А.М. БАКШАЕВ

Организация памяти ЭВМ

УЧЕБНОЕ ПОСОБИЕ

Конспект лекций
Часть I

Киров, 2000

1 Архитектура малых ЭВМ

1.1 Семейства ЭВМ и требования к ним

Расширение сферы применения средств вычислительной техники (СВТ), особенно их применение для решения научно-технических задач, задач автоматизации производства, САПР, работы с базами данных, обработки статистической информации и управления производством привело к необходимости создания вычислительных систем (ВС), отличающихся составом технических средств (ТС) и техническими характеристиками: производительностью, емкостью оперативной памяти (ОП), внешних запоминающих устройств (ВЗУ) и т.д.

Это привело к необходимости создания ЭВМ с единой архитектурой, но с переменным составом оборудования, который определяется выполняемыми ВС функциями. Такой подход означает выполнение отдельных функциональных устройств в виде модулей, которые могут объединяться в необходимом количестве в одной ЭВМ или говорят о ЭВМ проблемно-ориентированных на решение определенного класса задач. При этом существенное место для реализации такого подхода занимает сокращение числа типов (номенклатуры) выпускаемых семейств ЭВМ.

Основными требованиями для создания ВС с переменным составом оборудования, различной производительностью и стоимостью является их:

- информационная совместимость;
- программная совместимость снизу доверху;
- аппаратная совместимость;
- конструктивная совместимость;
- эксплуатационная совместимость.

В настоящее время широкое применение нашли несколько семейств ЭВМ различных западных фирм: Intel, Motorolla, DEC. При этом в каждом семействе для различных поколений микропроцессоров (МкПр) фирмы выпускают несколько типов процессоров и обрамления к ним (периферийных БИС). Например, фирмой Intel выпускаются МкПр следующих типов:

- высокопроизводительные центральные процессоры, предназначенные для сложных математических расчетов, но имеющие высокую стоимость;
- удешевленные варианты данного поколения МкПр с несколько худшими техническими характеристиками и предназначенные для использования в ЭВМ для учрежденческих применений, периферийных ЭВМ в составе сетей, во встроенных микроконтроллерных системах, где не требуется высокое быстродействие;
- МкПр с малым потреблением мощности, предназначенные для применения в компьютерах типа Note Book.

С развитием элементной базы и появлением новых типов МкПр существенно улучшаются технические характеристики ВС, но при этом для преемственности (совместимости) использования ранее разработанного программного обеспечения (ПО) любая модель семейства должна удовлетворять вышеперечисленным требованиям, т.к. стоимость разработки нового ПО на порядок и выше превышает стоимость аппаратных средств.

Информационная совместимость ЭВМ предполагает использование единых способов кодирования информации, форматов и типов данных, одинаковые или кратные длины машинных слов в различных моделях.

Программная совместимость означает, что программы, написанные для одной модели, должны выполняться для других моделей семейства. Это предполагает наличие единой системы и форматов

команд, режимов адресации, что позволяет использовать общие ОС и прикладное ПО для моделей одного семейства снизу доверху. Программной совместимостью объясняется наличие большого числа форматов команд и системы команд для старших моделей МкПр, что затрудняет их изучение.

Аппаратная совместимость заключается в возможности подключения к любой модели ЭВМ, состоящей из центрального процессора (ЦП) и ОП любых контроллеров периферийных устройств (ПУ), общих для всех моделей ряда. Это достигается за счет использования унифицированных интерфейсов ввода-вывода и единых протоколов обмена между ПУ и ЦП. Однако следует заметить, что практически каждая новая модель МкПр имеет свой состав аппаратных средств БИС, но которые программно и аппаратно совместимы с предыдущими версиями БИС.

Конструктивная совместимость подразумевает использование унифицированных панелей, блоков и ТЭЗов (плат) с единой системой назначения контактов разъемов и типов разъемов, конструктивного исполнения системного блока и разбивки ТС на конструктивные модули.

Эксплуатационная совместимость предполагает общие методы технической и математической эксплуатации и обслуживания, т.е. преемственность языков программирования, единых ОС, программ технического обслуживания и диагностики, единые методы профилактики ТС и т.д., что не требует переквалификации и дополнительного обучения обслуживающего персонала.

Все рассмотренные особенности и требования к семействам ЭВМ позволяют создавать системы переменной конфигурации с возможностью постепенного, по мере необходимости, наращивания вычислительной мощности ВС путем замены ЦП более производительным, расширения емкости ОП, подключения новых и замены устаревших ПУ. При этом за счет программной совместимости моделей одного семейства ЭВМ все ранее существующее программное обеспечение сохраняется.

Улучшение технико-экономических характеристик моделей семейства ведется в следующих направлениях:

* совершенствования элементной базы, т.е. применения новых более быстродействующих БИС и СБИС;
* повышения производительности ВС за счет применения новых технологий и структурных решений как для ЦП, так и для периферийных БИС;
* увеличения объема ОП и ВЗУ, совершенствования организации хранения данных, реализации виртуальной памяти, использования как внутренней на кристалле, так и внешней кэш-памяти;
* дальнейшего развития системы ПО как в области прикладных программ, так и ОС;
* создания мультипрограммных и многомашинных вычислительных систем, работающих в реальном времени, локальных и глобальных ВС;
* развития системы ввода-вывода и расширения номенклатуры ПУ;
* повышения надежности ВС, развития эффективных систем контроля и диагностики;
* за счет совершенствования технологии.

Технические средства семейств ЭВМ постоянно совершенствуются и развиваются в следующих направлениях:

* модификации - создании нескольких моделей одного и того же устройства, отличающихся друг от друга значением какого-либо параметра или набором функции. Как правило, модификации появляются вследствие доработки предыдущих версий устройства для устранения каких-либо недостатков или дополнения функциональных возможностей устройства без изменения его технических характеристик. Например, таймер ВИ54 является модификацией предыдущей модели ВИ53, в котором реализована дополнительная функция фиксации и считывания приказа обратного считывания;

* модернизации, т.е. замены устаревших образцов устройств новыми с улучшенными техническими, функциональными, конструктивными и эксплуатационными характеристиками. Например, для матричных принтеров выпускается множество модернизаций по ширине каретки, по количеству и расположению иголок, быстродействию и т.д.;
* создания принципиально новых устройств. Например, лазерный принтер является принципиально новой разработкой среди принтеров, как по принципу функционирования, так и по техническим характеристикам.

1.2 Логическая структура технических средств

Все модели одного семейства имеют общую архитектуру и принципы функционирования.

Под архитектурой ЭВМ понимают совокупность функциональных средств и принципов их взаимодействия, включающее описание пользовательских возможностей программирования, системы команд, режимов адресации и средств пользовательского интерфейса, организации памяти, операций ввода-вывода, управления и т.д.

Общность архитектуры разных моделей ЭВМ обеспечивает их совместимость с точки зрения пользователя. В контексте разработки аппаратных средств термин "архитектура" используется для описания принципа действия, конфигурации и взаимосвязей основных логических узлов и устройств ЭВМ.

Реализация конкретной архитектуры для ЭВМ одного семейства может быть различной на структурном уровне, но все они должны обладать свойством программной и информационной совместимости. То есть архитектура как логическое понятие определяет лишь общую концепцию построения и взаимодействия аппаратных средств и не накладывает жестких ограничений на конкретную техническую реализацию модели.

Всю элементную базу для построения МПС можно разделить на следующие группы:

* интегральные схемы малой и средней степени интеграции (ИС) (логические элементы и узлы);
* БИС памяти статического и динамического типа;
* программируемые БИС (ПЛМ, программируемые логические устройства (ПЛУ) и т.д.);
* периферийные БИС, используемые совместно с ЦП и предназначенные для сопряжения с УВВ либо в качестве самостоятельных устройств, выполняющих внешние функции по отношению к ЦП;
* секционированные микропроцессоры, позволяющие строить процессоры с произвольной системой команд и разрядности, ориентированные на решение определенного класса задач;
* однокристальные микропроцессоры (ЦП и сопроцессоры), имеющие жестко определенные технико-экономические характеристики по всем параметрам;
* микроконтроллеры (8-, 16- и 32-разрядные), используемые для построения встроенных систем управления объектами или обработки данных.

Многообразие выпускаемых БИС позволяет строить МПС любой проблемной ориентации и сложности с широким диапазоном изменения их технико-экономических характеристик.

В каждой группе рассмотренных типов БИС можно выделить ИС различных серий и функциональных возможностей, что несколько затрудняет ориентацию разработчика аппаратных средств при выборе требуемой элементной базы.

Даже внутри одной серии ИС несмотря на программную и информационную совместимость и общую архитектуру ЦП практически затруднительно изучить все возможные варианты структурных и схемотехнических решений, т.к. для однокристальных микропроцессоров в существенной степени изменяется не только структура ЦП, но и состав интерфейсных и периферийных БИС для каждой модели.

1.3 Особенности структурной организации малых ЭВМ

При рассмотрении принципов построения и состава ТС ЭВМ необходимо помнить, что все выпускаемые модели ЭВМ являются проблемно ориентированными на определенный класс решаемых задач, имеют различную стоимость и технико-экономические характеристики.

Однокристальные ЦП различных моделей нашли широкое применение для построения персональных ЭВМ (ПЭВМ), настольных рабочих станций, встроенных контроллеров, миниЭВМ для высокопроизводительных систем, работающих в мультипрограммных режимах работы, для построения многопроцессорных и многомашинных комплексов.

Отсюда, в зависимости от того, где применяется тот или иной тип ЦП, состав ТС и технические характеристики, включая стоимость, существенного отличаются.

Основными особенностями принципов организации ЭВМ являются:

* проблемная ориентация систем в достаточно широком диапазоне с возможностью гибкого изменения конфигурации системы и переориентации ее на другие классы задач;
* организация параллельной и конвейерной обработки информации как на уровне ЦП, так и на уровне модулей ЦП;
* использование табличных методов обработки данных и принятия решений, особенно ярко проявляется на примере преобразования виртуальных адресов в физические, организации средств защиты памяти и т.д.;
* в ряде систем реализована возможность создания адаптивно перенастраиваемых ВС, конфигурация которых изменяется в процессе решения задачи с целью наиболее эффективной организации вычислительного процесса и обеспечения живучести системы;
* аппаратная реализация ряда функций математического обеспечения (сопроцессоры), что позволяет в существенной степени повысить производительность ВС;
* многоуровневая иерархическая организация памяти ЭВМ с различными техническими характеристиками по емкости памяти и быстродействию.

Для МПС в структурной реализации получил дальнейшее развитие принцип 3М - модульность, магистральность и микропрограммируемость.

Модульная организация систем. Принцип модульной организации предполагает построение ЭВМ и ВС на основе набора модулей. Под модулем понимается конструктивно, функционально и электрически законченное устройство, позволяющее самостоятельно или в совокупности с другими модулями решать задачи заданного класса или выполнять определенные функции. При этом различают **функциональные и конструктивные модули.**

К **функциональным модулям**, как правило, относятся БИС, выполняющие строго определенные функции. Например, БИС операционных устройств, БМУ и т.д. для секционированных МкПр, БИС ЦП, блоки приоритетных прерываний, прямого доступа в память для однокристальных микропроцессоров, БИС микроконтроллеров. Из этих модулей собираются конструктивные ТЭЗы (платы) путем объединения функциональных модулей и наращивания разрядности обрабатываемых данных.

Конструктивные ТЭЗ (платы) также выполняют строго определенные функции, например, материнская плата, плата контроллера НМД, слоты памяти и т.д., или в систему может входить несколько ТЭЗ с одно- или разнотипными процессорами, на основе которых можно создавать любые конфигурации ЭВМ, отличающиеся друг от друга функциональными возможностями и техническими характеристиками. Модульный подход также способствует сокращению затрат и сроков проектирования, упрощает наращивание мощности и реконфигурацию систем, отодвигает время морального старения ТС.

При решении вопроса о функциональном составе модулей существуют две диалектические противоположности: **многофункциональность** (универсальность) и **специализация** модулей. Многофункциональные модули позволяют обеспечить:

* сокращение номенклатуры модулей;
* снижение затрат на проектирование и их изготовление;
* высокую серийность, а следовательно, и низкую стоимость.

Специализация модулей позволяет исключить избыточность структуры за счет оптимизации схемных решений, реализуемых алгоритмов и функций. Однако специализация модулей низшего конструктивного уровня ведет к необходимости иметь большое число разнотипных модулей, хотя и с высокими техническими характеристиками, за исключением создания систем специализированного назначения.

Примерами многофункциональных модулей могут служить ОУ, БМУ секционированных МПК, однокристальные ЦП, программируемые периферийные БИС, реализующие по несколько разнотипных программно настраиваемых режимов работы.

Специализированные модули позволяют получать хорошие характеристики по быстродействию, надежности, потребляемой мощности.

Магистральный способ обмена информацией. Выделяют два способа взаимосвязей модулей: принцип произвольных связей типа "каждый с каждым" и принцип упорядоченных связей - магистральный, позволяющий минимизировать число связей.

При магистральном принципе выделяют три типа шин: данных, адреса и управления, что позволяет обеспечить регулярность структуры МПС как на уровне БИС, так и на уровне связей между конструктивными модулями МПС.

Достоинства магистрального обмена:

* минимизация числа связей между конструктивными модулями;
* обеспечение стандартизации интерфейсов (Multibus, ISA, EISA и т.д.);
* сокращение числа выводов БИС;
* единые способы подключения и протоколы обмена между модулями.

Микропрограммная организация управления. Для современных МПС характерна многоуровневая организация управления на уровне микрокоманд: с жесткой и программируемой логикой. Принцип микропрограммного управления обеспечивает:

* наибольшую гибкость при организации многофункциональных микропроцессорных модулей за счет использования вызова подмикропрограмм, являющихся общими для нескольких алгоритмов;
* позволяет осуществлять проблемную ориентацию ЭВМ за счет программной настройки на требуемую систему команд или гибкость использования устройств за счет смены микропрограмм путем их загрузки в ОЗУ микропрограмм с внешнего носителя;
* использование макроопераций в МПС, т.е. часть алгоритмов, выполняемых с помощью подпрограмм, может выполняться по отдельным командам на микрокомандном уровне;
* увеличивает регулярность структур за счет использования ПЗУ и не требует при проектировании или доработке УУ в существенной степени изменять схему БМУ, а любые изменения сводятся к корректировке текстов микропрограмм;
* повышает надежность устройств за счет применения БИС памяти;
* упрощает контроль функционирования УУ, который сводится к контролю чтения содержимого ПЗУ микропрограмм.

Регулярность структуры. Принцип регулярности предполагает закономерную повторяемость элементов структуры и связей между ними.

Применение данного принципа позволяет:

* увеличить плотность интегрального исполнения БИС;
* сократить время топологического и схемотехнического проектирования БИС;
* сократить число типов функциональных и конструктивных элементов;
* повысить серийность, а следовательно, снизить стоимость БИС.

Принцип регулярности структур наиболее ярко проявляется при использовании структур и устройств типа памяти (РОН, ОЗУ, ПЗУ, ПЛМ), при использовании магистрального способа обмена, стандартизации интерфейсов, использовании принципа микропрограммного управления и т.д.

1.4 Архитектура малых ЭВМ

Как было отмечено выше, архитектура отображает аспекты структуры ЭВМ, которые являются видимыми для пользователя: систему команд, режимы адресации, форматы и типы данных, набор регистров ЭВМ, доступных пользователю. То есть термин "архитектура" используется для описания возможностей, предоставляемых ЭВМ, а термин "организация" определяет, как эти возможности реализованы.

Все ЭВМ содержат: процессор, состоящий из АЛУ и УУ, память и устройства ввода и вывода (УВВ) информации. Объединение функциональных устройств выполняется с помощью системы шин или интерфейса.

По принципам организации можно выделить:

* однопроцессорные системы с общей шиной;
* многопроцессорные системы с сильно связанной конфигурацией с общей или раздельными шинами;
* многопроцессорные системы со слабо связанной конфигурацией с общей или раздельными шинами.

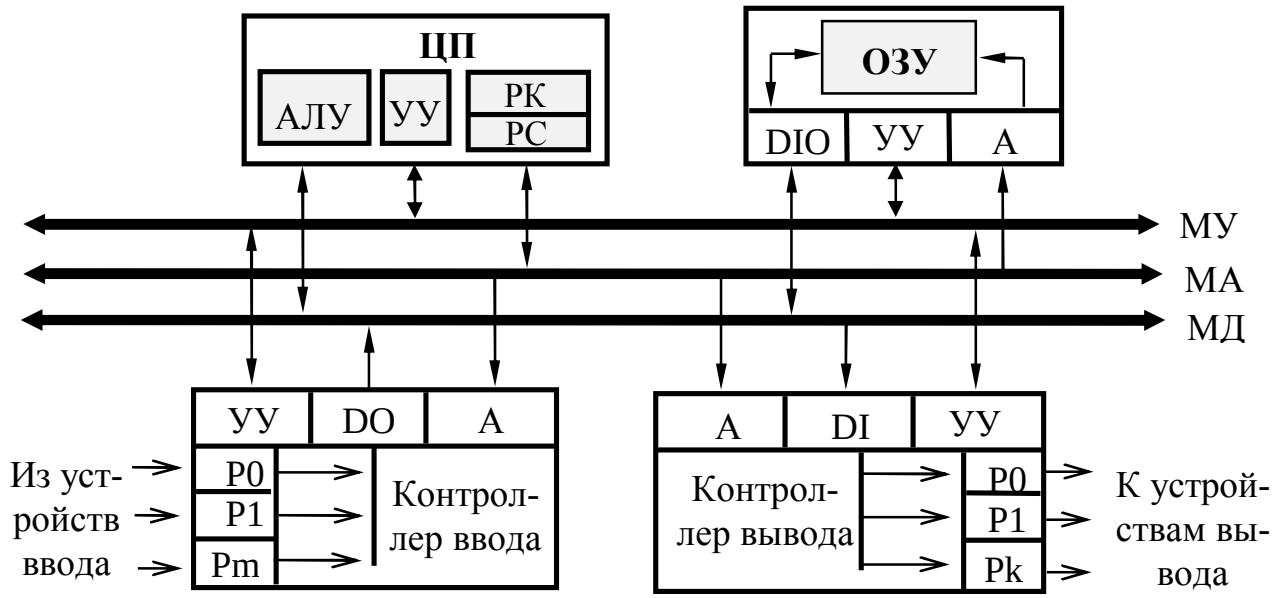


Рисунок 1.1 - Структура ЭВМ с общей шиной

Однопроцессорные системы с общей шиной (рисунок 1.1). К общей шине, состоящей из МД, МА и МУ, подключатся центральный процессор, память и набор устройств ввода и вывода. Каждое

УВВ имеет свой адрес порта ввода-вывода, по которому выполняются операции ввода или вывода данных по командам ЦП IN и OUT. Обращение к памяти выполняется по обычным машинным командам. Все устройства объединяются посредством одной общей шины, и параллельная работа устройств при такой организации невозможна. Достоинством схемы является простота технической реализации.

Многопроцессорные системы с раздельными шинами. (Многошинная организация). Допускает наличие нескольких типов шин, и для каждого способа обмена информацией с периферийными устройствами используется отдельная группа шин, например, для быстрого доступа к внешней кэш-памяти, для ввода-вывода информации при прямом доступе к памяти или сопроцессору и т.д. Протоколы обмена данными, структура шин и быстродействие при обмене для каждой из групп шин могут быть адаптированы к обслуживаемым устройствам (рисунок 1.2).

Многопроцессорные системы со слабо связанной конфигурацией.

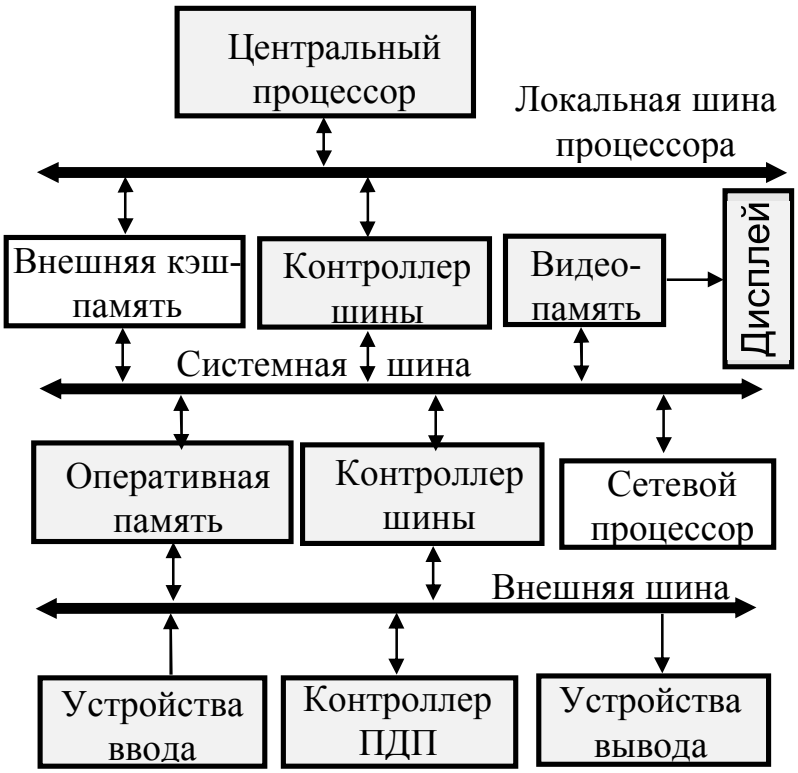


Рисунок 1.2 - Многошинная организация связей между устройствами

Слабосвязанные конфигурации применяют в ЭВМ с высокой производительностью. Любой модуль в такой системе может быть ведущим системной шины и содержать любой тип процессора, включая и несколько процессоров одного типа. При этом системные ресурсы разделяют несколько процессорных модулей, а проблему состязаний при доступе к шине должна решать логика управления системной шины на основе арбитража шины (АШ).

Каждый процессорный модуль имеет свою резидентную шину, к которой могут подключаться своя ОП и УВВ. При такой организации каждый процессорный модуль может работать параллельно с другими, используя свою память программ и данных, а по системной шине лишь изредка выполняется межпроцессорный обмен данными или дозагрузка программ и данных из системной памяти (рисунок 1.3).

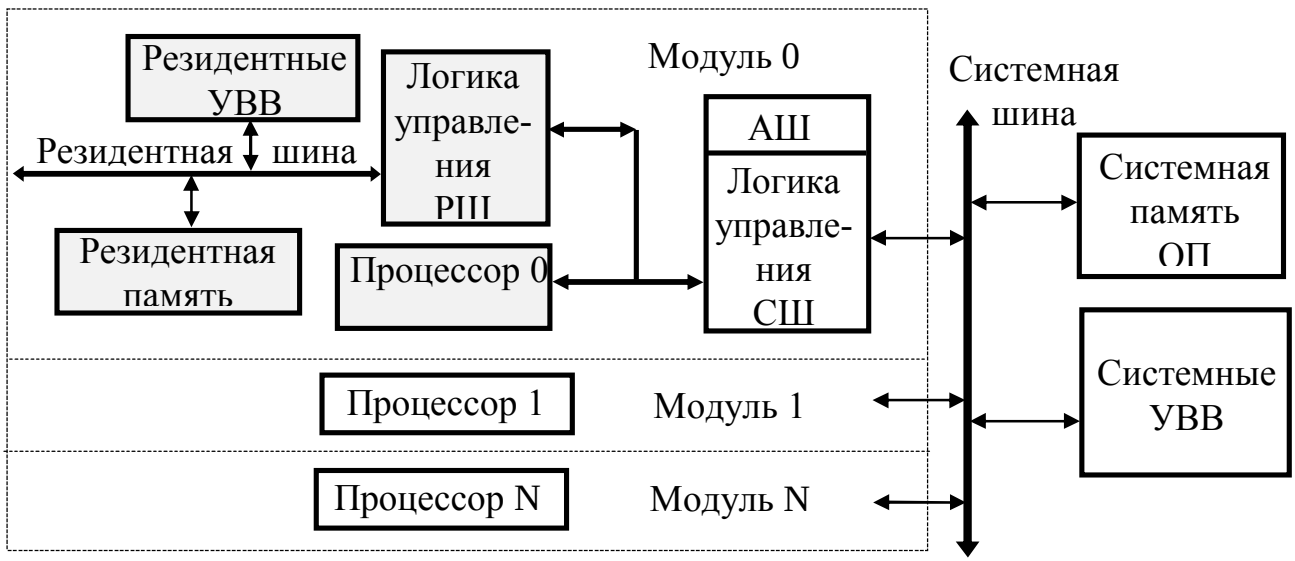


Рисунок 1.3 - Многопроцессорная организация интерфейсов

2 Организация памяти ЭВМ

2.1 Основные концепции организации памяти ЭВМ

Вычислительные возможности ЭВМ в основном определяются характеристиками ее памяти, которая, как и архитектура ЭВМ, должна удовлетворять следующим требованиям:

1. **Универсальность**, т.е. многофункциональное использование всех типов памяти для хранения программ и любых типов данных, а также обеспечение ее технических характеристик в заданных пределах.

2. **Общность решаемых задач**, т.е. память должна быть ориентирована на решение любого класса задач и обеспечивать максимальную эффективность организации вычислительного процесса.

3. **Высокое быстродействие** - память должна обеспечивать работу процессора в реальном времени и не снижать характеристики его производительности.

4. **Надежность.** В связи с ростом емкости памяти ЭВМ доля аппаратурных затрат на ее реализацию неуклонно возрастает, что приводит к увеличению числа сбоев и отказов памяти. Поэтому обеспечение высокой надежности памяти и достоверности результатов особенно возрастает.

5. **Низкая стоимость.** Так как емкость памяти и быстродействие постоянно возрастают, то и увеличивается и стоимость памяти. Поэтому данная проблема приобретает особую актуальность.

6. **Обеспечение совместного использования и доступа к памяти** множеством программ или процессоров в мультипрограммных и многопроцессорных комплексах.

7. **Дружественность**, т.е. ориентированность памяти к классам решаемых задач пользователя с целью обеспечения улучшения технических характеристик системы и использование удобного как программного, так и аппаратного интерфейса.

При этом требования 2-6 допускают количественную оценку, а 1 и 7 нельзя измерить количественно. Анализ требований к памяти показывает, что реализовать все перечисленные требования в одном типе (БИС) памяти невозможно, так как большинство их являются противоречивыми. Поэтому на практике используется множество типов ЗУ, которые отличаются своей архитектурой, техническими характеристиками (временем доступа, емкостью, стоимостью на бит информации). Также необходимо учитывать, что произведение времени доступа к памяти на стоимость бита информации является почти постоянной величиной для всех типов ЗУ.

Классификация архитектур памяти по требованиям к ним выглядит следующим образом:

1. **Быстродействующие памяти** (РОН, СОЗУ с прямым доступом, FIFO, LIFO на основе БИС СОЗУ, кэш-память, расслоение обращений к ОП, т.е. время доступа к памяти в 3-10 раз меньше, чем к ОП).

2. Быстродействующая **память большой емкости** - это ОП и дисковая кэш-память, располагаемая между основной и вторичной (дисковой) памятью (буферная память обычно типа FIFO), позволяющая снизить время доступа к вторичной памяти в 2-10 раз.

3. **Виртуальная память** (страничная, сегментная, сегментно-страничная организация памяти в общем адресном пространстве ЭВМ), когда для программиста память представлена как единое целое, а техническая реализация подразумевает использование емкости ОП и внешней дисковой памяти.

4. **Общая память** (совместное использование ОП множеством программ и процессоров на основе организации многопортового доступа к ОП и арбитражем доступов к памяти в мультипрограммных и мультипроцессорных системах).

5. **Высоконадежная память**:

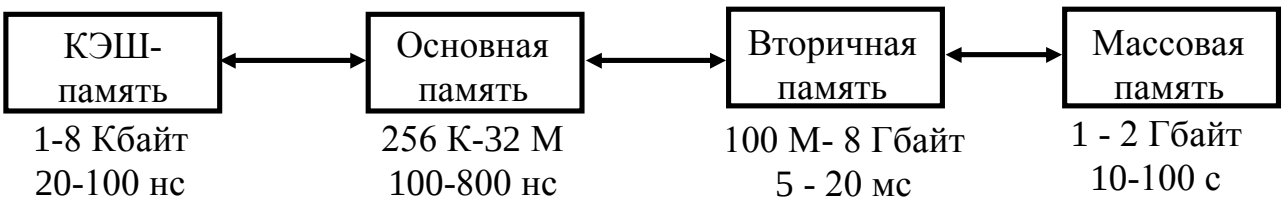
- с использованием корректирующих кодов для автоматического исправления одиночных ошибок;
- память с системой защиты по атрибутам доступа, защитой, основанной на использовании мандата, описывающего правила, права доступа, путь доступа и допустимые операции, выполняемые с памятью (защищенный режим работы) или еще называют с защитой от несанкционированного доступа.

6. **Интеллектуальная память** - это такая память, в которой для доступа к данным указывается не адрес ячейки памяти, а только часть данных и смысловая связь с другими данными, к которым уже был произведен доступ:

- ассоциативная память АЗУ;
- память правил, в которой хранятся совокупности правил для доступа к данным;
- память семантической структуры, в которой хранится семантическая сеть, используемая для поиска информации в соответствии со смысловой связью.

Такие виды памяти необходимы для различных экспертных и интеллектуальных систем, баз знаний, систем логического вывода и т.д.

Поэтому для увеличения емкости и повышения быстродействия без увеличения стоимости памяти необходимо использовать многоуровневую иерархическую организацию памяти.



Так как программы размещаются в последовательных ячейках памяти при естественном порядке следования команд, то такое расположение называется **пространственной локальностью программы.** Для ряда данных также характерно их размещение в соседних ячейках (массивы, исходные данные) и называется пространственной локальностью данных (обычно рекомендуется описывать данные по мере их использования в программе). С другой стороны, для программ характерна циклическая организация повторения некоторых участков программы, подпрограмм в течение какого-либо промежутка времени. К данным, как правило, доступ также осуществляется многократно. Такой процесс называется **временной локальностью**, а общее их название - **локальность по доступу.**

Применение блочной пересылки между различными уровнями памяти требует меньше времени, чем суммарное время пересылки отдельных данных этого блока, особенно при расслоении обращений к памяти.

Размер блока для обмена между основной памятью и вторичной порядка при страничной организации памяти до 4 Кбайт, т.к. емкость ОП достаточно велика (до нескольких Мб), поэтому выигрыш во времени для обмена данными блоками достаточно существенен.

При обмене кэш-памяти с ОП размер блоков (строк) невелик: 8-16 слов (16-128 байт), отсюда выигрыш по времени за счет пространственной локальности ниже, чем в предыдущем примере, но так как число строк в кэш-памяти велико (500-4000), то влияние временной локальности достаточно ощутимо.

2.2 Архитектура памяти с повышенным быстродействием

Одним из основных методов повышения производительности процессора является увеличение быстродействия памяти, при этом уменьшение времени обращения к ОП приводит к увеличению стои-

мости памяти, поэтому, как правило, применяют структурные методы повышения быстродействия. К ним относятся методы расслоения обращений к памяти:

1. Параллельная (пакетная) обработка доступов к памяти.

2. Конвейерная обработка обращений к памяти.

3. Использование буферных промежуточных памятей небольшой емкости с высоким быстродействием, одной из разновидностей которых является использование кэш-памяти.

2.2.1 Параллельная обработка

Так как для команд характерно последовательное размещение в ОП и выборка команд выполняется из соседних ячеек, а также при работе с массивами данных, то часто применяют метод расслоения обращений. Для этого память делится на модули, в каждом из которых одноименным адресам соответствуют соседние адреса ячеек памяти, а младшие разряды адреса определяют номер модуля, т.е. слова (рисунок 2.1).

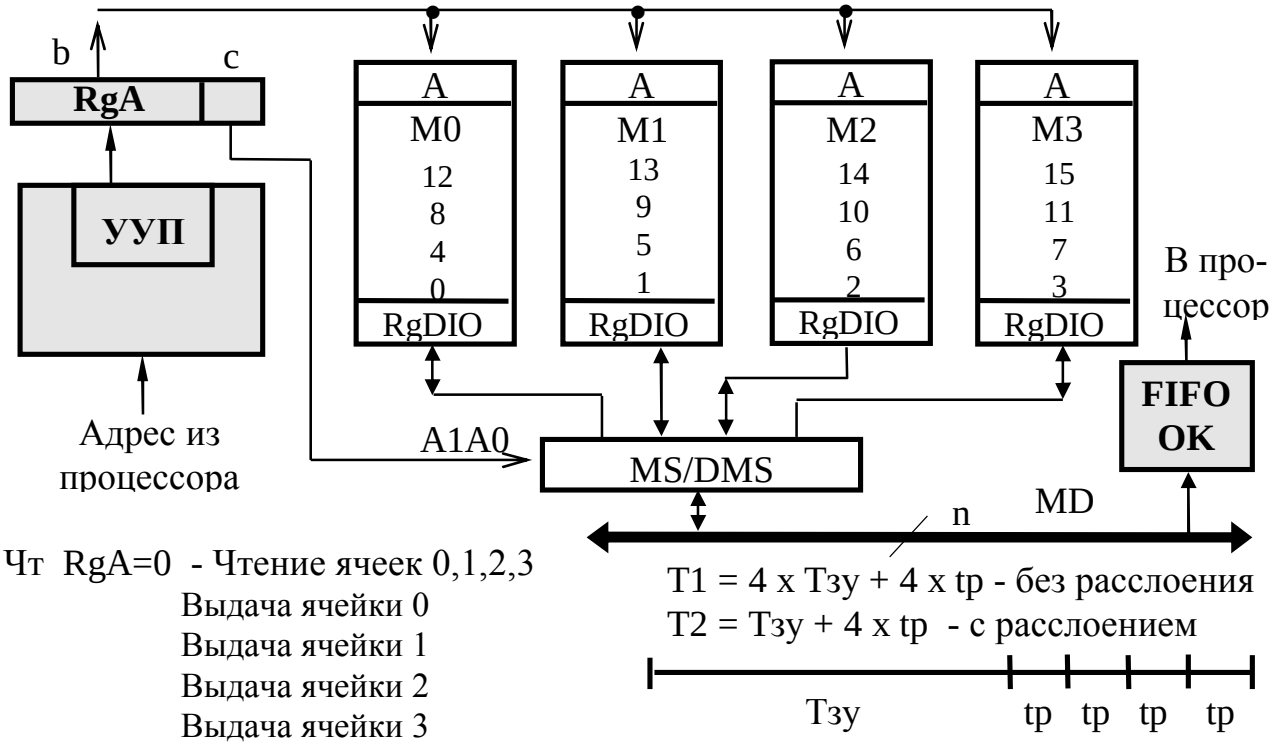


Рисунок 2.1 - Структура ОЗУ с параллельным доступом

Например, для организации очереди команд требуется одно обращение к ОП и k тактов передачи слов в буфер очереди команд (память типа FIFO).

При обработке массивов данных достаточно велика вероятность, что данные находятся в одной строке (по ширине выборки). Тогда при наличии схемы сравнения для сравнения старших разрядов адресов считанной и запрашиваемой строк данные могут быть уже считаны и хранятся в RgDIO.

Кроме того, метод может широко использоваться при совместном применении с кэш-памятью для обновления ОП и замещения кэш-памяти строками.

При наличии раздельных RgDI и RgDO с выходами на три состояния наличие MS/DMS необязательно, т.к. запись выполняется раздельно по модулям памяти, а считывание - параллельно, но передача данных по системной магистрали данных - только последовательно (с конвейерной передачей данных).

2.2.2 Конвейерный доступ

При конвейерном доступе ОП также строится по модульному принципу, каждый модуль имеет свой регистр адреса, а в составе УУП либо счетчик адресов для модификации адреса, либо буфер запросов (адресов) на основе FIFO для формирования очереди (потока) адресов при обращении к модулям. Каждый модуль должен иметь свой буфер регистра данных. Конвейерный доступ не требует ожидания окончания предыдущего цикла обращения к текущему модулю, а сразу инициирует цикл обращения к другому модулю, если тот свободен (рисунок 2.2).

Каждый модуль должен иметь свой блок УУ для управления доступом к модулям памяти и контроля конфликтных ситуаций по доступу. Наличие MS/DMS, как и в предыдущем случае, является необязательным.

Для задач общего назначения конвейеризация доступа дает выигрыш в производительности только при формирования очереди команд, замещении строк кэш-памяти, выборке строковых переменных и данных для сопроцессора, а также при обработке массивов данных за счет организации специальных пакетных циклов шины при доступе к последовательным адресам (например, пакетные кэшируемые и некэшируемые циклы шины ЦП). При этом в составе УУП не требуется FIFO очереди запросов, так как данные должны быть выровнены по ширине выборки, а каждый последующий адрес памяти данных отличается от предыдущего на единицу (т.е. на слово, передаваемое по МД).

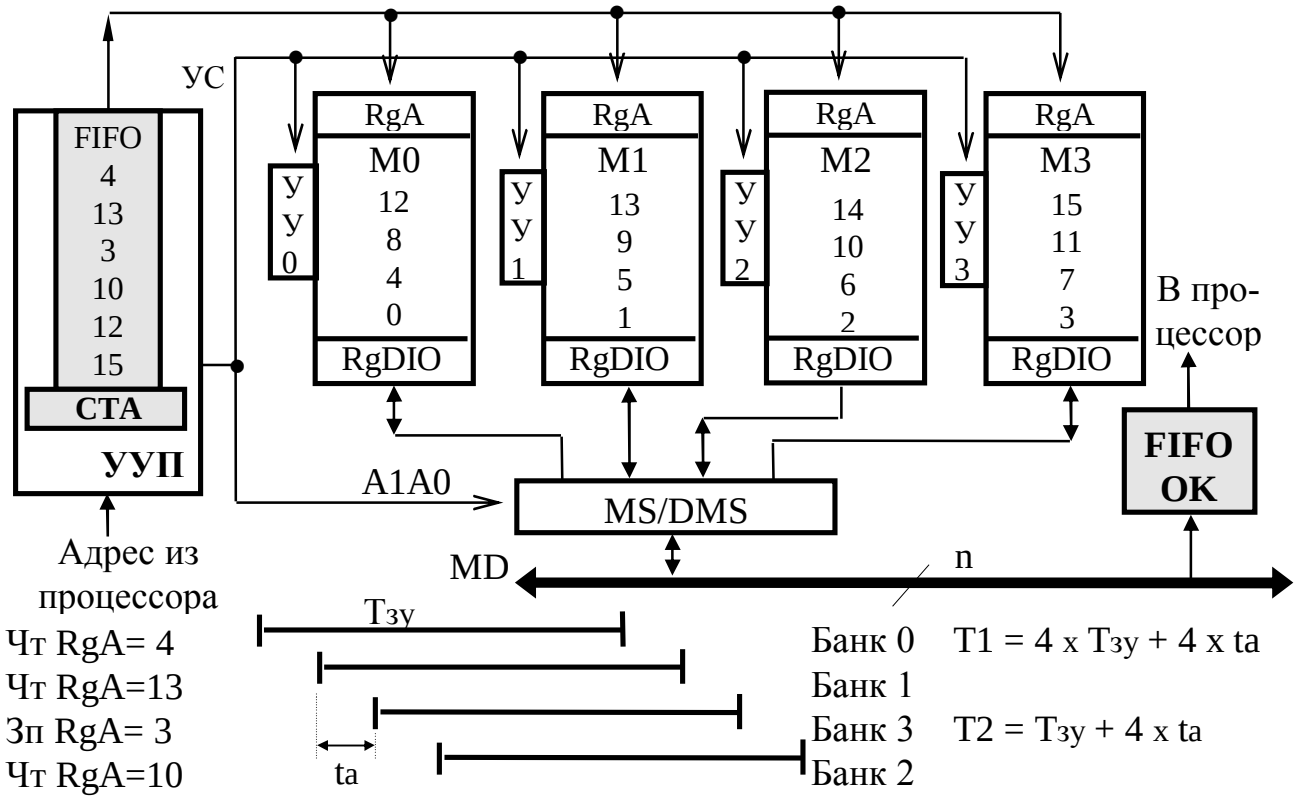


Рисунок 2.2 - Структура ОЗУ с конвейерным доступом

Кроме того, при конвейерном доступе к памяти можно организовать параллельный процесс регенерации модулей ОЗУ, незанятых в данный момент времени (непакетируемые циклы шины).

Наибольший эффект данный метод дает при применении в специализированных процессорах, например, при доступе к векторным операндам в векторном процессоре при обращении к адресам i, i+S, i+2S и т.д., отстоящим друг от друга на интервал S. Если данные размещаются в разных модулях

памяти, то общая производительность увеличивается в m/f(m,s) раз, где m - число модулей памяти, при этом при обращении используется очередь доступов к модулям памяти на основе FIFO.

Недостаток: если в течение m циклов доступа, следующих друг за другом, запрашивается доступ к тому же модулю, то до окончания предыдущего доступа к этому модулю запрос на последующий доступ должен блокироваться, т.е. наступает конфликт по доступу.

2.3 КЭШ - память

Кэш-память - это быстродействующая память, располагаемая между ЦП и ОП емкостью 1/1000-1/500 емкости ОП с временем доступа 1/5-1/10 от ОП. Вместе с ОП кэш-память образуют иерархическую структуру, и ее действие эквивалентно быстрому доступу к ОП.

Применение кэш-памяти позволяет сократить время обращения к ОЗУ Тзу в 5-10 раз. При обращении к памяти ЦП сначала должен определить, имеется ли копия строки в кэш-памяти и, если имеется, определить ее местоположение (адрес кэш-памяти, с которого начинается строка, и слова в строке).

Предварительно введем несколько понятий и определений.

Так как емкость кэш-памяти невелика, то при очередном обращении за информацией к кэш-памяти возможна ситуация, когда эта информация отсутствует в кэш-памяти. В подобных случаях необходимо выбрать одну из строк, хранящихся в кэш-памяти, и заменить ее на новую строку. Способ определения строки, удаляемой из кэш-памяти, называется **стратегией замещения кэш-памяти** (или назначение кандидата на удаление).

Для записи данных в кэш-память и ОП при выполнении команд в ЦП также существует несколько методов замещения старой информации. Эти методы называются **стратегией обновления оперативной памяти.**

На эффективность использования кэш-памяти существенно влияют пространственная и временная локальности. Кроме того, программы и данные существенно отличаются по локальности. Поэтому для каждого типа данных, имеющих различную локальность, часто используются кэш-памяти разного типа.

Механизмы преобразования адресов

Выделяют 4 способа размещения данных в кэш-памяти или механизма преобразования адресов строк:

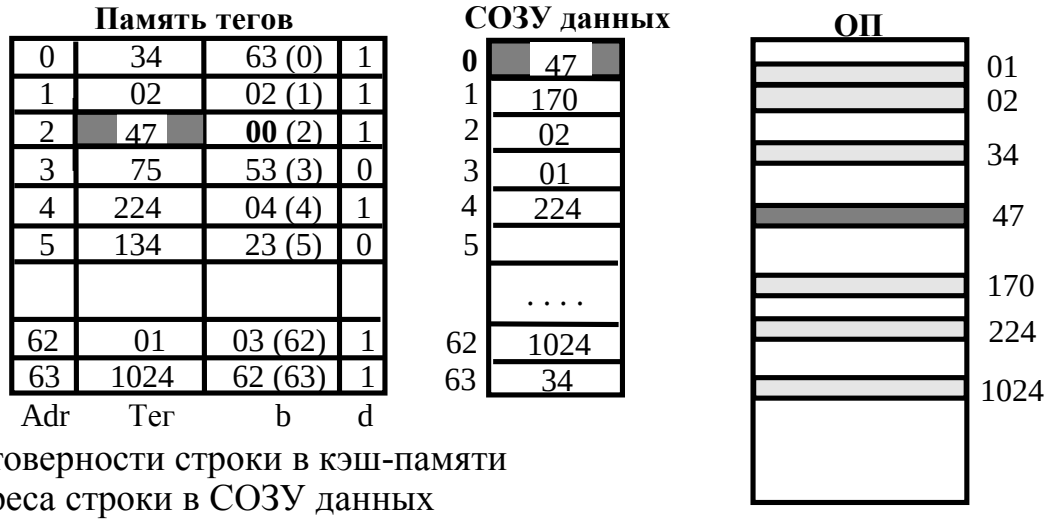
1. Полностью ассоциативное распределение.
2. Прямое распределение.
3. Частично-ассоциативное распределение.
4. Распределение секторов.

Пусть емкость кэш-памяти равна 512 слов, размер строки - 8 слов, а емкость основной памяти 64 Кслова.

2.3.1 Полностью ассоциативное распределение

ОП разбивается на строки по 8 слов в каждой (рисунок 2.3). Разрядность физического адреса слова (ФА) составляет 16 бит, следовательно, адрес строки определяется 13-ю старшими битами ФА, которые однозначно идентифицируют любую строку, хранимую в ОП. В кэш-памяти может быть размещено 512:8=64 строки. Для хранения идентификатора строк, находящихся в кэш-памяти, используют специальную память, называемую теговой памятью, а для хранения непосредственно данных (строк) используется СОЗУ данных. То есть каждому адресу строки в теговой памяти соответствует 8 слов (строка) в СОЗУ данных. Таким образом, для определения местоположения строки (в кэш-памяти или в

ОП) необходимо сравнить содержимое всех ячеек теговой памяти с 13-ю старшими разрядами ФА (тегом ФА) и если будет обнаружено совпадение, то значит строка располагается в быстродействующей кэш-памяти и операнд считывается из или записывается в СОЗУ данных кэш-памяти. Для получения адреса строки СОЗУ данных [b] в памяти тегов необходимо выполнить его формирование (адрес указан в скобках) или в дополнительном поле теговой памяти хранить этот адрес. Бит достоверности данных d=1 указывает на принадлежность строки кэш-памяти, а нулевое значение означает, что данная ячейка свободна (в ней размещаются недостоверные данные, принадлежащие, например, другой программе). Если в памяти тегов хотя бы один бит d=0, то при обнаружении несовпадения тегов при сравнении процедура обновления ОП не выполняется, а затребованная строка из ОП перемещается в кэш-память для дальнейшего использования.



d - бит достоверности строки в кэш-памяти
b - поле адреса строки в СОЗУ данных

Рисунок 2.3 - Связь кэш-памяти с основной памятью

Кэш-память строится на основе двух блоков памяти: первый выполняет функции теговой памяти для хранения номера строки ОП, находящейся в данный момент времени в СОЗУ данных. Таким образом, емкость теговой памяти составляет 64 13-разрядных слов, а кэш-данных - 64 строки по 8 n-разрядных слов или 64 x 8 = 512 слов.

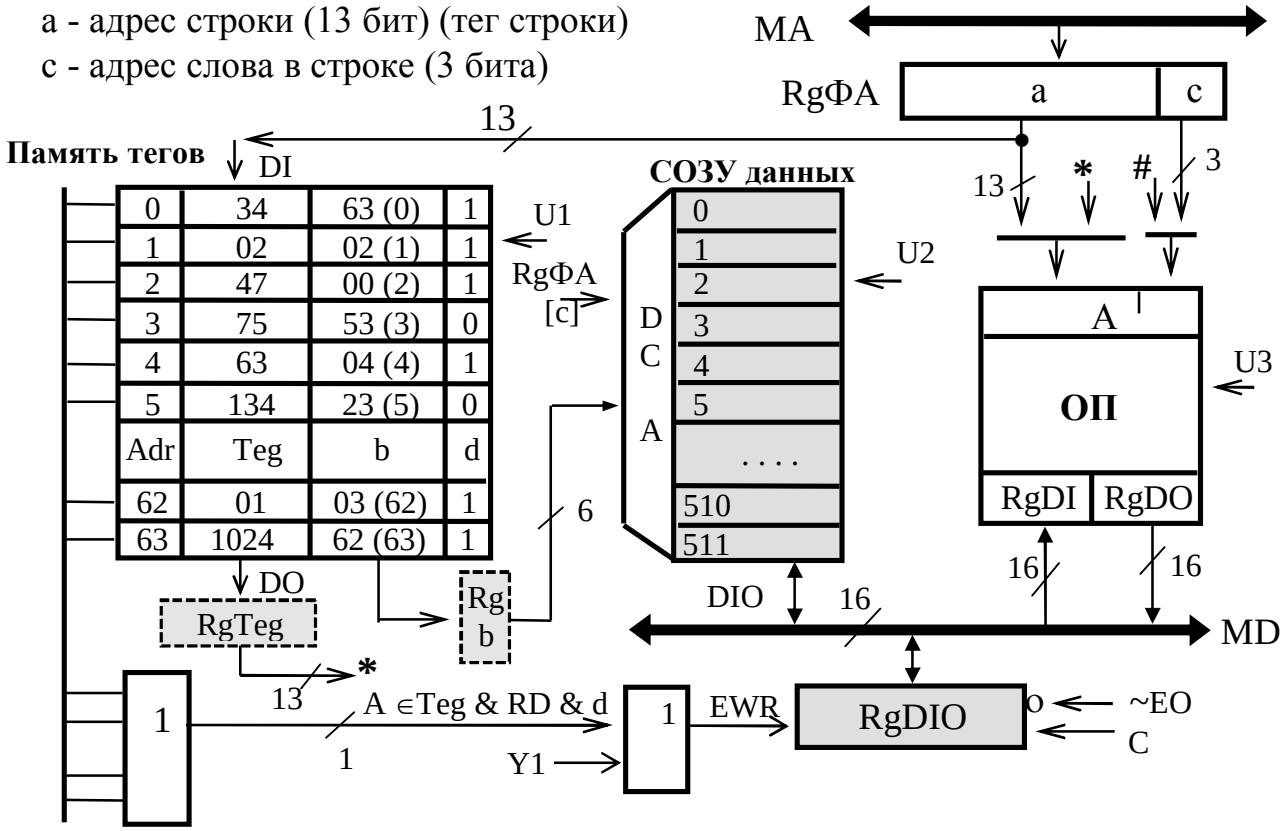
Память тегов строится на основе ассоциативного ЗУ (АЗУ), в котором старшие 13 бит ФА (поле a) строки используются в качестве адреса теговой памяти, которые параллельно сравниваются с содержимым всех ячеек памяти тегов. Если хотя бы один тег совпал, то это означает, что строка, содержащая текущий адрес i, находится в СОЗУ данных. В качестве старших разрядов адреса СОЗУ данных выступают 6 разрядов, считываемых из соответствующего поля [b] памяти тегов, к которым присоединяются три младших разряда ФА (поле [c]) и осуществляется обращение к СОЗУ данных по чтению или записи (рисунок 2.4).

Если ни один тег не совпадает со значением поля [a] RgФА ($A \neq Teg$), то выполняется процедура обновления строки из кэш-памяти в ОП и замещения кэш-памяти согласно принятой стратегии.

Данный метод допускает размещение каждой строки [bn] ОП на месте любой строки [bc] кэш-памяти, так как в качестве ассоциативного признака для поиска строки используется 13-разрядный тег, однозначно определяющий любую и только одну строку ОП.

Достоинства:

1. При замещении строк кандидатами на удаление могут выступать любые строки в кэш-памяти, в зависимости от принятой стратегии, а, следовательно, эффективное использование кэш-памяти для хранения наиболее активных строк.



- выходы трехразрядного счетчика слов CT S
Ui - сигналы управления блоком памяти ~CSi, ~WRi, ~RDi, (i=1-3)
Y1 - сигнал разрешения записи в RgDIO со стороны процессора

Рисунок 2.4 - Структурная схема полностью ассоциативной кэш-памяти без расслоения обращений

Недостатки:

1. Высокая стоимость из-за сложности реализации АЗУ.

2. Малая емкость БИС тегов из-за высокой плотности монтажа на кристалле, большая потребляемая мощность и число выводов БИС АЗУ.

3. Для обращения к кэш-памяти требуется два такта и для чтения, и для записи (такт чтения памяти тегов и такт чтения или записи в СОЗУ данных), так как памяти работать параллельно не могут.

4. Высокая стоимость и ограничения по емкости АЗУ (8-64 ячеек) делают этот метод практически неосуществимым.

При отсутствии запрашиваемой строки в кэш-памяти необходимо выполнить процедуры обновления ОП и замещения кэш-памяти. Время выполнения этих процедур в существенной степени зависит от принятых стратегий обновления ОП и замещения кэш-памяти, которые будут рассмотрены ниже, а также от принципов организации доступа к ОП и СОЗУ данных и структуры магистрали данных системного интерфейса.

На рисунках 2.4-2.6 приведены варианты подключения кэш-памяти и ОП в зависимости от принципов организации памяти (расслоения обращений) и структуры магистрали данных системного интерфейса.

Рассмотрим четыре варианта:

- ОП и СОЗУ данных без расслоения обращений;
- ОП с расслоением обращений на ширину выборки строки, СОЗУ данных без расслоения обращений;
- ОП и СОЗУ данных с расслоением обращений на ширину выборки строки.
- ОП с расслоением обращений на ширину выборки строки, СОЗУ данных с расслоением обращений и двусторонним доступом.

Первый вариант схемы включения представлен на рисунке 2.4. Процедура обновления ОП требует 8 обращений к СОЗУ данных и 8 обращений для записи строки в ОП, т.е. потребуется 8 x Тозу+ 8 x tсозу дан. и столько же обращений для замещения строки кэш-памяти.

Второй вариант (рисунок 2.5) требует наличия на выходах ОП RgDO с выходами с z-состоянием или мультиплексора на 8 входов, а на входах соответственно демультиплексора и регистра RgDI. Тогда процедура обновления ОП потребует 8 x tсозу для чтения строки из СОЗУ данных и записи в RgDI через DMS и одного обращения Тозу к ОП для записи строки и столько же обращений для замещения строки кэш-памяти.

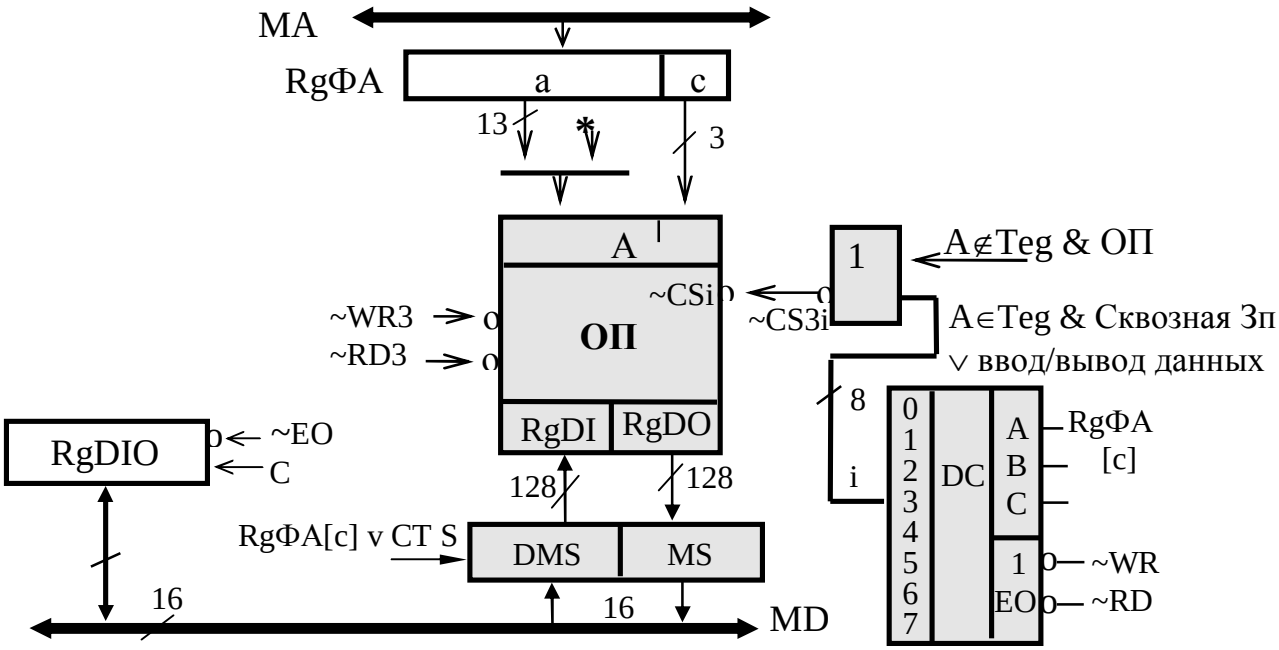


Рисунок 2.5 - Организация интерфейса памяти при расслоении обращений к оперативной памяти

При третьем варианте организации ОП и СОЗУ данных потребуется 8 x n-разрядная MD, что приводит к нарушению стандарта системного интерфейса (разрядность системной магистрали данных должна быть n-разрядной). Поэтому этот вариант является неприемлемым и не рассматривается.

Четвертый вариант организации магистралей, ОП и СОЗУ данных представлен на рисунке 2.6.

Схема включает двухпортовое СОЗУ данных, где порт А связан через MD с процессором и RgDIO. MD имеет разрядность n бит (т.е. без расслоения обращений), а порт В выполнен с расслоением обращений на ширину строки и связан с входами/выходами ОП через 8 x n-разрядную внутреннюю магистраль данных для обмена строками за одно обращение к ОП и СОЗУ данных. MD и BMD связаны между собой через контроллер шины (КШ) для выполнении сквозной записи данных в кэш-память и ОП и загрузки в ОП программ и данных из ВЗУ. КШ представляет собой схему мультиплексора/демультиплексора, управляемого контроллером кэш-памяти и ЦУУ. Процедура замещения строки

выполняется за одно обращение к ОП и одно обращение к кэш-памяти. Однако этот метод необходимо рассматривать только для перспективных разработок, когда ОП конструктивно будет размещаться на одной плате (ТЭЗе) с ЦП или между ОП и внутренней кэш-памятью подключается внешняя быстродействующая кэш-память большой емкости. Иначе надежность работы системы резко снижается, а реализация внешней дополнительной 128-разрядной шины данных не дает существенного выигрыша в быстродействии, так как при вероятности попадания в кэш-память 90% выигрыш составит 0,1 х 7 тактов = 0.7 такта на команду для рассматриваемого примера. Использование дополнительной шины данных оправдано только в мультипроцессорных системах при подключении к этой шине внешней кэш-памяти большой емкости для обеспечения независимого доступа к внешней кэш-памяти со стороны нескольких процессоров по приоритету с арбитражем шины.

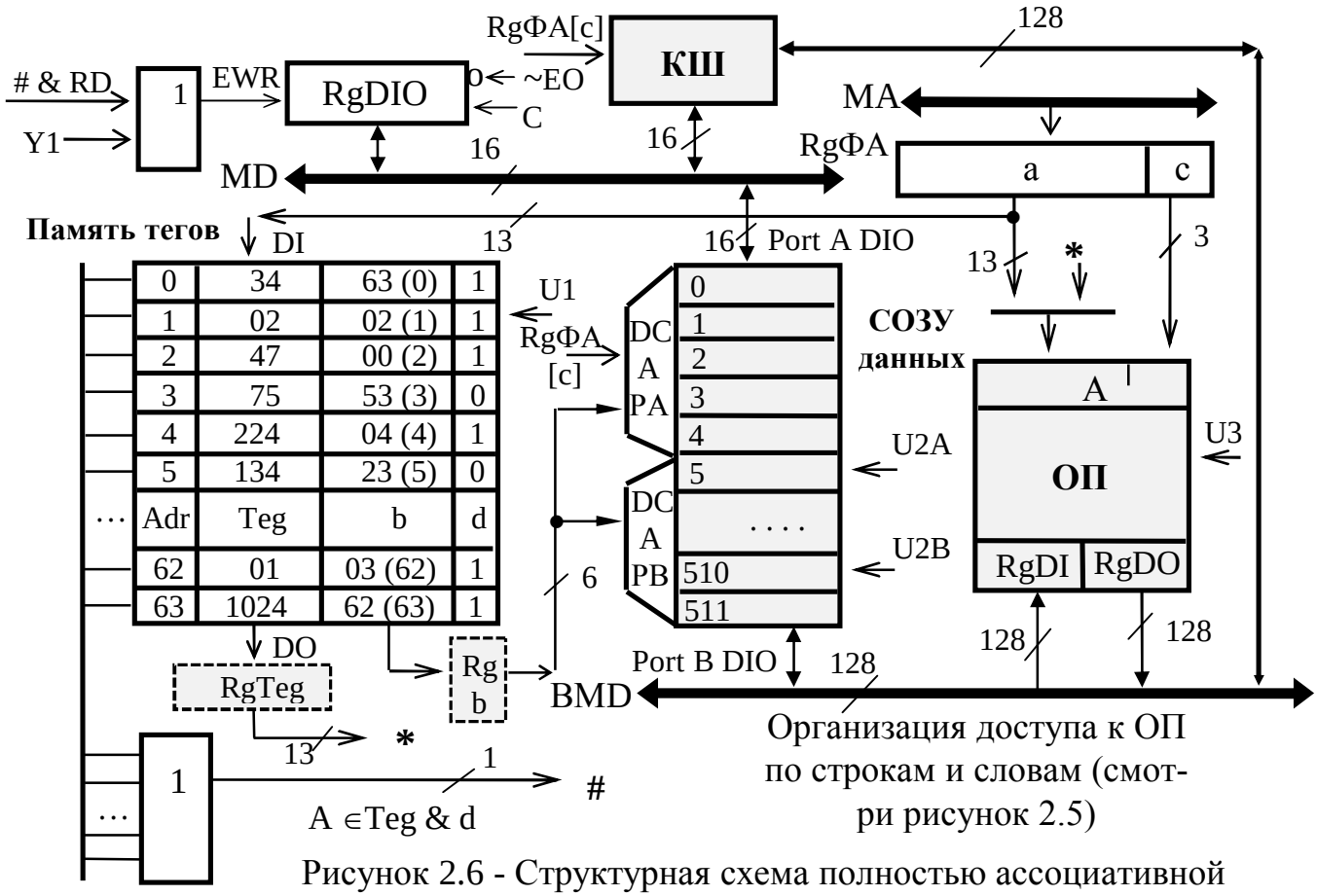


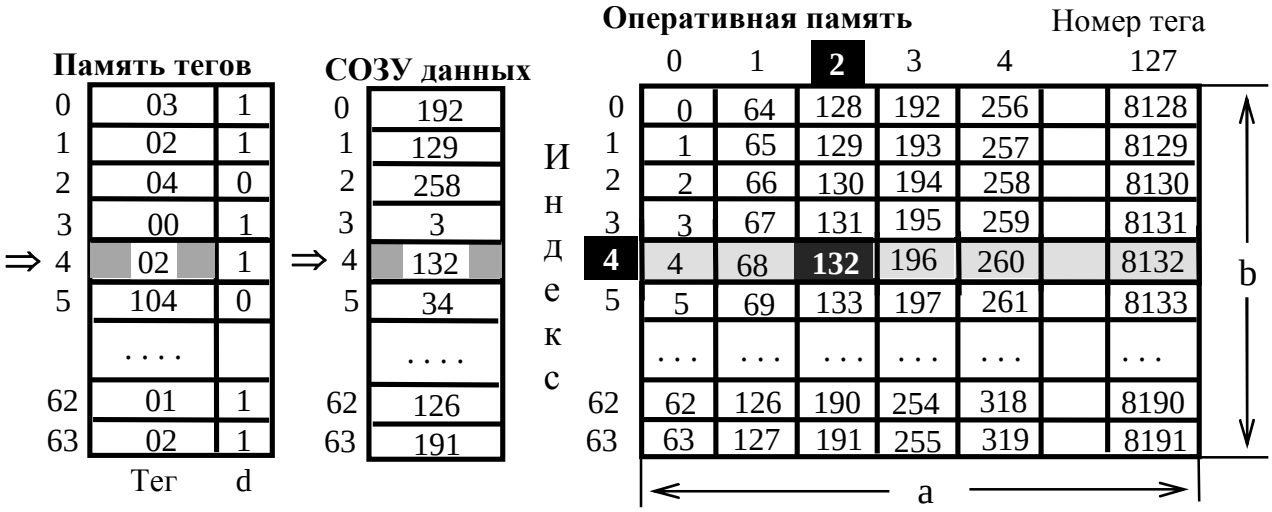
Рисунок 2.6 - Структурная схема полностью ассоциативной кэш-памяти с расслоением обращений

2.3.2 Прямое распределение

С целью уменьшения аппаратурных затрат для представления адреса строки (тега) разобьем формат ФА на 3 части: 3-разрядное поле [c] указывает на адрес слова в строке, разряды [a.b] - на адрес строки в ОП. Тогда структуру ОП можно представить в виде матрицы [a x b] (рисунок 2.7).

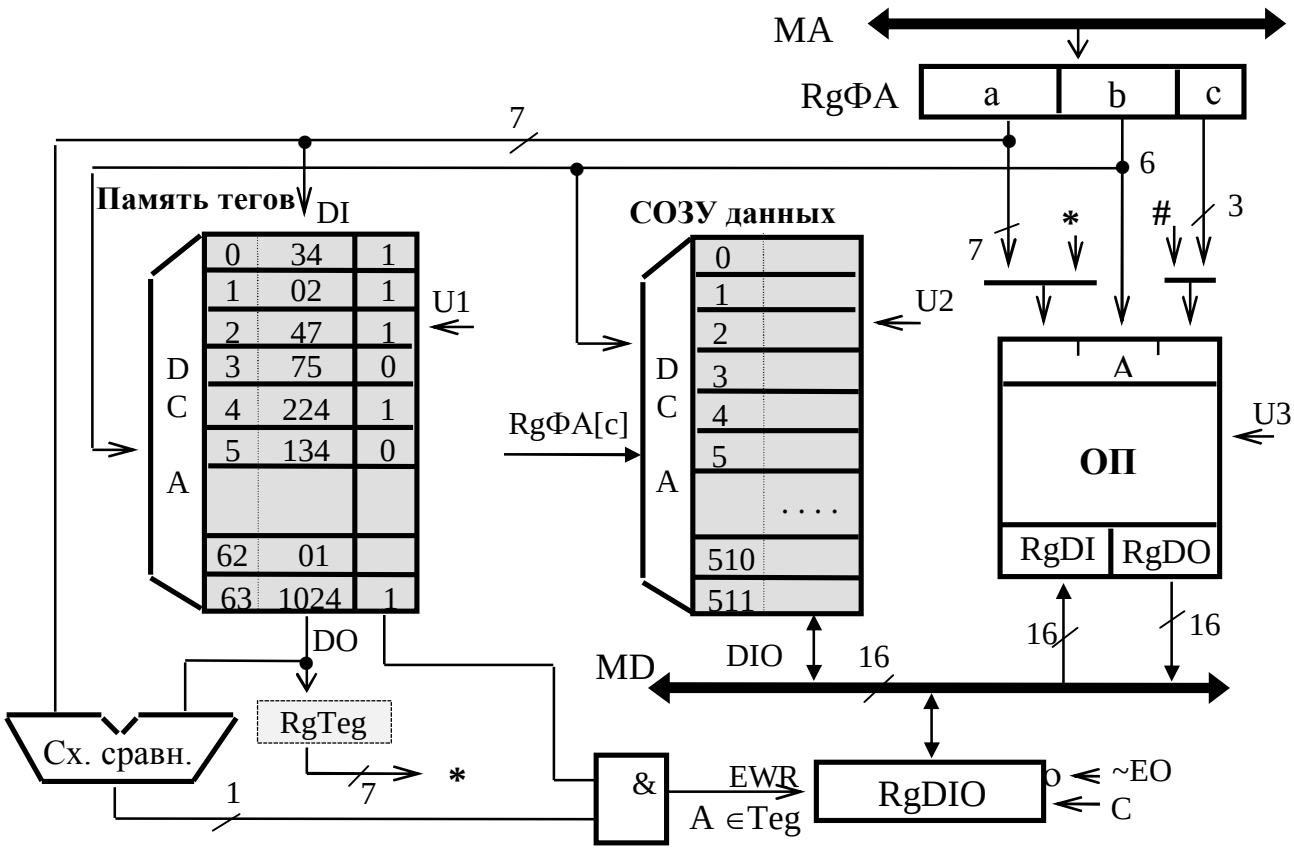
Тогда 6-разрядное поле [b] (индекс) указывает на множество строк (128), отстоящих друг от друга с шагом кратным 64 (например, для RgФА[b=4] это строки с номерами 4, 68, 132, 196,..., 8132), а 7-разрядное поле [a=2] можно использовать в качестве тега для выбора одной строки из этого ряда (строка 132). Такое разбиение позволяет сократить разрядность тега с 13 до 7 разрядов. Таким образом, поиск строки осуществляется в два этапа: поле [b] напрямую указывает (адресует) группу строк, а по тегу (старшим разрядам адреса) определяется принадлежность запрашиваемой строки кэш-памяти.

Кэш-память с прямым распределением строится на основе двух блоков СОЗУ с произвольным доступом на основе обычных БИС ЗУ статического типа с произвольным доступом (рисунок 2.8): первый блок выполняет функции теговой памяти для хранения старших разрядов номера строки ОП, находящейся в данный момент времени в СОЗУ данных кэш-памяти. Таким образом, емкость теговой памяти составляет 64 7-разрядных слов, а СОЗУ данных - 64 строки по 8 n-разрядных слов.



d - бит достоверности строки в кэш-памяти
a - тег; b - индекс (адрес группы строк); a.b - адрес строки в ОЗУ

Рисунок 2.7 - Связь кэш-памяти с основной памятью



- выходы трехразрядного счетчика слов СТ S
Ui - сигналы управления блоком памяти CSi, WRi, RDi, (i=1-3)

EWR=A∈Teg & RD ∨ Y1; Y1-сигнал разрешения записи в RgDIO из процессора

Рисунок 2.8 - Структурная схема кэш-памяти без расслоения обращений с прямым отображением

Следующие младшие 6 бит адреса (поле b) строки используются в качестве адреса теговой памяти и СОЗУ данных, указывающие на одну группу из 128 строк ряда, хранимых в ОП. Из теговой памяти считывается тег (номер столбца ОП) и сравнивается с 7-ью старшими разрядами адреса. Если они совпадают, то это означает, что строка, содержащая текущий адрес i, находится в СОЗУ данных. Параллельно считыванию тега осуществляется чтение данных из СОЗУ данных по адресу, образованному 9-ью младшими разрядами адреса i (поля [b.c]), и если $A \in Teg$, то данные выдаются (записываются) из/в СОЗУ данных на MD в/из RgDIO.

Если считанный тег отличается от поля [a] ($A \notin Teg$), то выполняется процедура обновления строки ОП из кэш-памяти и замещения кэш-памяти из ОП согласно принятой стратегии.

Таким образом, если $A \in Teg$, то при чтении время доступа к кэш-памяти составляет один такт, а время такта равно:

$t_{кэш} = (\max t_{teg} = (\tau_{teg} + \tau_{сх.ср.} + \tau\&), t_{созу\ дан.} = (\tau_{созу\ дан.} + \tau_{rgdio}))$.

При записи: $t_{кэш} = (2 \times \max (t_{teg}, t_{созу\ дан.}))$ (или два такта).

Достоинства:

1. Простота реализации на основе двух СОЗУ с произвольным доступом.

2. Достаточно высокое быстродействие, т.к. блоки памяти тегов и СОЗУ данных работают параллельно при чтении.

3. Память тегов и данных может быть достаточно большой емкости, т.к. кроме обычных БИС памяти необходима одна схема сравнения.

Недостаток: большая вероятность промахов при обращении к кэш-памяти или вероятность частого замещения строк кэш-памяти, т.к. место хранения строки в кэш-памяти однозначно определяется номером индекса строки (группы строк) (например, при частом обращении к строкам кратным 64: 63, 127, 191, 255,..., 8191 каждый раз требуется процедура обновления ОП и замещения кэш-памяти).

Принципы организации доступов к памяти с расслоением обращений и использование системного интерфейса на ширину выборки применимы к любому механизму преобразования адресов, так как данные методы являются универсальными и в дальнейшем не рассматриваются.

2.3.3 Частично-ассоциативное распределение

Является развитием метода прямого и ассоциативного распределения. При этом методе кэш-память строится из модулей, образующих группу строк - не менее двух модулей, например 4.

При данном распределении для выбора группы строк используется метод прямого распределения, а для выбора модуля в группе (строки) - метод полностью ассоциативного распределения (рисунок 2.9). Поле [b], как и при прямом распределении, адресует множество строк, отстоящих друг от друга с шагом кратным 64, а старшие разряды адреса также используются в качестве тега для выбора одной строки из этого множества. При этом в кэш-памяти для каждого адреса поля [b] можно хранить группу до четырех строк из этого множества. Выбор модуля (или одной строки из группы (четырех строк)) определяется по совпадению тегов (старших разрядов адреса) в данной группе и запрашиваемой строки.

Для доступа к массиву тегов, в котором хранятся 7 старших разрядов адреса (поле a), используются 6 младших разрядов адреса [b] строки (индекс группы строк), при этом одновременно осуществляется обращение и к СОЗУ данных. Из четырех модулей теговой памяти выбираются 7-разрядные теги (4 тега) строк и сравниваются с адресом тега из RgФА[a].

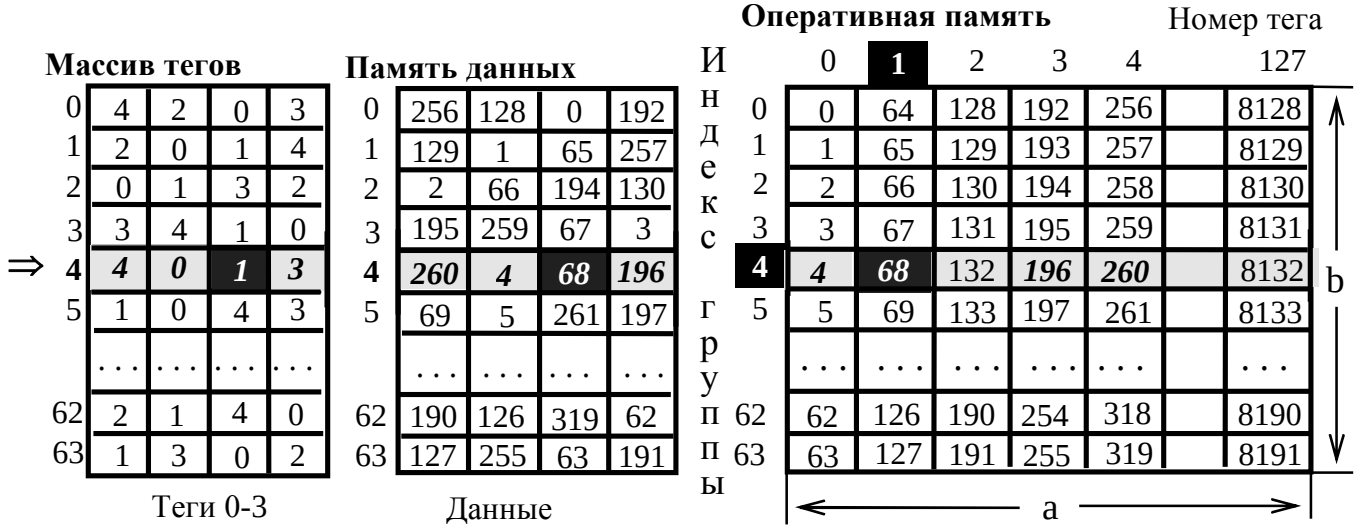


Рисунок 2.9 - Связь кэш-памяти с основной памятью

Если хотя бы один тег совпал, то вырабатывается сигнал A∈Teg и на выходе шифратора CD формируется двоичный номер модуля СОЗУ данных, в котором находится запрашиваемая строка. Так как память тегов и СОЗУ данных работают параллельно, то при чтении на выходе MS/DMS появляется запрашиваемое слово строки (из модуля 1, адрес слова определяется 9-ью младшими разрядами регистра адреса [b.c]) (рисунок 2.10).

Кандидатом на удаление из кэш-памяти однозначно назначается группа из четырех строк полем [b] RgФА, а конкретная строка определяется ассоциативно по одному из методов замещения строк.

Достоинства:

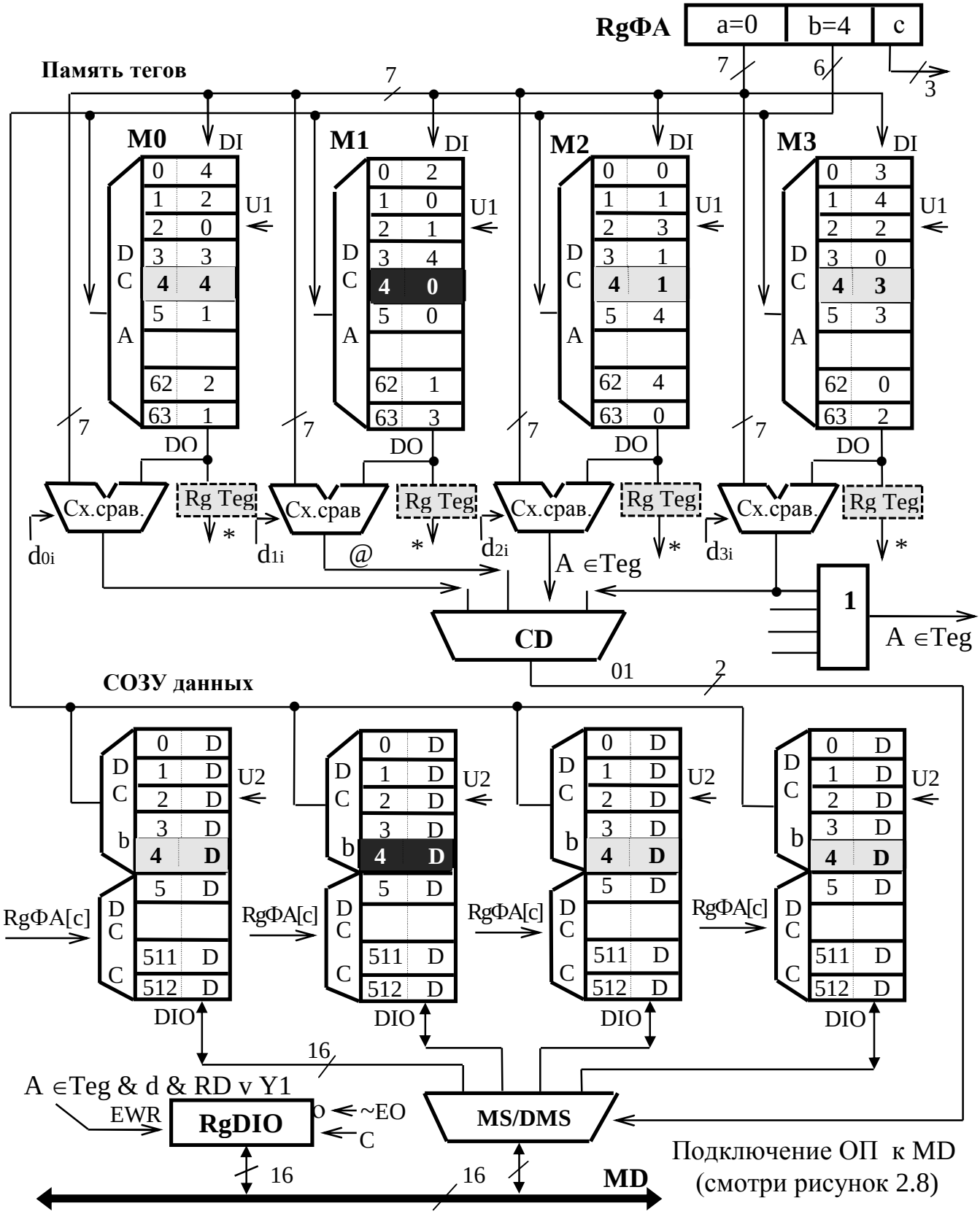
1. Высокое быстродействие: чтение за один такт. Время такта равно: tкэш={max tteg=(τteg +τcx.ср. +τcd + τлэ); tсозу.дан.=(τсозу дан. + τms + τrgdio)}, а запись за 2 такта tкэш = 2 × max {tteg, t созу.дан.};

2. Незначительные аппаратурные затраты (использование БИС памяти с произвольным доступом статического типа);

3. Возможность получения достаточно большой емкости кэш-памяти (при неизменной разрядности полей RgФА емкость кэш-памяти увеличилась в 4 раза);

4. Сокращение числа промахов по сравнению с методом прямого распределения, так как кандидат на замещение строки выбирается как один из четырех модулей по одной из стратегий замещения).

Модификацией структуры кэш-памяти может служить вариант, исключающий из структуры шифратор, а выходы схем сравнения соединяются с соответствующими входами выбора кристалла модулей ~CSi СОЗУ данных. Однако при этом время обращения к кэш-памяти и для чтения, и для записи будет составлять два такта.



- выходы трехразрядного счетчика слов СТ S; dji - бит достоверности строки;
Ui - сигналы управления блоками памяти CSi, WRi, RDi, (i=1-3);

@ - выход схемы сравнения, где совпали теги строк

Рисунок 2.10 - Структурная схема частично-ассоциативной кэш-памяти без расслоения обращений

2.3.4 Распределение секторов

Является развитием метода полностью ассоциативного распределения, основным недостатком которого было использование дорогостоящего АЗУ, информационная емкость и стоимость которого возрастает с увеличением емкости СОЗУ данных.

По методу распределения секторов вся ОП разбивается на секторы, каждый из которых состоит из фиксированного числа строк. То же делается с СОЗУ данных. Пусть сектор состоит из 16 строк, тогда старшие 9 бит адреса указывают на номер сектора, следующие 4 бита - адрес строки в секторе, а младшие 3 бита - адрес слова в строке.

Таким образом, при этом методе сектора в кэш-памяти распределены ассоциативно. Каждой строке, хранимой в СОЗУ данных, соответствует свой бит достоверности строки, который показывает, совпадает или нет содержимое данной строки с содержимым строки в ОП (рисунок 2.11).

Данный метод позволяет существенно снизить емкость памяти тегов на основе АЗУ при той же емкости СОЗУ данных по сравнению с полностью ассоциативным распределением либо увеличить емкость СОЗУ данных при неизменной емкости памяти тегов (для рассматриваемого примера емкость АЗУ уменьшена в 8 раз, а емкость СОЗУ данных увеличена в два раза).

Остальные достоинства и недостатки метода аналогичны полностью ассоциативному распределению.

Алгоритм работы кэш-памяти заключается в следующем:

1. Выполняется проверка бит достоверности секторов d в памяти тегов, и если все di=1, то параллельно производится ассоциативный поиск сектора в памяти тегов по 9-и старшим разрядам RgФА[a].

d - бит достоверности сектора в кэш- памяти
a - тег сектора;
b - номер строки в секторе;
a.b - адрес строки в ОЗУ
f - адрес сектора в СОЗУ данных и памяти бит достоверности строк
D - бит достоверности строки в секторе
X - недостоверная строка в секторе

Оперативная память Номер строки

Тег сектора	0	1	**2**	3	4		15
0	0	1	2	3	4		15
1	16	17	**18**	19	20		31
2	32	33	34	35	36		47
3	48	49	50	51	52		63
4	64	65	66	67	68		79
5	80	81	82	83	84		85
...	...	...	...	...	...		...
510	8160	8161	8162	8163	8164		8175
511	8176	8177	8178	8179	8180		8191

a

b

Память тегов сектров

Адрес	Тег	f	d
0	03	3	1
1	02	5	1
2	04	0	1
3	00	1	0
4	05	7	1
5	510	2	1
6	***01***	***4***	**1**
7	511	6	1

Биты достоверности строк в секторе (D)

Адрес f	012345 ...15
0	101100 ... 0
1	100010 ... 1
2	100000 ... 1
3	000100 ... 0
4	**101000 ... 0**
5	100000 ... 1
6	001100 ... 0
7	100010 ... 0

СОЗУ данных Номер строки

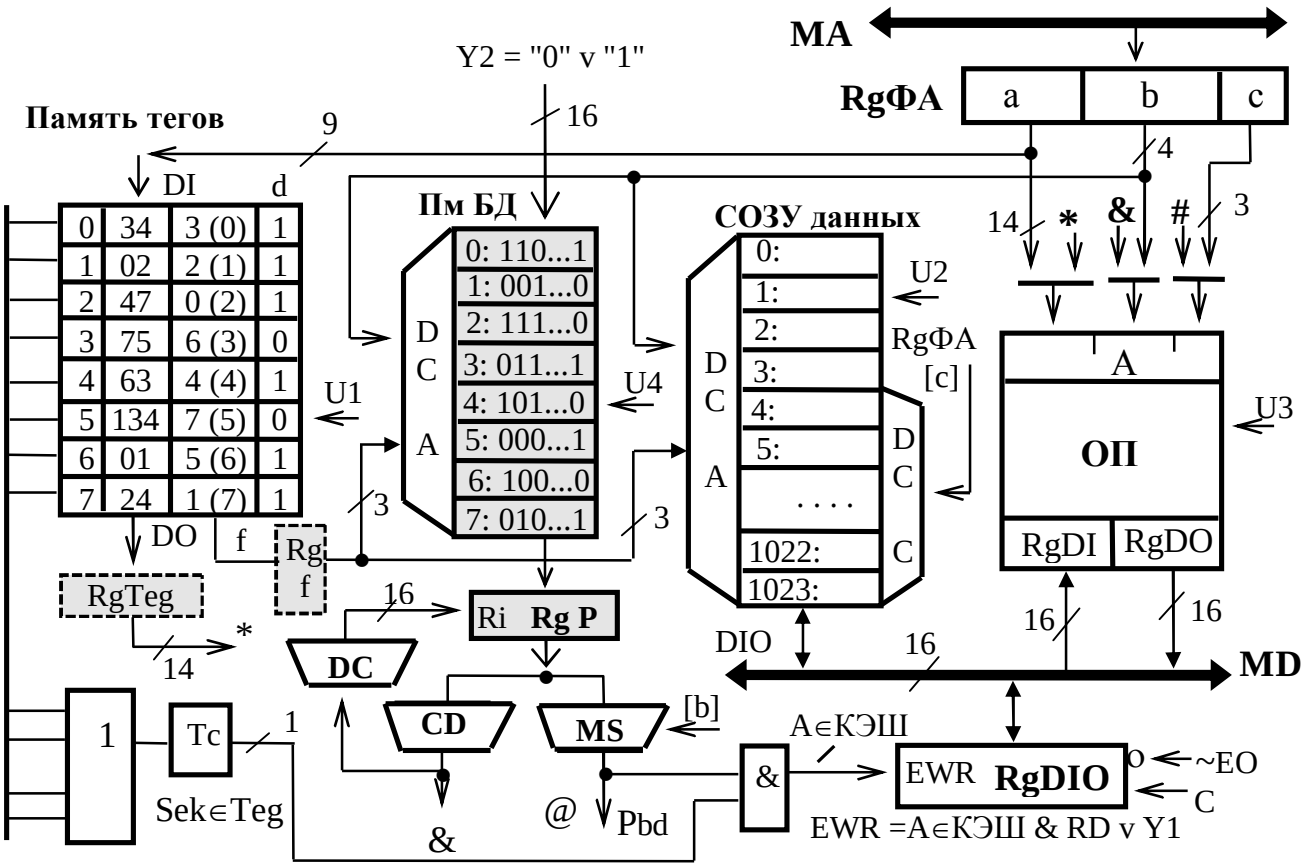
Адрес f	0	1	**2**	3	4		15
0	64	X	66	67	X		X
1	X	X	X	X	X		X
2	8160	X	X	X	X		8175
3	X	X	X	51	X		X
4	***16***	X	***18***	X	X		X
5	32	X	X	X	X		47
6	X	X	8178	8179	X		X
7	80	X	X	X	84		X

Рисунок 2.11 - Связь кэш-памяти секторов с основной памятью

2. Если адрес (сектор) не принадлежит АЗУ (памяти тегов), то определяется кандидат на удаление сектора из кэш-памяти по одной из принятых стратегий и осуществляется замена данного сектора на новый, т.е. удаляется сектор из кэш-памяти в ОП, причем удаляются только те строки, у которых бит достоверности равен "1", и бит достоверности удаленной строки сбрасывается.

3. Далее выполняется процедура замещения, но не всего сектора, а только строки, содержащей запрашиваемое слово, т.е. выполняется пересылка строки из ОП в СОЗУ данных, в память тегов записывается новый тег сектора, а бит достоверности строки устанавливается в "1".

4. Если Sek∈Teg, т.е. хотя бы одна строка сектора уже находится в СОЗУ данных, и если бит достоверности запрашиваемой строки равен "0", то эта строка пересылается из ОП в СОЗУ данных и ее бит достоверности устанавливается. Если же бит достоверности этой строки равен "1", то данное слово считывается (записывается) из/в СОЗУ данных.



- выходы трехразрядного счетчика слов СТ S
Ui - сигналы управления блоком памяти ~CSi, ~WRi, ~RDi, (i=1-4)
& - номер строки для удаления из кэш-памяти в ОЗУ (A∉Teg)
@ - бит достоверности строки при поиске строки в кэш- памяти (A∈Teg)
Y1 - сигнал разрешения записи в RgDIO данных из процессора

Рисунок 2.12 - Структурная схема кэш-памяти с секторным распределением без расслоения обращений

В отличие от остальных алгоритмов замещения рассмотренный метод, кроме выигрыша в емкости АЗУ, дает выигрыш на времени обновления кэш-памяти строками, а не секторами и при дальнейшем замещении строк в секторе СОЗУ данных память тегов секторов не обновляется.

На рисунке 2.12 приведена структурная схема кэш-памяти с секторным распределением. При выполнении обращения к памяти по чтению или записи сначала выполняется ассоциативный поиск на совпадение тега сектора, хранимого в АЗУ, с тегом поля [a] RgФА. Если Sek∈Teg, то по адресу [f],

сформированному или выбранному из памяти тегов выполняется обращение к памяти бит достоверности строк. В регистре RgP фиксируется 16-разрядный код, описывающий состояние строк сектора, а на выходе MS появляется признак бита достоверности запрашиваемой строки. Параллельно при чтении из СОЗУ данных считывается запрашиваемое слово по адресу [f.b.c], и если признак достоверности строки равен "1", то считанное слово защелкивается в RgDIO.

Если бит достоверности строки с выхода MS равен "0", то выполняется процедура замещения кэш-памяти: из ОП считывается запрашиваемая строка, загружается в СОЗУ данных по адресу [f.b] и в памяти бит достоверности строк устанавливается бит по этому же адресу.

При записи в общем случае требуется три обращения: к памяти тегов (Sek∈Teg), памяти бит достоверности строк для определения значения сигнала A∈кэш и записи в СОЗУ данных.

Если Sek∉Teg, то по одной из стратегий замещения назначается сектор для удаления из СОЗУ данных, по адресу [w] считывается содержимое бит достоверности строк сектора в RgP и по адресу [w.&] (где w - 3-разрядный адрес памяти тегов (секторов), назначенный в качестве кандидата на удаление из кэш-памяти, & - четырехразрядный номер строки в секторе, у которой бит достоверности равен "1") из СОЗУ данных считывается строка и переписывается в ОП по адресу [RgTeg . &] (где RgTeg - номер тега сектора, хранимого в АЗУ и назначенного в качестве кандидата на удаление). По окончании обновления очередной строки в RgP с выходов DC сбрасывается очередной бит достоверности строк и процедура обновления ОП продолжается до тех пор, пока содержимое RgP не будет равно нулю. Затем в память бит достоверности всех строк записывается 16-разрядный код нулей (сброс бит достоверности сектора), а в память тегов параллельно загружается новый тег сектора [a] из RgФА. Теперь содержимое памяти бит достоверности строки для запрашиваемого сектора и регистра RgP равны нулю, и выполняется алгоритм замещения строки по ранее описанной методике.

Таким образом, память бит достоверности строк при чтении из нее должна работать в режиме расслоения обращений на 16 бит, а при записи как в режиме расслоения обращений для сброса всех бит достоверности, если Sek∉Teg, и в режиме записи по одному биту, если Sek∈Teg.

2.3.5 Стратегии обновления ОП и замещения кэш-памяти

При **методах сквозной (прямой) записи**, если A∈Teg, то запись выполняется параллельно и в СОЗУ данных кэш-памяти, и в ОП, а при чтении данные выбираются из СОЗУ данных.

С распределением WTWA

Если адрес не принадлежит тегу (кэш-памяти), то и при чтении, и при записи нет необходимости удалять строку из кэш-памяти в ОП, а сразу реализуется процедура замещения кэш-памяти по одной из выбранных стратегий, т.е. из ОП считывается строка и записывается в СОЗУ данных на место строки, назначенной кандидатом на удаление, а в память тегов записывается новый тег считанной строки. За-

тем требуемое слово записывается или считывается из СОЗУ данных по ранее рассмотренному алгоритму, т.к. A∈Teg или при чтении выполняется запись слова в RgDIO с перехватом.

Таким образом, в общем случае данный метод дает выигрыш в быстродействии, если A∉Teg, так как не требуется выполнять процедуру удаления строки из кэш-памяти в ОП (ее копия всегда находится в ОП). Кроме того, выполнение командного цикла процессора продолжается сразу после записи данных в кэш-память, не дожидаясь окончания цикла записи в ОП.

В ряде ЭВМ при использовании данного метода между кэш-памятью и ОП включается небольшой буфер типа FIFO, в который при сквозной записи помещаются данные, если в данный момент времени ОП занята, что позволяет не дожидаться окончания записи предыдущих данных в ОП или ожидать освобождения доступа к ОП со стороны других устройств, а буфер FIFO будет разгружаться по мере освобождения ОП.

В микропроцессоре i486, если адрес не принадлежит кэш-памяти и выполняется запись, то замещение строки не выполняется, а запись производится только в ОП. Такая ситуация возможна только при выполнении команд пересылки MOV данных в ОП и вероятность обращения к этой строке является весьма незначительной.

Без распределения WTNWA

В отличие от предыдущего метода, если адрес не принадлежит тегу (кэш-памяти), то процедура замещения строки из ОП в кэш-память не выполняется, а операции чтения и записи выполняются с ОП. При этом методе в кэш-память первоначально загружаются наиболее активные строки. Особенно актуально применение данного метода для участков многократно выполняемых программ (циклы, ветви, подпрограммы и т.д.).

Существует несколько модификаций данного метода в зависимости от типа хранимых данных (программ или данных).

Методы обратной записи

При методах обратной записи, если A∈Teg, то запись выполняется только в кэш-память, а при чтении данные выбираются из СОЗУ данных.

Простая обратная запись SWB

Если A∉Teg (кэш-памяти), то по одной из выбранных стратегий замещения определяется строка, подлежащая удалению из кэш-памяти в ОП, и выполняется процедура удаления (перезаписи) выбранной строки в ОП как при чтении, так и при записи, а затем реализуется процедура замещения кэш-памяти, т.е. из ОП считывается запрашиваемая строка и записывается в СОЗУ данных на место удаленной строки, а в память тегов - новый тег считанной строки. Затем требуемое слово записывается или считывается из кэш-памяти по ранее рассмотренному алгоритму, т.к. теперь A∈Teg.

Данный метод проигрывает в быстродействии, если при замещении строк (A∉Teg) удаляется строка, которая не модифицировалась (не изменялась за время пребывания в кэш-памяти (особенно для программ)).

Флаговая обратная запись FWB

Является развитием предыдущего метода. Каждой строке ставится в соответствие бит флага записи слова в строку w, т.е. в структуру кэш-памяти вводится дополнительная одноразрядная память флагов емкостью, равной числу строк. Параллельно с чтением памяти тегов выполняется чтение памяти флагов. Если A∈Teg и выполняется запись в СОЗУ данных, то параллельно в одноименной ячейке памяти флагов устанавливается бит w=1. Если A∉Teg, то анализируется значение бита флага и процедура удаления строки из кэш-памяти в ОП выполняется только в том случае, если бит w был установ-

лен в "1", иначе данная процедура не выполняется, а сразу производится замещение строки из ОП в кэш-память со сбросом бита w замещаемой строки.

Существенный выигрыш данный метод дает, если в кэш-памяти хранится программа, т.к. в область программ запись не производится и не требуется процедура удаления строк из кэш-памяти в ОП.

Регистровая обратная запись PWB

Является модификацией метода SWB, при котором к магистрали данных между кэш-памятью и ОП подключается регистровый буфер (типа FIFO), который позволяет сократить время простоя кэш-памяти на выполнение процедур обновления ОП и замещения кэш-памяти.

В регистр помещается строка, подлежащая удалению из кэш-памяти, а ее запись в ОП выполняется после обновления кэш-памяти из ОП. Таким образом, обращение к кэш-памяти со стороны процессора возможно сразу после обновления строки в кэш-памяти, а запись удаленной из кэш-памяти строки в ОП будет отложена и выполнена параллельно с работой кэш-памяти. При этом выигрыш во времени составляет от одного обращения к ОП при использовании расслоения обращений к ОП до k x Тзу при использовании ОП без расслоения обращений.

Флаговая регистровая обратная запись FPWB

Является комбинацией методов PWB и FWB и не требует дополнительных комментариев.

Методы замещения кэш-памяти

Стратегии произвольного замещения не учитывают частоту обращений к строке со стороны ЦП, и велика вероятность, что кандидатом на удаление будет назначена строка, к которой ЦП часто обращается, что снижает общую производительность системы. Однако эти методы наиболее просты с точки зрения технической реализации.

Счетчик адресов в качестве кандидата на удаление (КНУ) определяет последовательные ячейки памяти тегов от 0 до N.

По стратегии FIFO удаляется строка, которая самой первой была переслана в кэш-память (по времени пребывания в кэш-памяти). Как правило, стратегия используется совместно с другими методами, например, со стратегией по биту неиспользования.

Методы, учитывающие активность строк, позволяют сократить число промахов при назначении кандидата на удаление из кэш-памяти, а следовательно, и повысить эффективность использования кэш-памяти.

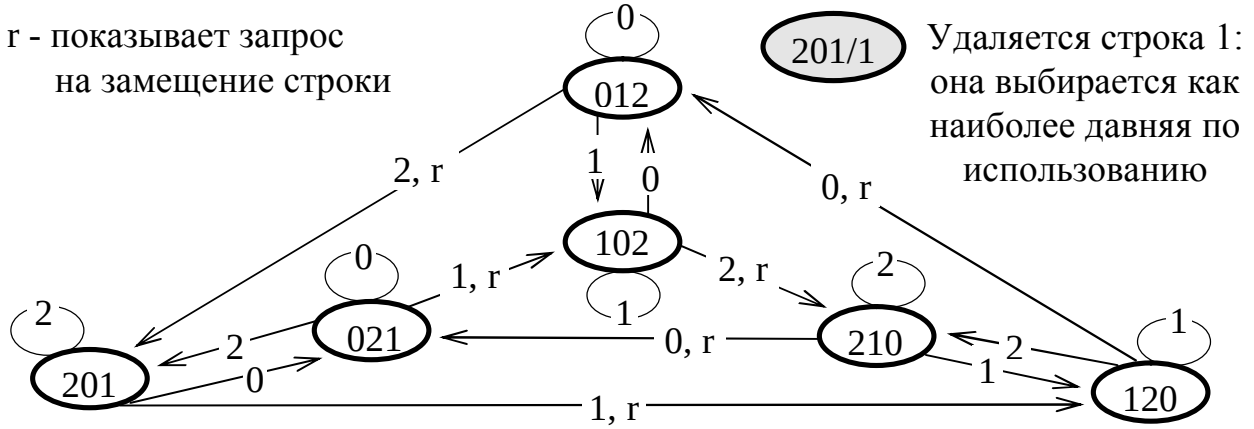
Стратегия LRU: удаляется наиболее давняя по использованию строка. Идея организации псевдо LRU-стека заключается в следующем: при каждом обращении к строке ее номер помещается в стек, в результате замене подлежит строка, хранящаяся в наиболее глубокой позиции стека, и эта строка подлежит удалению первой.

Обращение к строке: b d a a

abcd ⇒ bacd ⇒ dbac ⇒ adbc ⇒ adbc ⇒

Техническая реализация метода псевдо LRU на основе стека весьма проблематична, т.к. сдвиг информации в стеке выполняется не во всех ячейках, а носит случайный характер. Обычно метод псевдо LRU-стека реализуют на основе управляющего автомата с жесткой логикой. Реализация автомата усложняется по мере увеличения числа строк в кэш-памяти, т.к. число возможных комбинаций расположения данных в LRU (число состояний автомата) в общем случае без минимизации составляет n!, где n число замещаемых строк в кэш-памяти. Для 4 строк получим 4!=24 состояния автомата. Поэтому этот метод используется только при частично-ассоциативном распределении, когда число модулей не превышает 2-4, т.е. LRU имеет глубину 4 (для каждой группы строк (модулей)). Ниже приведен пример графа переходов состояний автомата для LRU-стека глубиной 3.

В ЦП i486 реализован следующий алгоритм замещения строк. Пусть B0, B1 и B2 - код текущего состояния автомата с учетом минимизации, однозначно определяющий номер строки в группе, которая подлежит замещению, а L0, L1, L2 и L3 замещаемые строки в группе строк. При каждом попадании в кэш-память биты состояния B0-B2 модифицируются путем изменения кода состояния. В таблице представлен алгоритм выбора замещаемой строки и формирования функций возбуждения для перехода в следующее состояние. После замещения строки новое состояние автомата заносится в блок LRU.



Код состояния B0 B1 B2	Замещаемая строка	Номер последней замещаемой строки	Изменение бита состояния
0 0 X	L0	Если L0 или L1,	то B0=1
0 1 X	L1	Если L2 или L3,	то B0=0
1 X 0	L2	Если L0,	то B1=1
1 X 1	L3	Если L1,	то B1=0
		Если L2,	то B2=1
		Если L3,	то B2=0

При технической реализации данного метода необходимо иметь память LRU состояний автомата для каждой группы строк, емкостью равной емкости одного модуля памяти тегов.

На рисунке 2.13 приведен фрагмент схемы памяти бит достоверности и LRU состояний для кэш-памяти с частично-ассоциативным отображением. Рассмотрим алгоритм работы схемы.

При чтении из памяти тегов параллельно по номеру индекса группы строк (поле b) из памяти LRU состояний считываются и загружаются в регистры бит достоверности строк Rgd и состояний RgS автомата содержимое бит достоверности строк данной группы (4 бита) и 3-разрядный код B0-B2 LRU состояния автомата для группы строк подлежащих замещению.

Если хотя бы один бит достоверности d=0, то удаление строки из кэш-памяти не производится, а запрашиваемая строка считывается из ОП и записывается в модуль, выбираемый сигналом ~CSi, формируемым с выходов дешифратора DC. После замещения строки в память тегов в одноименный модуль записывается новый тег загруженной строки, а в регистре бит достоверности Rgd устанавливается соответствующий бит. Переход к пункту 4.

Если все биты достоверности строк в группе равны "1" и A⊄Teg, то на выходе КС1 формируется код (номер) модуля памяти тегов строки, подлежащей замещению в двоичном или унитарном коде и на выходах MS2 формируется сигнал ~CSi выбора модуля из группы. В зависимости от принятой стратегии обновления ОП (сквозная или обратная запись) выполняется процедура удаления выбранной (одной из четырех) строк из кэш-памяти в ОП (при простой обратной записи), а затем процедура замещения затребованной строки из ОП в кэш-память.

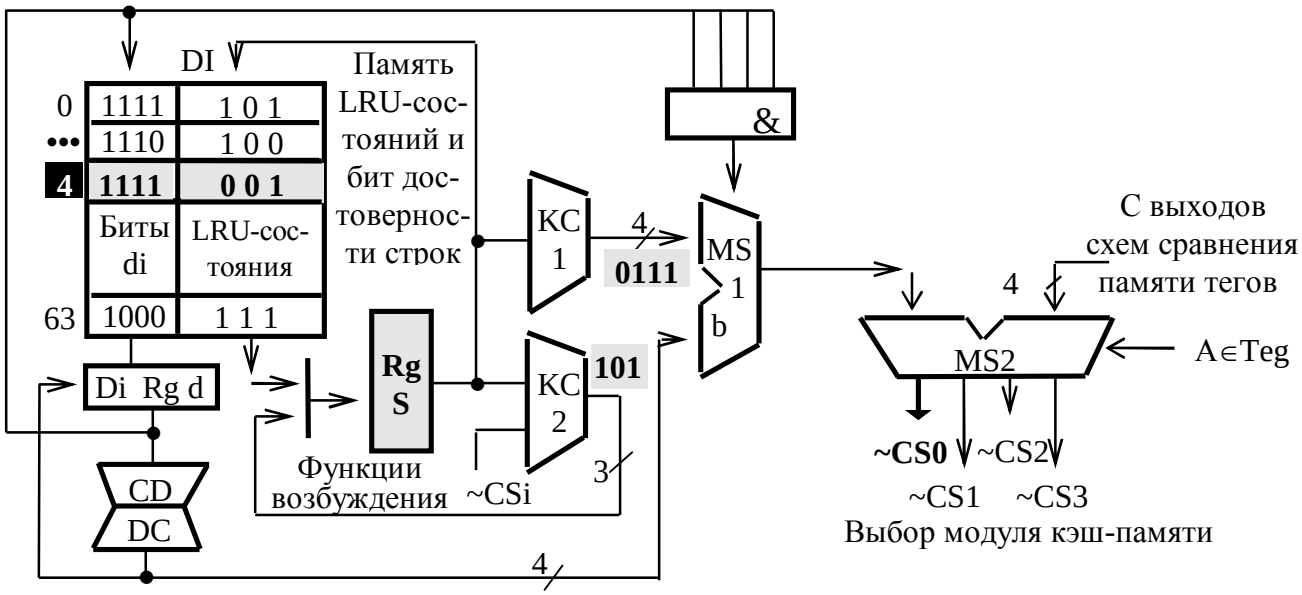


Рисунок 2.13 - Фрагмент схемы памяти бит достоверности и LRU состояний для кэш-памяти с частично-ассоциативным отображением

4. Выполнить требуемую операцию чтения или записи информации с кэш-памятью и параллельно на выходах КС2 в зависимости от набора входных сигналов ~CSi формируется код следующего состояния автомата, определяющего нового кандидата на замещение, и этот код состояния записывается в RgS, а затем в память LRU состояний параллельно со считанными ранее битами достоверности группы строк.

По признаку неиспользования: для каждой строки кэш-памяти вводится бит активности строки, который устанавливается при каждом обращении к строке при чтении или записи в кэш-память.

При частично-ассоциативном распределении память бит неиспользования строится по аналогии памяти бит достоверности в предыдущем примере (рисунок 2.13).

Для полностью ассоциативного и распределения секторов реализация данного метода требует значительных аппаратурных затрат и может использоваться только для памяти тегов небольшой емкости (16-64 ячеек).

На рисунке 2.14 приведен пример реализации схемы для полностью ассоциативной кэш-памяти с флагов ой обратной записью и назначением кандидата на удаление по признаку неиспользования.

Регистр битов активности можно построить на асинхронных RS-триггерах. Унитарный код, формируемый с выходов схем сравнения АЗУ при ассоциативном поиске, служит источником для установ-

ки бита активности по входу Si Rg BN. Биты активности, установленные в "0", определяют строки кэш-памяти, как кандидата на удаление из кэш-памяти в ОП или на замещение.

1. Если A∈Teg, то по адресу, сформированному схемой CD1 или выбираемому из памяти тегов (поле [f]), при выполнении записи в СОЗУ данных параллельно выполняется запись в память флагов кода "1".

2. Если A∉Teg, то по адресу, формируемому схемой CD2, назначается строка в качестве кандидата на удаление из кэш-памяти. По этому адресу из памяти флагов считывается значение бита w в триггер Tw для определения необходимости выполнения процедуры обновления ОП (при Tw=0 обновление ОП не выполняется).

3. Если Tw=1, то по этому же адресу строка считывается из СОЗУ данных и записывается в ОП по адресу RgTeg[a].

4. Из ОП считывается новая строка по адресу из RgФА[a] и загружается на место удаленной строки, а в памяти флагов соответствующий бит w сбрасывается, и параллельно в память тегов загружается новый тег из RgФА. Выполнить пункт 1.

Если в регистре признака неиспользования все биты будут установлены в "1", то произойдет автоматический сброс всех бит Rg BN и подсчет активности строк начинается сначала.

При увеличении емкости кэш-памяти для полностью ассоциативного и секторного распределений биты активности строк можно также хранить в отдельной одноразрядной памяти бит неиспользования, построенной на основе АЗУ. В качестве эталона для сравнения выступает код нуля, а с выходов АЗУ (поле [f]) будет сниматься код адреса строки, подлежащей замещению, если A∉Teg. Однако, такой подход требует значительных аппаратурных затрат и имеет высокую стоимость.

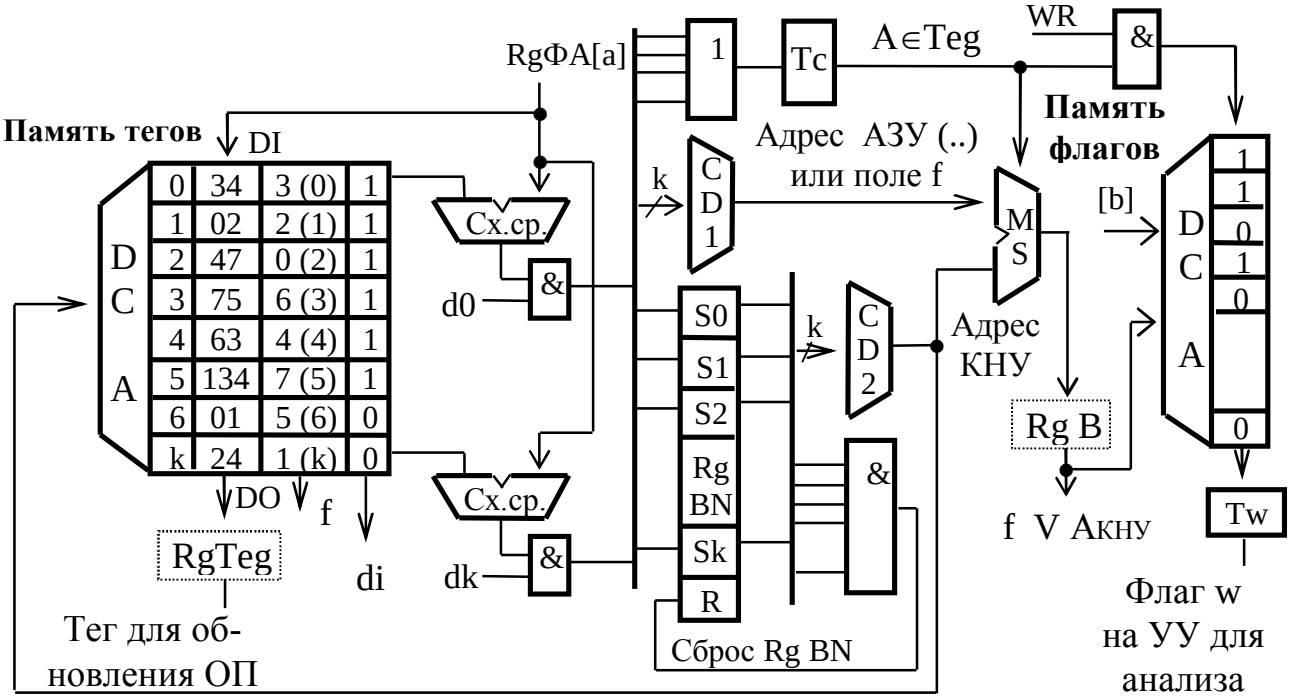


Рисунок 2.14 - Схема памяти тегов на основе АЗУ с памятью бит неиспользования и памятью флагов

Выбор стратегии замещения тесно связан с реализованным методом распределения кэш-памяти. Например, при прямом распределении кандидат на удаление однозначно определяется номером индекса, метод псевдо LRU-стека и биты неиспользования, как правило, применяется только для частично-ассоциативной кэш-памяти, FIFO - совместно с битом неиспользования для частично-ассоциативного

распределения и для полностью ассоциативного или секторного распределений при небольшой емкости АЗУ и т.д.

На рисунках 2.4, 2.5, 2.6, 2.8 и 2.12 на входах ОП показаны схемы мультиплексирования адреса: при сквозной записи операнда в ОП или при выполнении процедуры обновления ОП старшие разряды адреса выбираются из памяти тегов, а остальные в зависимости от метода распределения кэш-памяти (поле [b]) из RgФА и организации расслоения обращений ОП (счетчик слов в строке СТ S). При выполнении процедуры замещения кэш-памяти адрес строки берется из RgФА.

При использовании расслоений обращений в ОП (рисунок 2.5) для организации записи слова в ОП при сквозной записи и при вводе данных в ОП из ПУ в структуру УУП введен дешифратор выбора кристалла, позволяющий выполнять операции чтения/записи либо всей строки по ширине выборки для всех модулей памяти параллельно, либо в один из выбираемых дешифратором модулей памяти.

2.3.6 Методы увеличения быстродействия кэш-памяти

К методам увеличения быстродействия кэш-памяти можно отнести:

♦ применение метода расслоения обращений ОП при обновлении ОП и замещении кэш-памяти данных;

♦ использование СОЗУ данных кэш-памяти с двусторонним доступом и расслоением обращений СОЗУ данных и ОП;

♦ разделение кэш-памяти на независимые кэш-память команд и кэш-память данных с использованием различных стратегий обновления и замещения строк, т.к. для программ нет необходимости записи и удаления строк из кэш-памяти в ОП, если программа не модифицируется в процессе выполнения. Кроме того, процедуру замещения можно выполнять не строками, а блоками размером до 256 байт благодаря хорошей пространственной локальности программ, что в совокупности позволяет уменьшить время замещения;

♦ использование обходных буферов при обновлении ОП:

а) буфер FIFO при использовании сквозной записи для реализации отложенной записи в ОП;

б) введение регистрового буфера между кэш-памятью и ОП при удалении строки из кэш-памяти в ОП;

♦ использование флаговой обратной записи, что сокращает время на обновление ОП;

♦ использование двухуровневой кэш-памяти. Между кэш-памятью и ОП включается дополнительная кэш-память, называемая системной (или внешней) кэш-памятью. Для взаимодействия внутренней кэш-памяти с внешней используется метод сквозной записи, а для взаимодействия внешней кэш-памяти с ОП - метод обратной записи.

Основной выигрыш получается на операциях замещения, т.к. считывание строки во внутреннюю кэш-память выполняется из внешней кэш-памяти, имеющих небольшой разброс времени по быстродействию (1-2 такта). Если адрес не принадлежит внутренней и внешней кэш-памяти, то удаление строки в ОП производится только из внешней кэш-памяти, так как она хранит все копии строк. Процедура замещения строки из ОП может выполняться параллельно во внешнюю и внутреннюю кэш-память. Обе кэш-памяти могут быть реализованы по различным методам распределения, иметь различные стратегии обновления ОП и замещения кэш-памяти и емкость (для внешней кэш-памяти емкость достигает до 512-1024 кбайт).

2.3.7 Пример алгоритма работы контроллера кэш-памяти

На рисунке 2.15 приведен фрагмент микропрограммы работы контроллера кэш-памяти с распределением секторов с обратной флаговой записью строк сектора с замещением по признаку неиспользования сектора. ОП и СОЗУ данных построены с расслоением обращений на длину строки. Адрес и данные предварительно загружены в RgФА и RgDIO памяти.

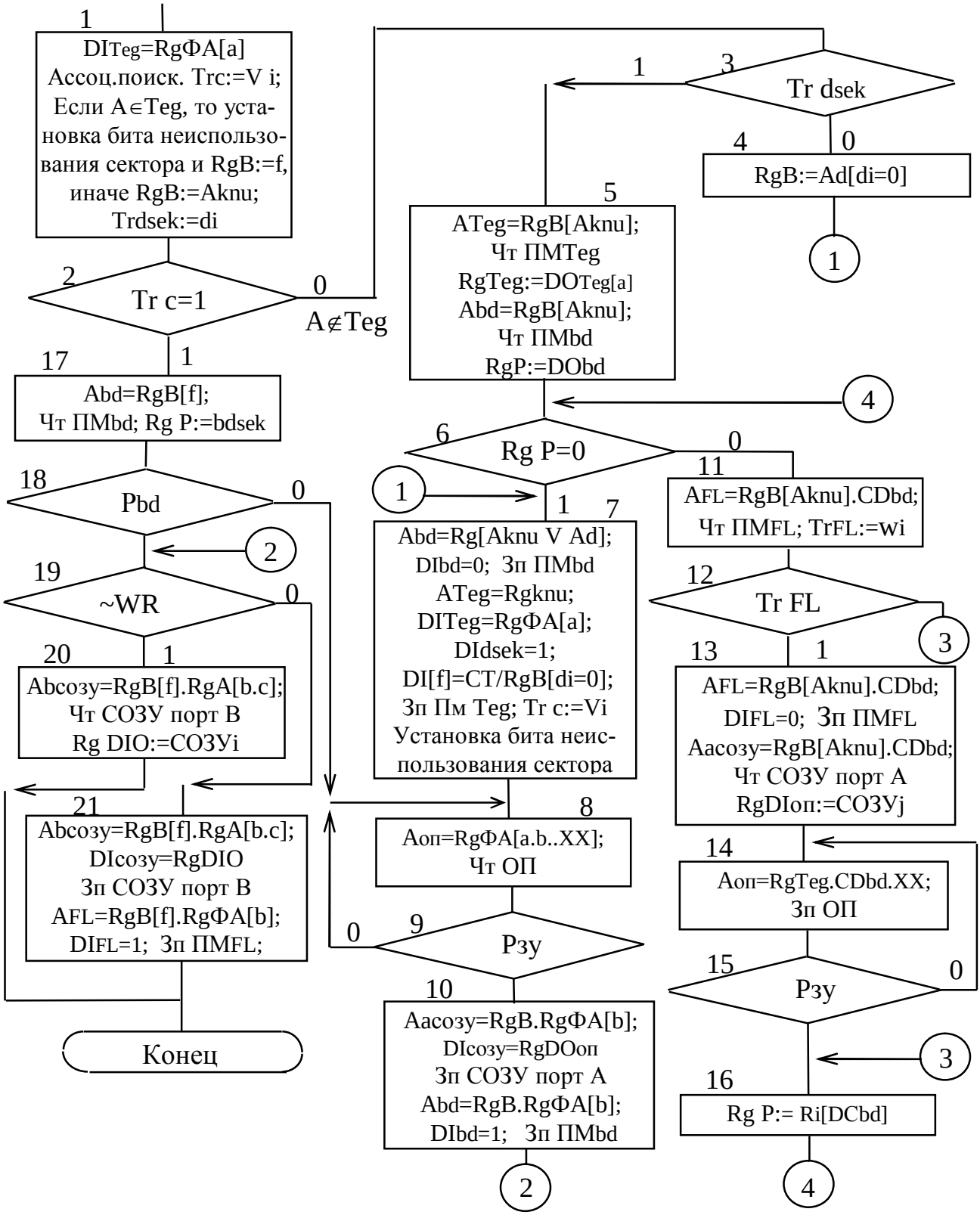


Рисунок 2.15 - Микропрограмма работы контроллера кэш-памяти

1. В блоке 1 на входы DI памяти тегов поступает номер тега из RgФА и выполняется ассоциативный поиск по всем ячейкам номеров тега секторов. В триггере Trc фиксируется значение признака A∈Teg сектора, и если A∈Teg, то устанавливается бит признака неиспользования сектора, а в буферном регистре RgB (рисунок 2.14) - значение адреса сектора СОЗУ данных f, хранимого в поле памяти тегов или формируемого схемой шифратора CD1. Триггер бита достоверности секторов Trdsek устанавливается в "0", если в АЗУ памяти тегов в поле бит достоверности сектора имеется хотя бы один недостоверный сектор, иначе Trdsek := 1.

2. Если A∈Teg (блок 2), то выполняется считывание значений бит достоверности строк сектора в Rg P (блок 17) (рисунок 2.12) и на выходе MS (@) формируется значение бита достоверности запрашиваемой строки Pbd. Если Pbd=0, то выполняется процедура замещения строки из ОП в кэш-память (блоки 8-10). Переход к пункту 9.

3. Иначе Pbd=1, т.е. строка достоверна и в блоках 19-21 выполняется запись данных или чтение из СОЗУ данных. Параллельно при записи в СОЗУ данных выполняется установка признака в памяти флагов. Конец.

4. Если A∉Teg, то сначала проверяется значение триггера достоверности сектора Trdsek (блок 3). Если Trdsek=0, то в памяти тегов имеется недостоверный сектор, подлежащий замещению без выполнения процедуры обновления ОП. В блоке 4 в RgB фиксируется адрес сектора, у которого бит достоверности di=0.

5. В блоке 7 в памяти бит достоверности строк сбрасываются все биты сектора. Выполняется загрузка нового тега, перезапись адреса сектора СОЗУ данных и установка бита достоверности сектора в памяти тегов. При этом автоматически в регистре бит неиспользования секторов будет установлен соответствующий бит и выполняется процедура замещения запрашиваемой строки из ОП в кэш-память (блоки 8-10). Переход к пункту 9.

Если бит достоверности сектора Trdsek=1, то все сектора в памяти тегов достоверны, и выполняется процедура обновления ОП строк сектора, назначенного в качестве кандидата на удаление.

В блоке 5 по адресу, сформированному в RgB (блок 1) в качестве сектора кандидата на удаление, из памяти тегов считывается значение поля тега [a] и фиксируется в буферном регистре RgTeg, а из памяти бит достоверности строк по этому же адресу считываются 16 бит достоверности строк сектора и фиксируются в RgP.

6. Далее анализируется значение Rg P (блок 6), и если он не равен нулю, то на выходах шифратора CD (рисунок 2.12) формируется адрес строки, у которой бит достоверности равен 1, а из памяти флагов выбирается значение признака записи в эту строку и фиксируется в TrFL (блок 11).

7. Если флаг равен нулю, то в Rg P сбрасывается бит достоверности строки с выходов DC (блок 16) и выполняется переход к блоку 6 для анализа содержимого Rg P на равенство нулю. Иначе в строку была произведена хотя бы одна запись и в памяти флагов сбрасывается соответствующий бит, а из СОЗУ данных считывается строка в RgDIоп (блок 13) и записывается в ОП на место старой по адресу, определяемому содержимым RgTeg и шифратором адреса строки с битом достоверности равным "1" (блоки 14-15).

В Rg P сбрасывается соответствующий бит достоверности строки (блок 16) и переход на анализ содержимого Rg P. Пункт 6 выполняется в цикле до тех пор, пока Rg P не станет равным нулю.

8. Если Rg P = 0, то выполняется пункт 5 (блок 7).

9. Из ОП считывается запрашиваемая строка (блоки 8-9), а в блоке 10 осуществляется запись считанной строки в СОЗУ данных и установка бита достоверности новой строки. Переход на пункт 3 для выполнения чтения или записи данных в СОЗУ данных.

На основе данного алгоритма можно разработать алгоритмы работы кэш-памяти и для других методов распределения и стратегий обновления ОП и замещения кэш-памяти.

Следует заметить, что эффективность использования кэш-памяти существенно зависит не только от правильно выбранных методов распределения кэш-памяти и стратегий обновления ОП и замещения кэш-памяти, но и от ее технических характеристик: емкости СОЗУ данных, времени доступа и размера строки. Так, чем больше емкость СОЗУ данных, тем больше данных может храниться в кэш-памяти, а следовательно, тем выше вероятность попадания при обращении к данным. Правильный выбор стратегии замещения кэш-памяти позволяет хранить в кэш-памяти только активные строки, а использование флаговой логики обновления ОП сокращает количество обращений к ОП. Сквозная запись с применением буфера записи в ОП позволяет реализовать принцип отложенной записи в ОП, что сокращает время простоя процессора при обращении к кэш-памяти. Таким образом, правильный выбор сочетания различных методов организации кэш-памяти позволяет существенно повысить вероятность обращения к ней до 85-95%, а разделение кэш-памяти на память программ и данных позволит не только иметь две кэш-памяти достаточно большой емкости, но и обеспечить применение различных стратегий обновления и замещения наиболее эффективных к каждому типу данных: команд и операндов.

На основе рассмотренных методов можно получить более 200 вариантов различных технических реализаций кэш-памяти.

2.4 Процессоры с конвейеризацией команд

При последовательном выполнении командный цикл процессора в общем случае включает следующие этапы:

1. Выборка команды IF.

2. Декодирование команды и вычисление эффективного адреса D.

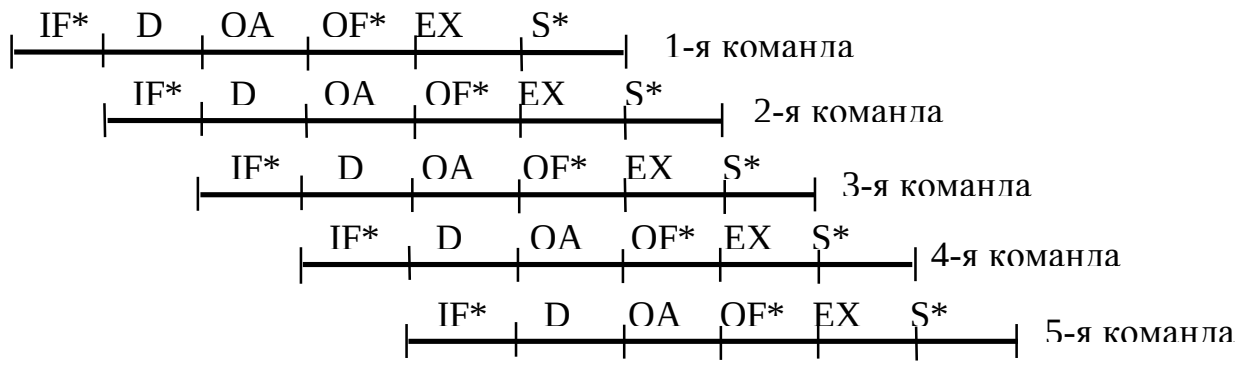
3. Формирование физического адреса операнда - преобразование логического адреса в физический OA.

4. Выборка операнда из ОП OF.

5. Выполнение команды в АЛУ EX.

6. Запись результата S.

С целью повышения производительности процессора часто используется метод конвейеризации команд, т.е. необходимо организовать непрерывное выполнение потока команд. Тогда для выполнения одной команды потребуется время, затрачиваемое на прохождение одной ступени конвейера (в идеальном случае).



Для реализации конвейера команд необходимо иметь **независимые аппаратные средства** для каждой ступени обработки с целью распараллеливания выполнения команд на разных стадиях обработки **с автономными устройствами управления (децентрализованное управление).**

Наиболее эффективно конвейер работает, если времена выполнения на каждой стадии командного цикла равны (синхронный принцип управления от 1 до n тактов на каждой стадии). Если же имеется некоторый разброс по времени, то его можно устранить, установив:

- в требуемых местах буферы, что обеспечивает сохранность информации для передачи на следующую еще занятую обработкой ступень;
- для каждой ступени осведомительные триггеры занятости и переход от одной ступени к другой будет выполняться только при условии, что обработка информации на предыдущей ступени завершена;
- на каждой ступени при наличии своего УУ можно обеспечить синхронно-асинхронный принцип управления, при котором короткие микрокоманды выполняются за один такт, а длинные - за 2-3 такта работы УУ, однако УУ всех ступеней конвейера должны работать на единой опорной частоте задающего ГИ, а суммарное время выполнения микрокоманд на каждой ступени должно быть одинаковым.

На рисунке 2.16 представлена структура процессора с применением буферов в возможных местах нарушения работы конвейера команд. Местоположение буфера и его глубина выбираются из конкретных алгоритмов работы процессора.

Ниже представлены временные диаграммы работы процессора с конвейеризацией команд без буферов и кэш-памяти и при наличии буферов и кэш-памяти с двусторонним доступом. Из диаграмм видно, что основное время простоя возникает из-за конфликтных ситуаций при обращении к ОП. Поэтому введение в структуру кэш-памяти, кроме промежуточных буферов, позволяет значительно сократить времена простоя, а кэш-память с двусторонним доступом и при необходимости раздельная кэш-память команд и кэш-память данных позволяют при отсутствии промахов обращения к кэш-памяти обеспечить безостановочную работу конвейера.

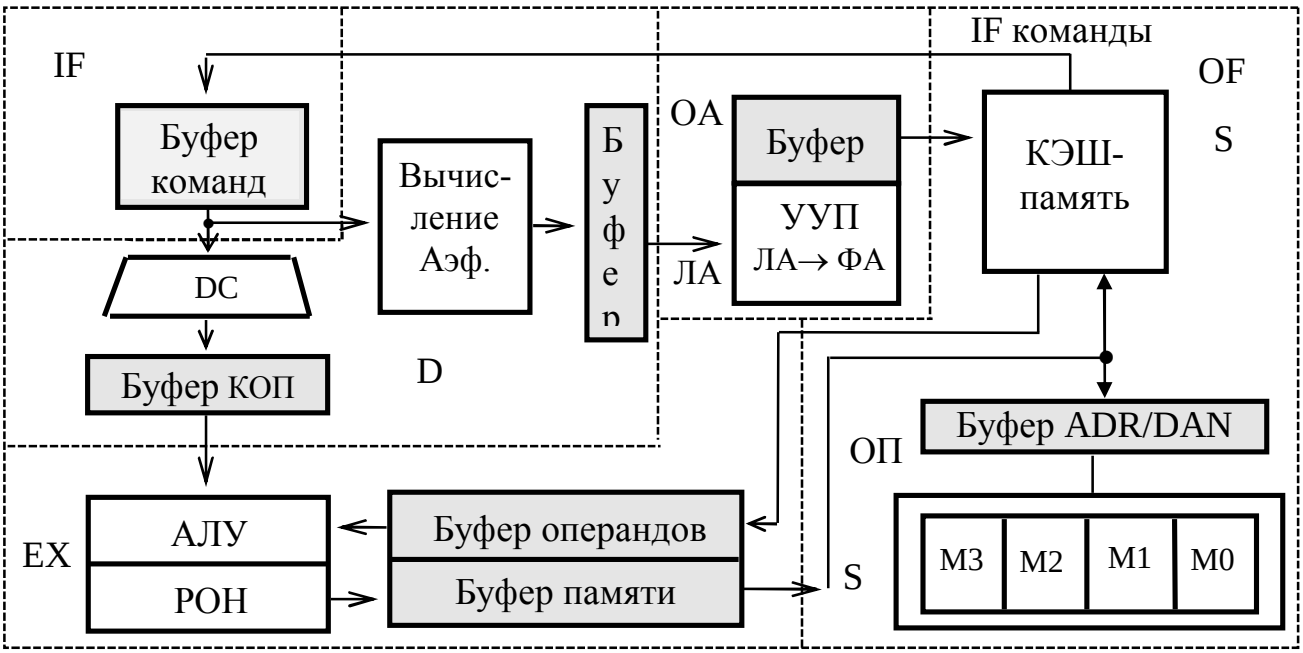
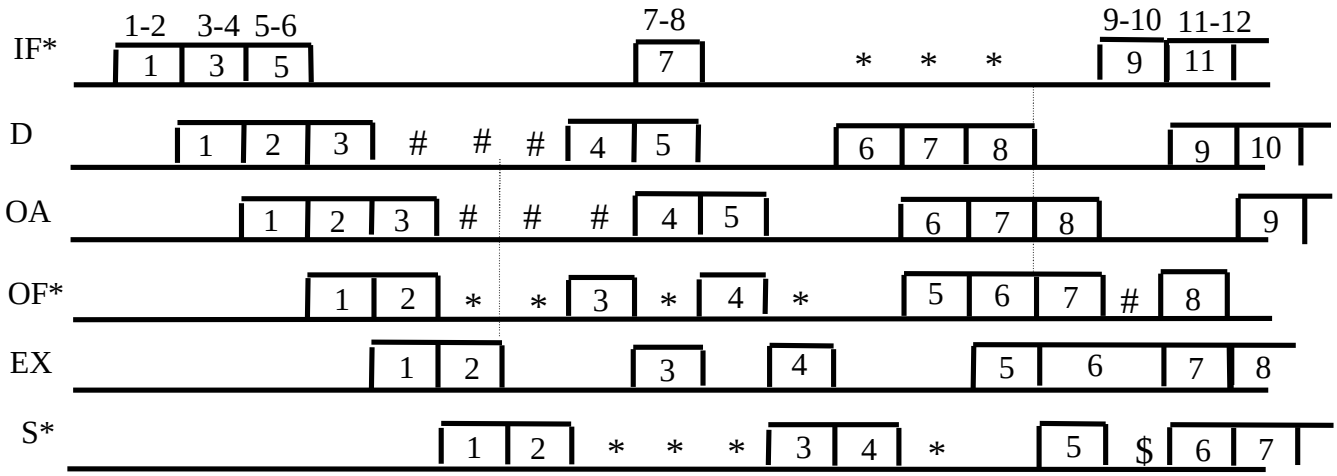


Рисунок 2.16 - Структура процессора с буферами между уровнями конвейеризации

Количество стадий (уровней) конвейеризации может быть различным и зависит от структуры процессора, максимального и минимального количества микрокоманд в командном цикле процессора, времени выполнения каждой микрокоманды и т.д. Необходимо обеспечить главное условие, чтобы время выполнения участка микропрограммы на каждой ступени было одинаковым.

Работа конвейера команд может быть нарушена по многим причинам, которые можно разделить на две группы:

- ♦ приводящие к приостановке выполнения преобразований на следующей ступени конвейера;
- ♦ приводящие к возобновлению работы конвейера, начиная с первой ступени.



* - простои из-за конфликтов по доступу к памяти

$ - простои из-за выполнения длинных операций в АЛУ

\# - простои из-за отсутствия буферов между уровнями

1-6 команды формата память-регистр с записью в память;
7-10 команды формата с записью в РОН

Рисунок 2.17 - Временная диаграмма работы конвейера команд без буферов и кэш-памяти

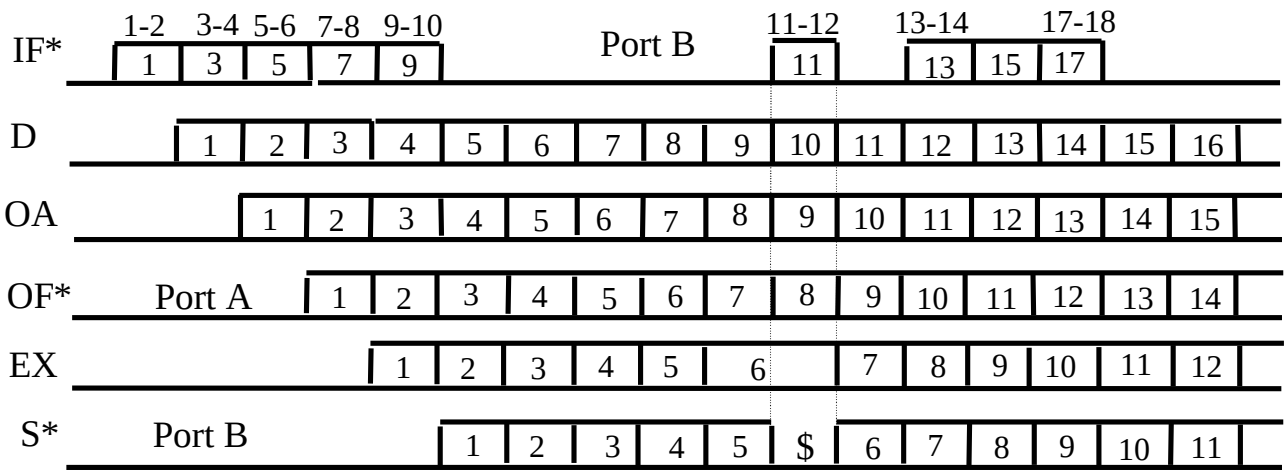


Рисунок 2.18 - Временная диаграмма работы конвейера команд при наличии буферов и кэш-памяти с двусторонним доступом

К первой группе причин относятся:

1. При необходимости использования результата операции, полученного в результате выполнения текущей команды, в следующей команде.

2. Определение адреса операнда в некоторой команде с помощью предыдущей команды.

3. Необходимость блочной (строковой) пересылки из ОП в кэш-память, если в последней отсутствует команда или операнд.

4. Наложение стадий выборки команды, операнда и/или записи результата, т.е. возникновение конфликтов при доступе к памяти.

5. Выполнение слишком длинных операций в АЛУ, требующих времени больше, чем для одного этапа преобразования команды и т.п. для других видов преобразований в процессоре.

Ко второй группе причин относятся:

Изменение последовательности выполнения команд для команд условного и безусловного перехода, вызова подпрограмм и т.д.

Возникновение прерываний как аппаратных (внешних и внутренних), так и программных.

Поэтому необходимо различать два подхода для устранения влияния перечисленных ситуаций: методы для обеспечения непрерывной работы конвейера, которые приводят к увеличению времени простоя процессора на каждой ступени конвейеризации, и методы, нарушающие последовательность выполнения команд, находящихся в буфере очереди команд и уже принятых к обработке на первых ступенях конвейера.

Причины первой группы легко устранить, если:

* пересылать некоторые данные отдельно от основного потока кратчайшим путем к местам, где они требуются (причины 1, 2);

* установить буферы (3, 5);

* разделить память (память программ и память данных, кэш-память программ и кэш-память данных, использование памятей с двухсторонним и многосторонним доступом) (4);

* использовать регистровый файл по назначению и отложенной записи в ОП результата вычислений (4).

* причину 5 можно устранить путем мультиплексирования АЛУ (введение в структуру специализированных АЛУ (ОУ) для длинных операций, например матричного типа) или его конвейеризации.

Причины второй группы: для команд условного перехода, если условие не выполняется, то порядок выполнения команд не нарушается и конвейер продолжает работу, а при выполнении условия, а также по командам безусловного перехода, CALL и при возникновении прерываний команды, введенные в конвейер и в очередь команд, считаются недействительными (сброс указателей стека очереди команд) и осуществляется перезапуск алгоритма со стадии выборки команды. При этом основной проблемой является блокировка выполнения команд, следующих за командой перехода после стадии декодирования этой команды и вычисления адреса перехода, путем перетрансляции потока микрокоманд в NOP либо после формирования адреса перехода блокировки записи результата в ОП и РОН на стадии S (операции чтения являются безобидными).

Для устранения причин второй группы существует ряд способов:

♦ с использованием буфера цикла команд достаточно большой емкости, в котором хранится весь программный цикл (подпрограмма), или нескольких таких буферов. Тогда при выполнении условия (вызове подпрограммы или прерывании) доступ осуществляется к буферу цикла, а не к ОП;

♦ множественного потока команд, при котором процессор имеет два буфера команд: основной и вспомогательный. Основной буфер используется для хранения очереди команд, если условие не задано

(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.

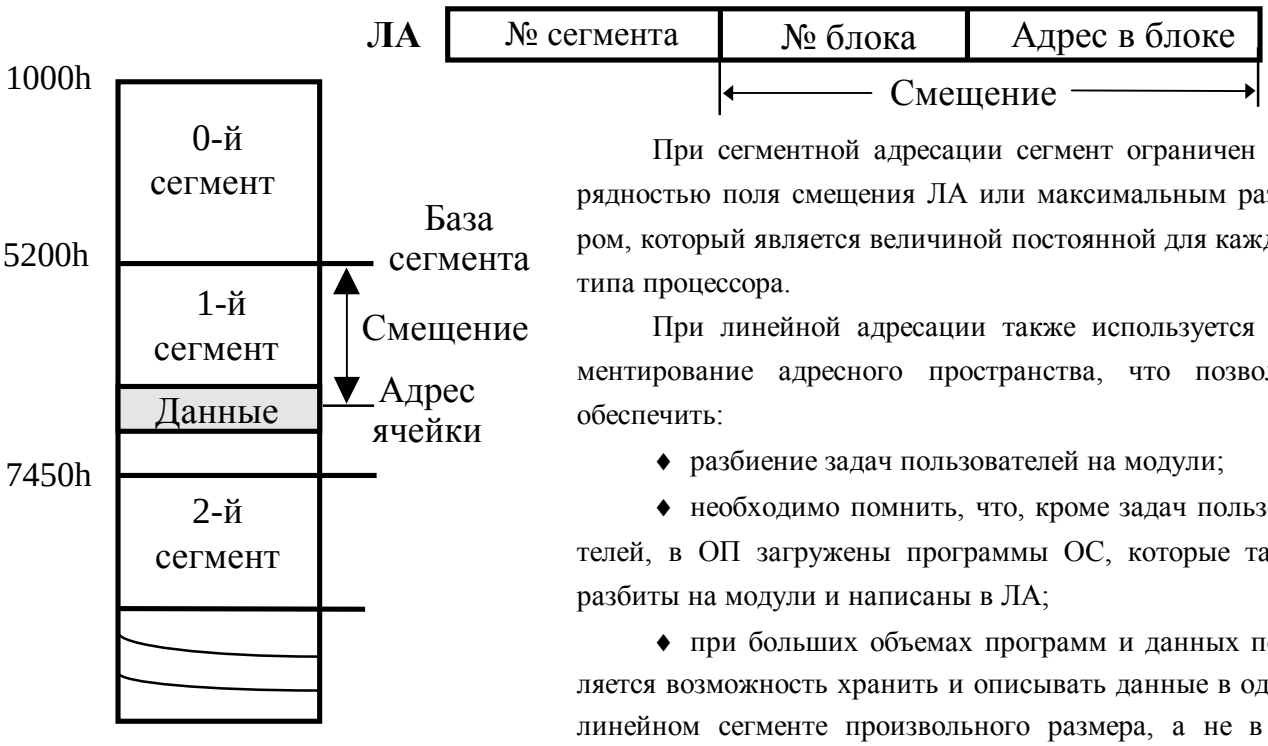


Рисунок 1.1 - Сегментированное адресное пространство

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

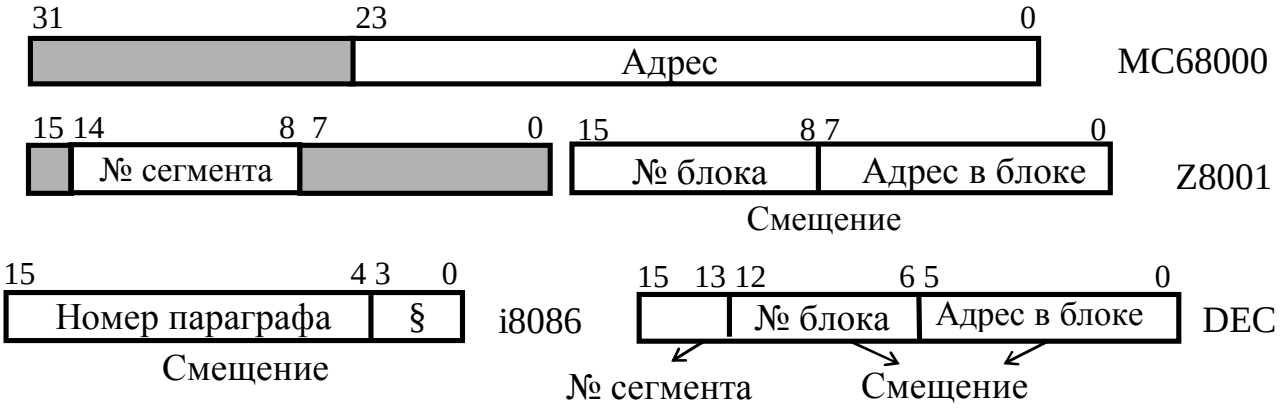


Рисунок 3.2 - Форматы логического адреса процессоров

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.

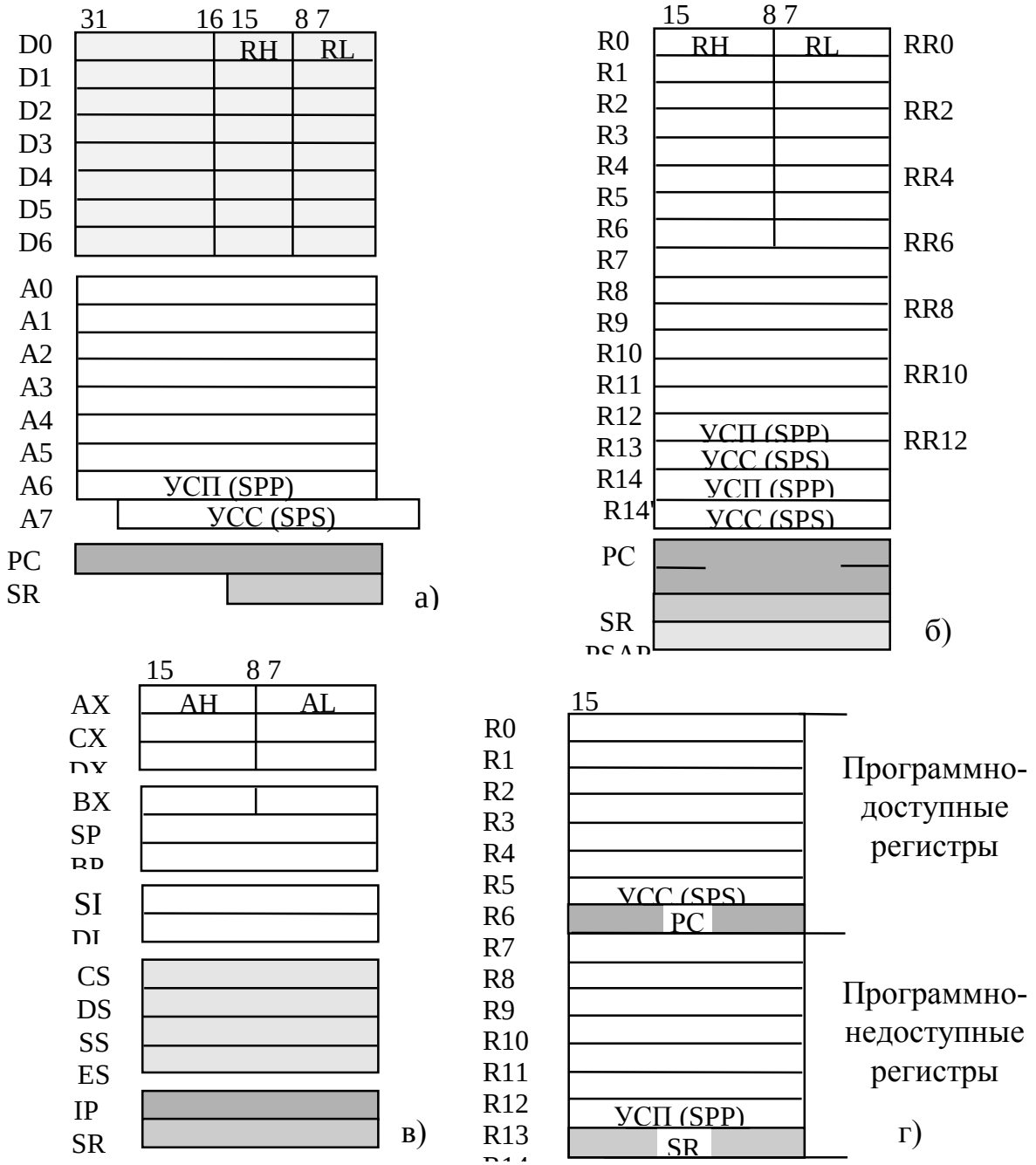


Рисунок 3.2 - Структура регистров процессоров MC6800 (а), Z800 (б), Intel 8086 (в), DEC (г)

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.

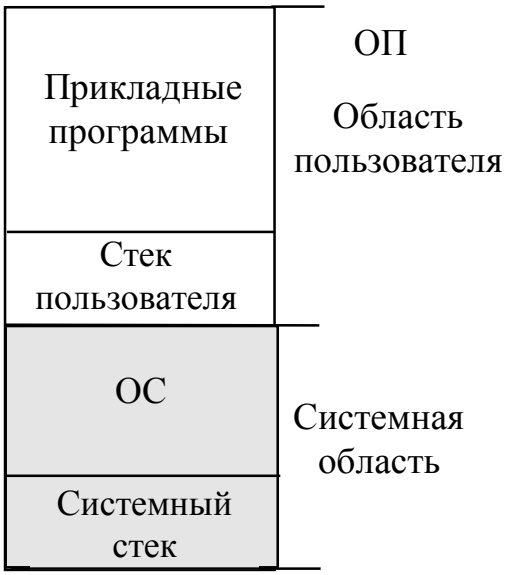
3.3 Режимы работы процессора и виды прерываний

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.

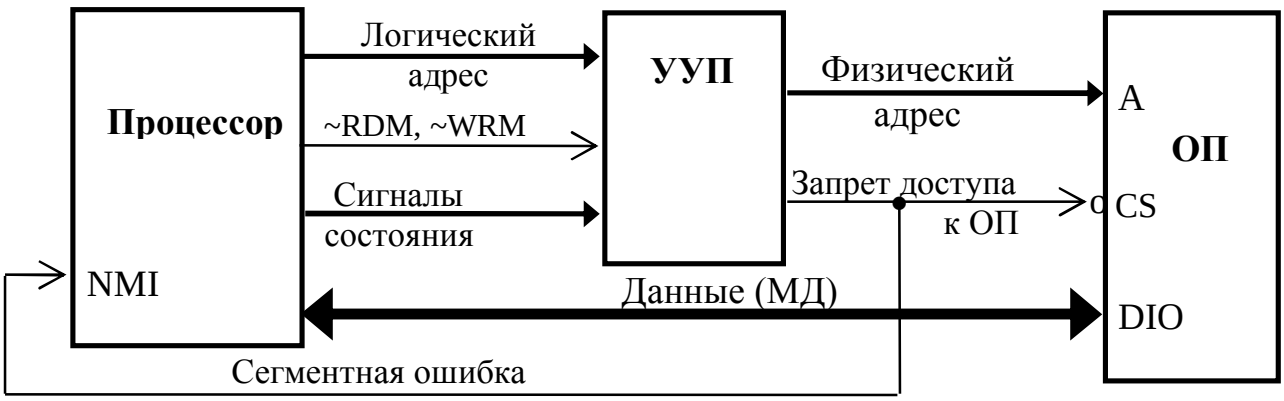


Рисунок 3.3 - Преобразование логического адреса в физический с помощью устройства управления памятью

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:

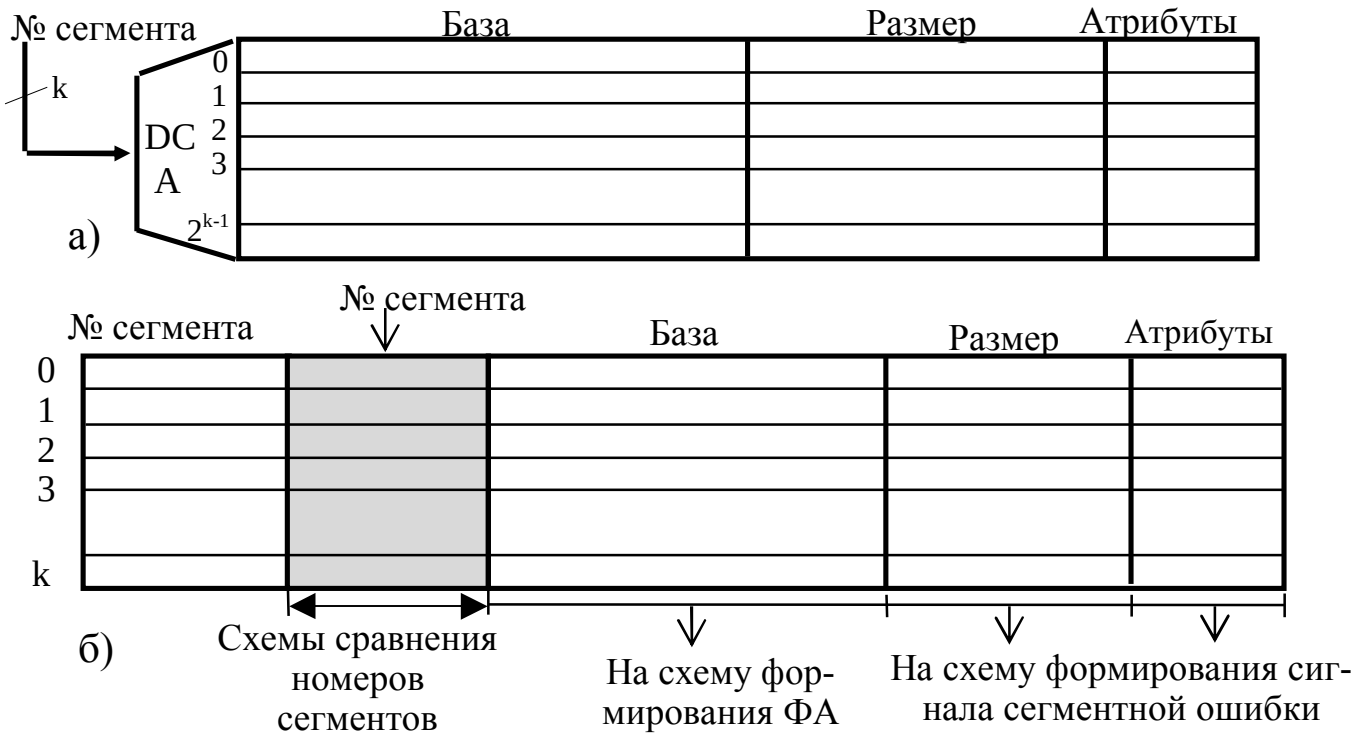


Рисунок 3.4 - Способы выбора регистров-дескрипторов сегментов

* 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).

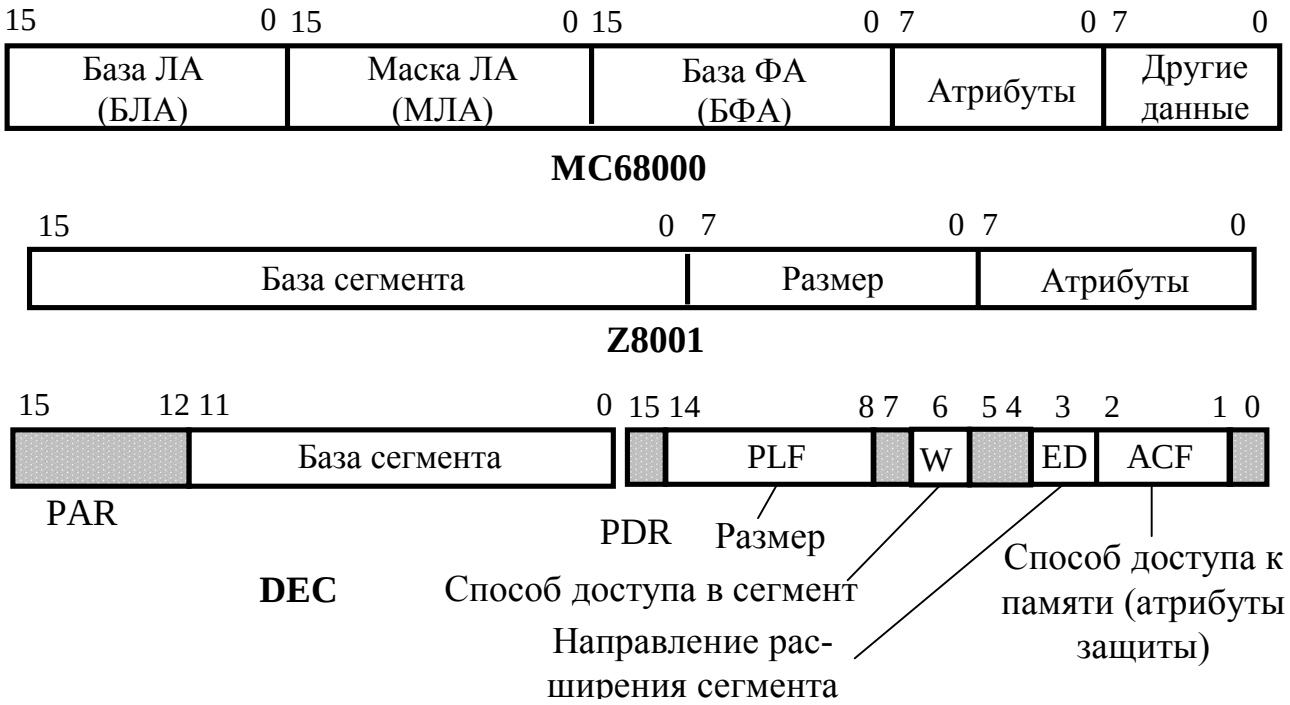


Рисунок 3.5 - Форматы дескрипторов процессоров MC68000, Z800 и DEC

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.

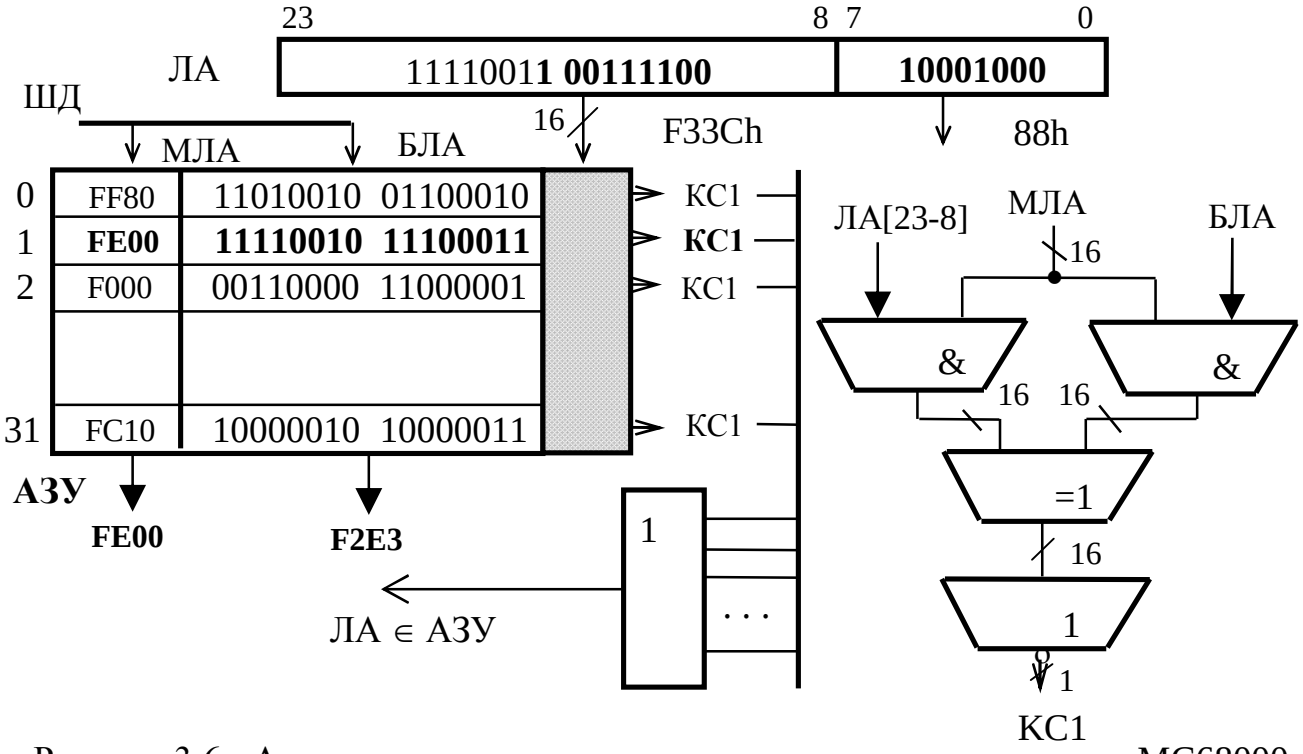


Рисунок 3.6 - Ассоциативная часть памяти дескрипторов процессора MC68000

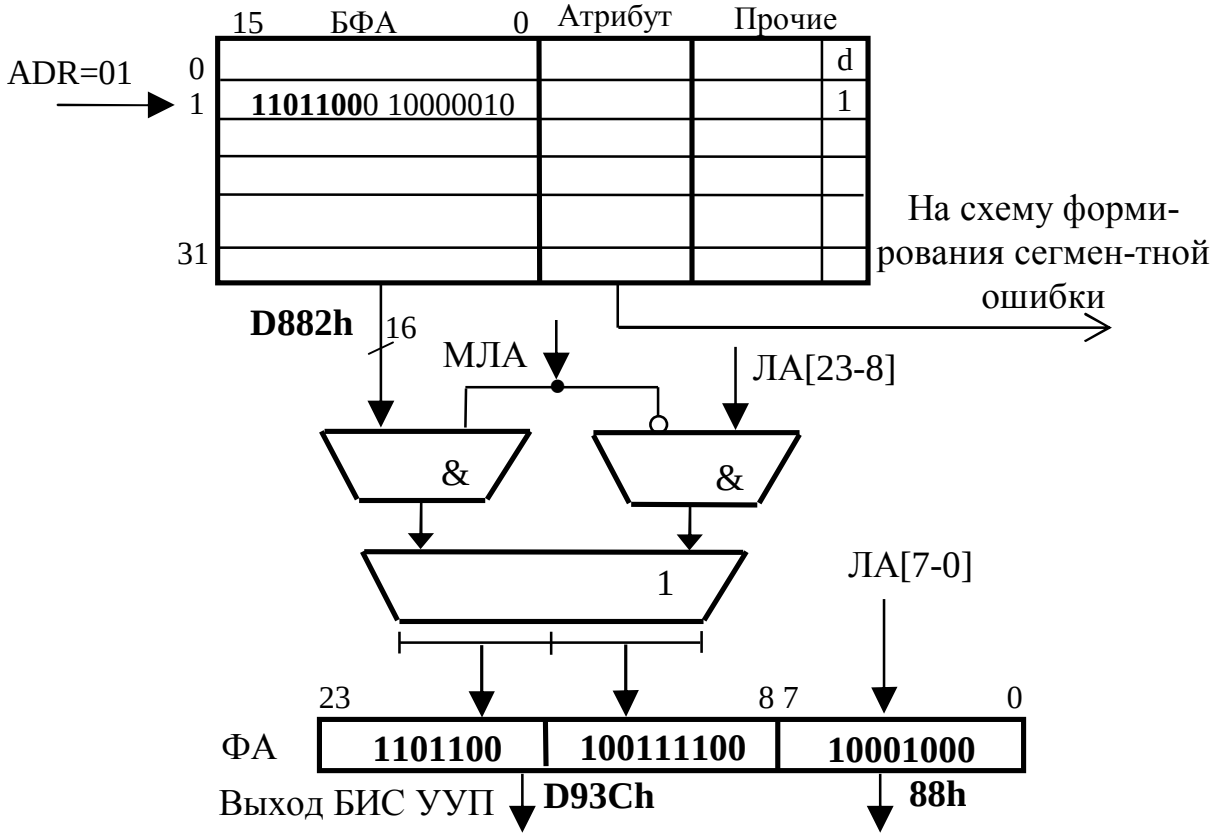


Рисунок 3.7 - Поле данных БФА дескрипторов процессора MC68000

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).

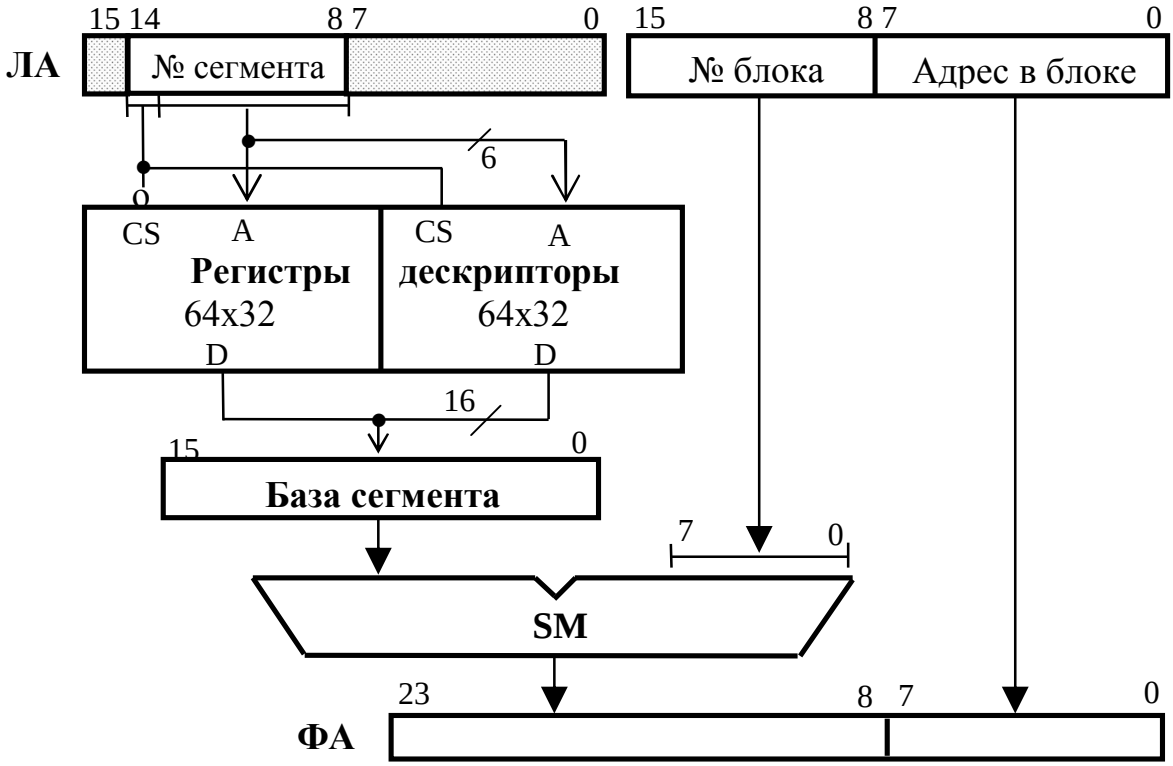


Рисунок 3.8 - Схема преобразования ЛА в ФА Z8001

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).

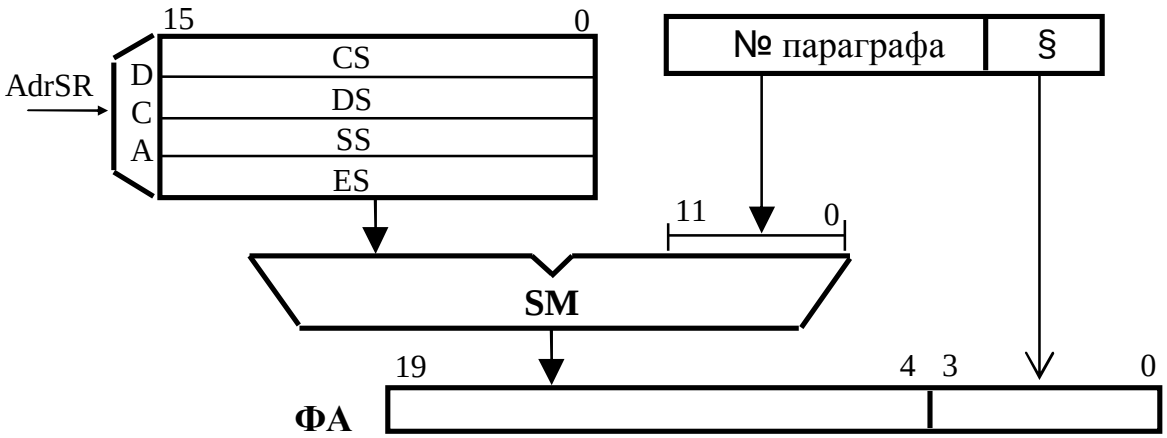


Рисунок 3.9 - Схема преобразования ЛА в ФА процессора i8086 (реальный режим)

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.

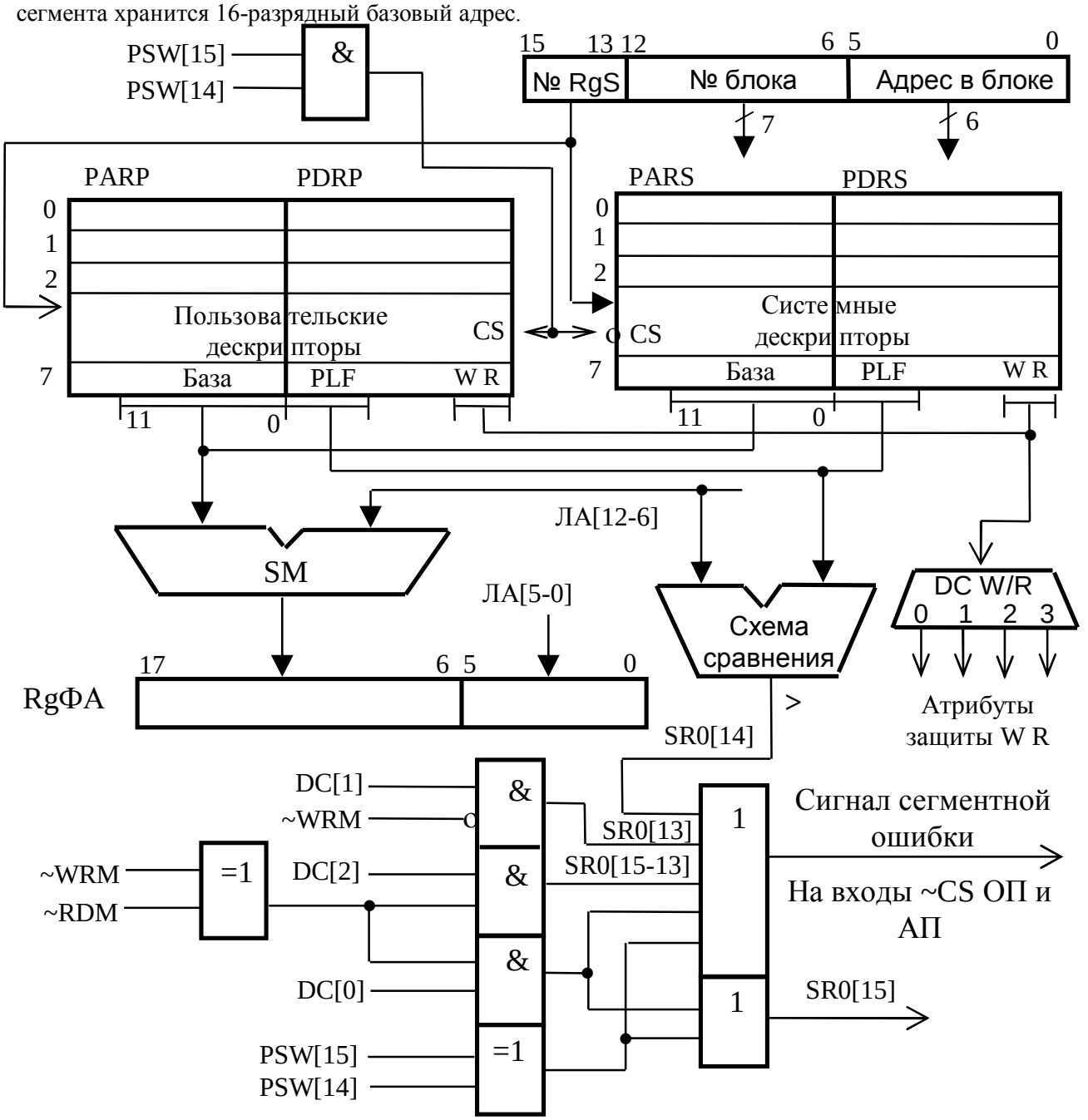


Рисунок 3.10 - Схема преобразования ЛА в ФА и формирования сегментной ошибки процессора фирмы DEC

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.

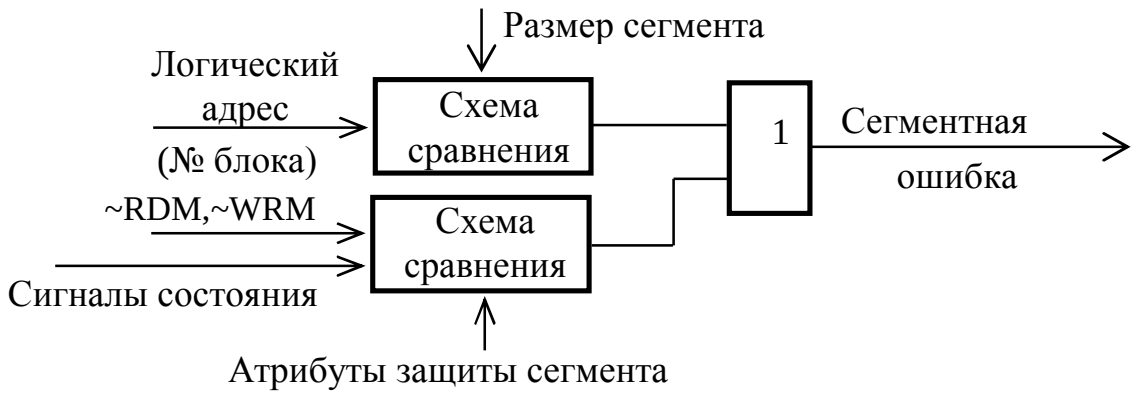


Рисунок 3.11 - Схема формирования сигнала сегментной ошибки при нарушении размера сегмента и атрибутов защиты по доступу к памяти

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).

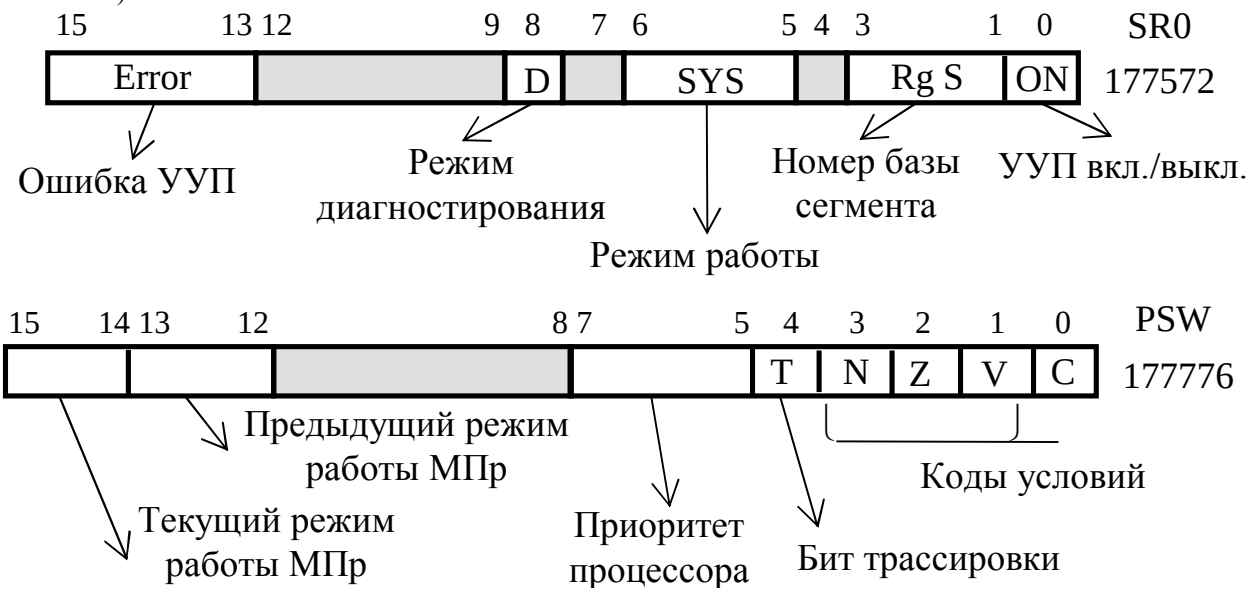


Рисунок 3.12 - Форматы регистра ошибок SR0 и слова состояния PSW процессора фирмы DEC

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).

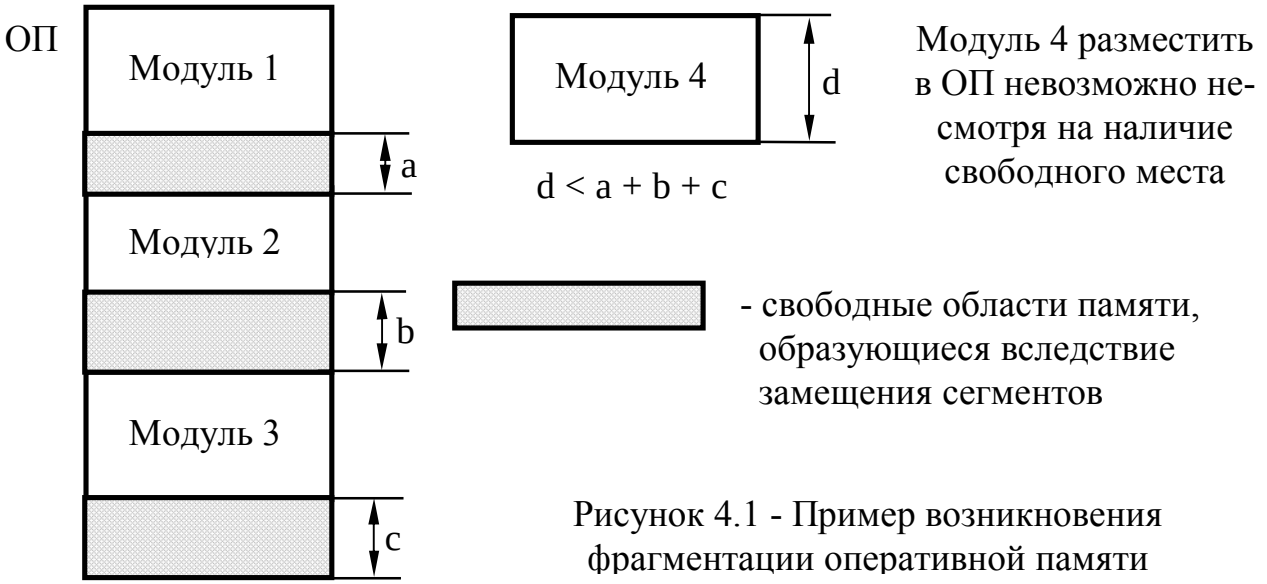


Рисунок 4.1 - Пример возникновения фрагментации оперативной памяти

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.

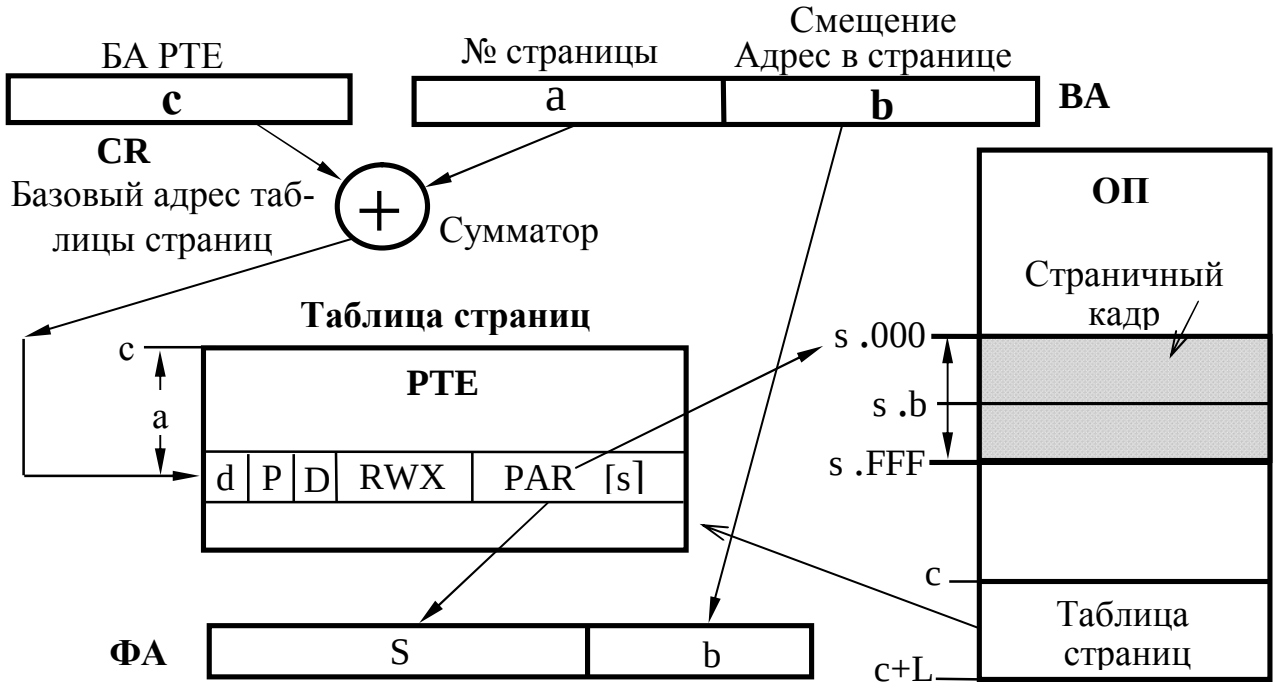


Рисунок 4.2 - Одноуровневая система преобразования виртуального страничного адреса в физический

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.

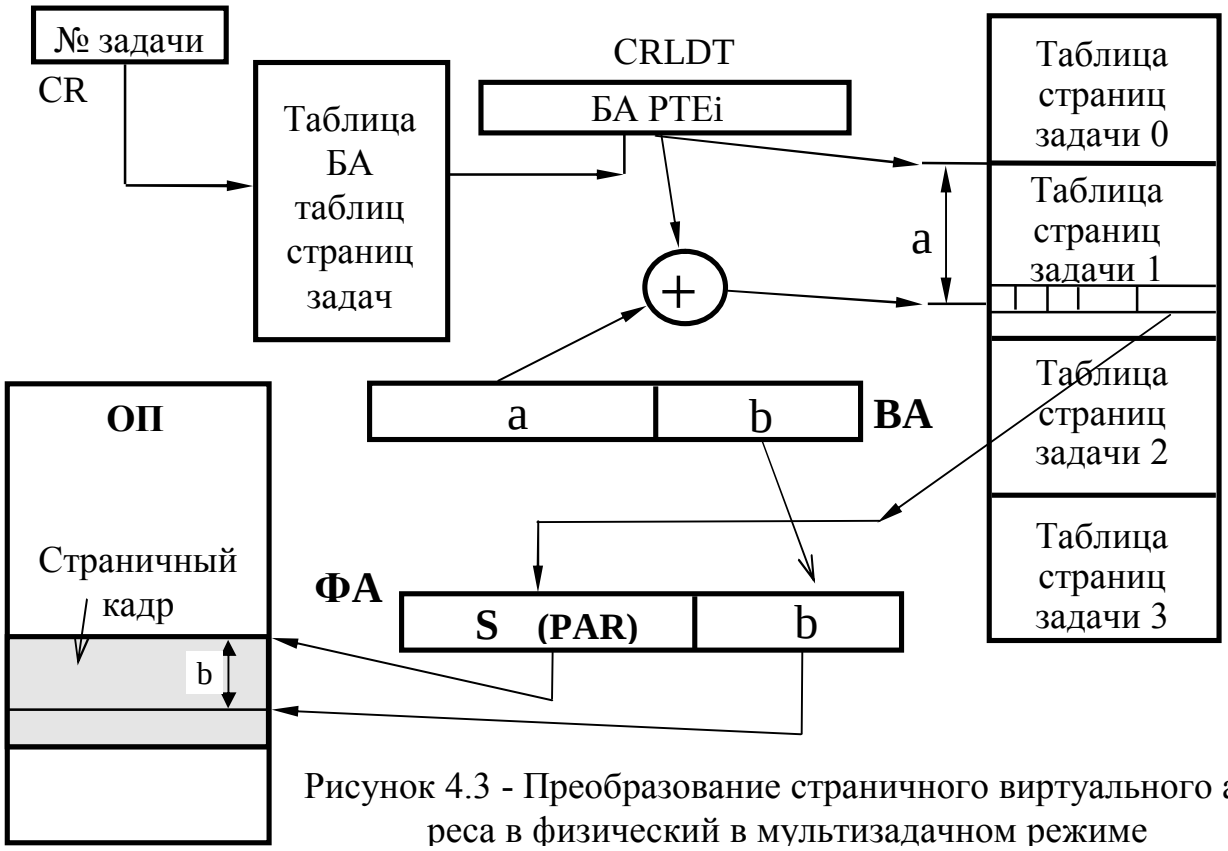


Рисунок 4.3 - Преобразование страничного виртуального адреса в физический в мультизадачном режиме

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).

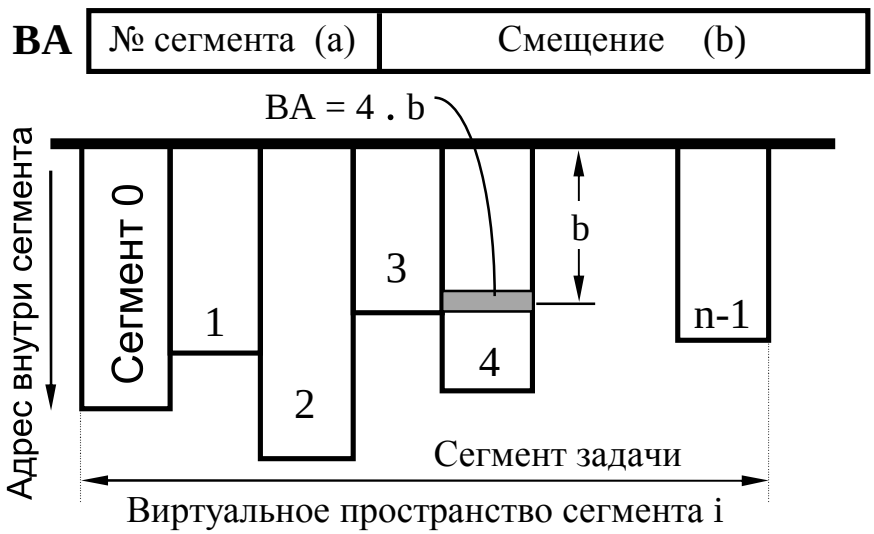


Рисунок 4.4 - Сегментное виртуальное адресное пространство задачи

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.

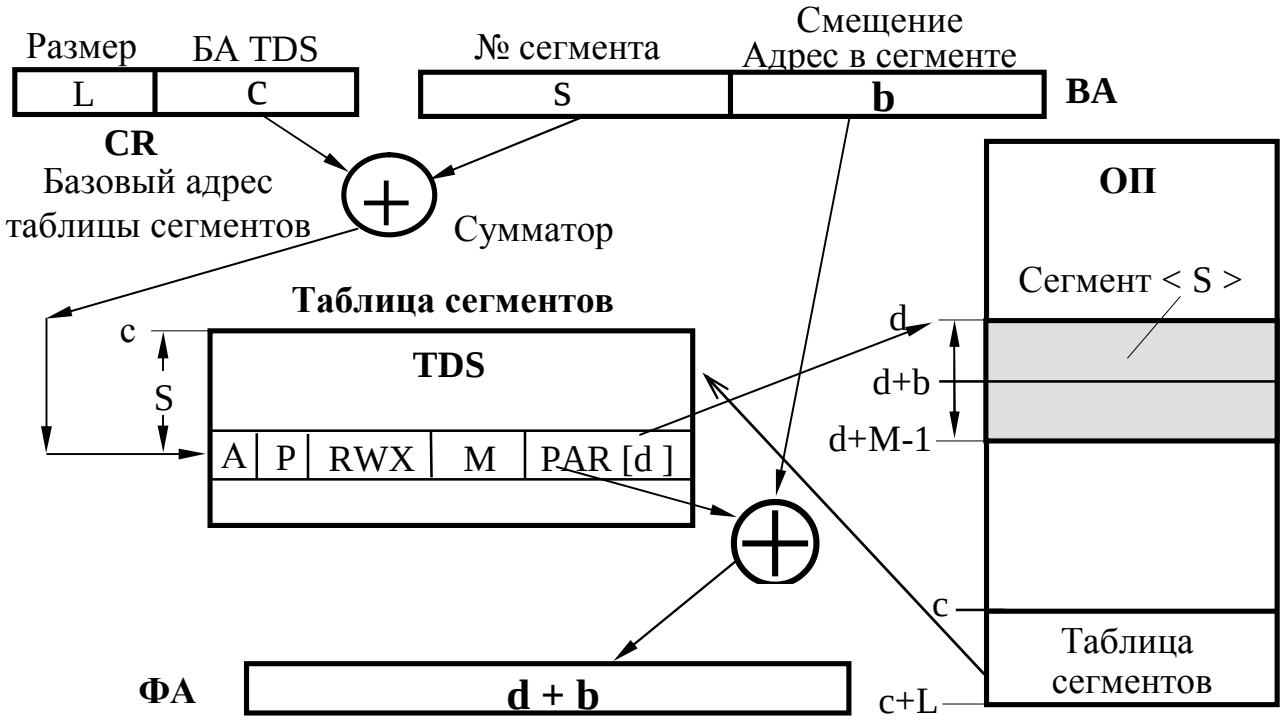


Рисунок 4.5 - Преобразования виртуального адреса в физический при сегментной организации памяти

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.

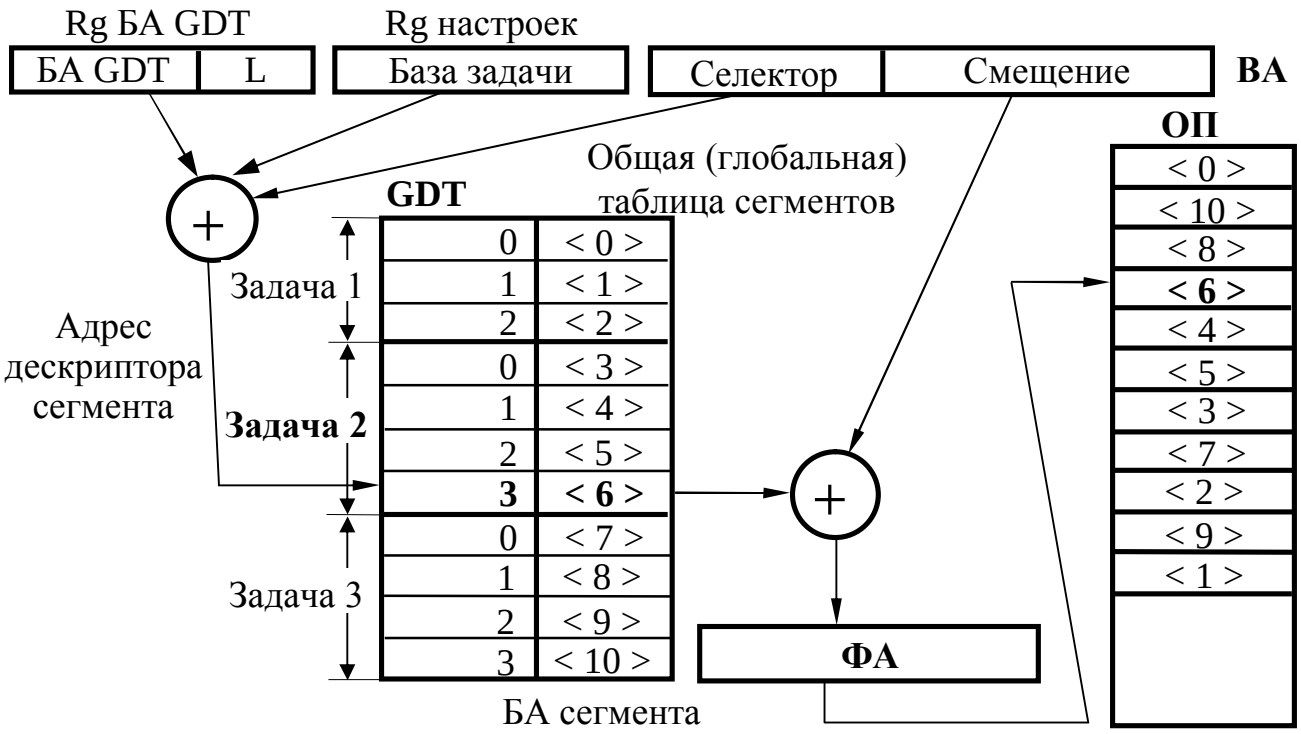


Рисунок 4.6 - Организация доступа к сегментам из разных задач через общую (глобальную) таблицу сегментов (дескрипторов)

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.

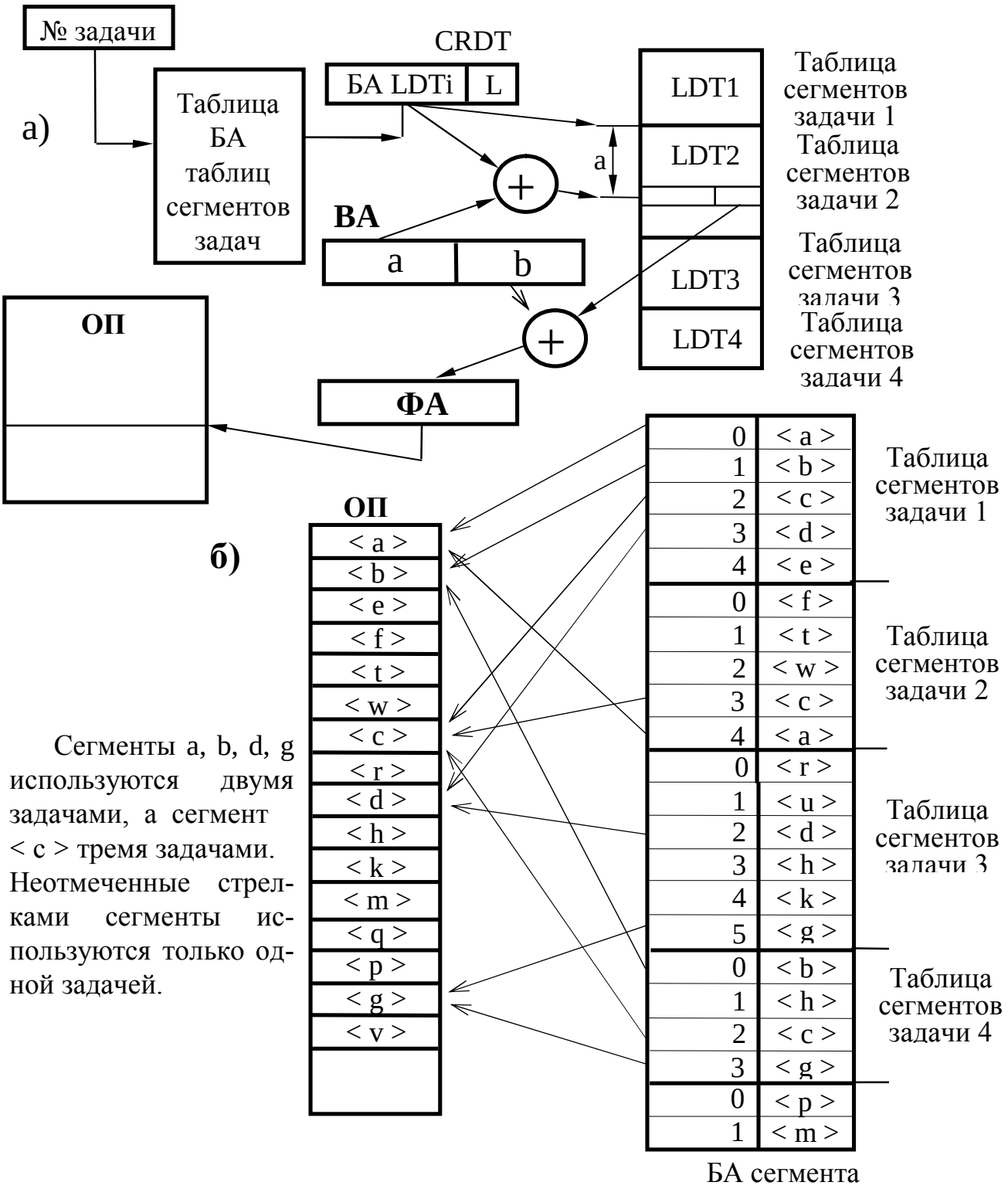


Рисунок 4.7 - Преобразование сегментного виртуального адреса в физический в мультипрограммном режиме (а); совместное использование сегмента несколькими задачами (б)

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).

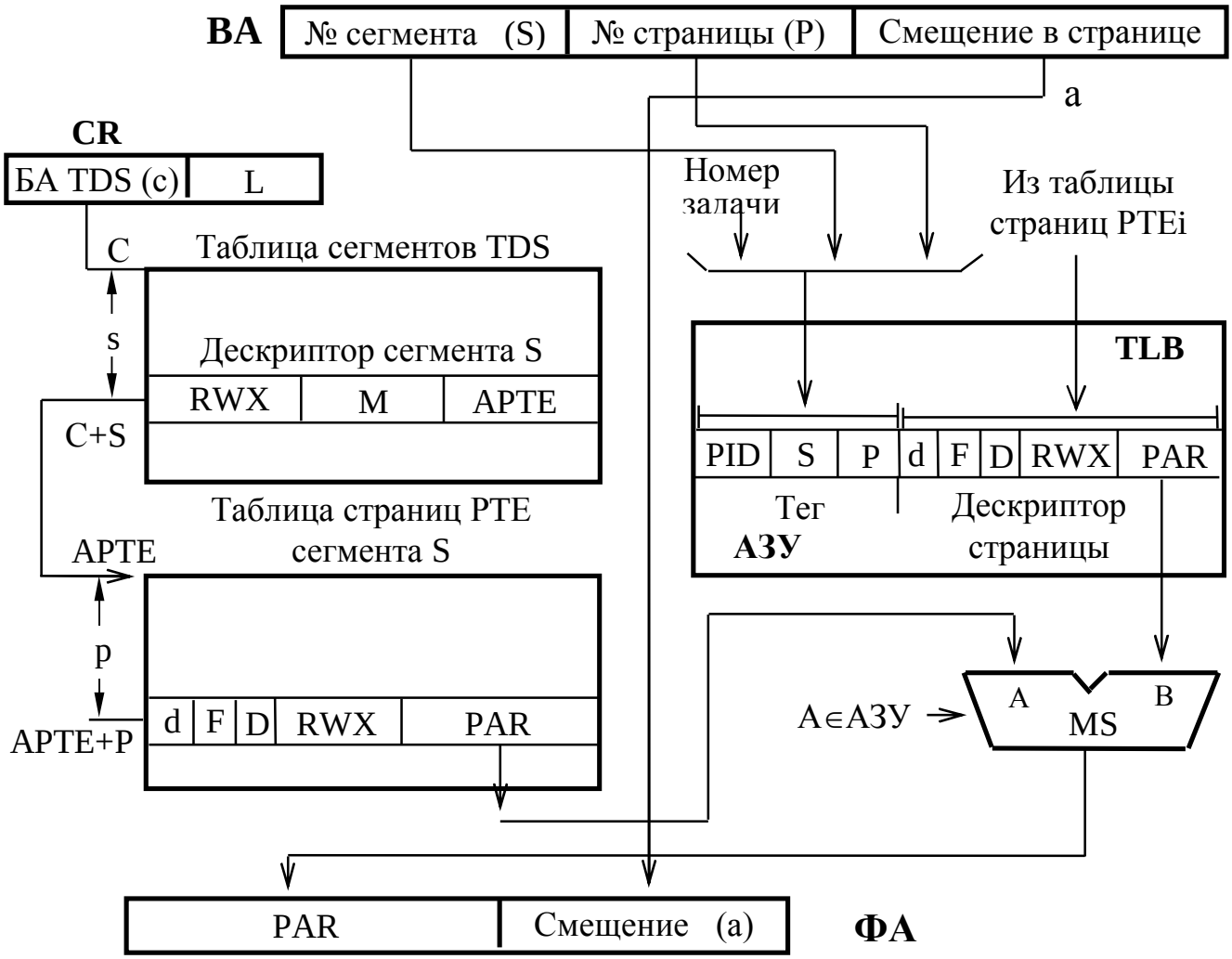


Рисунок 4.8 - Преобразование ВА в ФА при сегментно-страничной организации памяти с использованием ассоциативного буфера TLB

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

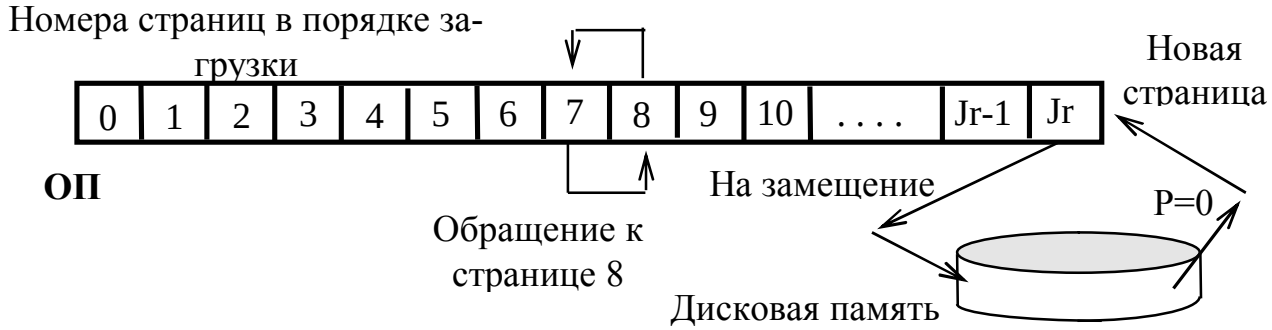
A=0 за это время (не было обращений к странице), то время неиспользования увеличивается на единицу Тн:=Тн + 1, записываемое в определенное поле дескриптора страницы. Удалению подлежит страница с максимальным временем неиспользования Тн.

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

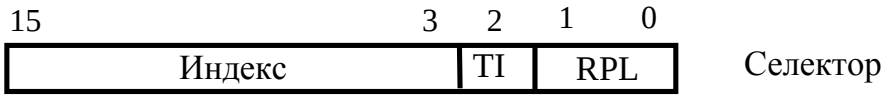
В реальном режиме максимальный объем адресуемой ОП составляет 1 Мбайт, а размеры сегментов не превышают 64 Кбайт. В защищенном режиме линейное адресное пространство увеличивается до 4 Гбайт (2^{32} байт). С точки зрения программистов, защищенный режим предоставляет большее адресное пространство и поддерживает 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.

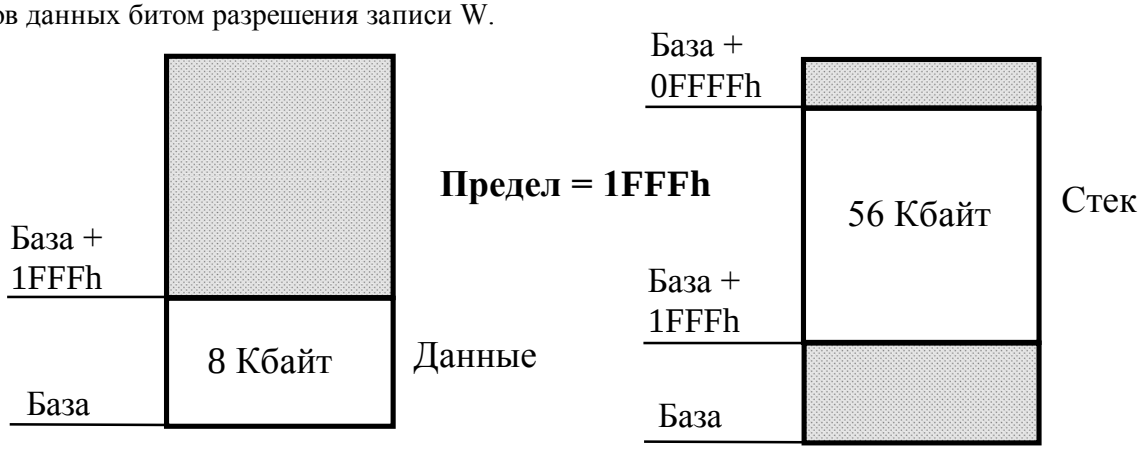


Рисунок 4.10 - Интерпретация поля предела для сегментов данных и стека

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.

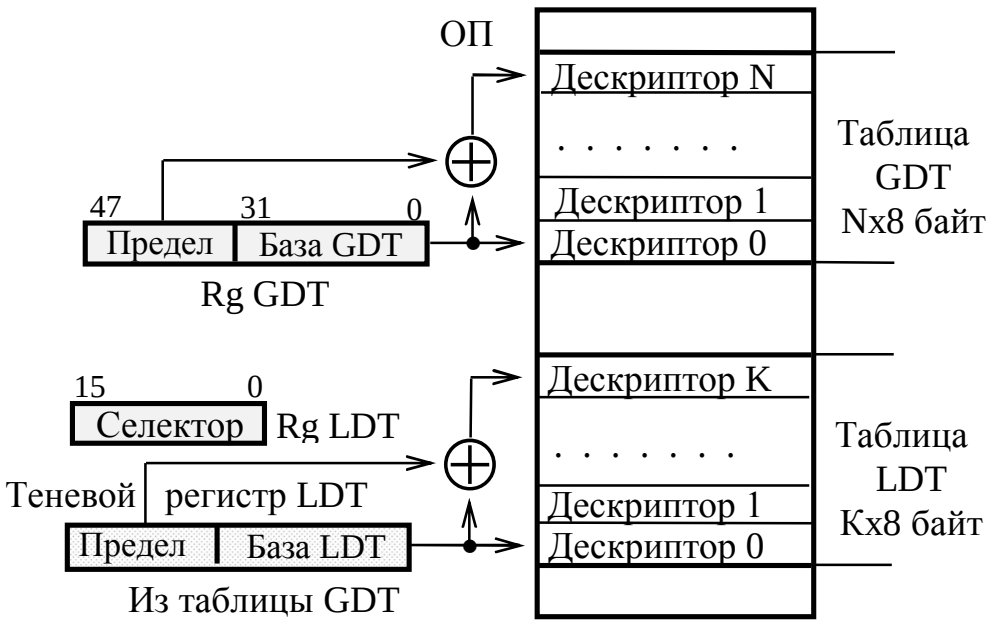


Рисунок 4.11 - Таблицы дескрипторов и системные регистры

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:

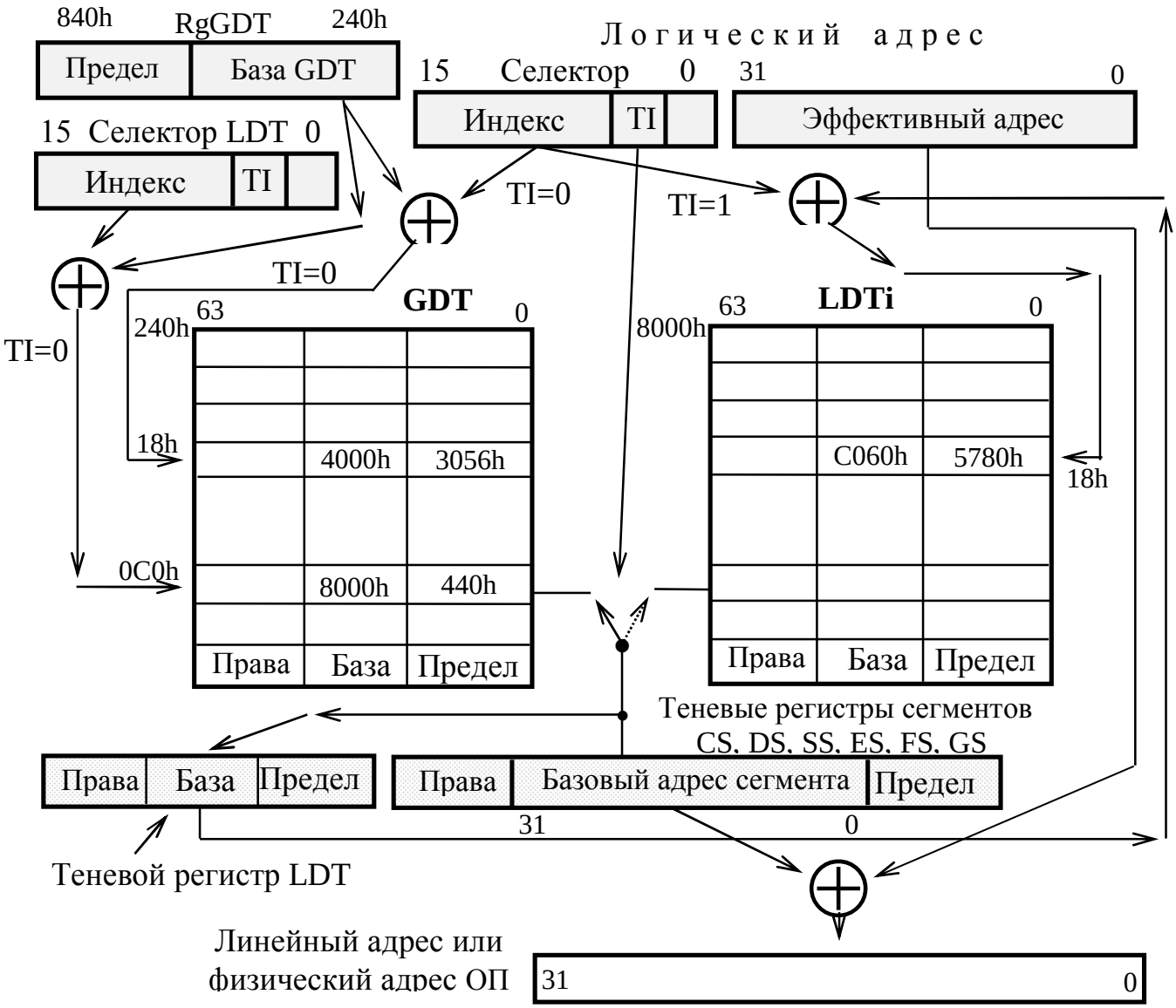


Рисунок 4.12 - Преобразование логического адреса в линейный (ФА)

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.

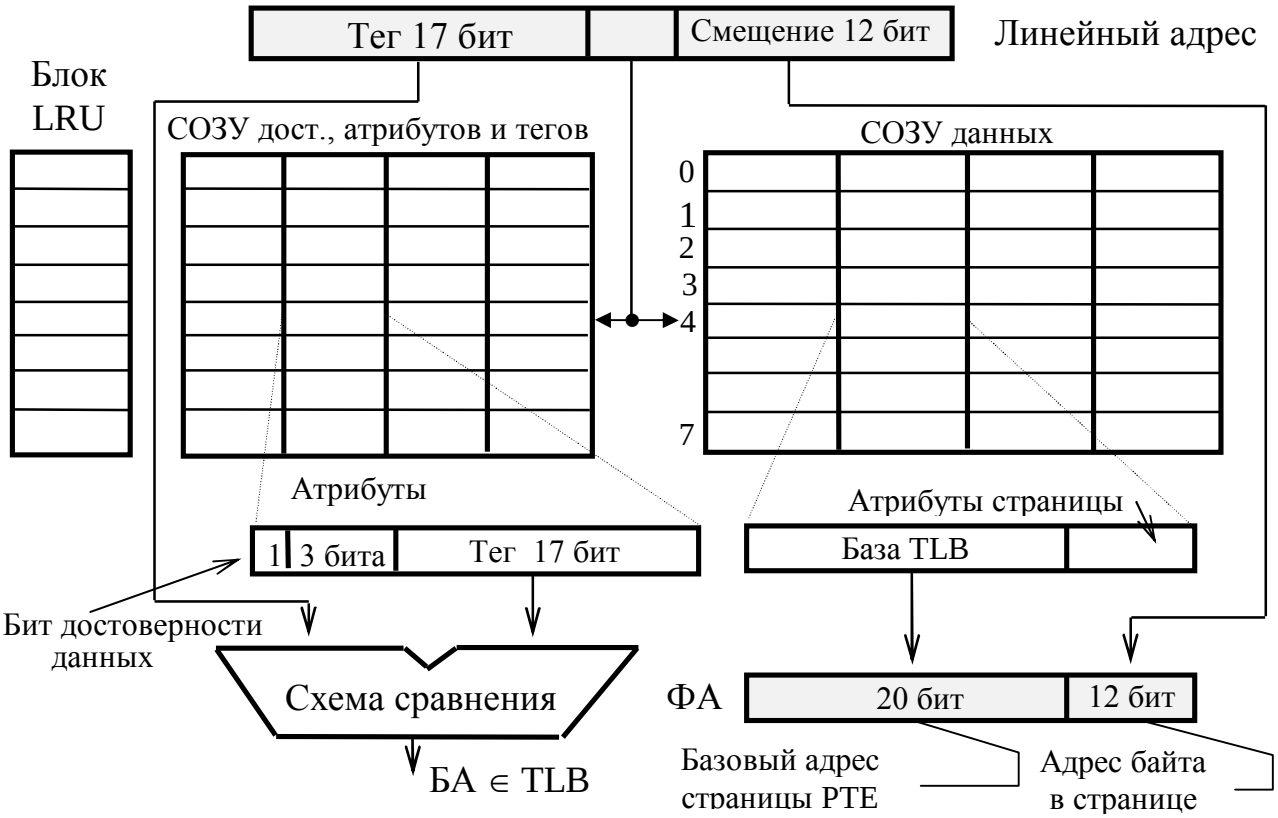


Рисунок 4.15 - Организация буфера TLB для преобразования линейного адреса в физический адрес

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.

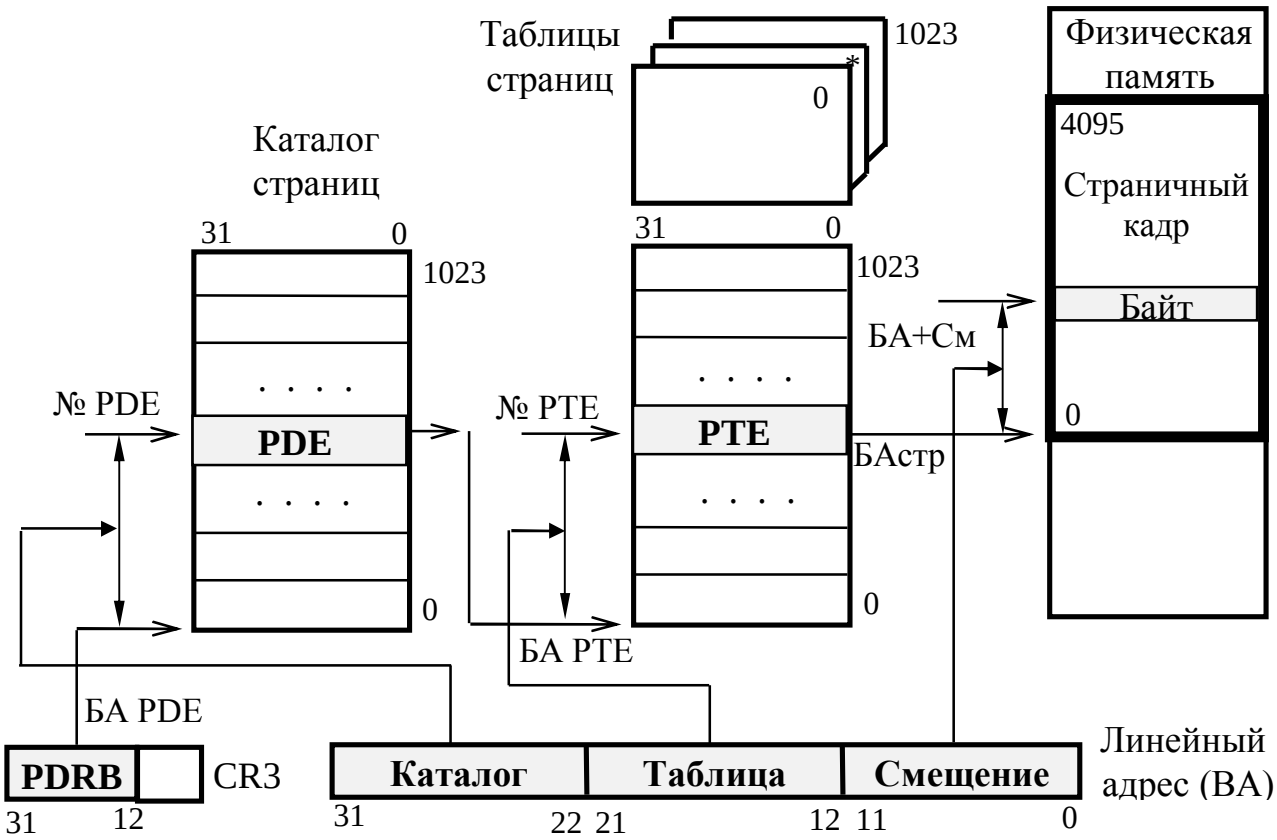


Рисунок 4.16 - Двухэтапное преобразование линейного адреса в ФА

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).

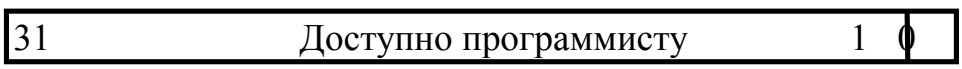
Бит D устанавливается также аппаратно **только при записи** в страницу и только в элементе PTE (в элементе каталога PDE этот бит не определен и не участвует в алгоритмах работы страничного преобразования).

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.

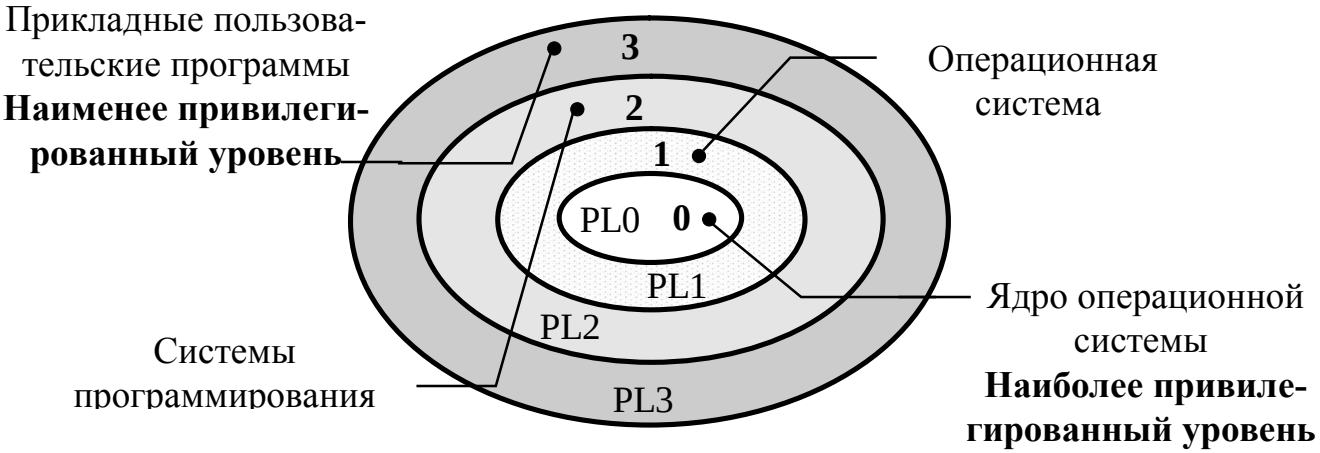


Рисунок 4.18 - Уровни привилегий или кольца защиты

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.

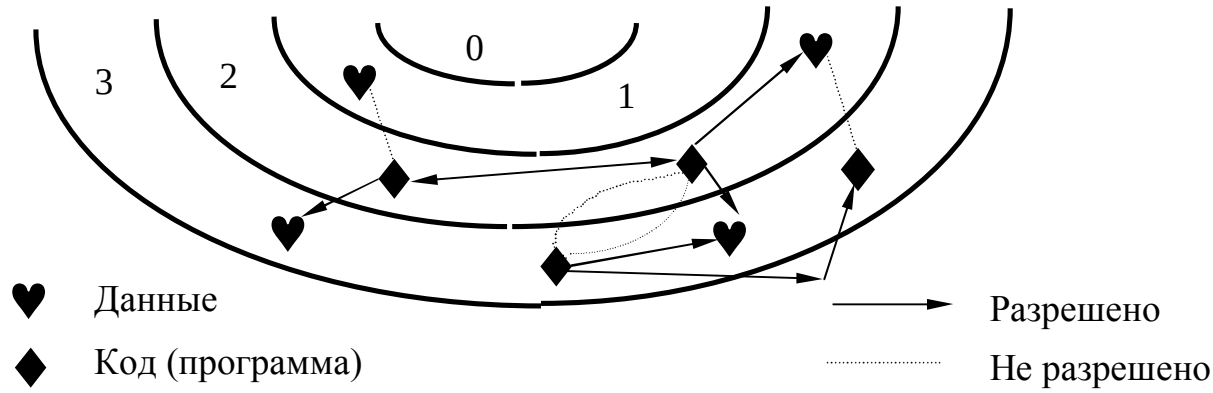


Рисунок 4.19 - Обращения в пределах колец защиты и между ними

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.

5 Построение памяти повышенной надежности

5.1 Корректирующие коды

В ряде ЭВМ повышенной надежности в ОП, ВЗУ на МЛ и МД, при передаче информации по каналам связи применяют корректирующие коды, позволяющие не только обнаруживать, но и автоматически исправлять ошибки. К таким кодам относятся:

- код Хэмминга;
- групповые коды;
- циклические коды и другие.

Код Хэмминга получают путем добавления к информационным разрядам слова дополнительных контрольных разрядов, которые формируются перед записью (передачей) информации путем подсчета четности суммы единиц для определенных групп информационных разрядов. Контрольная аппаратура при считывании (или приеме) информации путем аналогичных подсчетов образует новый контрольный код или корректирующее число, которое равно 0 при отсутствии одиночной ошибки либо указывает место ошибки (синдром ошибки), а ошибочный разряд автоматически корректируется (исправляется) путем изменения его состояния на противоположное.

Разрядность контрольного кода определяется из соотношения:

$$2^r \geq n + 1 \quad \text{или} \quad 2^r - r - 1 \geq k,$$

где k - число разрядов информационного слова;

r - число дополнительных контрольных разрядов;

n = r + k - передаваемое кодовое число.

Отсюда, если r=5, то можно контролировать от 11 до 26 информационных разрядов, а при r=6 от 27 до 57 разрядов.

Все корректирующие коды можно классифицировать по обнаруживающей способности:

- КО-ОД - коррекция одиночных ошибок и обнаружение двойных (код Хэмминга, коды с нечетными весами столбцов проверочной матрицы) в основном используются для однобитовых БИС памяти, т.к. выход из строя даже всей БИС не нарушает работу ОЗУ;

- КО-ОД-ООГ - код КО-ОД с дополнительной способностью обнаружения кратной ошибки в одной группе (под группой подразумеваются БИС ОЗУ с организацией k-бит слова (4 бита, 8 бит));

- КОГ-ОДГ - коды с коррекцией ошибок в одной группе и обнаружением в двух группах, т.к. ошибка в одной группе может вызвать ошибку в соседней группе (ошибка в байте вызывает ошибку в соседнем байте (БИС памяти));

- КД-ОТ - с коррекцией двойных и обнаружением тройных ошибок.

Благодаря корректирующему коду ОЗУ может нормально работать при наличии как случайных отказов (сбоев), так и постоянных отказов.

5.2 Формирование кода Хэмминга

Для образования контрольного кода Хэмминга (КХ) используется проверочная матрица H размерности r x n, где r - число контрольных бит, а n = k + r, где k - разрядность информационного слова, а n - длина кодового слова, записываемого в память.

Введем следующие обозначения:

V = (V1,V2,V3,...,Vn) - записанное в ОЗУ кодовое слово;

U = (U1,U2,U3,...,Un) - считанное из ОЗУ кодовое слово.

При записи в ОЗУ и при считывании выполняется процесс формирования контрольных бит с помощью проверочной матрицы H.

$$H = |\ P, Ir\ |, \qquad (5.1)$$

где P - двоичная матрица размерности r x k, a Ir - единичная матрица размерности r x r. Первые k бит кодового слова представляют собой информационные биты, r последних - контрольные.

Проверочная матрица H размерностью r x n определяет условия, которым должно удовлетворять считанное из ОЗУ кодовое слово. Эти условия задаются в виде бит, сумма по модулю 2 которых должна быть равна нулю (синдрома ошибок), т.е. должно выполняться условие:

$$S = H\ Uт = 0, \qquad (5.2)$$

где Uт - трансформированный вектор U.

Код, описываемый матрицей вида (5.1), является разделимым.

Если записанное и считанное кодовые слова отличаются хотя бы в одном бите, то разница между U и V определяет вектор ошибки E=(E1,E2,....,En), т.е. U = V + E, т.е. если Ei ≠ 0, то i-й бит считанного слова содержит ошибку.

Проверка считанного слова заключается в вычислении синдрома ошибки S:

$$S = H\ Uт = H\ (\ Vт + Eт\ ). \qquad (5.3)$$

Если S - нулевой вектор, то считается, что ошибок нет. В противном случае синдром определяет вектор ошибки. Синдром является суммой по модулю 2 тех столбцов h матрицы H, которым соответствуют ошибки. Если столбец hi нулевой, то ошибка в этой позиции кодового слова не окажет влияния на синдром и не позволит обнаружить ошибку. Если два столбца матрицы H совпадают, то одиночные ошибки в каждой из этих позиций кодового слова дают один и тот же синдром.

Отсюда, матрица H кода КО-ОД должна удовлетворять следующим условиям:

1) векторы-столбцы матрицы H не должны быть нулевыми и должны отличаться друг от друга;

2) сумма двух векторов-столбцов матрицы H не должна равняться нулю или любому третьему вектору-столбцу матрицы H.

Рассмотрим процесс формирования контрольного кода и синдрома ошибок на примере 5-разрядного информационного слова. Для контрольного кода потребуется 4 дополнительных разряда ($2^r - r - 1 \geq n$) и один дополнительный бит для определения двойной ошибки:

d4 d3 d2 d1 d0 - информационное слово;

c4 d3 c2 c1 c0 - номер бита кода Хэмминга;

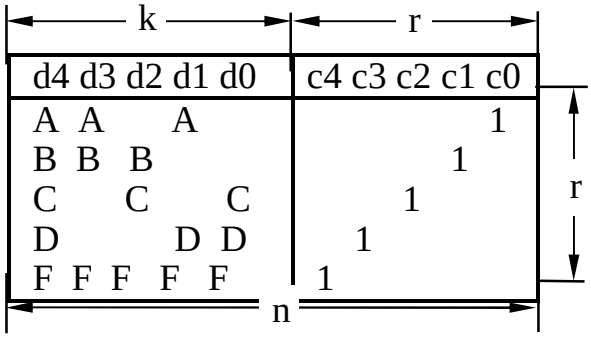
CT C3 C2 C1 C0 - контрольный код Хэмминга, где бит CT необходим для обнаружения двойной ошибки.

В соответствии со сформулированными требованиями разработаем проверочную матрицу для данного кода (таблица 5.1).

В таблице 5.2 приведены примеры внесения одиночных и двойных ошибок в разряды считанного информационного слова или кода Хэмминга CT-C0. Синдром ошибки ST-S0 может быть получен сложением по модулю 2 считанного из ОЗУ контрольного кода и вновь полученного контрольного кода от считанного информационного слова:

$$ST\text{-}S0 = (CT\text{-}C0)\text{считанный} \oplus (CT\text{-}C0)\text{полученный}.$$

Таблица 5.1 - Проверочная матрица для формирования кода КО-ОД



Определение кода Хэмминга

$$C0 = d4 \oplus d3 \oplus d1 \oplus c0$$
$$C1 = d4 \oplus d3 \oplus d2 \oplus c1$$
$$C2 = d4 \oplus d2 \oplus d0 \oplus c2 \qquad (5.4)$$
$$C3 = d4 \oplus d1 \oplus d0 \oplus c3$$
$$CT = d4 \oplus d3 \oplus d2 \oplus d1 \oplus d0 \oplus c4$$

Таблица 5.2

Код Хэмминга CT C3 C2 C1 C0 **1 0 0 1 1**	Исходное число d4 d3 d2 d1 d0 **1 0 1 1 1**	Синдром ошибки ST S3 S2 S1 S0	Местоположение и тип ошибки (вектор ошибки)
0 1 0 1 0	1 0 1 **0*** 1	1 1 0 0 1	d1 00010 . 00000
0 1 1 1 1	1 0 1 1 **0***	1 1 1 0 0	d0 00001 . 00000
0 0 0 0 0	1 **1*** 1 1 1	1 0 0 1 1	d3 01000 . 00000
0 1 1 0 0	**0*** 0 1 1 1	1 1 1 1 1	d4 10000 . 00000
0* 0 0 1 1	1 0 1 1 1	1 0 0 0 0	CT 00000 . 10000
1 0 0 **0*** 1	1 0 1 1 1	0 0 0 1 0	C1 00000 . 00010
1 **1*** 0 1 1	1 0 1 1 1	0 1 0 0 0	C3 00000 . 01000
1 0 1 1 0	1 0 1 **0*** **0***	0 0 1 0 1	Двойная ошибка
1 1 1 0 0	1 **1*** 1 1 **0***	0 1 1 1 1	Двойная ошибка
1 0 1 1 0	1 **1*** **0*** 1 1	0 0 1 0 1	Двойная ошибка
0 0 0 0 0	1 0 **0*** **0*** **0***	1 0 0 1 1	d3 - неверное обнаружение (3-кратная ошибка)

Существует несколько алгоритмов для нахождения вектора ошибок E по синдрому ошибки с определением двойной ошибки или с выделением всех ошибок, кроме одиночных, в группу неисправимых ошибок. В таблице 5.3 приведен алгоритм определения типа ошибки по синдрому ошибки.

Таблица 5.3

ST	Биты S3-S0	Тип ошибки
0	Все нули	Нет ошибки
1	Все нули	Ошибка бита CT
1	Один бит или более равен 1	Исправимая ошибка, если ошибочный бит определяется по таблице 5.1
0	Один бит равен 1	Ошибка бита C3-C0
0	Более одного бита равны 1	Двойная ошибка

Другой алгоритм для обнаружения и исправления одиночной ошибки в информационном слове и в битах контрольного кода Хэмминга без выделения двойной ошибки можно представить в следующем виде:

1) Проверяется условие ST-S0 = S = 0. Если S=0, то считанное слово безошибочное.

2) Если $S \neq 0$, необходимо найти соответствие между синдромом ошибки и столбцом матрицы H. Поиск выполняется с помощью дешифратора или ПЗУ.

3) Если S совпадает с i-м столбцом матрицы H, то это значит, что i-й бит кодового слова содержит ошибку и должен быть исправлен путем его инвертирования.

4) Если S не совпадает ни с одним из столбцов матрицы H, то ошибка неисправима (двойная или большей кратности).

Например, если считано слово:

Слово	Код Хэмминга	Синдром ошибки
1 0 1 **0***1	0 1 0 1 0	1 1 0 0 1

т.е. E=00010 . 00000 и синдром ошибки ST = 1, S=1001 совпадает с первым столбцом матрицы H, что свидетельствует об ошибке в этой позиции кодового слова (бит d1).

5.3 Аппаратура кода Хэмминга

Представим фрагменты технической реализации блоков системы контроля ОП. На рисунке 5.1 приведена функциональная схема генератора кода КО-ОД и схема свертки определения бит нечетности, реализованные на элементах сложения по модулю 2 (GKH) по формулам (5.4).

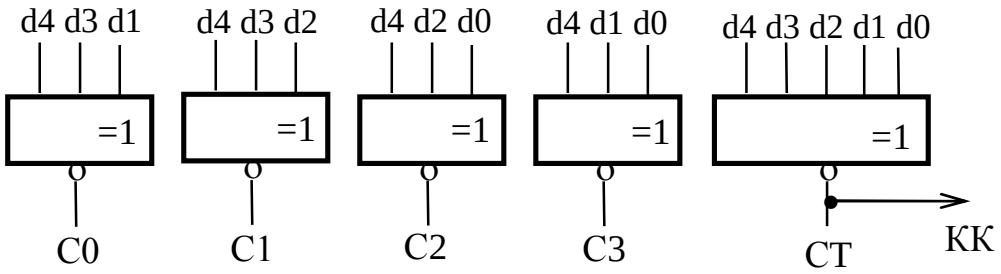


Рисунок 5.1 - Схема генератора КХ и свертки по нечетности

На рисунке 5.2 схема генерации синдрома ошибки реализована на элементах "Исключающее ИЛИ" с формированием бита ST. Элемент ИЛИ формирует сигнал равенства синдрома ошибки нулю, а совместная дешифрация сигналов ST и SK дает четыре комбинации осведомительных сигналов классификации типа ошибки (вариант 1) или на элементе "И" формируется сигнал неисправимой ошибки (вариант 2).

Схема определения ошибочного бита (рисунок 5.3) может быть реализована на обычном дешифраторе на четыре входа, на выходах которого формируется унитарный код с одной "1" только при обнаружении одиночной ошибки. Этот код в дальнейшем выполняет функции вектора ошибки E и используется в схеме коррекции ошибки.

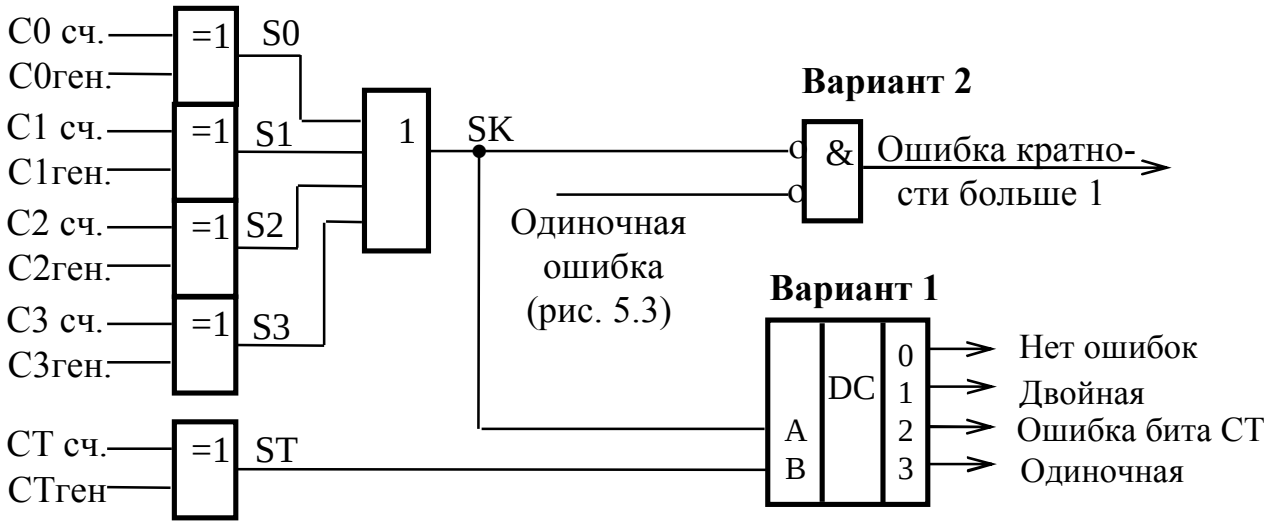


Рисунок 5.2 - Схема определения синдрома ошибки ST-S0, сравнения и классификации типа ошибки

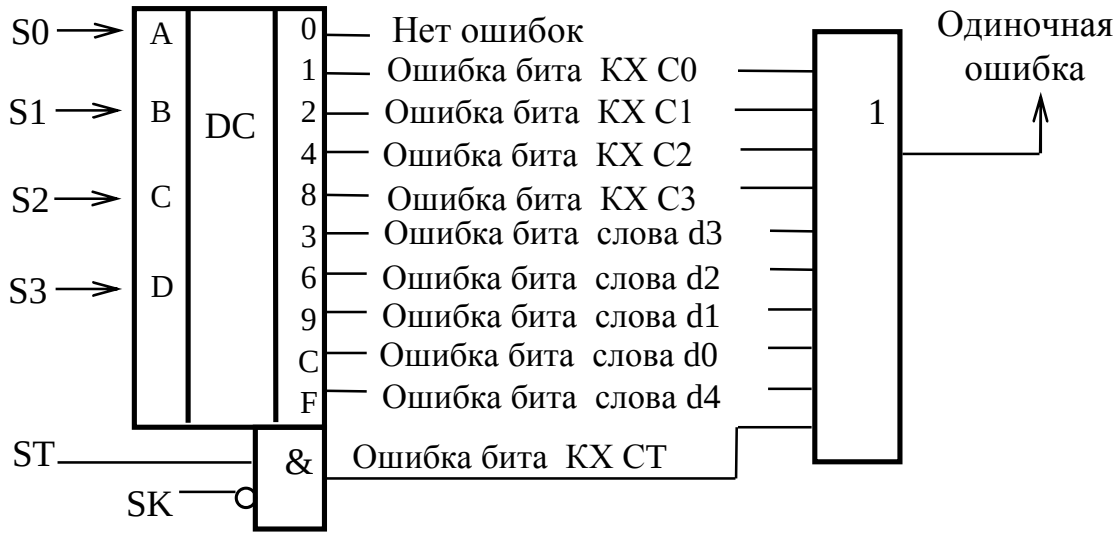


Рисунок 5.3 - Схема определения ошибочного бита (вектора ошибок E) на основе DC

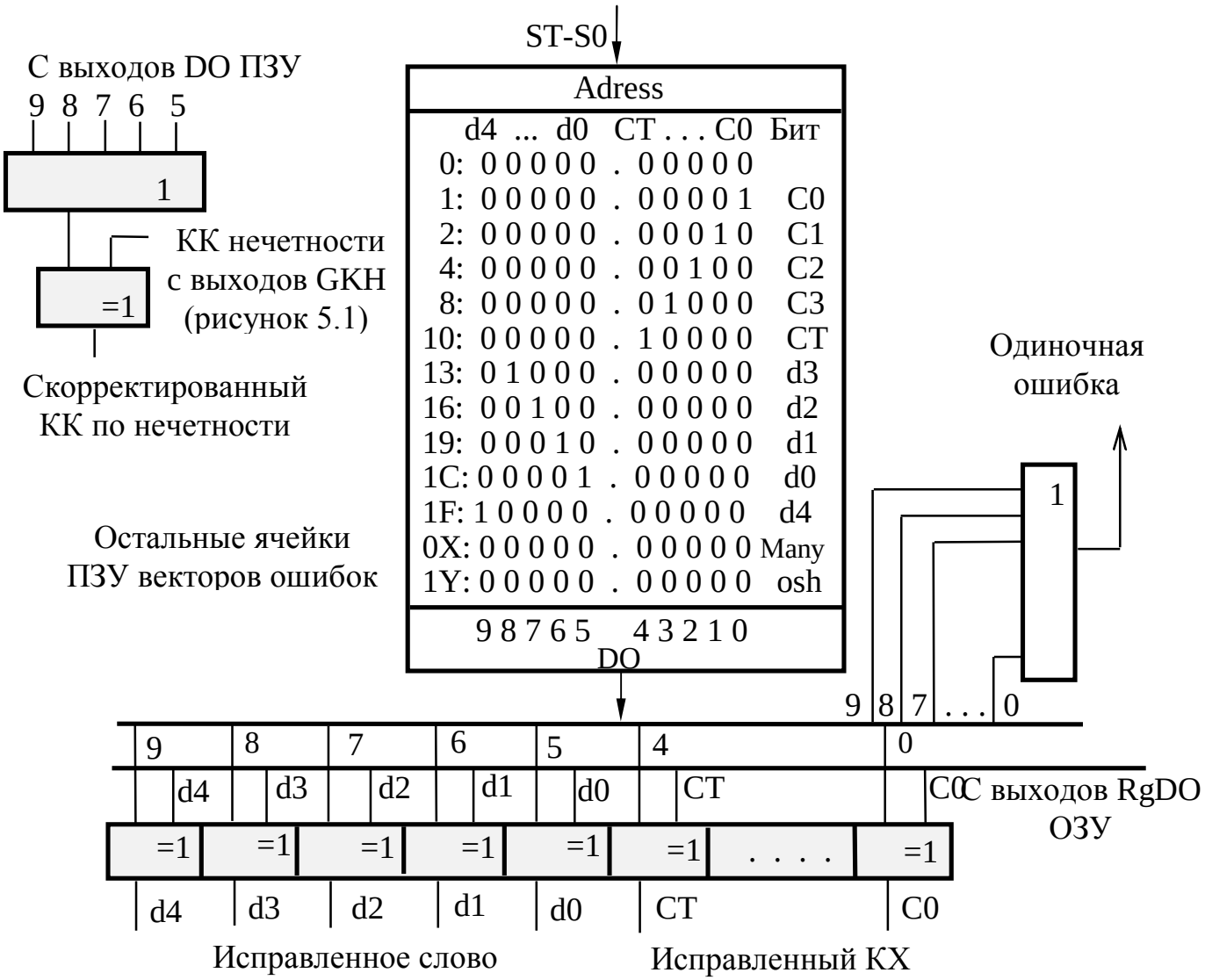


Рисунок 5.4 - Схемы формирования вектора ошибок на основе ПЗУ и коррекции одиночной ошибки в слове, КХ и контрольном коде нечетности

При исправлении одиночной ошибки в информационном слове требуется выполнить коррекцию и контрольного бита нечетности, т.к. этот контрольный код был сформирован параллельно с контрольным кодом считанного из ОЗУ слова с одиночной ошибкой (рисунок 5.4). Коррекция контрольного бита нечетности выполняется путем его инверсии только для одиночных ошибок (при формировании вектора ошибки) параллельно с исправлением ошибки в информационных разрядах слова, считанного из ОП, путем инверсии бита, указанного вектором ошибки Е.

Схема определения ошибочного бита может быть реализована и на ПЗУ (но задержка может получиться больше) (рисунок 5.4). Такая реализация позволяет гибко изменять типы матриц Н путем перепрограммирования (прошивки) БИС ПЗУ, но может увеличить время коррекции слова.

5.4 Аппаратурная реализация системы контроля по коду КО-ОД

Код Хэмминга обеспечивает исправление всех одиночных ошибок и обнаружение двойных. Рассмотрим пример организации системы контроля ОП для 64-разрядного информационного слова, поступающего из процессора вместе с битами контрольного кода по нечетности для каждого байта слова (8 бит контрольного кода КК). Это позволяет в дальнейшем получать контрольные биты нечетности байт непосредственно из схемы генерации кода Хэмминга, что минимизирует аппаратурные затраты и позволяет легко переходить от кода Хэмминга СТ-С0 к контрольному коду по нечетности К7-К0.

Рассмотрим алгоритм работы системы контроля КО-ОД. **При записи:**

♦ информация из процессора поступает в RgDI в виде 64-разрядного слова и 8 бит нечетности для каждого байта слова (рисунок 5.5), а адрес для записи - в RgA;

♦ с выхода RgDI через MS1 слово поступает на генератор кода Хэмминга и бит нечетности GKH для формирования кода Хэмминга СТ-С0 и контрольных бит нечетности байт К7-К0. На схеме сравнения (Сх.ср.) определяется правильность приема информации из процессора в RgDI путем сравнения бит контрольного кода нечетности, принятых из процессора, и бит нечетности, сформированных схемой GKH;

♦ если сигнал ошибки передачи по нечетности не выработан, то сформированный код Хэмминга СТ-С0 с выходов GKH вместе с информационными разрядами слова записывается в ОП;

♦ если выработан сигнал ошибки передачи по нечетности, то выполняется попытка повторной передачи информации из процессора с контрольными битами КК для классификации типа отказа: постоянный или случайный (сбой) (иногда до 8 раз);

♦ если попытка восстановления информации классифицируется как постоянный отказ, то устанавливается триггер ошибки и вырабатывается сигнал прерывания от схем контроля, иначе запись слова и КХ СТ-С0 в ОП.

Чтение из ОП.

1. Из ОП считываются 64 бита слова и 8 бит КХ СТ-С0. Считанное 64-разрядное слово через MS1 поступает на схемы генератора кода Хэмминга и формирования бит нечетности байт.

2. Полученные в GKH биты СТ-С0н сравниваются со считанными из ОП битами СТ-С0сч. и при их несовпадении схема определения синдрома ошибки (Сх.ОСО) формирует код синдрома ошибок ST-S0, определяющего номер ошибочного бита в слове, а также тип ошибки на схеме классификации ошибок (одиночная или двойная).

3. Адрес слова ОП, его синдром ошибки и тип ошибки запоминаются в специальном регистре ошибок для последующей записи его значения в журнал ошибок ОС.

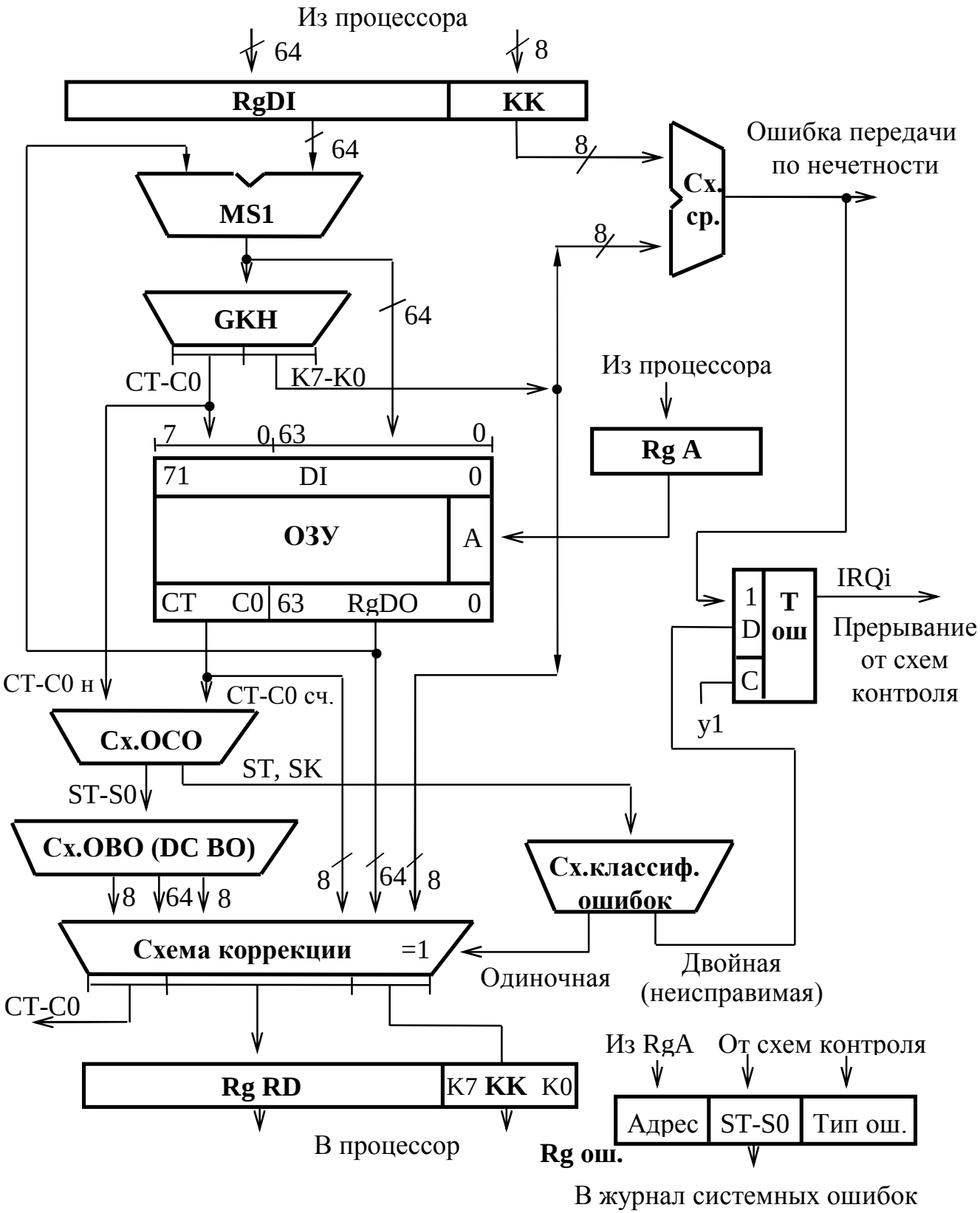


Рисунок 5.5 - Структурная схема контроля ОЗУ по коду Хэмминга

4. При одиночной ошибке схема определения вектора ошибки (Сх.ОВО) формирует код вектора ошибки Е, а в схеме коррекции ошибок выполняется ее исправление путем инверсии ошибочного бита,

а также выполняется изменение бита нечетности того байта, в котором выполнена коррекция. Скорректированная информация поступает в RgRD, а из него в процессор.

5. Неисправимая (двойная) ошибка ОП вызывает установку бита ошибки в Тош и прерывание от схем контроля. Все случаи ошибок, обнаруженные при обращении к памяти, фиксируются в журнале ошибок ОС.

5.5 Самоконтролируемая система контроля по коду Хэмминга

При работе системы контроля ошибки могут возникать как в основной аппаратуре, так и в самих схемах контроля. Для обнаружения ошибок в аппаратуре контроля (рисунок 5.6) в схему введем дополнительный (дублирующий) генератор КХ и формирования бит нечетности GKH2.

Тогда в предыдущий алгоритм необходимо ввести некоторые изменения и дополнения.

Запись в ОП выполняется без изменений алгоритма.

При чтении в пункте 1 считанное из ОП слово сначала поступает в RgDO, и параллельно выполняется алгоритм проверки работоспособности схем контроля, т.е. данные из RgDO через MS1 поступают на входы GKH1 для получения нового кода Хэмминга и бит нечетности считанного слова и параллельно для формирования нового кода Хэмминга и бит нечетности на входы GKH2. Полученные контрольные биты нечетности и кода Хэмминга СТ-С0 сравниваются на схемах сравнения (Сх.ср.2 и Сх.ср.3) и принимается решение о работоспособности аппаратуры контроля, т.е. если хотя бы одна схема сравнения вырабатывает сигнал неравенства кодов, то формируется сигнал прерывания от схем контроля и фиксируется тип ошибки в регистре ошибок.

5.6 Методы исправления двойных ошибок

5.6.1 Последовательная коррекция

Метод основан на применении кода КО-ОД и позволяет исправлять большинство двойных неисправимых ошибок. Алгоритм работы схемы имеет много общего с первым методом исправления одиночных ошибок, но основан на запоминании синдромов жестких исправимых одиночных ошибок в специальной памяти, построенной на основе ассоциативного запоминающего устройства (АЗУ).

Запись в ОП выполняется аналогично предыдущим методам.

1. При считывании из ОП одновременно осуществляется обращение к АЗУ для формирования признака $A \in Teg$.

2. Если адрес не принадлежит тегу, то при одиночной ошибке ее синдром ST-S0 и адрес (Teg) записываются в АЗУ, а одиночная ошибка и КК нечетности исправляются средствами кода КО-ОД.

3. Если при считывании выработан сигнал двойной ошибки и $A \notin Teg$, то вырабатывается сигнал прерывания от схем контроля о неисправимой ошибке (двойная ошибка обнаружена в момент возникновения).

4. Если $A \in Teg$ и выработан сигнал одиночной ошибки, то она исправляется схемой коррекции кода КО-ОД, а ее синдром, если он не равен нулю, вновь записывается в АЗУ на место старого синдрома, так как при первой записи синдрома ошибка могла быть вызвана случайным самоустранимым сбоем и в дальнейшем произойдет неправильное исправление ошибки. Если новый синдром равен нулю, то в ячейке АЗУ соответствующий бит достоверности данных сбрасывается. Также выполняется коррекция КК нечетности одного из байт слова.

5. Если A∈Teg и выработан сигнал двойной ошибки, то начинает работу схема последовательной коррекции: первая ошибка корректируется по информации из АЗУ памяти синдромов ST'-S0' и исправленное слово вновь записывается в RgDO для формирования нового кода СТ-С0н и синдрома ошибки ST-S0, а оставшаяся одиночная ошибка и КК нечетности исправляются схемой коррекции кода КО-ОД.

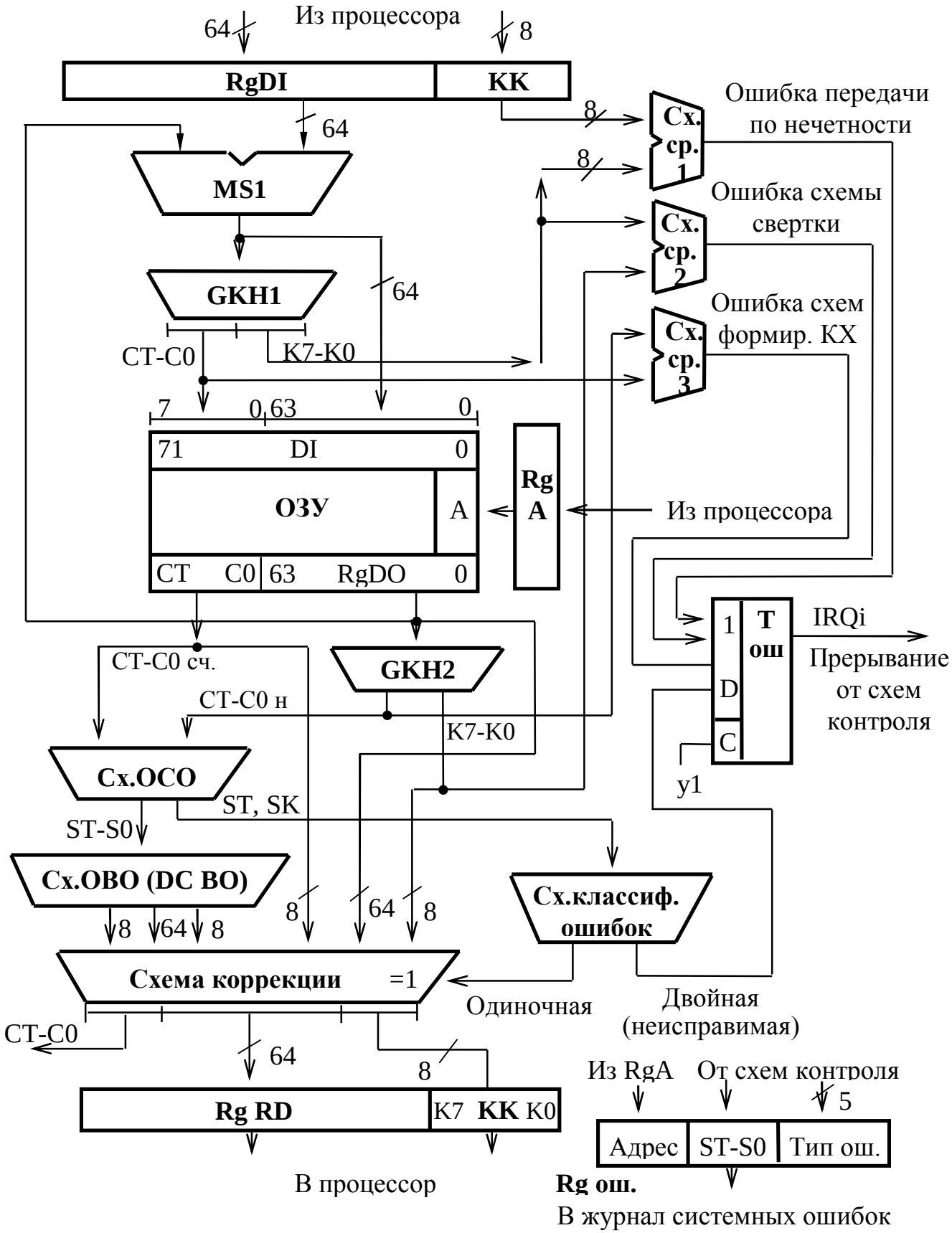


Рисунок 5.6 - Структурная схема самопроверяемой схемы контроля ОЗУ по коду Хэмминга

Схема управления коррекцией известной жесткой ошибки, хранимой в АЗУ, запускается схемой классификации типа ошибки.

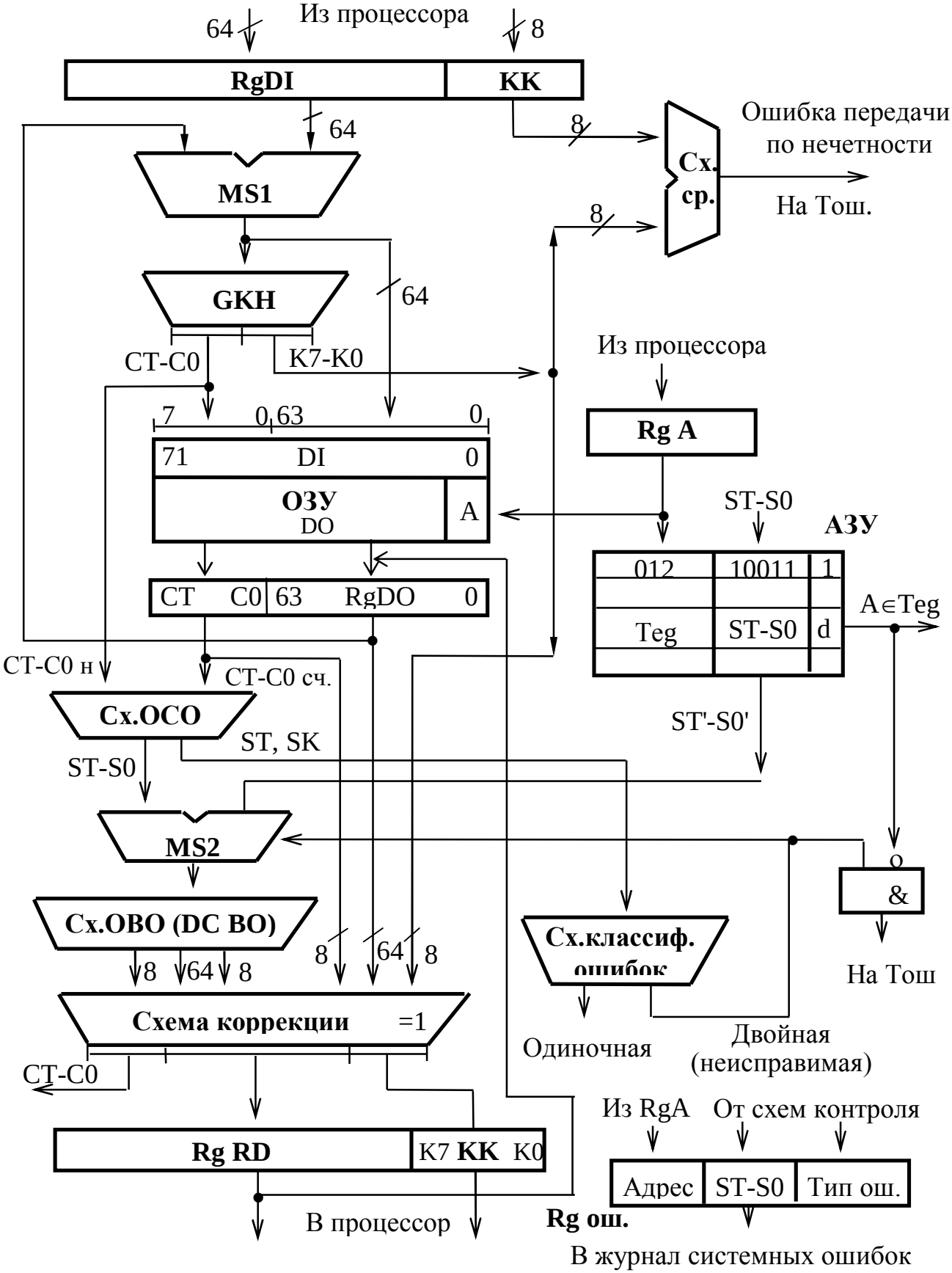


Рисунок 5.7 - Схема последовательной коррекции по коду Хэмминга

5.6.2 Алгоритмические методы исправления двойных ошибок

Двойные неисправимые ошибки могут быть частично исправлены методами алгоритмической коррекции неисправимых ошибок. Существуют следующие методы коррекции неисправимых ошибок:

* перезапись в инверсном коде;
* запись в ошибкозащищенном коде;
* последовательная коррекция.

Перезапись в инверсном коде. Алгоритм состоит из следующих шагов:

1. Чтение слова W1 из ОЗУ.

2. Если в слове W1 обнаружена одиночная ошибка, то она исправляется схемами коррекции КО-ОД и слово передается по назначению.

3. Если в слове W1 обнаружена неисправимая ошибка, то в эту же ячейку ОЗУ записывается инверсия считанного слова W2 = ~W1 считанное. Затем слово снова считывается и инвертируется W3 = ~W2 считанное;

4. Если в слове W3 схемами контроля обнаружена одиночная ошибка, то она исправляется схемами коррекции КО-ОД, а исправленное слово передается по назначению (и в некоторых алгоритмах записывается в ту же ячейку ОЗУ для устранения ошибки, если при записи был случайный сбой).

5. Если ошибок нет, то слово W3 считается безошибочным и передается по назначению.

6. Если в слове W3 обнаруживается неисправимая ошибка, то считается, что она алгоритмически неисправима и вырабатывается прерывание от схем контроля.

Пусть в ОЗУ обнаружена двойная ошибка в различных сочетаниях: одна "мягкая" (случайный сбой) и одна "жесткая" (отказ), две "жестких" и две "мягких":

Исходное слово W1 (записанное)	**1**1001100	**1**1001100	**1**1001100	**11**001100
Тип ошибки	Ж М	Ж Ж	М М	ЖЖ
W1 считанное с ошибками	**0**1001101	**0**1001101	**0**1001101	**00**001100
W2 записанное = ~W1 считанное	10110010	10110010	10110010	11110011
W2 считанное	00110010	00110011	10110010	00110011
W3 = ~W2 считанное	**1**1001101	**1**1001100	**0**1001101	**11**001100

Синдром жесткой ошибки	Одиночная	Устранены	Двойная	Устранены
W1 считанное ⊕ W2 считанное	**0**1111111	**0**1111110	11111111	**00**111111
	1	2	3	4

Из примеров видно, что любая "жесткая" ошибка по алгоритму самоустраняется, а одиночная "мягкая" исправляется аппаратурой КО-ОД. Двойная "мягкая" ошибка является неустранимой (пример 3).

Таким образом, алгоритмическое исправление позволяет корректировать 2Ж + М < d ошибок, где d - минимальное кодовое расстояние корректирующего кода.

5.7 Методы обеспечения отказоустойчивости памяти

К методам обеспечения отказоустойчивости ОЗУ относятся:

* разнесение неисправимых двойных ошибок путем логической перестановки адресов неисправных БИС памяти:
  - ◊ постоянная (ручная) логическая перестановка адресов;
  - ◊ управляемая логическая перестановка адресов;
  - ◊ автоматическая логическая перестановка адресов;

- подключение резервных ТЭЗ памяти;
- алгоритмическая коррекция неисправимых ошибок.

Использование метода разнесения двойных неисправимых ошибок путем логической перестановки адресов неисправных БИС памяти основано на использовании кода КО-ОД путем замены одной двойной ошибки двумя одиночными исправимыми ошибками.

На рисунке 5.8 показано размещение адресов БИС памяти при инверсии некоторых бит адреса, поступающих на адресные входы этих БИС. При такой организации имеет место измененный порядок выборки ячеек ОЗУ в каждой БИС памяти. Например, при RgA=0010 в БИС0 выбирается ячейка с адресом 0010, в БИС1 - 0011, БИС2 - 0000, БИС3 - 0001 и т.д.

Таким образом, если на ТЭЗе памяти для каждого адресного входа предусмотреть переключатель с инвертором адресного бита, то при обнаружении двойной ошибки обслуживающий персонал, выполнив переключение адресного входа неисправной БИС памяти, фактически выполнит разнесение двойной ошибки на две одиночные (ручная перестановка). Этот метод наиболее эффективно работает только при использовании однобитных БИС памяти. После ручного переключения адреса обязательно выполняется прогон тестов памяти для определения правильности разнесения всех двойных ошибок, выполняется перезагрузка ОС и прикладных программ, после чего ЭВМ может продолжить работу.

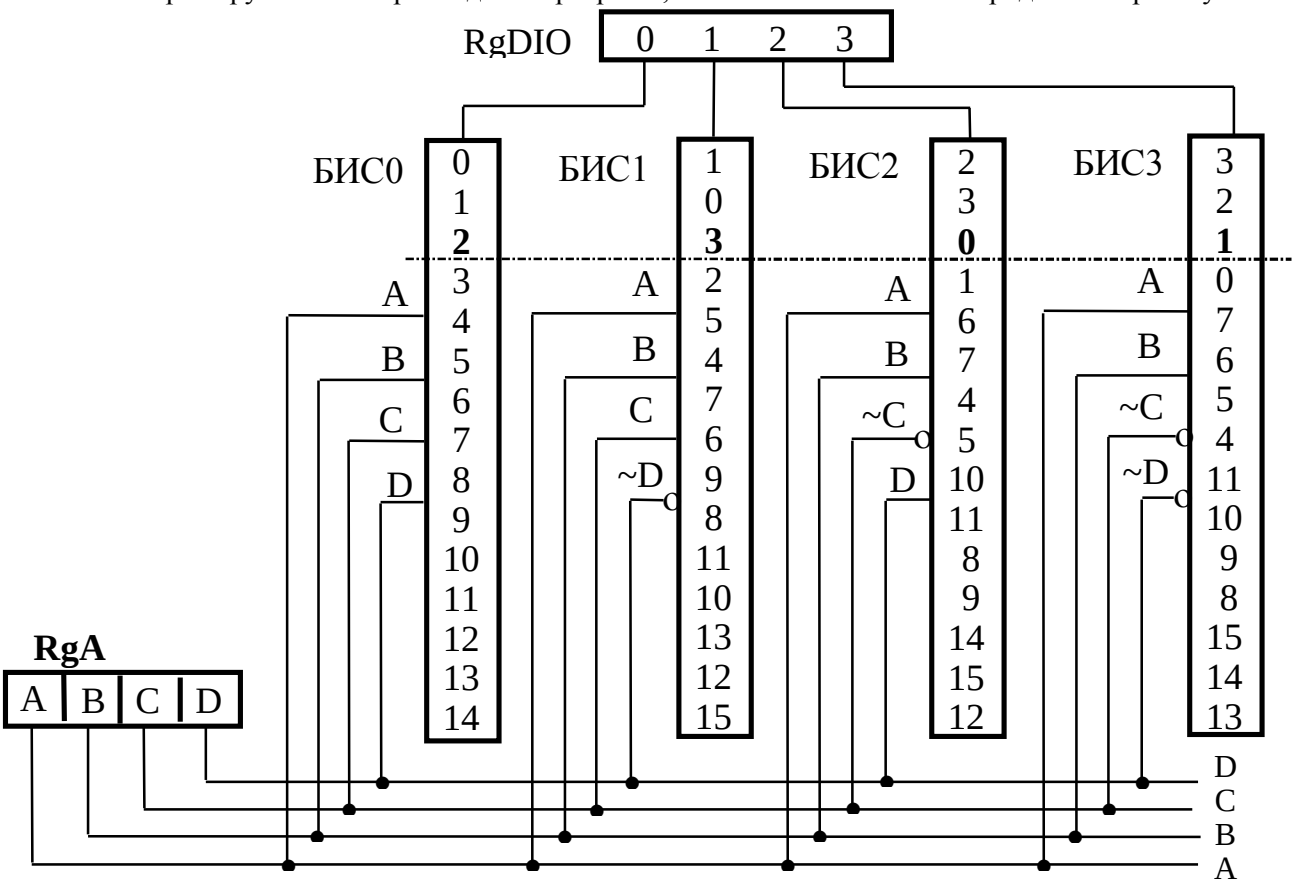


Рисунок 5.8 - Изменение адресации к БИС памяти при логической перестановке адресов

Управляемая логическая перестановка адресов позволяет несколько автоматизировать процедуру переключения (инверсии) адресных бит путем инверсии адресных бит через схему сложения по модулю два (рисунок 5.9), на вторые входы которых подается информация из регистра управления. В зависимости от состояния RgУ на адресные входы БИС памяти информация поступает в прямом или

обратном коде. Если в БИС ОЗУ имеется несколько неисправимых ошибок, то возникает задача подбора таких состояний RgУ, которые обеспечили бы разнесение всех неисправимых ошибок.

Для этих целей существуют алгоритмы вычисления значений RgУ.

1. Начальное значение всех RgУi и множества неисправных адресов FP равно нулю (i=1).
2. Формируется множество неисправных адресов:

$FPi = FPi\text{-}1 \cup (RgAi \oplus RgYi)$.

3. i := i+1. При обнаружении двойной ошибки формируется список запрещенных значений RgУi путем сложения по mod 2 всех изменяемых адресов из множества FPi с очередным адресом неисправной БИС: B=FPi-1⊕RgAi. В качестве значения RgУi выбирается один из кодов, не содержащихся в множестве B.

4. Переход к пункту 2 до тех пор, пока в множестве B не будет неиспользованных кодов для всех RgУi.

RgA ОЗУ (ЛА)	Номер БИС ОЗУ	Состояние RgУi	Физический адрес на входах БИС ОЗУ
00111	0	10101	10010
	1	11011	11100
	2	00100	00011
	3	00110	00001

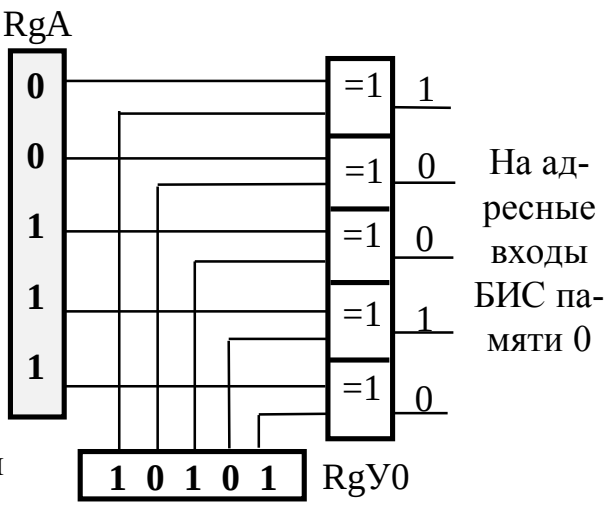


Рисунок 5.9 - Управляемая логическая перестановка адресов

Автоматическая логическая перестановка адресов. В этом варианте RgУ реализован в виде регистра сдвига с линейными обратными связями. Начальное состояние всех RgУ нулевое.

При появлении первой неисправимой (двойной) ошибки все регистры **устанавливаются в разное исходное состояние**, соответствующее значению 1, x, x^2,..., x^k mod g(x) генерирующего многочлена g(x).

При каждой последующей неисправимой ошибке на все регистры RgУ поступает сигнал сдвига.

Как видно из рисунка 5.10, при всех модификациях состояний RgУ нет ни одного состояния, где значения в RgУ (а следовательно, и на адресных входах БИС памяти) совпадают, чем и обеспечивается автоматическое разнесение двойных ошибок при каждом сдвиге информации в RgУ.

На рисунке 5.10 t0 - исходное состояние адресов БИС, t1 - адресов после обнаружения первой двойной ошибки после занесения по входам S1 начального значения полинома, t2, t3 - состояние адресов после каждого последующего сдвига информации в RgУ при обнаружении неисправимой ошибки.

Подключение резервных БИС (ТЭЗ) памяти. На рисунке 5.11 показано ОЗУ, состоящее из k БИС, каждая из которых служит для наращивания емкости ОЗУ и резервной БИС. Для автоматического подключения резервной БИС и логического отключения неисправной при обнаружении неисправности устанавливается триггер ошибки и система контроля или тестирования определяет номер неисправной БИС и загружает этот номер в регистр маски.

Схема мультиплексора выбора кристалла (MS CS) при каждом обращении к неисправной БИС переадресует их к резервной, а DC1 и одна из схем И-НЕ обеспечивает блокирование подачи сигнала выбора кристалла ~CSi на неисправную БИС. При этом выходы DC1 разрешаются только при установ-

ке триггера ошибки в "1" (при Тош=0 выходы DC1 имеют единичное значение). Таким образом, ЭВМ сохраняет работоспособность без вмешательства обслуживающего персонала до замены вышедшей из строя БИС памяти. Как и при использовании методов логической перестановки адресов, после реконфигурации памяти требуется перезагрузка ОС и прикладных программ пользователя.

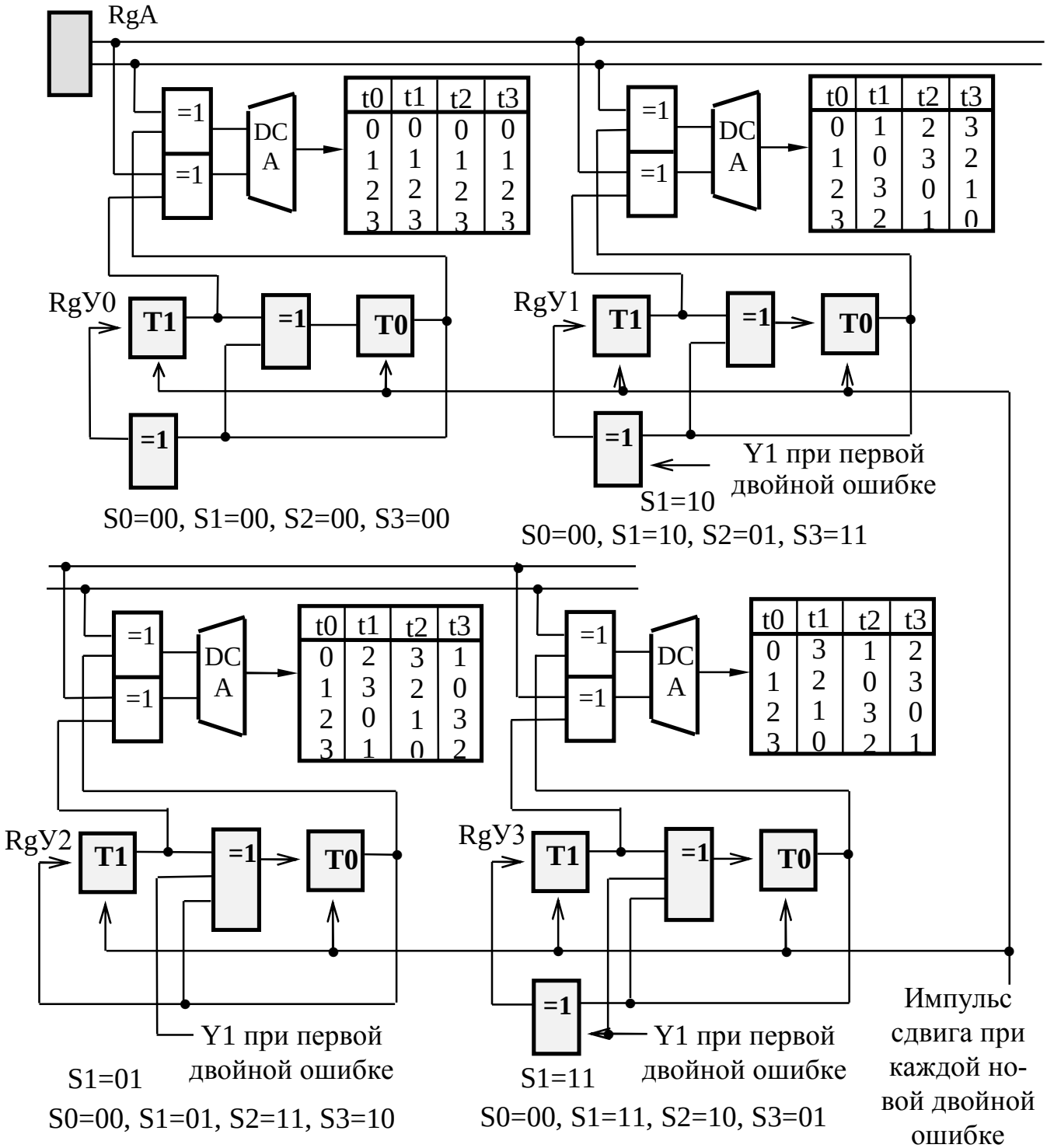
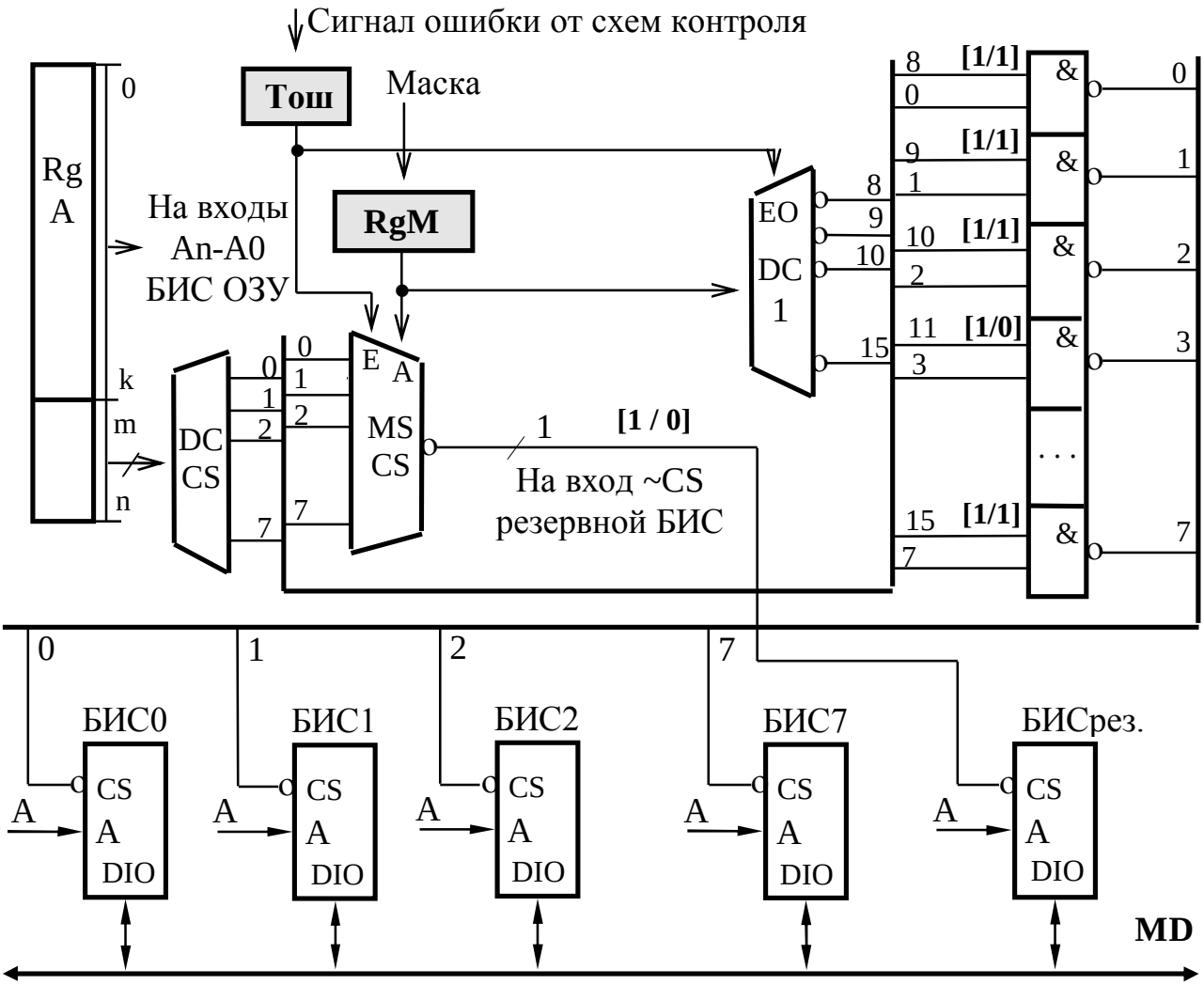


Рисунок 5.10 - Схема автоматической логической перестановки адресов



[1 / 0] - первая цифра означает состояние выходов схемы при Тош=0, вторая при Тош=1 и RgM=3 (обнаружена ошибка в модуле с номером 3)

Рисунок 5.11 - Метод замещения резервной БИС (ТЭЗ) памяти

Литература

1. Лю Ю-Чжен, Гибсон Г. Микропроцессоры семейства 8086/8088. Архитектура, программирование и проектирование микрокомпьютерных систем: Пер. с англ.- М.: Радио и связь, 1987.- 512 с.

2. Брамм П., Брамм Д. Микропроцессор 80386 и его программирование. М.: Мир, 1990.

3. Морс С.П., Д. Альберт. Архитектура микропроцессора 80286. М.: Радио и связь, 1990.

4. Б.Э. Смит, М.Т. Джонсон. Архитектура и программирование микропроцессора Intel 80386. М.: ТОО "Конкорд", 1992.

5. Григорьев В.Л. Микропроцессор i486. Архитектура и программирование (в 4-х книгах). Книга 1. Программная архитектура. - М.: ГРАНАЛ, 1993. - с.346.

6. Григорьев В.Л. Микропроцессор i486. Архитектура и программирование (в 4-х книгах). Книга 2. Аппаратная архитектура. Книга 3. Устройство с плавающей точкой. Книга 4. Справочник по системе команд. - М.: ГРАНАЛ, 1993. - с.382.

7. Паппас К., Марри У. Микропроцессор 80386: Справочник: Пер. с англ. - М.: Радио и связь, 1993. - 320 с.

8. Амамия М., Танака Ю. Архитектура ЭВМ и искусственный интеллект: Пер. с японск. - М.: Мир, 1993. - 400 с.

9. Бродин В.Б., Шагурин И.И. Микропроцессор i486: Архитектура, программирование, интерфейс. - М.: Диалог-МИФИ, 1993. - 240 с.

10. Гук М. Аппаратные средства IBM PC: Энциклопедия. - СПб.; М.; Харьков; Минск: Питер. 1999. - 816 с.

11. Корнеев В.В., Киселев А.В. Современные микропроцессоры. - М.: Изд-во "Нолидж", 1998. - 240 с.

12. Фролов А.В., Фролов Г.В. Аппаратное обеспечение персонального компьютера. - М.: Диалог-МИФИ, 1997. - 319 с.

13. Мансфельд Г., Эркамп Й., Дралле Ш. Аппаратные средства ПК. Секреты и советы. / Пер. с немецкого. - М.: Бином, 1997. - 192 с.

14. Шевкопляс Б.В. Микропроцессорные структуры. Инженерные решения: Доп.1. - М.: Радио и связь, 1993. - 253 с.

Содержание