

Тема 9. Методы управления основной памятью

Основные вопросы:

- 9.1. Отображение адресного пространства программы на область основной памяти
- 9.2. Адресная структура команд и планирование ресурсов
- 9.3. Виртуальная память
- 9.4. Система прерываний в ЭВМ

9.1. Отображение адресного пространства программы на область основной памяти

Алгоритмы распределения, использования, освобождения ресурсов и предоставления к ним доступа предназначены для наиболее эффективной организации работы всего комплекса устройств ЭВМ.

Рассмотрим их на примере управления основной памятью.

Для выполнения программы при ее загрузке в основную память ей выделяется часть *машинных ресурсов* - они необходимы для размещения

- команд,
- данных,
- управляющих таблиц и
- областей ввода-вывода,

т.е. производится **трансляция адресного пространства откомпилированной программы в местоположение в реальной памяти**.

Выделение ресурсов может быть осуществлено самим программистом (особенно если он работает на языке, близком машинному языку), но может производиться и ОС.

Если выделение ресурсов производится перед выполнением программы, такой процесс называется **статическим перемещением**, в результате которого программа “привязывается” к определенному месту в памяти вычислительной машины. Если же ресурсы выделяются в процессе выполнения программы, это называется **динамическим перемещением**, в этом случае программа не привязана к определенному месту в реальной памяти. Динамический режим можно реализовать только с помощью ОС.

При **статическом перемещении** возможны две ситуации.

1. Реальная память больше требуемого адресного пространства программы. В этом случае загрузка программы в реальную память производится, начиная с 0-го адреса (см. рис. 9.1.).



Рис. 9.1. Загрузка программы в реальную память (объем реальной памяти больше адресного пространства программы)

Загружаемая программа *A* является *абсолютной программой*, так как никакого изменения адресов в адресном пространстве, подготовленном компилятором, при загрузке в основную память не происходит - программа располагается с 0-го адреса реальной памяти.

2. Реальная память меньше требуемого адресного пространства программы (см. рис. 9.2).

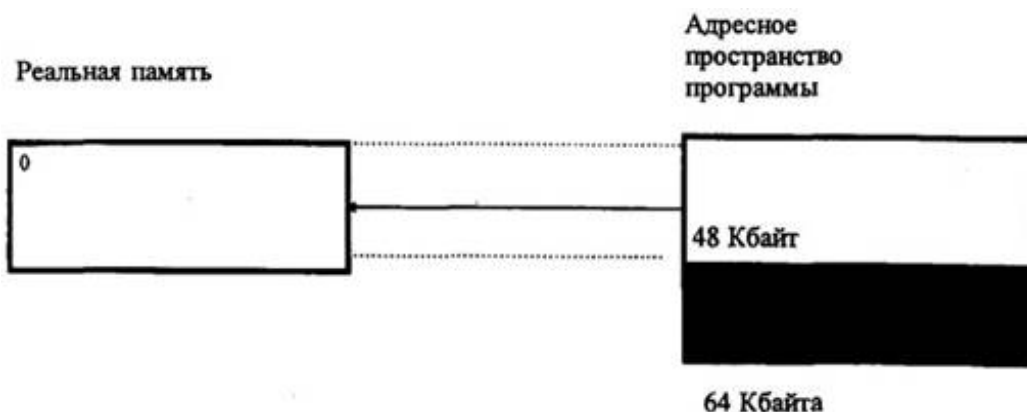


Рис. 9.2. Загрузка программы в реальную память (объем реальной памяти меньше адресного пространства программы)

В этом случае *программист* (или *ОС*) вынужден решать проблему, как организовать выполнение программы. Методов решения проблемы существует несколько: можно создать *оверлейную структуру* (т.е. разбить программу на части, вызываемые в ОП по мере необходимости), сделать модули программы *реентерабельными* (т.е. допускающими одновременную работу модуля по нескольким обращениям из разных частей программы или из различных программ) и т.д. В некоторых ОС адреса откомпилированной (с 0 адреса) программы могут быть преобразованы в адреса реальной памяти, отличные от 0. При этом создается *абсолютный модуль*, который требует размещения его в памяти всегда с одного и того же адреса.

При мультипрограммном режиме, если, например, имеем программы *A*, *B* и *C*, для которых известно, что программа *A* выполняется при размещении в памяти с адреса 60 Кбайт до 90 Кбайт, *B* - с 60 Кбайт до 90 Кбайт, *C* - с 50 Кбайт до 120 Кбайт, организовать их совместное выполнение невозможно, так как им необходим один и

тот же участок реальной памяти. Эти программы будут ждать друг друга либо их нужно заново редактировать с другого адреса.

При работе в мультипрограммном возможна ситуация, когда между программами образуются незанятые участки памяти, как, например, изображено на рис. 9.3.

Реальная память	
ОС	
20 Кбайт	
Программа А	0Кбайт
	Программа D
	50Кбайт
10 Кбайт	
Программа В	
20 Кбайт	
Программа С	

Рис. 9.3. Пример фрагментации реальной памяти

Общий объем незанятой памяти, составляющий 50 Кбайт, достаточен, чтобы загрузить и программу D, находящуюся в ожидании. Но ее не удастся загрузить, так как свободные участки памяти не являются смежными. Такое состояние называется **фрагментацией** реальной памяти. Оно характерно для систем со статическим перемещением.

В системах с динамическим перемещением программ перемещающий загрузчик размещает программу в свободной части памяти (см.рис.9.4.) и допускает использование несмежных ее участков.

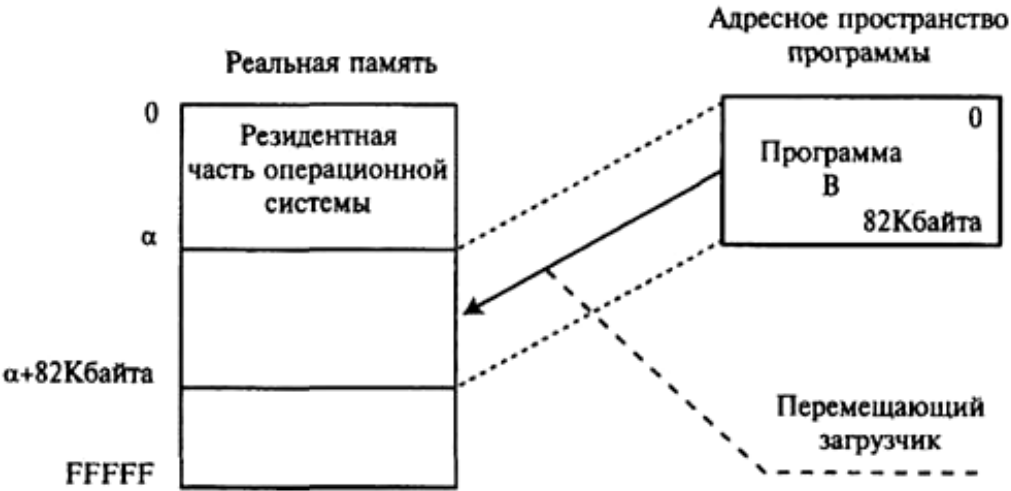


Рис. 9.4. Размещение программы в свободной части ОП

В этом случае имеется больше возможностей для организации мультипрограммной работы, следовательно, и для более эффективного использования временных ресурсов ЭВМ.

9.2. Адресная структура команд и планирование ресурсов

При больших размерах реализуемых программ возникают некоторые противоречия при организации мультипрограммного режима работы, трудности динамического распределения ресурсов.

Сегментная организация памяти

В настоящее время разработано несколько способов решения этих противоречий. Например, для борьбы с фрагментацией основной памяти адресное пространство программы может быть разбито на отдельные *сегменты*, слабо связанные между собой. Тогда (см. рис. 9.5) программа *D* общей длиной 50 Кбайт может быть представлена в виде ряда сегментов, загружаемых в различные области ОП. Это позволяет использовать реальную память, теряемую из-за фрагментации.

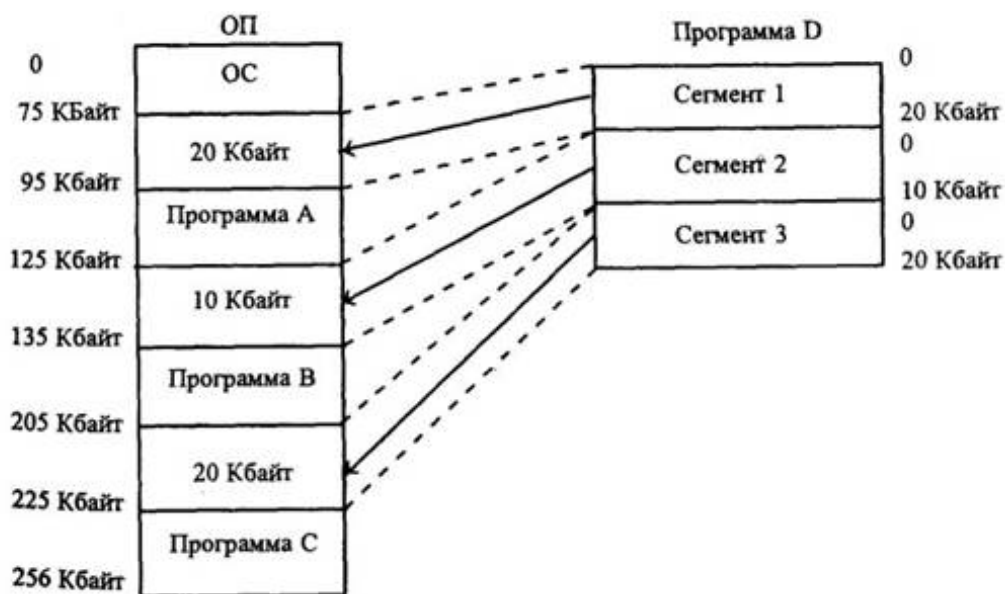


Рис. 9.5. Фрагментация ОП. Загрузка сегментированной программы

Адреса в каждом сегменте начинаются с 0. При статическом перемещении программы в процессе загрузки ее в основную память адреса должны быть привязаны к конкретному месту в памяти, на что уходит много времени, отвлекаются вычислительные ресурсы. Более эффективной является *динамическая трансляция адресов* (ДТА), которая заключается в том, что сегменты загружаются в основную память без трансляции адресного пространства (т.е. без изменения адресов в программе с учетом физического размещения в памяти команд и данных), а трансляция адресов каждой команды производится в процессе ее выполнения. Этот тип трансляции называется *динамическим перемещением* и осуществляется специальными аппаратными средствами ДТА.

Каждый сегмент программы должен иметь свое имя. Форма имени сегмента может быть любой, например, как на рис. 9.6. *а*, *б*.

При таком представлении адрес будет состоять из двух частей: s, i , где s - имя сегмента, i - адрес внутри сегмента.

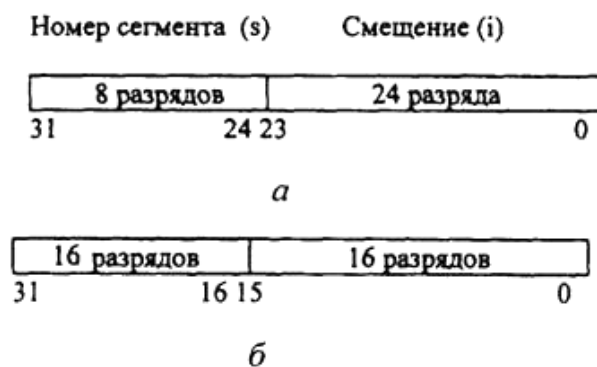


Рис. 9.5. Форма имени сегмента: *а* — при выделении 8-ми разрядов; *б* — при выделении 16-ти разрядов

Если имеется 32-битовая адресная структура, максимальная длина адреса в единственном сегменте будет длиной 32 разряда. Если 16 разрядов из 32 отвести под номер сегмента (а 16 — под смещение), то в этом случае все адресное пространство программы может состоять из 2^{16} сегментов. Сегмент может содержать $2^{16}=64$ Кбайта (т.е. иметь адреса от 0 до 65535).

При другой структуре адреса изменяются количество сегментов и их длина.

Структура адресов накладывает два важных ограничения:

- **ограничивается максимальное число сегментов**, которое может существовать в адресном пространстве программы;
- **ограничивается максимальное смещение любого адреса в сегменте.**

При загрузке в основную память сегментированной программы каждый сегмент перемещается в реальную память отдельно, причем участки основной памяти могут быть или не быть смежными. **Трансляция адресов не происходит — сегменты по-прежнему содержат свои относительные адреса.**

Процессор может обращаться к основной памяти, используя только абсолютные адреса.

Для динамической трансляции адресов (при определении абсолютных адресов по известным относительным, содержащим номер сегмента и смещение) ОС строит специальные таблицы, устанавливающие соответствие между сегментируемым адресным пространством программы и действительными адресами сегментов в реальной памяти. На рис. 9.6 приведен пример трансляции адресного пространства программы *D* в основную память.

Каждая строка таблицы сегментов содержит адрес начала сегмента в реальной памяти. Для каждого сегмента имеется одна строка таблицы.

Таблицу сегментов содержит каждая выполняемая программа.

В дополнение к таблице сегментов для динамической трансляции адреса используется специальный управляющий регистр, называемый **регистром начала таблицы сегментов** (PHTC или STOR (segment table origin register)). В этот регистр занесен адрес таблицы сегментов выполняемой в данный момент программы.

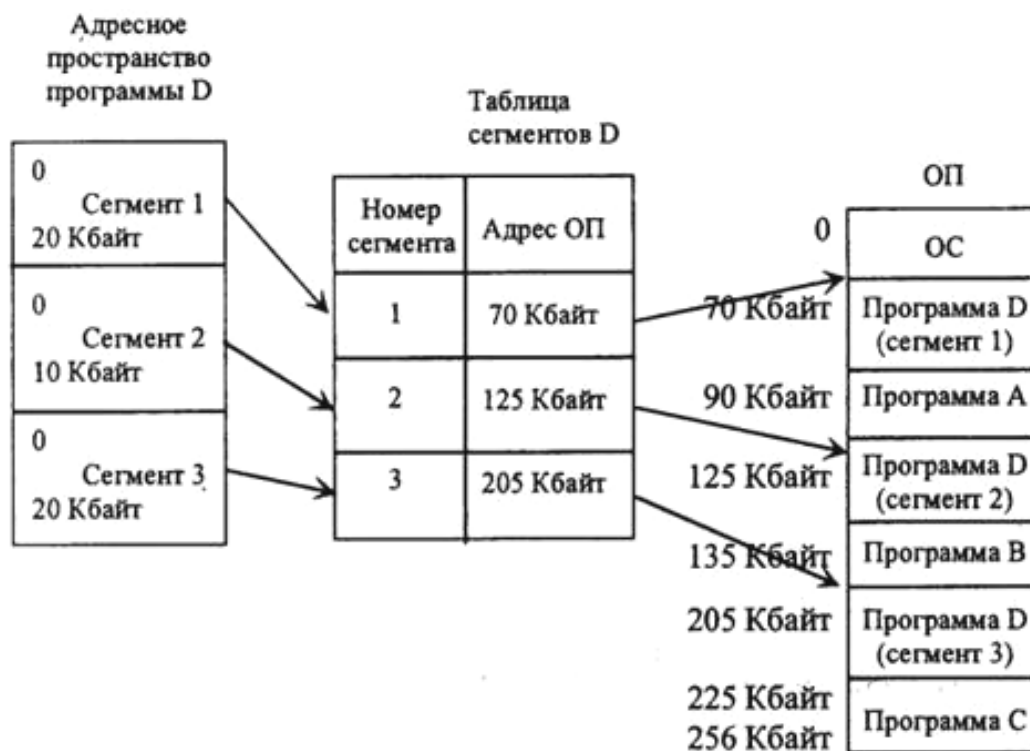


Рис. 9.6. Динамическая трансляция адресов при сегментной организации программы

На рис. 9.7 изображено выполнение программы *D*. В РНТС находится адрес таблицы сегментов этой программы. Если программа *B* прервет выполнение программы *D*, то в РНТС будет занесен начальный адрес таблицы сегментов программы *B*.

Допустим, для выполняемой программы *D* начальный адрес таблицы сегментов — 68000. В реальной вычислительной машине все действия выполняются в шестнадцатеричной системе счисления, здесь для простоты проведем вычисления в десятичной системе счисления.

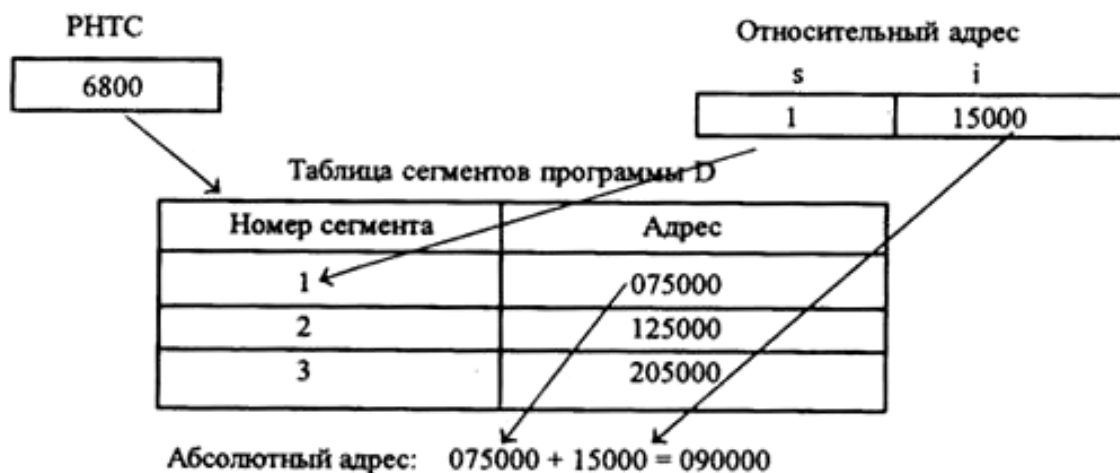


Рис. 9.9. Технология динамической трансляции адресов

Для обращения к адресу 15000 сегмента 1 производятся следующие действия:

- РНТС указывает на начало таблицы сегментов программы D - 68000;
- номер сегмента в относительном адресе используется как индекс при обращении к таблице сегментов. В данном примере обращение производится к 1-й строке;

• адрес, хранимый в выбранной строке таблицы сегментов, есть адрес начала сегмента в реальной памяти. Смещение в относительном адресе добавляется к начальному адресу, и результат является адресом в реальной памяти: $15000 + 75000 = 90000$.

Для относительного адреса (сегмент 3, смещение 13000) будет получен абсолютный адрес 218000.

При ДТА такое определение адресов ведется в процессе выполнения каждой команды!!

Если операционной системе понадобится переместить исполняемую программу в другую часть памяти (например, чтобы исключить фрагментацию), сначала надо будет переслать команды и данные сегмента. Затем строку таблицы сегментов для данного сегмента нужно изменить так, чтобы она содержала новый адрес, и выполнение программы может быть продолжено. Это дает возможность динамического управления реальной памятью в процессе выполнения программы.

Использованием сегментации программ достигается уменьшение фрагментации основной памяти, но полностью фрагментация не устраняется — остаются фрагменты, длина которых меньше длины сегмента программы.

Сегментно-страничная организация памяти

Если сегменты разделить на одну или несколько единиц, называемых **страницами**, которые имеют **фиксированный размер**, то поскольку размер страницы достаточно мал по сравнению с обычным размером сегментов, неиспользуемые фрагменты ОП значительно сокращаются в объеме — будет иметь место так называемая фрагментация внутри страниц. Следовательно, потери все-таки останутся, но они будут существенно меньше.

Сегментно-страничная организация добавляет еще один уровень в структуре адресного пространства программы. Теперь адресное пространство программы дробится на сегменты, внутри сегментов — на страницы и адреса внутри страниц. Структура адреса представлена на рис.9.12: (s, p, i) , где s — имя сегмента внутри адресного пространства программы; p — имя страницы; i — адрес внутри страницы.

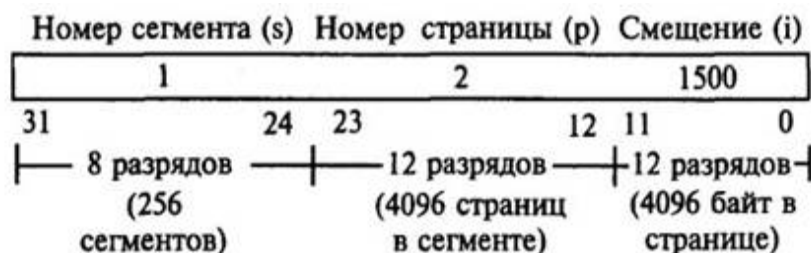


Рис. 9.8. Адресная структура при сегментно-страничной организации памяти

Формирование сегментно-страничной структуры выполняется автоматически с помощью операционной системы.

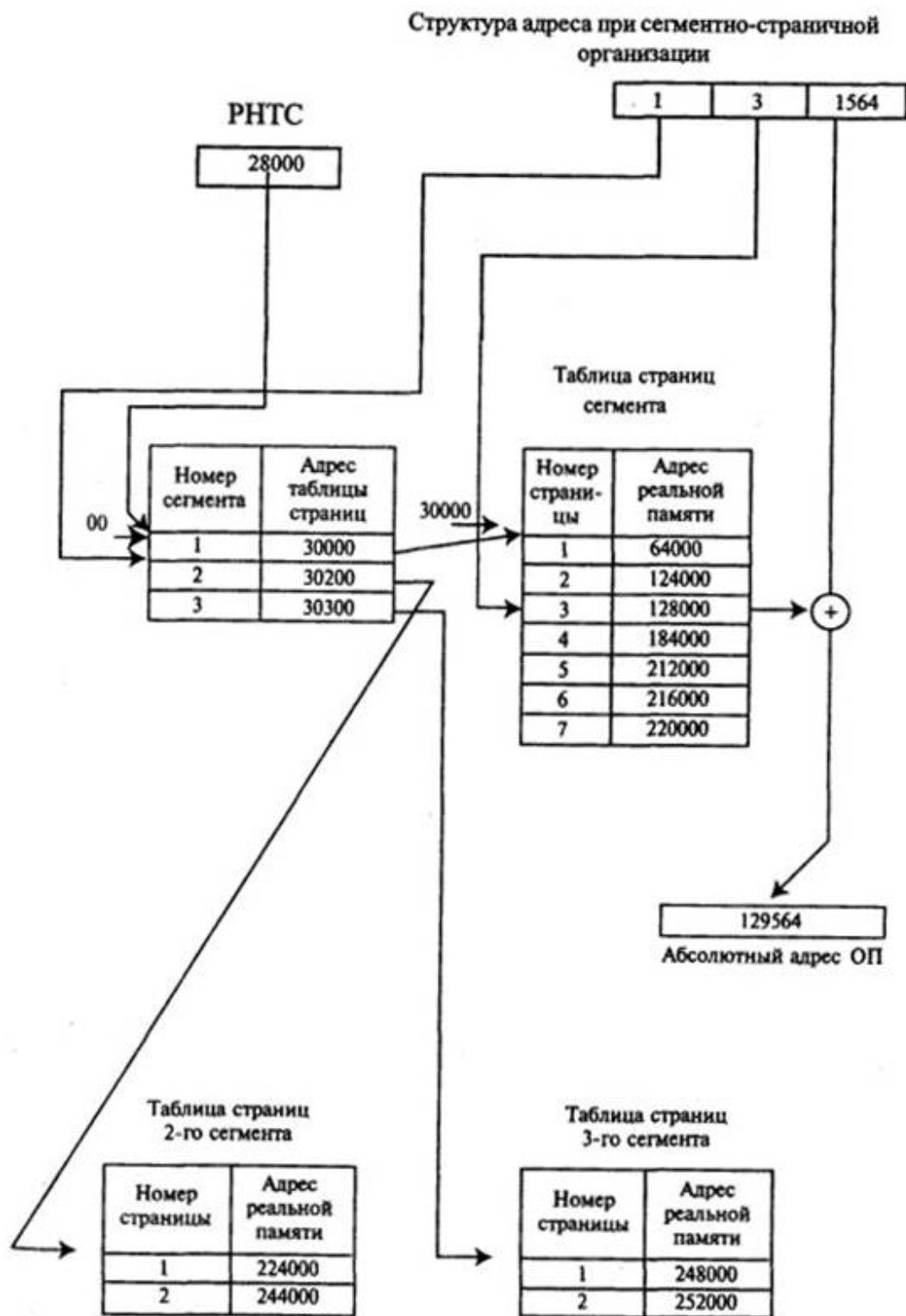


Рис. 9.9. Структурная схема формирования абсолютного адреса при сегментно-страничной организации ОП

Для динамической трансляции адресов (ДТА) каждому сегменту необходимы одна таблица сегментов и несколько таблиц страниц (см. рис. 9.8).

ДТА будет выполняться следующим образом:

- регистр начала таблицы сегментов содержит начальный адрес таблицы сегментов выполняемой программы — 28000;
- номер сегмента в относительном адресе используется как индекс для обращения к записи таблиц сегментов. Эта запись идентифицирует начало таблицы страницы (реальный адрес) — 30000;
- номер страницы в относительном адресе используется как индекс для обращения к записи таблицы страниц. Эта запись идентифицирует начало страничного блока, содержащего эту страницу - 128000;
- смещение в относительном адресе и местоположение страничного блока объединяются вместе, формируя абсолютный адрес — 129564. В реальной системе адрес страничного блока и смещение связываются, т.е. соединяются вместе для образования абсолютного адреса.

Все преимущества динамического перемещения с использованием сегментации и страничной организации достигаются благодаря аппаратуре и программному обеспечению, а не пользователям системы.

Специальные программы во время загрузки разбивают адресное пространство программы на сегменты и страницы, строят таблицы сегментов и страниц.

Средства ДТА автоматически транслируют адрес в процессе выполнения программы.

9.3. Виртуальная память

При наличии иерархической структуры запоминающих устройств, на реальном объеме памяти, значительно меньшем максимального, можно имитировать работу с максимальной памятью. Режим имитации памяти максимально допустимого для данной ЭВМ объема при имеющейся реальной памяти меньшей по объему называется **режимом виртуальной памяти**.

Теоретически доступная пользователю ОП, объем которой определяется только разрядностью адресной части команды, и которая не существует в действительности, называется **виртуальной памятью**.

Виртуальная память имеет сегментно-страничную организацию и всегда реализуется в иерархической системе памяти ЭВМ. Часть ее размещается в **страничных блоках основной памяти**, а часть — в ячейках **внешней страничной памяти (slot)**. Внешняя страничная память является частью внешней памяти. Ячейка (слот) — это записываемая область во внешней страничной памяти (например, на жестком магнитном диске). Она того же размера, что и страница.

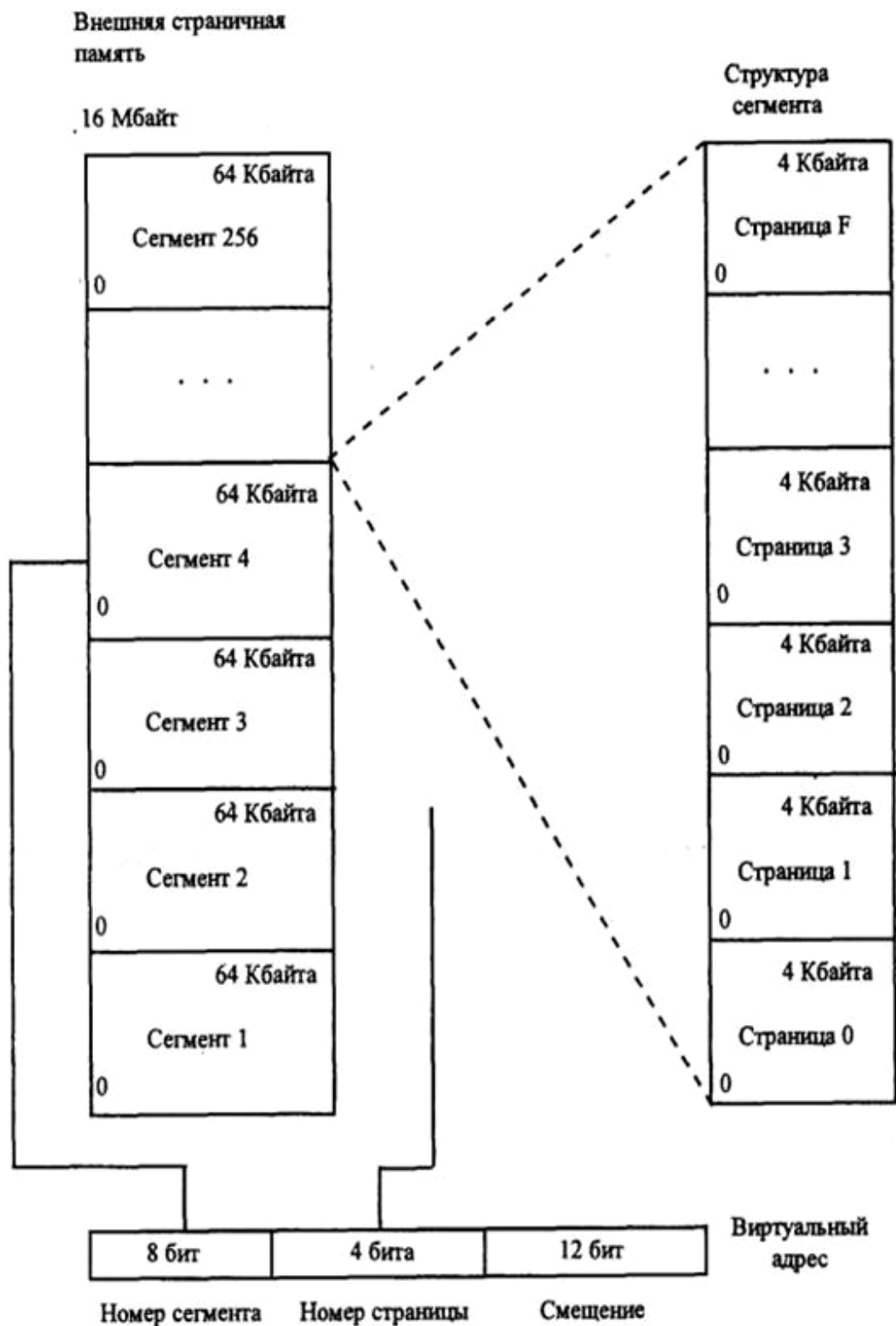


Рис. 9.10. Пример структуры внешней страничной памяти и виртуального адреса

Например, ЭВМ с 24-разрядным адресом (2^{24}) может иметь адресное пространство в 16 Мбайт, с 32-разрядным адресом (2^{32}) - 4 Гбайт. Структура такой памяти показана на рис.9.10.

Все программные страницы физически располагаются в ячейках внешней страничной памяти. *Виртуальная же память существует только как продукт*

деятельности ОС, функционирующей на основе совместного использования внешней и страничной памяти.

Загрузить программу в виртуальную память — значит переписать несколько программных страниц из внешней страничной памяти в основную память. Если в процессе выполнения программы *A* система обнаружит, что требуемой страницы нет в реальной памяти, она должна переслать копию этой страницы из внешней страничной памяти в реальную память. Этот механизм называется *принудительным страничным обменом*.

При расшифровке виртуального адреса номер сегмента с помощью таблицы сегментов соотносится с адресом таблицы страниц. Таблица страниц содержит номер страницы и адрес страничного блока. В виртуальном режиме к таблице страниц добавляется еще одна колонка, содержащая *бит недоступности*. Нулевое состояние этого бита означает, что соответствующая страница загружена в реальную память. Единичное состояние означает, что страница недоступна, ее надо переписать в реальную память из внешней. Местоположение страницы во внешней памяти указывается в таблице внешних страниц. [Виртуальная память_допол.docx](#)

4.3. Система прерываний ЭВМ

Современная ЭВМ представляет собой комплекс автономных устройств, каждое из которых выполняет свои функции под управлением местного устройства управления независимо от других устройств машины. ЦП активизирует работу устройства, передавая устройству команду и все необходимые для ее исполнения параметры. После начала работы устройства ЦП отключается от него и переходит к обслуживанию других устройств или к выполнению других функций.

Можно считать, что ЦП *переключает свое внимание* с устройства на устройство и с функции на функцию. На что именно обращено внимание ЦП в каждый данный момент, определяется выполняемой им программой.

Во время работы в ЦП поступает (и вырабатывается в нем самом) большое количество различных сигналов. Сигналы, которые выполняемая в ЦП программа способна воспринять, обработать и учесть, составляют *поле зрения* ЦП или другими словами - входят в зону его внимания.

Например, если процессором исполняется программа сложения двух двойных слов, которая анализирует регистр флагов ЦП, то в “поле ее зрения” находятся:

- флаги микропроцессора, определяющие знаки исходных данных и результата,
- наличие переноса из тетрады или байта,
- переполнение разрядной сетки и др.

Такая программа готова реагировать на любой из сигналов, находящихся в ее зоне внимания. Поскольку именно программа управляет работой ЦП, то она определяет и “зону внимания” ЦП. Но если во время выполнения такой программы нажать какую-либо клавишу, то эта программа “не заметит” сигнала от этой клавиши, так как он не входит в ее “поле зрения”.

Для того чтобы ЦП, выполняя свою работу, имел возможность реагировать на события, происходящие вне его зоны внимания, наступления которых он “не ожидает”, существует *система прерываний ЭВМ*. При отсутствии системы

прерываний все заслуживающие внимания события должны находиться в поле зрения процессора, что сильно усложняет программы и требует большой их избыточности. Кроме того, поскольку момент наступления события заранее не известен, процессор в ожидании какого-либо события может находиться длительное время, и чтобы не пропустить его появления, ЦП не может “отвлекаться” на выполнение какой-либо другой работы. Такой режим работы (режим сканирования ожидаемого события) связан с большими потерями времени ЦП на ожидание.

Кроме сокращения потерь на ожидание, режим прерываний позволяет организовать выполнение такой работы, которую без него реализовать просто невозможно. Например, при появлении неисправностей, нештатных ситуаций режим прерываний позволяет организовать работу по диагностике и автоматическому восстановлению в момент возникновения нештатной ситуации, прервав выполнение основной работы таким образом, чтобы сохранить полученные к этому времени правильные результаты. Тогда как без режима прерываний обратить внимание на наличие неисправности система могла только после окончания выполняемой работы (или ее этапа) и получения неправильного результата.

Таким образом, *система прерываний позволяет микропроцессору выполнять основную работу, не отвлекаясь на проверку состояния сложных систем при отсутствии такой необходимости, или прервать выполняемую работу и переключиться на анализ возникшей ситуации сразу после ее появления.*

Помимо требующих внимания нештатных ситуаций, которые могут возникнуть при работе микропроцессорной системы, процессору полезно уметь “переключать внимание” и на различные виды работ, одновременно выполняемые в системе. Поскольку управление работой системы осуществляется программой, этот вид прерываний должен формироваться программным путем.

В зависимости от места нахождения источника прерываний они могут быть разделены на *внутренние* (программные и аппаратурные) и *внешние* прерывания (поступающие в ЭВМ от внешних источников, например, от клавиатуры или модема).

Принцип действия системы прерываний заключается в следующем:

при выполнении программы после каждого рабочего такта микропроцессора изменяются содержимое регистров, счетчиков, состояние отдельных управляющих триггеров, т.е. **изменяется состояние процессора**. Информация о состоянии процессора лежит в основе многих процедур управления вычислительным процессом. Не вся информация одинаково актуальна, есть **существенные элементы**, без которых невозможно продолжение работы. Эта информация должна **сохраняться при каждом “переключении внимания процессора”**.

Совокупность значений наиболее существенных информационных элементов называется **вектором состояния** или **словом состояния процессора** (в некоторых случаях она называется **словом состояния программы**).

Вектор состояния в каждый момент времени должен содержать информацию, достаточную для продолжения выполнения программы или повторного пуска ее с точки, соответствующей моменту формирования данного вектора.

Вектор состояния формируется в соответствующем регистре процессора или в группе регистров, которые могут использоваться и для других целей.

Наборы информационных элементов, образующих векторы состояния, отличаются у ЭВМ разных типов.

В IBM PC вектор состояния включает содержимое счетчика команд, сегментных регистров, регистра флагов и аккумулятора (регистра AX).

При возникновении события, требующего немедленной реакции со стороны машины, ЦП прекращает обработку текущей программы и переходит к выполнению другой программы, специально предназначенной для данного события, по завершении которой возвращается к выполнению отложенной программы. Такой режим работы называется *прерыванием*.

Каждое событие, требующее прерывания, сопровождается специальным сигналом, который называется *запросом прерывания*. Программа, затребованная запросом прерывания, называется *обработчиком прерывания*.

Запросы на прерывание могут возникать из-за сбоев в аппаратуре (зафиксированных схемами контроля), переполнения разрядной сетки, деления на нуль, выхода за установленные для данной программы области памяти, затребования периферийным устройством операции ввода-вывода, завершения этой операции ввода-вывода или возникновения при этой операции особых условий и т.д.

Некоторые из этих запросов порождаются самой программой, но время их возникновения невозможно предсказать заранее.

При наличии нескольких источников запросов прерывания часть из них может поступать одновременно. Поэтому в ЭВМ устанавливается определенный порядок (дисциплина) обслуживания поступающих запросов. Кроме того, в ЭВМ предусматривается *возможность разрешать или запрещать прерывания определенных видов*.

Например, ПЭВМ IBM PC может выполнять 256 различных прерываний, каждое из которых имеет свой номер (двухразрядное шестнадцатеричное число).

Все прерывания делятся на две группы:

- прерывания с номера 00h по номер 1Fh называются *прерываниями базовой системы ввода-вывода (BIOS -Basic Input-Output System)*;
- прерывания с номера 20h по номер FFh называются *прерываниями DOS*.

Прерывания DOS имеют более высокий уровень организации, чем прерывания BIOS, они строятся на использовании модулей BIOS в качестве элементов.

Прерывания делятся на три типа:

- *аппаратурные,*
- *логические и*
- *программные.*

Аппаратурные прерывания вырабатываются устройствами, требующими внимания микропроцессора: прерывание № 2 — отказ питания; № 8 — от таймера; № 9 — от клавиатуры; № 12 — от адаптера связи; № 14 — от НГМД; № 15 — от устройства печати и др.

Запросы на **логические прерывания** вырабатываются внутри микропроцессора при появлении “нештатных” ситуаций: прерывание № 0 — при попытке деления на 0; № 4 — при переполнении разрядной сетки арифметико-логического устройства; № 1 — при переводе микропроцессора в пошаговый режим работы; № 3 — при достижении программой одной из контрольных точек. Последние два прерывания используются отладчиками программ для организации пошагового режима выполнения программ (трассировки) и для остановки программы в заранее намеченных контрольных точках.

Запрос на **программное прерывание** формируется по команде $INTn$, где n — номер вызываемого прерывания.

Запрос на аппаратное или логическое прерывание вырабатывается в виде специального электрического сигнала.