

Тема 4 (Повтор материала 1-го семестра)

Функциональная и структурная организация ЭВМ

Оглавление

1. ОБЩИЕ ПРИНЦИПЫ ФУНКЦИОНАЛЬНОЙ И СТРУКТУРНОЙ ОРГАНИЗАЦИИ ЭВМ	1
2. ОРГАНИЗАЦИЯ ФУНКЦИОНИРОВАНИЯ ЭВМ С МАГИСТРАЛЬНОЙ АРХИТЕКТУРОЙ	2
3. ОРГАНИЗАЦИЯ РАБОТЫ ЭВМ ПРИ ВЫПОЛНЕНИИ ЗАДАНИЯ ПОЛЬЗОВАТЕЛЯ	5
4. ОТОБРАЖЕНИЕ АДРЕСНОГО ПРОСТРАНСТВА ПРОГРАММЫ НА ОСНОВНУЮ ПАМЯТЬ.....	7
4.1. Адресная структура команд микропроцессора и планирование ресурсов.....	10
4.2. Виртуальная память.....	14
4.3. Система прерываний ЭВМ.....	17

1. Общие принципы функциональной и структурной организации ЭВМ

Электронные вычислительные машины включают, кроме аппаратурной части и программного обеспечения (ПО), большое количество функциональных средств. К ним относятся коды, с помощью которых обрабатываемая информация представляется в цифровом виде:

- арифметические коды - для выполнения арифметических преобразований числовой информации;
- помехо- защищенные коды, используемые для защиты информации от искажений;
- коды формы, определяющие, как должна выглядеть обрабатываемая в ЭВМ информация при отображении; цифровые коды аналоговых величин (звука, “живого видео”) и др.

Кроме кодов, на функционирование ЭВМ оказывают влияние:

- алгоритмы их формирования и обработки,
- технология выполнения различных процедур (например, начальной загрузки операционной системы, принятой в системе технологии обработки заданий пользователей и др.);
- способы использования различных устройств и организация их работы (например, организация системы прерываний или организация прямого доступа к памяти),
- устранение негативных явлений (например, таких, как фрагментация памяти) и др.

Будем считать, что коды, система команд, алгоритмы выполнения машинных операций, технология выполнения различных процедур и взаимодействия hard и soft, способы использования устройств при организации их совместной работы, составляющие идеологию функционирования ЭВМ, образуют функциональную организацию ЭВМ.

Реализована идеология функционирования ЭВМ может быть по-разному:

- аппаратурными,
- программно-аппаратурными или
- программными средствами.

При аппаратурной и программно-аппаратурной реализации могут быть применены регистры, дешифраторы, сумматоры; блоки жесткого аппаратурного управления или микропрограммного с управлением подпрограммами (комплексами микроопераций); устройства или комплексы устройств, реализованные в виде автономных систем (программируемых или с жестким управлением) и др.

При программной реализации могут быть применены различные виды программ - обработчики прерываний, резидентные или загружаемые драйверы, com-, exe- или tsr - программы, bat- файлы и др.

Будем считать, что способы реализации функций ЭВМ составляют структурную организацию ЭВМ. Тогда элементная база, функциональные узлы и устройства ЭВМ, программные модули различных видов (обработчики прерываний, драйверы, com-, exe-, tsr-программы, bat-файлы и др.) являются структурными компонентами ЭВМ.

При серьезных конструктивных различиях ЭВМ могут быть совместимыми, т.е. приспособленными к работе с одними и теми же программами (*программная совместимость*) и получению одних и тех же результатов при обработке одной и той же, однотипно представленной информации (*информационная совместимость*). Если аппаратурная часть ЭВМ допускает их электрическое соединение для совместной работы и предусматривает обмен одинаковыми последовательностями сигналов, то имеет место и *техническая совместимость* ЭВМ.

Совместимые ЭВМ должны иметь одинаковую функциональную организацию: информационные элементы (символы) должны одинаково представляться при вводе и выводе из ЭВМ, система команд должна обеспечивать в этих ЭВМ получение одинаковых результатов при одинаковых преобразованиях информации. Работой таких машин должны управлять одинаковые или функционально совместимые операционные системы (а для этого должны быть совместимы методы и алгоритмы планирования и управления работой аппаратурно-программного вычислительного комплекса). Аппаратурные средства должны иметь согласованные питающие напряжения, частотные параметры сигналов, а главное - состав, структуру и последовательность выработки управляющих сигналов.

2. Организация функционирования ЭВМ с магистральной архитектурой

ЭВМ представляет собой совокупность устройств, выполненных на больших интегральных схемах, каждая из которых имеет свое функциональное назначение. Комплект интегральных схем, из которых состоит ЭВМ, называется *микропроцессорным комплектом*. В состав микропроцессорных комплектов входят: системный таймер, микропроцессор (МП), сопроцессоры, контроллер прерываний, контроллер прямого доступа к памяти, контроллеры устройств ввода-вывода и др.

Все устройства ЭВМ делятся на *центральные* и *периферийные*. Центральные устройства полностью электронные, периферийные устройства могут быть либо электронными, либо электромеханическими с электронным управлением.

В центральных устройствах основным узлом, связывающим микропроцессорный комплект в единое целое, является *системная магистраль*. Она состоит из трех узлов, называемых шинами: шина данных (ШД), шина адреса (ША), шина управления (ШУ). В состав системной магистрали входят регистры-зашелки, в которых запоминается передаваемая информация, шинные формирователи, шинные арбитры, определяющие очередность доступа к системной магистрали, и др.

Логика работы системной магистрали, количество разрядов (линий) в шинах данных, адреса и управления, порядок разрешения конфликтных ситуаций, возникающих при одновременном обращении различных устройств ЭВМ к системной магистрали, образуют *интерфейс системной шины*.

В состав центральных устройств ЭВМ входят: центральный процессор, основная память и ряд дополнительных узлов, выполняющих служебные функции: контроллер прерываний, таймер и контроллер прямого доступа к памяти (ПДП).

Периферийные устройства делятся на два вида: *внешние ЗУ* (НМД, НГМД, НМЛ) и *устройства ввода-вывода* (УВВ): клавиатура, дисплей, принтер, мышь, адаптер каналов связи (КС) и др.

Управляющая работой ЭВМ программа перед началом выполнения загружается в основную память. Адрес первой выполняемой команды передается микропроцессору и запоминается в счетчике команд.

Начало работы процессора заключается в том, что адрес из счетчика команд (в котором всегда хранится адрес очередной команды) выставляется на шину адреса системной магистрали. Одновременно на шину управления выдается команда: **выборка из ОП**, которая воспринимается основной памятью. Получив с шины управления системной магистрали команду, основная память считывает адрес с шины адреса, находит ячейку с этим номером и ее содержимое выставляет на шину данных, а на шину управления выставляет сигнал о выполнении команды. Процессор, получив по шине управления сигнал об окончании работы ОП, вводит число с шины данных на внутреннюю магистраль МП и через нее пересылает введенную информацию в регистр команд.

В регистре команд полученная команда разделяется на кодовую и адресную части. Код команды поступает в блок управления для выработки сигналов, настраивающих МП на выполнение заданной операции, и для определения адреса следующей команды (который сразу заносится в счетчик команд). Адресная часть команды выставляется на шину адреса системной магистрали (СМ) и сопровождается сигналом **выборка из ОП** на шине управления. Выбранная из ОП информация через шину данных поступает на внутреннюю магистраль МП, с которой вводится в арифметическое устройство (АУ). На этом заканчивается подготовка МП к выполнению операции, и начинается ее выполнение в АЛУ.

Результат выполнения операции выставляется микропроцессором на шину данных, на шину адреса выставляется адрес ОП, по которому этот результат необходимо записать, а на шину управления выставляется команда **запись в ОП**. Получив с шины управления команду, ОП считывает адрес и данные с системной магистрали, организует запись данных по указанному адресу и после выполнения команды выставляет на шину управления сигнал, обозначающий, что число записано. Процессор, получив этот сигнал, начинает выборку очередной команды: выставляет адрес из счетчика команд на шину адреса, формирует команду **выборка из ОП** на шине управления и т.д.

В каждом цикле, получив команду в регистр команд и выделив код операции, процессор определяет, к какому устройству она относится. Если команда должна выполняться процессором, организуется ее выполнение по описанному циклу. Если же команда предназначена для выполнения в другом устройстве ЭВМ, центральный процессор (ЦП) передает ее соответствующему устройству. Процесс передачи команды другому устройству предусматривает следующие действия: ЦП выставляет на шину адреса СМ адрес интересующего его устройства. По шинам управления передается сигнал **поиск устройства**. Все устройства, подключенные к системной магистрали, получив этот сигнал, читают номер устройства с шины адреса и сравнивают его со своим номером.

Устройства, для которых эти номера не совпадают, на эту команду не реагируют. Устройство с совпавшим номером вырабатывает сигнал отклика по шине управления. ЦП, получив сигнал отклика, в простейшем случае выставляет имеющуюся у него команду на шину данных и сопровождает ее по шине управления сигналом **передаю команду**. Получив сигнал о приеме команды, ЦП переходит к выполнению очередной своей команды, выставляя на шину адреса содержимое счетчика команд.

В более сложных случаях, получив сигнал, что устройство откликнулось, прежде чем передавать команду, ЦП запрашивает устройство о его состоянии. Текущее состояние устройства закодировано в байте состояния, который откликнувшееся устройство передает процессору через ШД системной магистрали. Если устройство включено и готово к работе, то байт состояния - нулевой. Наличие в нем единиц свидетельствует о нештатной ситуации, которую ЦП пытается проанализировать и в необходимых случаях извещает оператора о сложившейся ситуации.

Взаимодействие МП с внешними устройствами предусматривает выполнение логической последовательности действий, связанных с поиском устройства, определением его технического состояния, обменом командами и информацией. Эта логическая последовательность действий вместе с устройствами, реализующими ее, получила название *интерфейс ввода-вывода*.

Для различных устройств могут использоваться разные логические последовательности действий, поэтому интерфейсов ввода-вывода может в одной и той же ЭВМ использоваться несколько. Если их удастся свести к одному, универсальному, то такой интерфейс называется *стандартным*. В IBM PC есть два стандартных интерфейса для связи ЦП с внешними устройствами: параллельный (типа Centronics) и последовательный (типа RS-232).

Интерфейсы постоянно совершенствуются, поэтому с появлением новых ЭВМ, новых внешних устройств и даже нового программного обеспечения появляются и новые интерфейсы. Так, в программном обеспечении, разработанном ведущими фирмами, все шире используется новый интерфейс Plug and Play (Включи и играй), который предназначен для облегчения системной настройки ЭВМ при подключении новых устройств, к машине. Этот интерфейс позволяет подключить с помощью кабеля новое устройство, а после включения ЭВМ ее программное обеспечение автоматически определяет состав подключенных устройств, их типы и настраивает машину на работу с ними без вмешательства системного оператора.

Если при обращении ЦП к внешнему устройству продолжение выполнения основной программы центральным процессором возможно только после завершения операции ввода-вывода, то ЦП, запустив внешнее устройство, переходит в состояние ожидания и находится в нем до тех пор, пока внешнее устройство не сообщит ему об окончании обмена данными. Это приводит к простоему большинства устройств ЭВМ, так как в каждый момент времени может работать только одно из них. Такой режим работы получил название *однопрограммного* - в каждый момент времени все устройства находятся в состоянии ожидания, и только одно устройство выполняет основную (и единственную) программу.

Для ликвидации таких простоев и повышения эффективности работы оборудования внешние устройства сделаны автономными. Получив от ЦП необходимую информацию, они самостоятельно организуют свою работу по обмену данными. Процессор же, запустив внешнее устройство, пытается продолжить выполнение программы. При необходимости (если встретятся соответствующие команды) он может запустить в работу несколько других устройств (так как внешние устройства работают значительно медленнее процессора). Если же ему приходится переходить в режим ожидания, то, пользуясь тем, что в ОП может одновременно находиться не одна, а несколько программ, ЦП переходит

к выполнению очередной программы. При этом создается ситуация, когда в один и тот же момент времени различные устройства ЭВМ выполняют либо разные программы, либо разные части одной и той же программы, такой режим работы ЭВМ называется *многопрограммным*.

3. Организация работы ЭВМ при выполнении задания пользователя

Организация процессов ввода, преобразования и отображения результатов относится к сфере системного программного обеспечения. Это сложные процессы, ^которые чаще всего делаются “прозрачными”, т.е. незаметными для пользователя. Один из них - реализация задания пользователя: профессиональный пользователь (программист) пишет задание для ЭВМ в виде программы на *алгоритмическом языке*. Написанное задание (программа) представляет собой *исходный модуль*, сопровождаемый управляющими предложениями, указывающими операционной системе ЭВМ, на каком языке написана программа и что с ней надо делать. Если программа пишется на алгоритмическом языке, то управляющие предложения - на языке управления операционной системой (в ЕС ЭВМ и IBM 360/370 этот язык называется -Job Control Language, в MS DOS IBM PC - это язык команд DOS, иногда оформляемый в виде bat - файла).

Исходный модуль перед исполнением должен быть переведен на внутренний язык машины. Эта операция выполняется специальной программой - *транслятором* (рис. 1). Трансляторы выполняются в виде двух разновидностей: *интерпретаторы* и *компиляторы*. Интерпретатор после перевода на язык машины каждого оператора алгоритмического языка немедленно исполняет полученную машинную программу. Компилятор же сначала полностью переводит всю программу, представленную ему в виде исходного модуля (ИМ), на язык машины. Получаемая при этом машинная программа представляет собой *объектный модуль* (ОМ). Результат работы компилятора может быть записан в библиотеку объектных модулей (БОМ) или передан другим программам для дальнейшей обработки, поскольку полученная машинная программа не готова к исполнению по двум причинам.

Во-первых, она содержит *неразрешенные внешние ссылки* (т.е. обращение к программам, которые не содержатся в исходном модуле, но необходимы для работы

основной программы, например, к стандартным программам алгоритмического

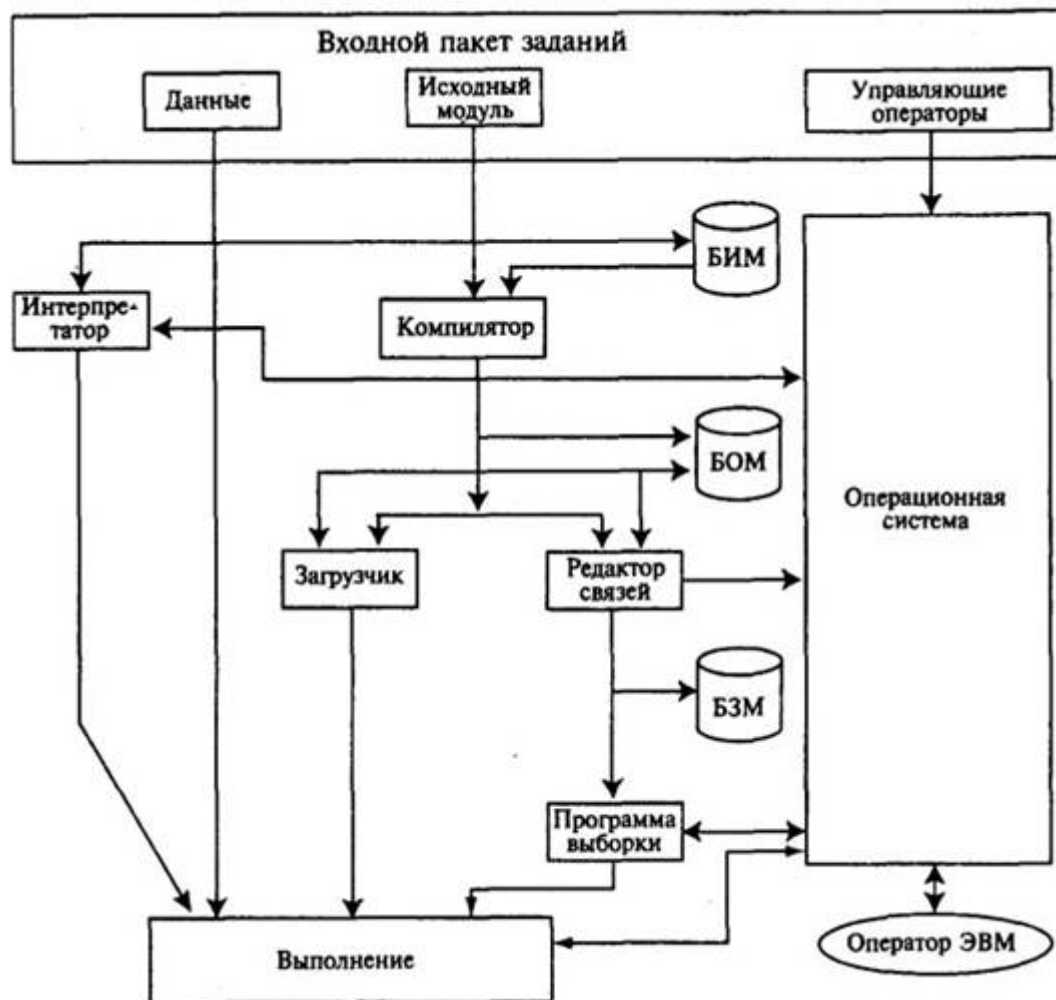


Рис. 1. Обработка заданий операционной системой

языка, таким, как *извлечение корня квадратного, вычисление тригонометрических функций* и т.д.). Во-вторых, объектный модуль представляет собой машинную программу *в условных адресах* - каждый объектный модуль начинается с адреса 0h, тогда как для исполнения программа должна быть “привязана” к конкретным *физическим адресам* основной памяти.

Недостающие программы должны быть взяты из *библиотек компилятора* (которые могут быть написаны в виде исходных либо в виде объектных модулей) и добавлены к основной программе. Эту операцию выполняет *редактор связей*. В результате работы редактора связей образуется *загрузочный модуль* (ЗМ), который помещается в соответствующую библиотеку загрузочных модулей (БЗМ). В загрузочном модуле все ссылки разрешены, т.е. он содержит все необходимые стандартные программы, но привязки к памяти у загрузочного модуля нет.

Привязка к памяти загрузочного модуля производится *программой выборки*, которая переносит загрузочный модуль из библиотеки загрузочных модулей (обычно хранящейся на магнитном носителе) в основную память и во время этого переноса корректирует адреса, учитывая, с какого адреса основной памяти размещается загрузочный модуль. После перемещения загрузочного модуля в основную память программа выборки инициирует ее выполнение.

Представление машинной программы в виде исходных, объектных и загрузочных модулей позволяет реализовать наиболее эффективные программные комплексы. Например, если по одной и той же программе необходимо много раз производить расчеты, то неэффективно тратить каждый раз время на трансляцию и редактирование программы - ее нужно оформить в виде загрузочного модуля и хранить в соответствующей библиотеке. При обращении к такой программе сразу будет вызываться программа выборки для загрузки соответствующего модуля (а этапы компиляции и редактирования связей будут опускаться) - время на выполнение программы существенно сократится.

Если же программа только отлаживается или после каждого просчета ее нужно будет модернизировать, то получение загрузочного модуля и обращение к программе выборки будут лишними операциями. Для их обхода вместо редактора связей может быть применен *загрузчик* - программа, сочетающая в себе функции редактирования связей и загрузки полученной машинной программы в основную память для исполнения. Но при использовании загрузчика многократные просчеты по программе

4. Отображение адресного пространства программы на основную память

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), но может производиться и операционной системой.

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 операционной системы.

При статическом перемещении может встретиться два случая.

1. Реальная память больше требуемого адресного пространства программы. В этом случае загрузка программы в реальную память производится, начиная с 0-го адреса (рис.2).

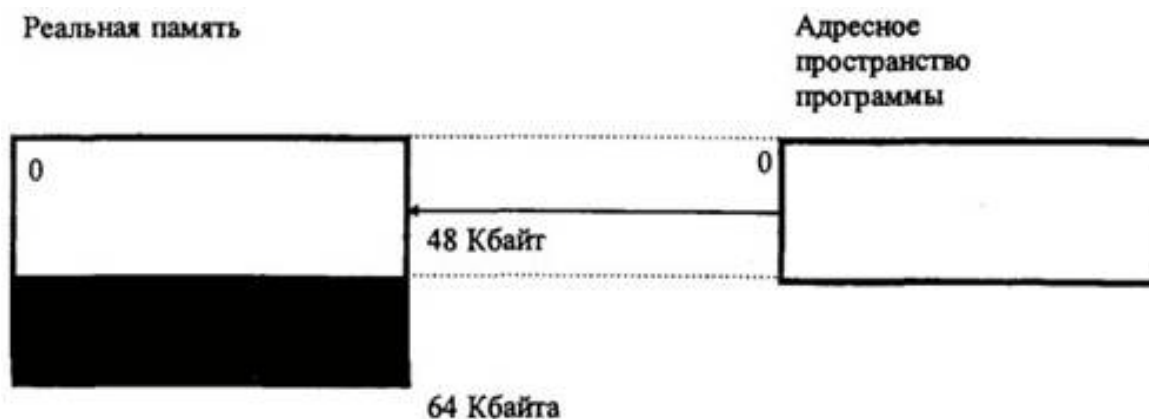


Рис.2. Загрузка программы в реальную память (объем реальной памяти больше адресного пространства программы)

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 (рис.3).

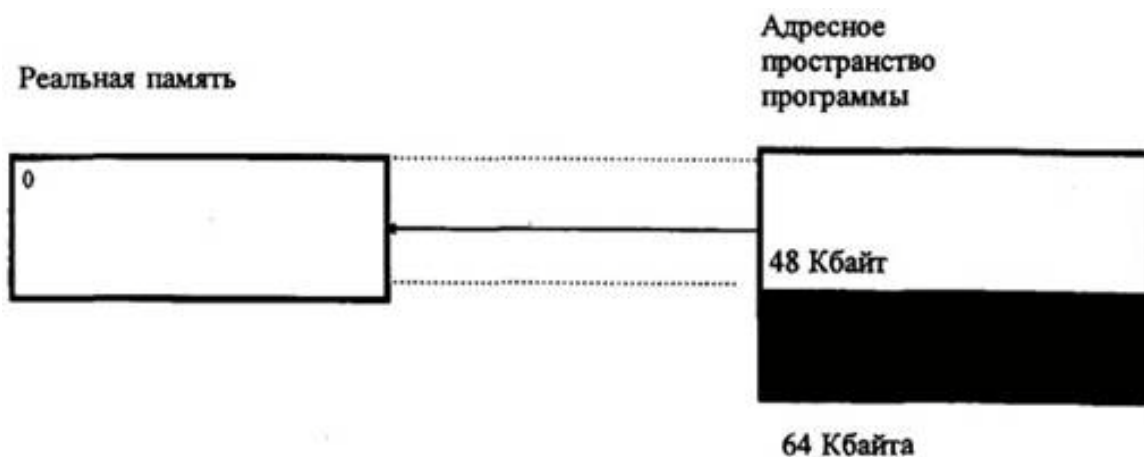


Рис.3. Загрузка программы в реальную память (объем реальной памяти меньше адресного пространства программы)

Реальная память	
ОС	
20Кбайт	
Программа А	Программа D
	0Кбайт

	50Кбайт
10Кбайт	
Программа В	
20Кбайт	
Программа С	

Рис. 4. Фрагментация реальной памяти

В этом случае программист (или операционная система) 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 операционных системах 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 имеем программы **А**, **В** и **С**, для которых известно, что программа **А** выполняется при размещении в памяти с адреса 60 Кбайт до 90 Кбайт, **В** - с 60 Кбайт до 90 Кбайт, **С** - 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 режиме может сложиться ситуация, когда между программами образуются незанятые участки памяти. На рис.4

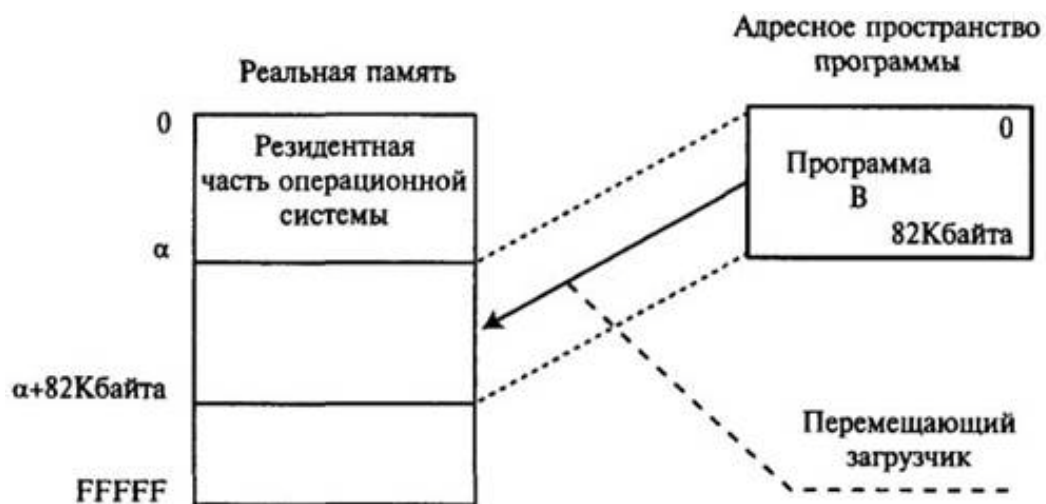


Рис.5. Размещение программы в свободной части ОП

общий объем незанятой памяти, составляющий 50 Кбайт, достаточен, чтобы загрузить и программу **Д**, 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 (рис.5) и допускает использование несмежных ее участков.

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

4.1. Адресная структура команд микропроцессора и планирование ресурсов

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

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 (рис.6) 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.

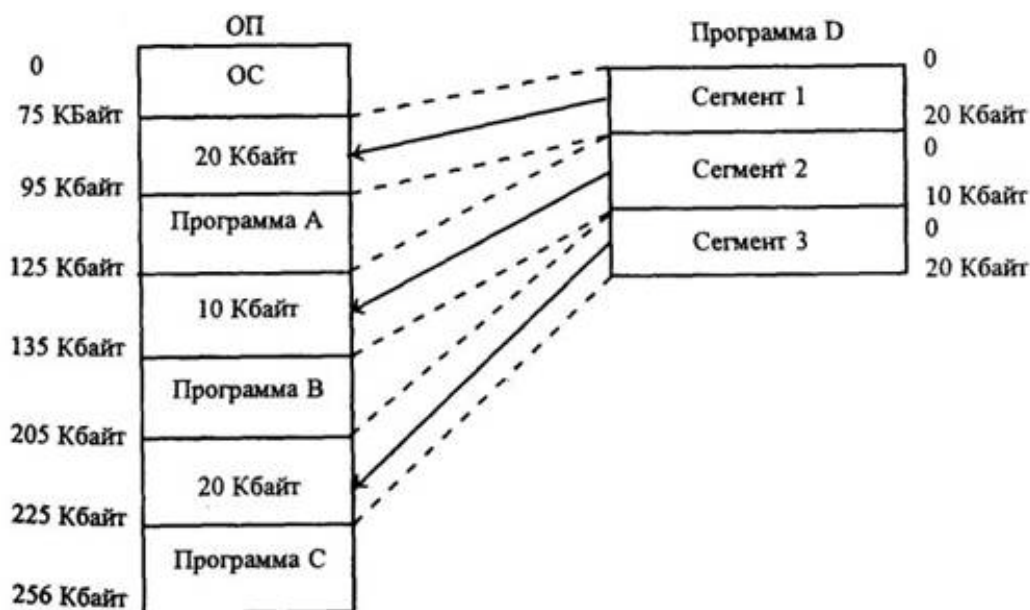


Рис. 6. Фрагментация ОП. Загрузка сегментированной программы

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 номер (рис.7 а, б).

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

Если ЭВМ имеет 32-битовую адресную структуру, 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 $2^{16} = 64$ Кбайта 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.

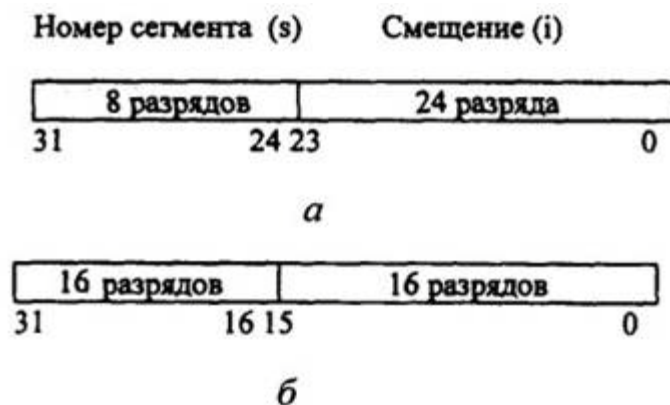


Рис.7. Форма имени сегмента: *а* - при выделении 8 разрядов; *б* - при выделении 16 разрядов

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.

Для динамической трансляции адресов (т.е. при определении абсолютных адресов по известным относительным, содержащим номер сегмента и смещение) операционная система строит специальные таблицы, устанавливающие соответствие между сегментируемым адресным пространством программы и действительными адресами сегментов в реальной памяти (рис.8).

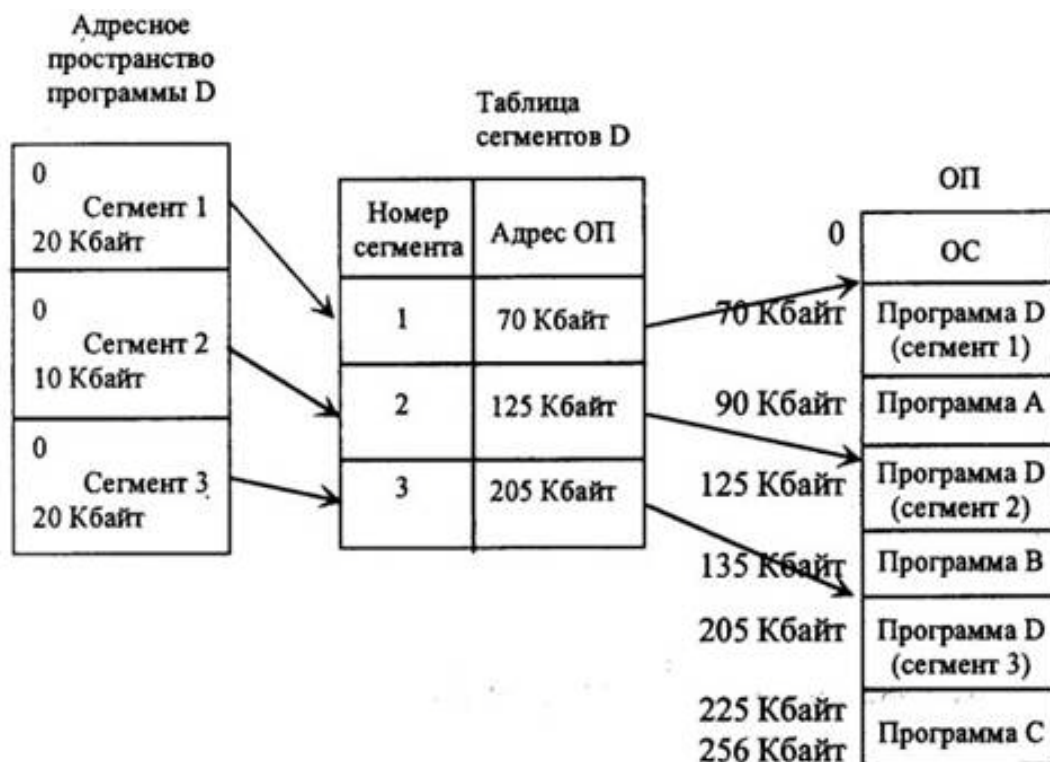


Рис.8. Динамическая трансляция адресов при сегментной организации программы

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

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* (РНТС или STOR (segment table origin register)). В этот регистр занесен адрес таблицы сегментов выполняемой в данный момент программы.

На рис.9 изображено выполнение программы D. В РНТС находится адрес таблицы сегментов этой программы. Если программа **В** прервет выполнение программы **D**, то в РНТС будет занесен начальный адрес таблицы сегментов программы **В**.

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

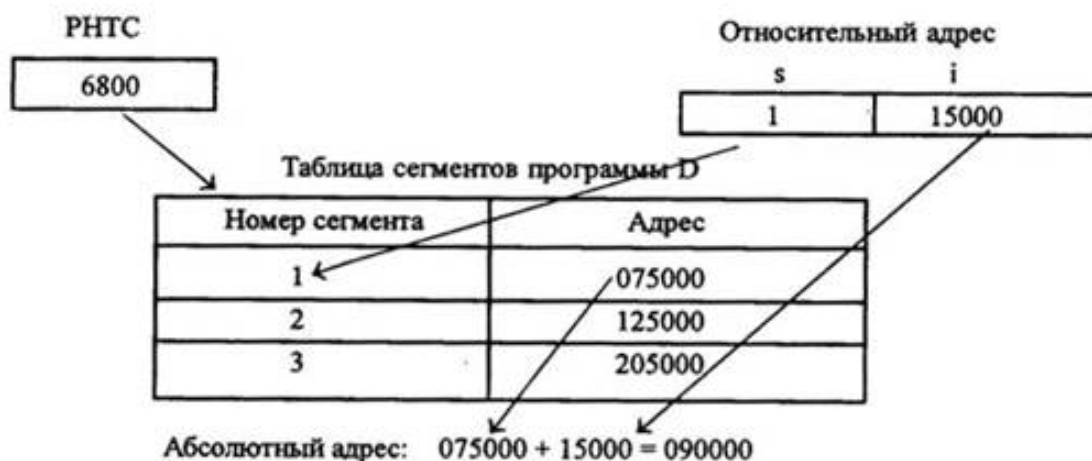


Рис. 9. Технология динамической трансляции адресов

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.

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: (s, p, i) - рис.10, где s - имя сегмента

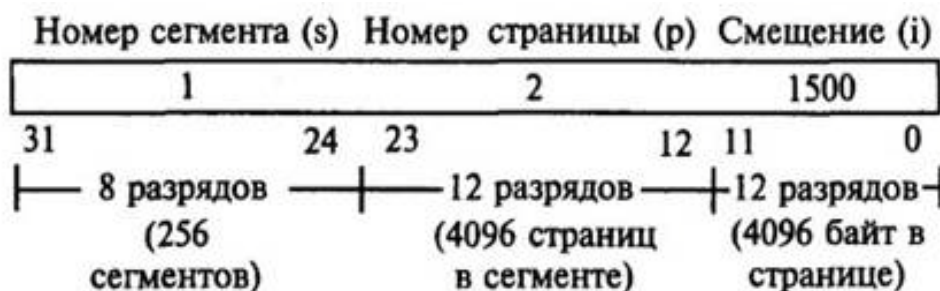


Рис.10. Адресная структура при сегментно-страничной организации памяти внутри адресного пространства программы; p - имя страницы; i - адрес внутри страницы.

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

Для динамической трансляции адресов (ДТА) каждому сегменту необходимы одна таблица сегментов и несколько таблиц страниц (рис.4.11).

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 страниц (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.

4.2. Виртуальная память

Имея иерархическую структуру запоминающих устройств, на реальном объеме памяти, значительно меньшем максимального, можно имитировать работу с максимальной памятью. В этом случае программист работает так, как будто ему предоставлена реальная память максимально допустимого для данной ЭВМ объема, хотя имеющаяся реальная память значительно меньше по объему. Такой режим работы называется *режимом виртуальной памяти*.

Имея иерархическую структуру запоминающих устройств, на реальном объеме памяти, значительно меньшем максимального, можно имитировать работу с максимальной памятью. В этом случае программист работает так, как будто ему

предоставлена реальная память максимально допустимого для данной ЭВМ объема, хотя имеющаяся реальная память значительно меньше по объему. Такой режим работы называется *режимом виртуальной памяти*.

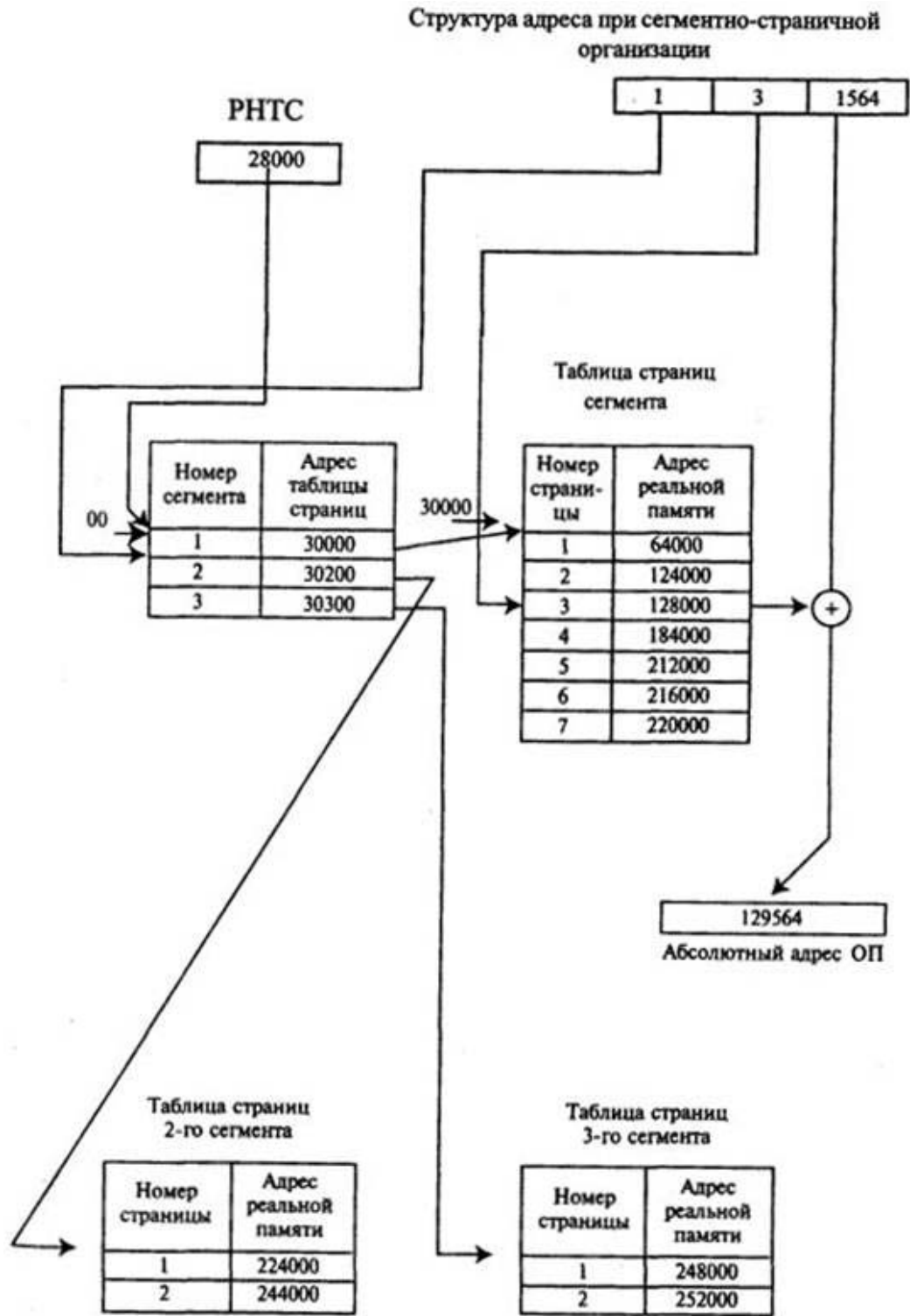


Рис. 11. Структурная схема формирования абсолютного адреса при сегментно-страничной организации ОП

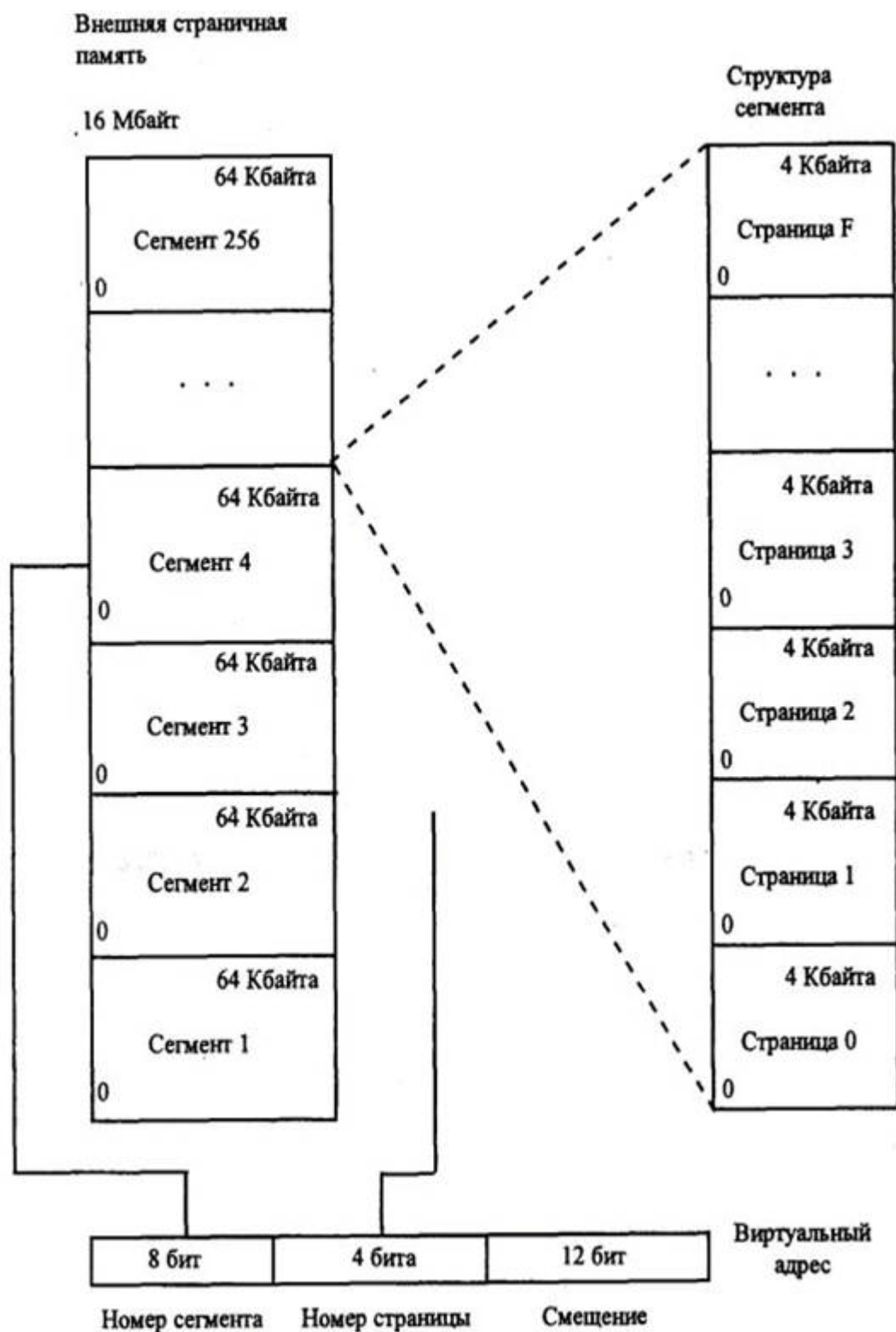


Рис. 12. Структура виртуальной памяти

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 реализована 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.

Вычислительная система с 24-разрядным адресом может иметь адресное пространство в 16 777 216 байт (16 Мбайт), с 32-разрядным адресом - 4 Гбайт. Структура такой памяти показана на рис.12.

Все программные страницы физически располагаются в ячейках внешней страничной памяти. Виртуальная же память существует только как продукт деятельности операционной системы (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 А 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.

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. Включает устройство в работу центральный процессор. Он передает устройству команду и все необходимые для ее исполнения параметры. После начала работы устройства центральный процессор отключается от него и переходит к обслуживанию других устройств или к выполнению других функций.

Можно считать, что центральный процессор *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 (а поскольку именно

программа управляет работой ЦП, то она определяет и “зону внимания” центрального процессора). 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 аппаратные)* 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.

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.