной частью которого является ассоциативный буфер преобразования TLB, реализованный в виде частично-ассоциативной памяти, состоящей из четырех групп (модулей) общей емкостью 32 32-разрядных слова (рисунок 4.15).

Рисунок 4.14 - Одноэтапное преобразование линейного адреса в физический

Работа буфера TLB основана на общей идеологии кэш-памяти. Основная таблица страниц хранится в ОП, а базовые адреса активных страниц (наименее давно используемых), с которыми работают в текущий момент времени программы в блоке данных частично-ассоциативной памяти. В устройстве страничного преобразования из линейного адреса по 3-разрядному номеру индекса из памяти тегов выбираются четыре тега, базовые адреса которых загружены в блок данных, и сравниваются с тегом из линейного адреса (старшие 17 разрядов линейного адреса). Если один из тегов совпадает, то из СОЗУ данных параллельно выбирается 20-разрядный адрес страничного кадра, отображаемого на физическую память (базовый адрес страницы в ОП) и 12 бит, описывающих страницу (назначение битов смотри рисунок 4.17). ФА ОП получается операцией конкатенации выбранного 20-разрядного базового адреса страницы и 12-разрядного смещения из линейного адреса.

В типичных системах TLB удовлетворяет до 99% запросов на доступ к таблицам страниц. В качестве стратегии замещения в буфере TLB применяется алгоритм псевдо-LRU, как и во внутренней кэш-памяти.

Если при страничном преобразовании в TLB не обнаружено совпадение тегов, то выполняется процедура замещения информации из таблицы страниц, находящейся в памяти.

Кратко рассмотрим этот процесс. В процессоре реализовано двухэтапное или двухуровневое страничное преобразование линейного адреса в физический на уровне ОП, представленное на рисунке 4.16, что позволяет существенно сократить емкость ОП для хранения таблиц страниц по сравнению с одноэтапным преобразованием (рисунок 4.14).

В состав ассоциативного буфера TLB также входит регистр управления CR3, в котором хранится 20-разрядный физический базовый адрес каталога страниц текущей задачи, он называется регистром базового адреса каталога страниц PDBR. Каталог страниц постоянно находится в ОП и не участвует в свопинге.

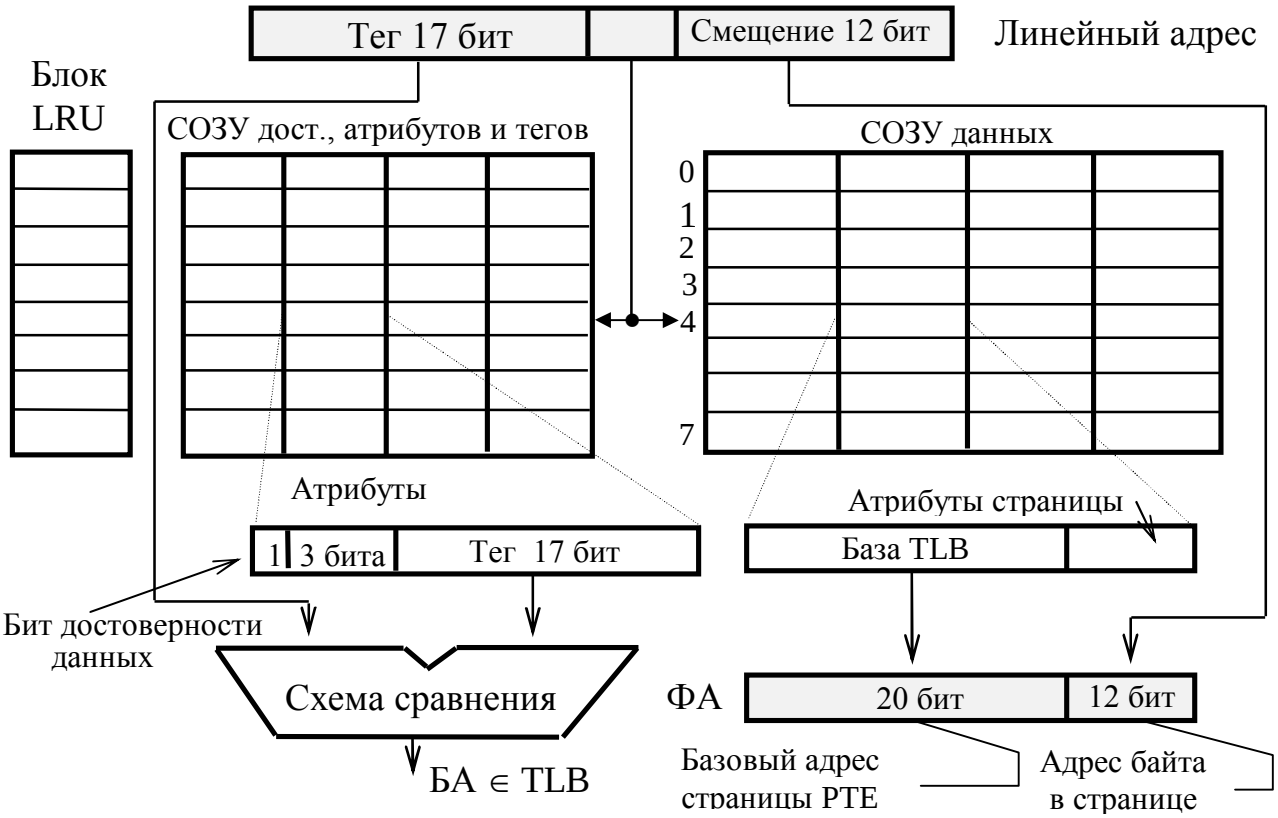


Рисунок 4.15 - Организация буфера TLB для преобразования линейного адреса в физический адрес

Корневая таблица называется таблицей страниц первого уровня или просто каталогом страниц, содержит 1024 32-разрядных дескриптора, называемых элементами каталога страниц PDE. Каждый элемент таблицы PDE адресует подчиненную таблицу страниц (таблицу страниц второго уровня), т.е. всего допускается иметь до 1024 подчиненных таблиц страниц.

Каждая из таблиц страниц содержит 1024 32-разрядных дескриптора, называемых элементами таблицы страниц - PTE, и каждый из элементов PTE, в свою очередь, адресует страничный кадр в физической памяти.

Собственно преобразование линейного адреса в физический состоит из следующих этапов:

- старшие 10 бит 31-22 линейного адреса, сдвинутые на два разряда влево, логически складываются с содержимым регистра базового адреса каталога страниц CR3, и по этому адресу из каталога страниц выбирается один из 1024 элементов PDE, который определяет 20-разрядный адрес таблицы страниц (одну из 1024 таблиц страниц);
- средние 10 бит 21-12 линейного адреса, сдвинутые на два разряда влево, логически складываются с содержимым выбранного элемента PDE каталога страниц и по этому адресу из таблицы страниц выбирается один из 1024 элементов PTE, который определяет 20-разрядный адрес страничного кадра в физической памяти (базовый адрес страницы в ОП);
- 20-разрядный базовый адрес страничного кадра PTE совместно с 12 младшими разрядами атрибутов страницы загружаются в ассоциативный буфер TLB страничного преобразования на место назначенной на удаление страницы по алгоритму псевдо LRU-стека, а также выбранные 20 бит базового адреса страничного кадра совместно с 12 битами 11-0 линейного адреса образуют 32-разрядный физический адрес памяти, по которому производится обращение.

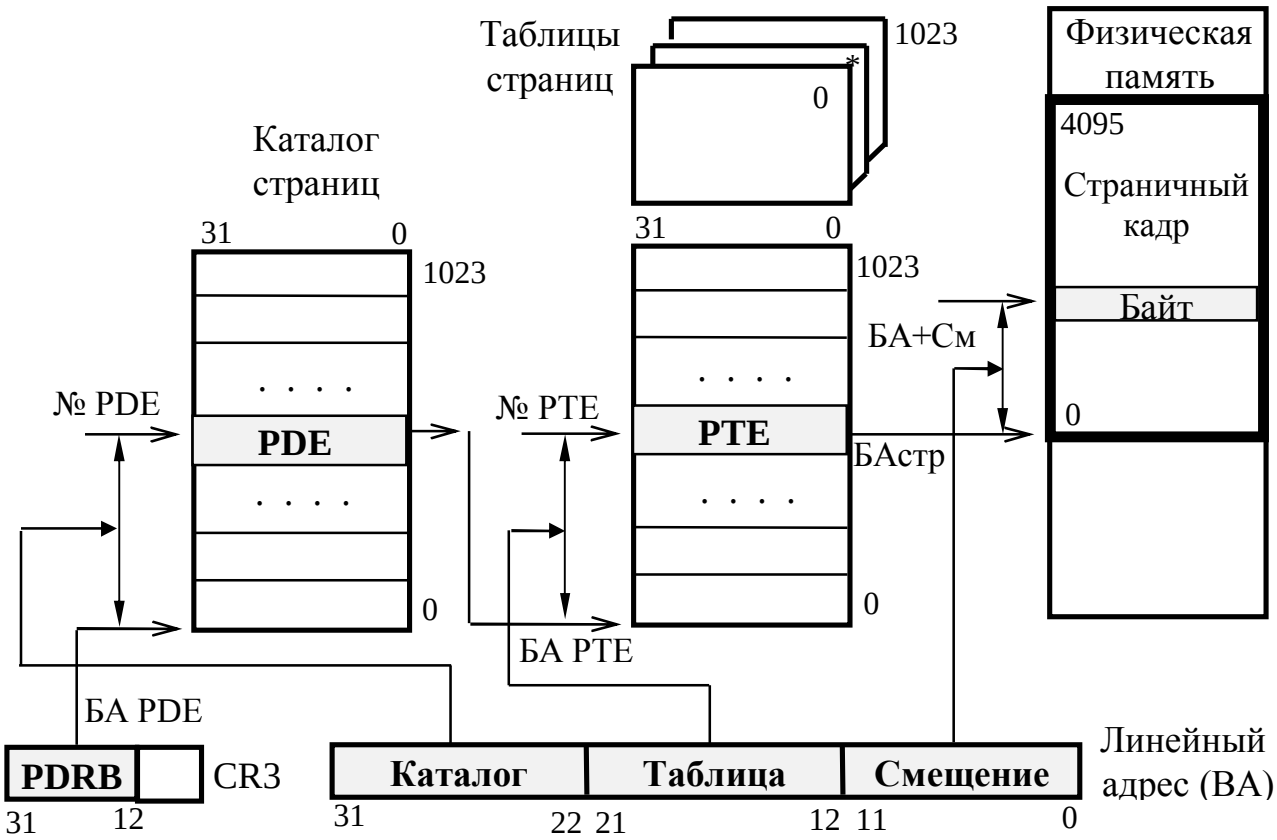


Рисунок 4.16 - Двухэтапное преобразование линейного адреса в ФА

При обращениях к элементам PDE и PTE параллельно производится несколько проверок, в которых принимают участие младшие 12 разрядов элементов таблиц каталога и страниц, имеющие одинаковый формат:

Рисунок 4.17 - Формат элемента таблицы страниц (слова TBL)

Поле - адрес страничного кадра. В таблице каталогов (элемент PDE) в этом поле указывается адрес таблицы страниц, а в таблице страниц (элемент PTE) -базовый адрес страницы, содержащей данные или команды.

Биты системного программиста. Биты 11-9 аппаратно не устанавливаются, а могут быть использованы разработчиками операционных систем, например, для хранения информации о активности страниц, загруженных в ассоциативный буфер TLB (т.е. о том, как часто они используются).

Биты обращения A (Accessed) **и записи D в страницу** (Dirty) **(неудачный перевод "грязный")** содержат информацию об использовании страницы.

Бит A устанавливается аппаратно при каждом обращении к странице **при записи или при чтении** из нее, т.е. при обращении к странице первого и второго уровней до выполнения операции чтения или записи в таблицу, а сбрасывается программно (сбросом бита A может управлять программист).

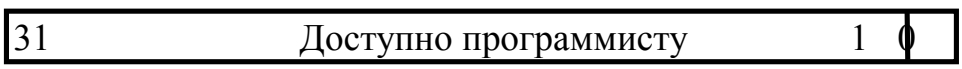
Бит D устанавливается также аппаратно **только при записи** в страницу и только в элементе PTE (в элементе каталога PDE этот бит не определен и не участвует в алгоритмах работы страничного преобразования).

Периодически проверяя и сбрасывая биты A во всех элементах таблиц страниц, ОС может определить наиболее активные (часто используемые) страницы с привлечением поля достоверности [11-9], в котором можно фиксировать число обращений к странице между каждым сбросом бита A.

ОС привлекает бит D при возвращении страницы на диск. Бит D сбрасывается в 0 при загрузке страницы в память, и при необходимости освобождения места в ОП ОС принимает решение о необходимости удаления данной страницы на диск перед загрузкой новой затребованной страницы с диска в ОП (прототип бита флага для обновления ОП в кэш-памяти).

Состояние D=0 показывает, что содержимое страничного кадра в ОП не изменялось (не было записи в страницу) и на диске имеется точная копия этой страницы, поэтому не требуется и свопинг для нее. А D=1 указывает на необходимость свопинга страницы, т.е. предварительную перезапись страницы на диск на место старой страницы.

Бит присутствия P (Present) показывает местоположение страницы: в физической памяти (P=1) или на диске (P=0). При P=0 в таблице страниц хранится информация о местоположении отсутствующей страницы на диске в формате:



Если бит P=0 в каком-либо элементе (PDE или PTE), то при обращении к этому элементу со стороны программы возникает особый случай страничного нарушения и в режиме виртуальной памяти реализуется следующий алгоритм замещения страниц (свопинга):

1. Если в ОП нет свободной области памяти для дозагрузки страниц с диска, то ОС по значению бита обращения A (возможно с привлечением поля достоверности (биты 11-9)) определяет страницу в ОП с P=1 в качестве кандидата на удаление (потерявшего активность) на диск.

2. По значению бита D данной страницы принимается решение о необходимости выполнения процедуры свопинга страницы. Если бит D=1, то ОС копирует страницу из ОП на диск и отмечает ее местоположение в таблице страниц. Далее ОС копирует затребованную страницу с диска в ОП, загружает адрес страничного кадра в элемент таблицы страниц и устанавливает бит P=1. Также могут устанавливаться другие биты, например, атрибут защиты страницы R/W.

3. Так как в буфере TLB может остаться копия старого элемента таблицы страниц, то ОС очищает его (сброс бита достоверности в TLB). Текущий элемент страницы загружается в TLB буфер на место, определяемое нулевым значением бита достоверности или кодом, формируемым блоком LRU в качестве кандидата на замещение.

4. Осуществляется рестарт команды, вызвавшей особый случай прерывания.

Биты считывания/записи R/W и пользователь/супервизор U/S применяются в механизме защиты страниц и рассматриваются ниже.

Биты управления кэшированием. Биты PCD запрещения кэширования страницы и PWT сквозной записи применяются для управления кэшированием на уровне страниц и также рассматриваются ниже.

4.6.4 Защита по привилегиям

Механизм защиты процессора опирается на описание различных системных объектов (сегментов) с помощью дескрипторов. В каждом дескрипторе имеется двухбитное поле уровня привилегий дескриптора – DPL, которое определяет, каким программам разрешается доступ к дескриптору и, следовательно, описываемому им объекту (сегменту).

Термин привилегия подразумевает права или возможности, которые обычно не разрешаются. Процессор Intel (кроме 8086) поддерживает 4 уровня привилегий 0, 1, 2, 3: чем меньше номер, тем выше уровень привилегии. Число программ, выполняемых на каждом уровне, уменьшается с увеличением уровня привилегии (уменьшением номера привилегии).

При выполнении почти каждой команды осуществляется проверка защиты по привилегиям для следующих ситуаций:

- возможности выполнения текущей команды (для привилегированных команд);
- возможности обращения к данным других программ;
- возможности передачи управления (переходу) в другой сегмент кода (программ), имеющему другой уровень привилегии по отношению к текущему кодовому сегменту.

Привилегированные команды. В систему команд процессора i486 входит 19 привилегированных команд, которые разрешено выполнять только ОС с разным уровнем привилегии.

К первой группе команд относятся команды, изменяющие сегментацию (содержимое сегментных регистров) или влияют на сам механизм защиты (загрузки дескрипторов, селекторов таблиц и т.п.). Эти команды разрешены только на уровне привилегии 0 (PL0-программы). При попытке выполнения этих команд на других уровнях генерируется нарушение общей защиты (прерывание 13) и ОС прекращает работу.

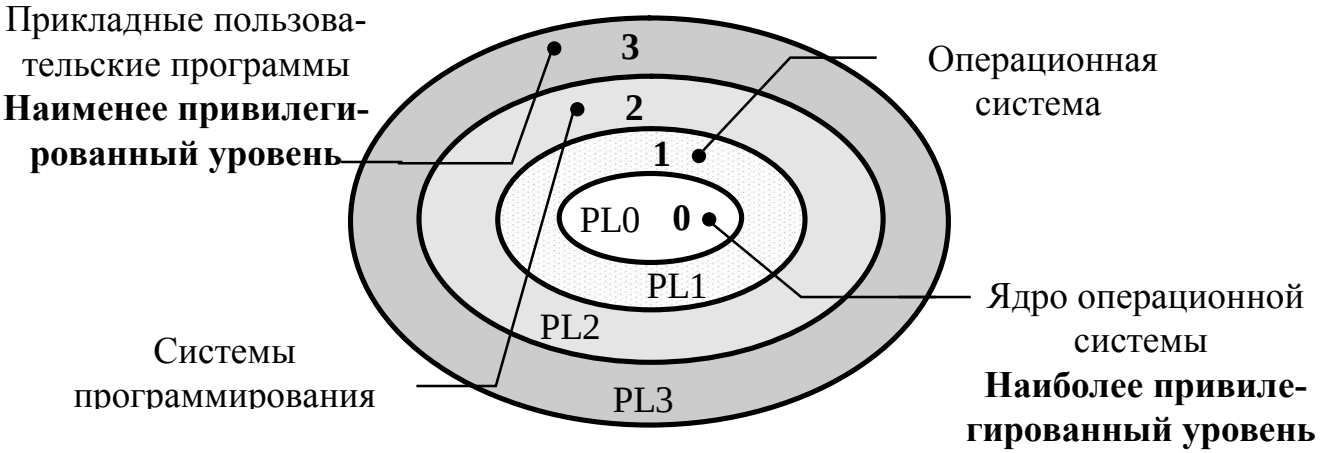


Рисунок 4.18 - Уровни привилегий или кольца защиты

Вторую группу привилегированных команд составляют команды, модифицирующие состояние флажка прерываний IF и команды ввода-вывода IN и OUT. Для этих команд программа не обязательно должна иметь уровень привилегий 0.

Защита доступа к данным. Большинство программ в мультизадачной среде разделяют сегменты данных, т.е. несколько программ могут использовать одни сегменты данных. **Программам не разрешается считывание/запись данных из сегментов, которые имеют более высокий уровень привилегий,** т.е. «движение» к данным внутрь колец защиты запрещается и любая такая попытка приводит к формированию нарушения общей защиты.

Передача управления. Ограничения защиты для вызова (перехода) на выполнение других кодовых сегментов (программ), находящихся на других уровнях привилегий, еще более жесткие, так как

передача управления (с помощью команд FAR CALL и JMP) **разрешается только программам, уровни привилегий которых совпадают,** т.е. находятся на одном уровне кольца защиты.

Для устранения этого недостатка в процессоре предусмотрены специальные средства , которые позволяют переходить с одного уровня кольца защиты на другой. Иначе, прикладные программы с уровня PL3 не смогут воспользоваться программами (драйверами и т.п.) операционной системы на уровнях с более высоким уровнем привилегии (т.е. программами, находящимися на уровнях 0 - 2).

Общие правила защиты по привилегиям показаны на рисунке 1.23.

Уровни привилегий задаются в трех местах:

1. Уровень привилегий сегмента определяет поле DPL дескриптора в байте AR прав доступа.
2. Уровень привилегий выполняющегося кода (программы) называется текущим уровнем привилегий CPL, и он задается полем RPL селектора в регистре CS.
3. Младшие два бита селектора содержат поле запрашиваемого уровня привилегий RPL или уровень привилегий запросчика.

Тогда основное правило защиты доступа к данным имеет вид:

CPL (т.е. PL программы) <= DPL (т.е. PL данных).

То есть значение DPL дескриптора сегмента кода (программы) должно быть не больше значения DPL дескриптора сегмента данных (программа не должна быть менее привилегированна, чем данные, к которым она обращается). Иначе, процессор генерирует прерывание о нарушении общей защиты. Проверка привилегий осуществляется при загрузке селектора в один из сегментных регистров DS, ES, FS или GS вместе с контролем атрибутов защиты, описанных в предыдущем параграфе, и при нарушении привилегий загрузка селектора не производится.

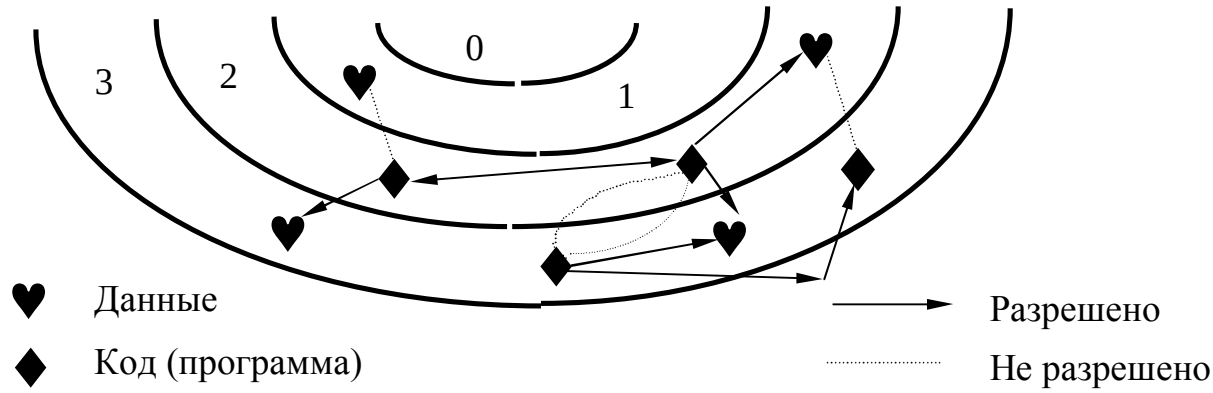


Рисунок 4.19 - Обращения в пределах колец защиты и между ними

При загрузке селектора в сегментный регистр стека SS правила защиты ужесточаются:

CPL (т.е. PL программы) = DPL (т.е. PL данных),

т.е. не разрешается использовать стек даже с меньшими привилегиями.

Для защиты сегментов данных процессор также использует значение поля RPL селектора данных.

Поле RPL показывает уровень привилегий источника селектора и при передачах не изменяется. Поэтому содержимое поля RPL должно быть численно меньше или равно значению DPL выбираемого дескриптора, показывая не меньшие привилегии источника, так как один и тот же дескриптор могут использовать несколько программ с разным значением поля RPL в селекторе. При несоблюдении этого условия доступ к ОП блокируется и формируется нарушение общей защиты.

При загрузке селектора сегмента данных и стека процессор сравнивает значение DPL выбираемого дескриптора не прямо со значением CPL, а с максимальным значением из CPL и RPL:

DPL >= max (CPL, RPL) = EPL.

Значение EPL называется эффективным уровнем привилегий и показывает меньший уровень привилегий. Следовательно, при загрузке селектора, в котором RPL содержит большее значение, чем требуется (т.е. больше значения CPL), запрещается доступ к сегменту данных, использование которого в противном случае было бы разрешено.

Защита сегмента кода (программ) запрещает передачу управления сегменту кода, находящегося на другом уровне привилегий и также имеет несколько дополнений.

Передача управления в другой сегмент осуществляется по командам межсегментного перехода (типа FAR) JMP, CALL и возврата из подпрограммы RET. Адрес передачи управления задается 48-разрядным указателем селектор:смещение, который содержится либо в самой команде (прямая передача управления), либо берется из памяти (косвенная передача управления).

Адресная часть команды

При выполнении таких команд изменяются регистры CS и EIP. Отсюда контроль межсегментной передачи управления заключается в проверке достоверности селектора, загружаемого в регистр CS.

При загрузке селектора в регистр CS процессор предпринимает следующие проверки:

Проверяется целевой дескриптор (дескриптор сегмента, которому передается управление) сегмента кода, т.е. в поле прав доступа дескриптора указан ли атрибут именно исполняемого сегмента (сегмента кода).

Контролируется значение DPL целевого дескриптора, которое должно быть точно равно значению CPL (т.е. равно уровню привилегии текущей программы, из которой передается управление), или в селекторе значение поля RPL должно быть меньше или равно значению CPL, или следует установить RPL=0, и тогда это поле не действует.

Целевой сегмент кода (программы) должен быть отмечен присутствующим (бит P=1 в дескрипторе).

Новое значение для EIP должно находиться в пределах нового сегмента кода (контроль по размеру).

При условии прохождения всех этих проверок процессор продолжает выполнение передачи управления по указанному адресу. Иначе формируется нарушение общей защиты (прерывание 13) или нарушение неприсутствия (прерывание 11).

Организация передачи управления между уровнями привилегий рассматривается в дисциплине "Системное программное обеспечение".

4.6.5 Защита на уровне страниц

Защита на уровне страниц в основном предназначена для предотвращения взаимодействия программ друг с другом. Контроль достоверности обращения к памяти осуществляется параллельно со страничным преобразованием адреса (т.е. не ухудшает характеристики производительности процессора) и является более простым, чем при сегментации памяти, так как не использует полей типа страни-

цы (кода или данных) и предела (размер всех страниц одинаков и определяется разрядностью смещения).

Различают две разновидности контроля на уровне страниц: **ограничение адресуемой области памяти и контроль типа обращений.** Доступом к страницам управляют биты элемента таблицы страниц (рисунок 4.17): PCD, PWT, U/S, R/W. Защита применяется к таблицам страниц первого и второго уровня (элементу каталога и страницы).

Ограничение адресуемой области памяти. Для страниц используется не четыре, а только два уровня привилегий:

Уровень супервизора (U/S=0) для операционной системы, других системных программ (драйверов устройств и т.д.) и защищенных системных данных (таблиц страниц и т.д.).

Уровень пользователя (U/S=1) для прикладных программ и данных.

При CPL= 0, 1 или 2 процессор работает на уровне супервизора и ему доступны все страницы, а при CPL=3 процессор работает на уровне пользователя и ему доступны только страницы уровня пользователя. Механизм защиты состоит в сравнения бита U/S с полем CPL.

Контроль типа обращений. В механизме защиты предусмотрены только два типа доступа к памяти:

- разрешение доступа только для считывания из страницы (R/W=0);
- разрешение доступа для считывания/записи из/в страницу (R/W=1).

Таким образом, используется двухуровневая защита страниц:

- если процессор работает на уровне пользователя, то записать информацию можно только в страницы, которые

  а) принадлежат уровню пользователя (U/S=1, CPL=3):

  б) отмечены как допускающие считывание и запись (R/W=1);

  в) страницы уровня супервизора недоступны с уровня пользователя ни для считывания, ни для записи (U/S=1, CPL=0, 1, 2).

При попытке нарушения этих правил генерируется прерывание особого случая общей защиты.

Следует отметить, что атрибуты защиты элемента каталога (таблицы страниц первого уровня PDE) могут отличаться от атрибутов ее элемента таблицы страниц второго уровня PTE, так как процессор контролирует атрибуты защиты элементов PDE и PTE раздельно, начиная с первого уровня, что подразумевает четыре возможные комбинации атрибутов защиты R/W для пользовательского режима. Совместный эффект можно описать выражением R/W[TBL]= R/W[PDE] & R/W[PTE]. Данное значение атрибута защиты и устанавливается в бите защиты элемента таблицы страниц, загружаемого в TLB буфер при промахе. Атрибут защиты страниц, используемых в режиме супервизора всегда имеет атрибут разрешения считывания и записи (R/W=1).

Когда разрешено страничное преобразование, сначала реализуется защита сегментов, а затем защита страниц и только в том случае, если при сегментации не обнаружено нарушений защиты памяти.

Бит PCD управляет кэшированием страницы во внутреннюю кэш-память: при PCD=1 кэширование страницы запрещено. Это бывает необходимо для страниц, которые содержат порты ввода-вывода с отображением на память или для страниц, кэширование которых не дает выигрыша в быстродействии (линейные участки программ или программы инициализации).

Бит PWT можно использовать для запрещения сквозной записи при обращениях только к внешней кэш-памяти (PWT=1 - сквозная запись), так внутренняя кэш-память работает только со сквозной записью при любом значении бита PWT. При PWT=0 для страницы разрешается обратная запись только при обращении к внешней кэш-памяти.