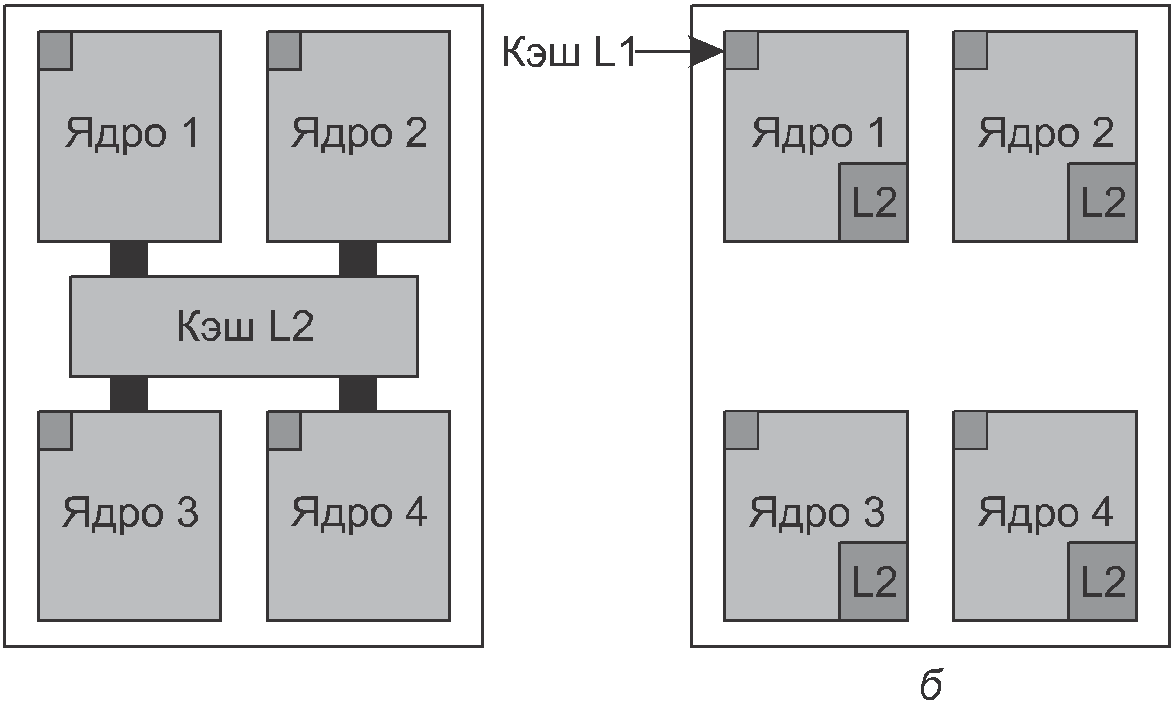
**Рис. 1.8.** Четырехъядерный процессор: *а* — с общей кэш-памятью второго уровня (L2);



*б* — с отдельными блоками кэш-памяти L2



Кстати, с точки зрения абсолютных чисел нет ничего лучше, чем современные графи- ческие процессоры (Graphics Processing Unit, GPU). На их кристаллах содержатся тысячи крохотных ядер. Они очень хорошо подходят для множества небольших про- изводимых параллельно вычислений, таких как визуализация многоугольников в гра- фических приложениях. Но для выполнения последовательных задач они не годятся. К тому же их трудно программировать. Хотя графические процессоры могут найти применение и для операционных систем (например, при кодировании или обработке сетевого трафика), не похоже, что на них могла бы запускаться основная часть самой операционной системы.

**1.3.3. Память**

Второй основной составляющей любого компьютера является память. В идеале память должна быть максимально быстрой (работать быстрее, чем производится выполнение одной инструкции, чтобы работа центрального процессора не замедлялась обраще- ниями к памяти), довольно большой и чрезвычайно дешевой. Никакая современная технология не в состоянии удовлетворить все эти требования, поэтому используется другой подход. Система памяти создается в виде иерархии уровней (рис. 1.9). Верхние уровни обладают более высоким быстродействием, меньшим объемом и более высокой удельной стоимостью хранения одного бита информации, чем нижние уровни, иногда в миллиарды и более раз.



**Рис. 1.9.** Типичная иерархия памяти. Приведенные значения весьма приблизительны

Верхний уровень состоит из внутренних регистров процессора. Они выполнены по той же технологии, что и сам процессор, и поэтому не уступают ему в быстродействии. Следовательно, к ним нет и задержек доступа. Внутренние регистры обычно предо- ставляют возможность для хранения 32 × 32 бит для 32-разрядного процессора или

64 × 64 бит — для 64-разрядного. В обоих случаях этот объем не превышает 1 Кбайт. Программы могут сами управлять регистрами (то есть решать, что в них хранить), без вмешательства аппаратуры.

Затем следует **кэш-память**, которая управляется главным образом аппаратурой. Опе- ративная память разделяется на **кэш-строки**, обычно по 64 байт, с адресами от 0 до

63 в кэш-строке 0, адресами от 64 до 127 в кэш-строке 1 и т. д. Наиболее интенсивно используемые кэш-строки оперативной памяти сохраняются в высокоскоростной кэш- памяти, находящейся внутри процессора или очень близко к нему. Когда программе нужно считать слово из памяти, аппаратура кэша проверяет, нет ли нужной строки в кэш-памяти. Если строка в ней имеется, то происходит результативное обращение к кэш-памяти (cache hit — кэш-попадание), запрос удовлетворяется за счет кэш-памяти без отправки запроса по шине к оперативной памяти. Обычно результативное обраще- ние к кэшу занимает по времени два такта. Отсутствие слова в кэш-памяти вынуждает обращаться к оперативной памяти, что приводит к существенной потере времени. Кэш-память из-за своей высокой стоимости ограничена в объеме. Некоторые маши- ны имеют два или даже три уровня кэша, причем каждый из последующих медленнее и объемнее предыдущего.

Кэширование играет существенную роль во многих областях информатики, это от- носится не только к кэшированию строк оперативной памяти. Довольно часто для повышения производительности к кэшированию прибегают везде, где есть какой-либо объемный ресурс, который можно поделить на фрагменты, часть из которых использу- ется намного интенсивнее всех остальных. Операционные системы используют кэши- рование повсеместно. Например, большинство операционных систем держат интенсив- но используемые файлы (или фрагменты файлов) в оперативной памяти, избегая их многократного считывания с диска. Точно так же результаты преобразования длинных имен файлов вроде /home/ast/projects/minix3/src/kernel/clock.c в дисковый адрес, по которому расположен файл, могут кэшироваться, чтобы исключить необходимость в повторных поисках. И наконец, может кэшироваться для дальнейшего использования результат преобразования адресов веб-страниц (URL) в сетевые адреса (IP-адреса). Можно привести массу других примеров использования технологии кэширования.

В любой системе кэширования довольно скоро возникает ряд вопросов.

1. Когда помещать в кэш новый элемент?

2. В какую область кэша помещать новый элемент?

3. Какую запись удалять из кэша, когда необходимо получить в нем свободное про- странство?

4. Куда именно в памяти большей емкости помещать только что «выселенный»

элемент?

Не каждый из этих вопросов имеет отношение к кэшированию. Например, в процессе кэширования строк оперативной памяти в кэше центрального процессора при каждом неудачном обращении к кэш-памяти в нее, как правило, будет вводиться новый эле- мент. При вычислении нужной кэш-строки для размещения нового элемента обычно используются некоторые старшие биты того адреса памяти, на который осуществляется

ссылка. Например, при наличии 4096 кэш-строк по 64 байта и 32-разрядных адресов биты с 6-го по 17-й могли бы использоваться для определения кэш-строки, а биты с 0-го по 5-й — для определения байта в кэш-строке. В этом случае элемент, подлежащий удалению, совпадает с тем элементом, в который попадают новые данные, но в других системах такой порядок может и не соблюдаться. Наконец, когда кэш-строка перепи- сывается в оперативную память (если она была изменена в процессе кэширования), место в памяти, в которое ее нужно переписать, однозначно определяется адресом, фигурирующим в запросе.

Применение кэширования оказалось настолько удачным решением, что многие со- временные процессоры имеют сразу два уровня кэш-памяти. Первый уровень, или **кэш L1**, всегда является частью самого процессора и обычно подает декодированные команды в процессорный механизм исполнения команд. У многих процессоров есть и второй кэш L1 для тех слов данных, которые используются особенно интенсивно. Обычно каждый из кэшей L1 имеет объем 16 Кбайт. Вдобавок к этому кэшу процессо- ры часто оснащаются вторым уровнем кэш-памяти, который называется **кэш L2** и со- держит несколько мегабайт недавно использованных слов памяти. Различия между кэш-памятью L1 и L2 заключаются во временной диаграмме. Доступ к кэшу первого уровня осуществляется без задержек, а доступ к кэшу второго уровня требует задержки в один или два такта.

При разработке многоядерных процессоров конструкторам приходится решать, куда поместить кэш-память. На рис. 1.8, *а* показан один кэш L2, совместно использующийся всеми ядрами. Такой подход применяется в многоядерных процессорах корпорации Intel. Для сравнения на рис. 1.8, *б* каждое ядро имеет собственную кэш-память L2. Именно такой подход применяет компания AMD. Каждый из подходов имеет свои аргументы «за» и «против». Например, общая кэш-память L2 корпорации Intel требует использования более сложного кэш-контроллера, а избранный AMD путь усложняет поддержание согласованного состояния кэш-памяти L2 разных ядер.

Следующей в иерархии, изображенной на рис. 1.9, идет оперативная память. Это глав- ная рабочая область системы памяти машины. Оперативную память часто называют оперативным запоминающим устройством (**ОЗУ**), или памятью с произвольным до- ступом (Random Access Memory (**RAM**)). Ветераны порой называют ее core memory — памятью на магнитных сердечниках, поскольку в 1950–1960-е годы в оперативной па- мяти использовались крошечные намагничиваемые ферритовые сердечники. Прошли десятилетия, но название сохраняется. В настоящее время блоки памяти имеют объем от сотен мегабайт до нескольких гигабайт, и этот объем стремительно растет. Все за- просы процессора, которые не могут быть удовлетворены кэш-памятью, направляются к оперативной памяти.

Дополнительно к оперативной памяти многие компьютеры оснащены небольшой по объему неизменяемой памятью с произвольным доступом — постоянным запоминаю- щим устройством (**ПЗУ**), оно же память, предназначенная только для чтения (Read Only Memory (**ROM**)). В отличие от ОЗУ она не утрачивает своего содержимого при отключении питания, то есть является энергонезависимой. ПЗУ программируется на предприятии-изготовителе и впоследствии не подлежит изменению. Эта разновид- ность памяти характеризуется высоким быстродействием и дешевизной. На некото- рых компьютерах в ПЗУ размещается начальный загрузчик, используемый для их запуска. Такой же памятью, предназначенной для осуществления низкоуровневого управления устройством, оснащаются некоторые контроллеры ввода-вывода.

Существуют также другие разновидности энергонезависимой памяти, которые в от- личие от ПЗУ могут стираться и перезаписываться, — электрически стираемые про- граммируемые постоянные запоминающие устройства (**ЭСППЗУ**), они же **EEPROM** (Electrically Erasable PROM), и флеш-память. Однако запись в них занимает на не- сколько порядков больше времени, чем запись в ОЗУ, поэтому они используются для тех же целей, что и ПЗУ. При этом они обладают еще одним дополнительным свой- ством — возможностью исправлять ошибки в содержащихся в них программах путем перезаписи занимаемых ими участков памяти.

Флеш-память также обычно используется как носитель информации в портативных электронных устройствах. Если упомянуть лишь два ее применения, то она служит

«пленкой» в цифровых фотоаппаратах и «диском» в переносных музыкальных пле- ерах. По быстродействию флеш-память занимает промежуточное положение между ОЗУ и диском. Также, в отличие от дисковой памяти, если флеш-память стирается или перезаписывается слишком часто, она приходит в негодность.

Еще одна разновидность памяти — CMOS-память, которая является энергозависимой. Во многих компьютерах CMOS-память используется для хранения текущих даты и вре- мени. CMOS-память и схема электронных часов, отвечающая за отсчет времени, получа- ют питание от миниатюрной батарейки (или аккумулятора), поэтому значение текущего времени исправно обновляется, даже если компьютер отсоединен от внешнего источника питания. CMOS-память также может хранить параметры конфигурации, указывающие, например, с какого диска системе следует загружаться. Потребление энергии CMOS- памятью настолько низкое, что батарейки, установленной на заводе-изготовителе, часто хватает на несколько лет работы. Однако когда батарейка начинает выходить из строя, на компьютере могут проявиться признаки «болезни Альцгеймера» и он станет «забывать» то, что помнил годами, например с какого жесткого диска следует производить загрузку.

**1.3.4. Диски**

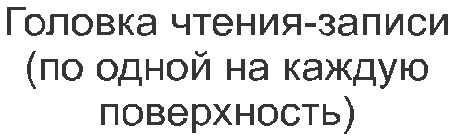
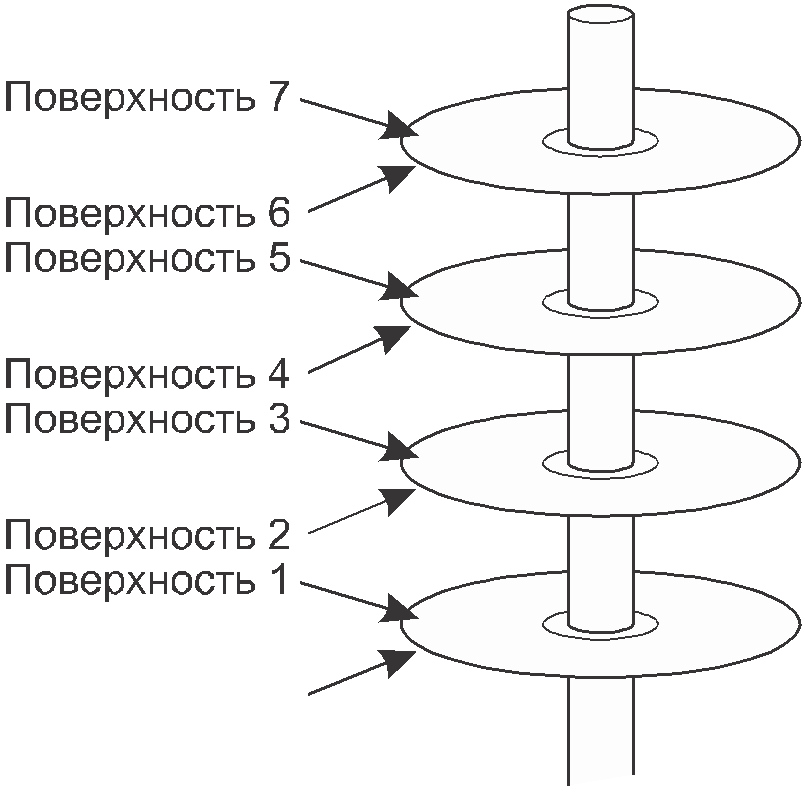
Следующим после оперативной памяти уровнем нашей иерархии памяти является маг- нитный (жесткий) диск. Дисковый накопитель в пересчете на бит информации на два порядка дешевле, чем ОЗУ, а его емкость зачастую на два порядка выше. Единственная проблема состоит в том, что время произвольного доступа к данным примерно на три порядка медленнее. Причина в том, что диск является механическим устройством, конструкция которого условно показана на рис. 1.10.

Жесткий диск состоит из одной или нескольких металлических пластин, вращающихся со скоростью 5400, 7200, 10 800 и более оборотов в минуту. Механический привод по- ворачивается на определенный угол над пластинами, подобно звукоснимателю старого проигрывателя виниловых пластинок на 33 оборота в минуту. Информация записыва- ется на диск в виде последовательности концентрических окружностей. В каждой за- данной позиции привода каждая из головок может считывать кольцеобразный участок, называемый дорожкой. Из совокупности всех дорожек в заданной позиции привода составляется цилиндр.

Каждая дорожка поделена на определенное количество секторов, обычно по 512 байт на сектор. На современных дисках внешние цилиндры содержат больше секторов, чем внутренние. Перемещение привода с одного цилиндра на другой занимает около 1 мс. Перемещение к произвольно выбранному цилиндру обычно занимает от 5 до 10 мс в зависимости от конкретного накопителя. Когда привод расположен над нужной

|  |  |
| --- | --- |
|  |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Рис. 1.10.** Схема конструкции жесткого диска



дорожкой, накопитель должен выждать, когда нужный сектор попадет под головку. Это приводит к возникновению еще одной задержки от 5 до 10 мс в зависимости от скорости вращения диска. После попадания требуемого сектора под головку произво- дится операция чтения или записи со скоростью от 50 Мбайт/с (для низкоскоростных дисков) до 160 Мбайт/с (для высокоскоростных).

Порой речь заходит о таких дисках, которые на самом деле дисками не являются, на- пример о твердотельных накопителях — SSD (Solid State Disks). У них нет движущихся частей, дисковых пластин, а данные хранятся во флеш-памяти. Они напоминают диски только тем, что содержат большой объем данных, который при отключении питания не теряется.

Многие компьютеры поддерживают схему, которая называется **виртуальной памятью**. Ее мы довольно основательно рассмотрим в главе 3. Она дает возможность запускать про- граммы, превышающие по объему физическую память компьютера, за счет помещения их на диск и использования оперативной памяти как некой разновидности кэша для наи- более интенсивно исполняемых частей. Эта схема требует прозрачного для программы преобразования адресов памяти, чтобы конвертировать адрес, сгенерированный програм- мой, в физический адрес, по которому слово размещено в ОЗУ. Такое отображение адре- сов осуществляется частью центрального процессора, называется блоком управления памятью (Memory Management Unit (**MMU**)), или **диспетчером памяти** (см. рис. 1.6).

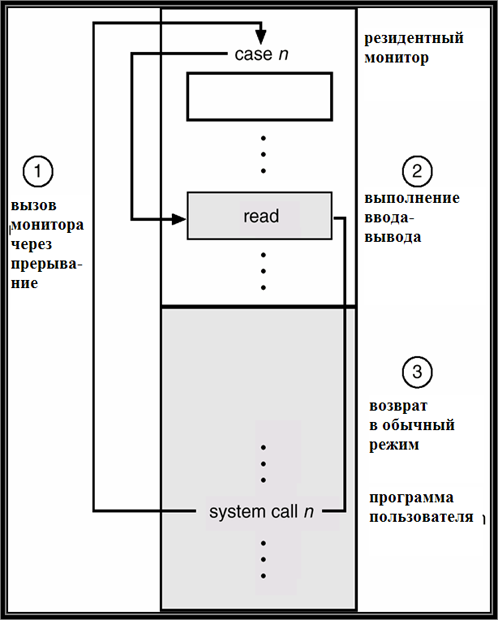
Использование кэширования и MMU может оказать существенное влияние на про- изводительность. При работе в мультипрограммном режиме, когда осуществляется переключение с одной программы на другую, иногда называемое **переключением контекста** (context switch), может потребоваться сброс всех измененных блоков из кэш-памяти и изменение регистров отображения в MMU. Обе эти операции обходятся слишком дорого, и программисты всеми силами стараются их избежать. Некоторые последствия применяемых ими тактических приемов мы рассмотрим чуть позже.

**Аппаратная защита памяти и процессора**

В целях совместного использования *системных ресурсов* (памяти, процессора, *внешних устройств*) несколькими программами, требуется, чтобы аппаратура и *операционная система* обеспечили невозможность влияния некорректно исполняемой программы на другие пользовательские программы. Для этого необходима аппаратная *поддержка*, как *минимум*, двух режимов исполнения программ – **пользовательского (** **непривилегированного** **) режима (** **user mode** **) –**для выполнения программ пользователей – и **системного (** **привилегированного** , **режима ядра - system mode** , **monitor mode** **)**- для модулей операционной системы. Идея двух режимов в том, чтобы выполняемые в привилегированном режиме модули ОС могли выполнять распределение и выделение *системных ресурсов*, в частности, формировать новые адреса, а пользовательские программы, в результате ошибок или преднамеренных атак, выполняясь в обычном режиме, не могли бы обратиться в область памяти операционной системы или другой задачи, изменять их и этим нарушать их *целостность*. Для определения текущего режима выполнения команд в аппаратуре вводится**бит режима**, равный 0 для системного и 1 – для пользовательского режима. При прерывании или сбое аппаратура автоматически переключается в системный режим. Некоторые привилегированные команды, изменяющие системные ресурсы и *состояние системы*(например, *регистр состояния* процессора), должны выполняться только в системном режиме, что защитит системные ресурсы от случайной или преднамеренной порчи при выполнении этих команд обычной пользовательской программой.

Для **защиты ввода-вывода**все команды ввода-вывода считаются привилегированными. Необходимо гарантировать, чтобы пользовательская *программа* никогда не получила управление в системном режиме и, в частности, не могла бы записать новый*адрес* в *вектор* прерываний, который, как уже отмечалось, содержит адреса подпрограмм обработки прерываний, в частности, связанных со вводом-выводом.

Использование системного вызова для выполнения ввода-вывода иллюстрируется на [рис. 4.7](http://www.intuit.ru/studies/courses/641/497/lecture/11272?page=3#image.4.7).

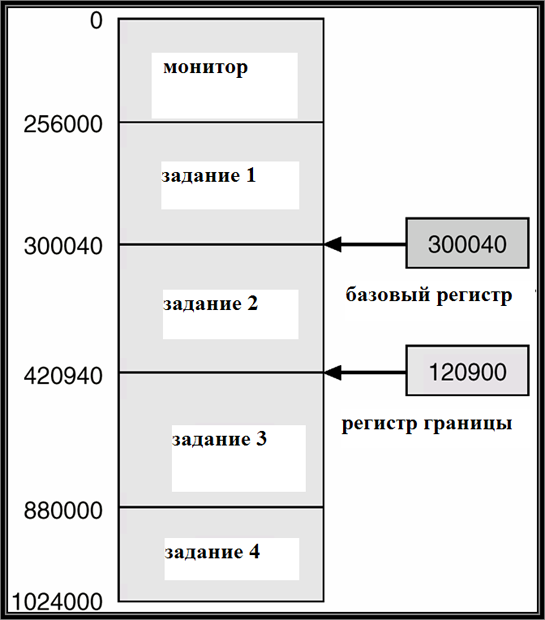


**Рис. 4.7.**Использование системного вызова для выполнения ввода-вывода.

На схеме *системный вызов* номер **n**вызывает программируемое *прерывание* (*trap*), вызывается ОС в привилегированном режиме, и *по*номеру системного вызова определяется *операция ввода-вывода*, которая должна быть выполнена *по* данному прерыванию. Затем в привилегированном режиме выполняется *операция ввода-вывода*, после чего происходит *прерывание* и возврат в пользовательскую программу, выполняемую в обычном режиме.

Для **защиты памяти**необходимо обеспечить защиту, *по* крайней мере, для вектора прерываний и подпрограмм обслуживания прерываний. Например, недопустимо разрешить пользовательской программе формировать в обычном режиме произвольный *адрес*и обращаться *по* нему, так как при этом может быть нарушена *целостность* *системных областей* памяти. Чтобы этого избежать, в аппаратуре вводятся два регистра, которые отмечают границы *допустимой области* памяти, выделенной пользовательской программе. Это **базовый регистр (base register)**,содержащий начальный *адрес* области памяти, выделенной пользовательской программе, и **регистр границы (limit register)**,содержащий размер пользовательской области памяти. *Память* вне отмеченного диапазона считается защищенной, т.е. обращения к ней из пользовательской программы не допускаются (при попытке такого обращения возникает *прерывание*).

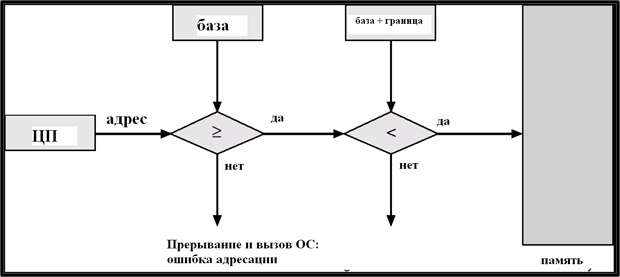
Использование базового регистра и регистра границы иллюстрируется на [рис. 4.8](http://www.intuit.ru/studies/courses/641/497/lecture/11272?page=3#image.4.8).



**Рис. 4.8.**Использование регистра базы и регистра границы для защиты памяти

На схеме заданию 2 выделена область памяти, начиная с адреса 300040 (хранящегося в регистре базы), длиной 120900 (хранящейся в регистре границы), т.е. *по* *адрес* 420939 включительно. Обращение, например, *по* адресу 420940 из программы задания 2 приводит к прерыванию как недопустимое – срабатывает *защита памяти*.

Схема аппаратной защиты адресов памяти иллюстрируется [рис. 4.9](http://www.intuit.ru/studies/courses/641/497/lecture/11272?page=3#image.4.9).

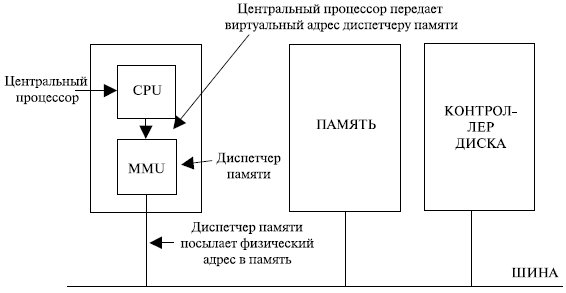
[](http://www.intuit.ru/EDI/18_03_15_2/1426630694-20867/tutorial/901/objects/4/files/4_9.png)

[увеличить изображение](http://www.intuit.ru/EDI/18_03_15_2/1426630694-20867/tutorial/901/objects/4/files/4_9.png)  
**Рис. 4.9.**Схема аппаратной защиты адресов памяти

1. **Страничная организация виртуальной памяти**

Большинство систем виртуальной памяти используют технику, называемую страничной организацией памяти [[32](http://www.intuit.ru/studies/courses/631/487/literature#literature.32), [37](http://www.intuit.ru/studies/courses/631/487/literature#literature.37)]. Любой процесс, реализуемый в компьютере, может обратиться к множеству адресов в памяти. Адреса могут формироваться с применением индексации, базовых регистров, сегментных регистров и другими путями. Эти программно формируемые адреса, называемые виртуальными адресами, формируют *виртуальное* *адресное пространство*. На компьютерах без виртуальной памяти виртуальные адреса подаются непосредственно на шину памяти и вызывают для чтения или записи *слово* в физической памяти с тем же самым адресом.

Когда используется *виртуальная память*, виртуальные адреса не передаются напрямую шиной памяти. Вместо этого они передаются диспетчеру памяти (*MMU* – *Memory Management* Unit), который отображает виртуальные адреса на физические адреса памяти, как показано на [рис. 6.9](http://www.intuit.ru/studies/courses/631/487/lecture/11057?page=3#image.6.9). Здесь *диспетчер* памяти показан как часть микросхемы процессора, как обычно и бывает чаще всего. Но логически он мог бы быть отдельной микросхемой, как было в недавнем прошлом.



**Рис. 6.9.**Диспетчер памяти

Все имеющееся в настоящее время множество реализаций виртуальной памяти различается в основном способом структуризации виртуального адресного пространства.

Сам термин "*виртуальная память*" ассоциируется с системами, использующими страничную организацию. Впервые сообщение о виртуальной памяти на основе страничной организации появилось в 1962 году в работе Kilburn I и др. "One-Level *Storage System*", и вскоре после этого *виртуальная память* стала широко применяться в коммерческих системах.

В настоящее время выделяют три метода реализации виртуальной памяти.

1. *Страничная виртуальная память* организует перемещение данных между основной памятью и диском страницами – частями виртуального адресного пространства фиксированного и сравнительно небольшого размера.
2. Сегментная виртуальная память предусматривает перемещение данных сегментами – частями виртуального адресного пространства произвольного размера, полученного с учетом смыслового значения данных.
3. Сегментно-*страничная виртуальная память* использует двухуровневое деление: виртуальное адресное пространство делится на сегменты, а затем сегменты делятся на страницы. Единицей перемещения данных является страница.

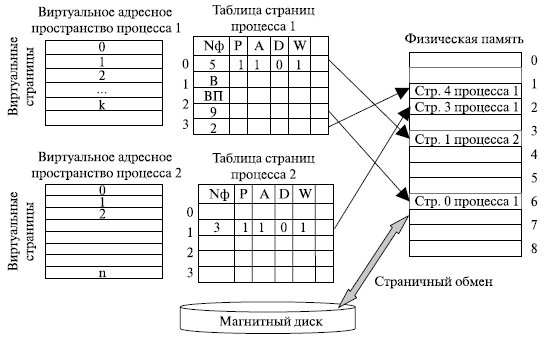
Для временного хранения сегментов и страниц на диске отводится специальная область либо специальный *файл* (страничный *файл*или *файл* подкачки – *paging file*). Текущий размер страничного файла является важным параметром, оказывающим влияние на возможности операционной системы: чем больше страничный *файл*, тем больше приложений может одновременно выполнять ОС (при фиксированном размере оперативной памяти). Однако необходимо понимать, что увеличение числа одновременно работающих приложений за счет увеличения размера страничного файла замедляет их работу, так как значительная часть времени при этом тратится на перемещение данных на *диск* и обратно.

Размер страничного файла в современных ОС является настраиваемым параметром, который выбирается администратором системы для достижения компромисса между уровнем программирования и быстродействием системы.

При страничной организации *виртуальное* *адресное пространство* каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые *виртуальными страницами* (*Virtual* pages). В общем случае размер виртуального адресного пространства не кратен размеру страницы, поэтому последняя страница дополняется фиксированной областью.

Вся оперативная *память* машины также делится на части такого же размера, называемые физическими страницами (или блоками, или кадрами). Размер страницы выбирается равным степени двойки: 1024, 2048, 4096 *байт* и т.д. Это позволяет упростить механизм преобразования адресов.

При создании процесса ОС загружает в операционную *память* несколько его *виртуальных страниц* (начальные страницы кодового сегмента и сегмента данных). Копия всего виртуального адресного пространства процесса находится на диске. Смежные виртуальные страницы не обязательно находятся в смежных физических страницах. Для каждого процесса ОС создает таблицу страниц – информационную структуру, содержащую записи обо всех виртуальных страницах процесса ([рис. 6.10](http://www.intuit.ru/studies/courses/631/487/lecture/11057?page=3#image.6.10)).



**Рис. 6.10.**Таблицы страниц виртуальной памяти

*Запись* таблицы (*дескриптор* страницы) включает следующую информацию:

1. номер физической страницы (N ф.с.), в которую загружена данная *виртуальная страница*;
2. признак присутствия Р, устанавливаемый в единицу, если данная страница находится в оперативной памяти;
3. признак модификации страницы D, который устанавливается в единицу всякий раз, когда производится запись по адресу, относящемуся к данной странице;
4. признак обращения А к странице, называемый также битом доступа, который устанавливается в единицу при каждом обращении по адресу, относящемуся к данной странице;
5. другие управляющие биты, служащие, например, для целей защиты или совместного использования памяти на уровне страниц.

Перечисленные признаки в большинстве моделей процессов устанавливаются аппаратно схемами процессора при выполнении операций с памятью. *Информация* из таблицы страниц используется для решения вопроса о необходимости перемещения той или иной страницы между памятью и диском, а также для преобразования виртуального адреса в физический. Сами таблицы страниц, так же как и описываемые ими страницы, размещаются в оперативной памяти.

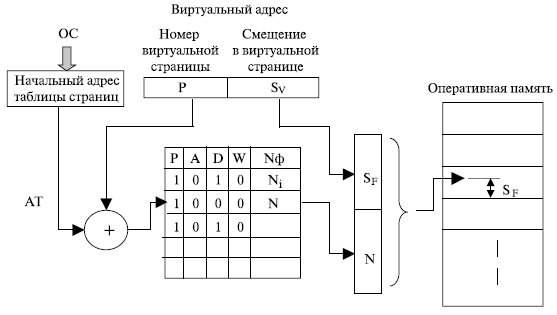
Поскольку процесс может задействовать большой объем виртуальной памяти (например, в *Windows* 2000 он равен 232 = 4 Гбайт), при использовании страницы объемом 4 Кбайт (212) потребуется 220 записей в таблице страниц для каждого процесса. Понятно, что выделять такое количество оперативной памяти под таблицы страниц нецелесообразно. Для преодоления этой проблемы большинство схем виртуальной памяти хранит таблицы страниц не в реальной, а в виртуальной памяти. Это означает, что сами таблицы страниц становятся объектами страничной организации. При работе процесса как *минимум* часть его таблицы страниц должна располагаться в основной памяти, в том числе *запись* о странице, выполняющейся в настоящий момент. *Адрес* таблицы страниц включается в *контекст процесса*. При активизации очередного процесса ОС загружает *адрес* его таблицы страниц в специальный *регистр*.

При каждом обращении к памяти выполняется *поиск* номера *виртуальной страницы*, содержащей требуемый *адрес*, затем *по* этому номеру определяется нужный элемент таблицы страниц и из него извлекается описывающая страницу *информация*. Далее анализируется признак присутствия, и если данная *виртуальная страница* находится в оперативной памяти, то выполняется преобразование виртуального адреса в физический. Если же нужная *виртуальная страница* в данный момент выгружена на *диск*, то происходит страничное *прерывание*.

Выполняющий процесс переводится в состояние ожидания, активизируя процесс из очереди процессов, находящихся в состоянии готовности. Параллельно *программа* обработки страничного прерывания находит на диске требуемую виртуальную страницу и пытается ее загрузить в оперативную *память*. Если в памяти имеется свободная *физическая страница*, то *загрузка* выполняется немедленно. Если же свободных страниц нет, то на основании принятой в данной системе *стратегии замещения* страниц решается вопрос о том, какую страницу следует выгрузить из оперативной памяти.

После того как выбрана страница, которая должна покинуть оперативную *память*, обнуляется ее *бит присутствия* и анализируется ее признак модификации. Если удаляемая страница за время последнего требования в оперативной памяти была модифицирована, то ее новая версия должна быть переписана на *диск*. Если нет, то принимается во внимание, что на диске уже имеется предыдущая копия этой *виртуальной страницы*, и никакой записи на *диск* не производится. *Физическая страница* объявляется свободной. Из соображений безопасности в некоторых системах освобождаемая страница обнуляется, чтобы невозможно было использовать содержимое выгруженной страницы. Для хранения информации о положении вытесненной страницы в страничном файле ОС может задействовать специальные поля таблицы страниц.

*Виртуальный адрес* при страничном распределении может быть представлен в виде пары ( P, Sv ), где Р – номер *виртуальной страницы* процесса (*нумерация* страниц начинается с 0), а Sv – смещение в пределах *виртуальной страницы* ([рис. 6.11](http://www.intuit.ru/studies/courses/631/487/lecture/11057?page=3#image.6.11)).*Физический адрес* также может быть представлен в виде пары ( N, Sf ), где N – номер физической страницы, а Sf – смещение а пределах физической страницы. Задача подсистемы виртуальной памяти состоит в отображении пары значений ( P, Sv ) в пару (N, Sf ).



**Рис. 6.11.**Преобразование виртуального адреса

Чтобы понять механизм реализации этого отображения, следует остановиться на двух базисных свойствах страничной организации. Как уже отмечалось, объем страницы, как виртуальной, так и физической, выбирается равным степени двойки – 2к ( k = 8 и более). Отсюда следует, что смещение Sv и Sf может быть получено отделением k младших разделов в двоичной записи виртуального и, соответственно, физического адреса страницы. При этом оставшиеся старшие *разделы* адреса представляют собой двоичную *запись* номера виртуальной и, соответственно, физической страницы. Дополнив эти номера к нулям, можно получить начальный *адрес* виртуальной и физической страниц.

Второе свойство – линейность адресного пространства виртуальной и физической страницы – приводит к тому, что Sf = Sv. Отсюда следует простая схема преобразования виртуального адреса в физический.

При обращении к памяти *по* некоторому виртуальному адресу ( P, Sv ) аппаратные схемы процессора выполняют следующие действия.

1. Из специального регистра процессора извлекается начальный адрес АТ таблицы страниц активного процесса. С помощью сумматора \sum по значениям АТ, Р, L (длина отдельной записи в таблице страниц) определяется адрес нужной записи в таблице страниц:

A = АТ + (Р \* L).

1. Считывается номер соответствующей физической страницы – N.
2. К номеру физической страницы присоединяется смещение Sv.

В итоге полученный *физический адрес* оперативной памяти представляется парой значений ( N, Sf ).

Рассмотрим пример, поясняющий основные характеристики организации страничной виртуальной памяти. Пусть *компьютер* имеет оперативную *память* объемом Еоп = 256 Мбайт, размер страницы выбран равным Естр= 4 Кбайт. В этом случае количество физических страниц равно

Nf = Еоп / Естр = 256\*2020 / 4\*210 = 64.000 страниц.

Для отображения физического адреса произвольного байта оперативной памяти потребуется K = log2 256\*2020 = 28 двоичных разрядов.

Число разрядов для отображения смещения в странице M = log2 4 Кбайт = log2 4096 = 12.

Если *процессор* имеет 32-разрядную структуру, то на номер *виртуальной страницы* отводится 32-12=20 двоичных разрядов. Таким образом, число *виртуальных страниц* равно Nв = 220 (примерно 1 млн *виртуальных страниц*).

Для каждой *виртуальной страницы* в таблице страниц должна быть *запись*, содержащая номер *виртуальной страницы* (20 двоичных разрядов), начальный *адрес* соответствующей ей физической страницы плюс дополнительные разряды, характеризующие свойства страницы (присутствие, модификация, обращение и т.п.), на которые потребуется 1 *байт*. Поскольку *адрес* начала физической страницы кратен 4096, то на него достаточно 28 – 12 = 16 двоичных разрядов (остальные 12 разрядов заполняются нулями). Таким образом, одна *запись* таблицы страниц займет 20 + 16 + 8 = 44 двоичных разрядов или 6 *байт*. Общий объем таблицы страниц составит 6 \* Nв = 6 Мбайт.

Реально при выборе структуры записи таблицы страниц нужно учитывать следующие факторы. Современные компьютеры позволяют наращивать объем оперативной памяти (например, в ПК она может почти достигать объема виртуальной памяти и даже более). Поэтому на *адрес* физической страницы в нашем примере следует выделить 32-12 = 20 двоичных разрядов. С другой стороны, нет необходимости в записи (дескрипторе) *виртуальной страницы* иметь *поле* с номером *виртуальной страницы* (20 разрядов), так как*адрес* нужной записи можно вычислять, как это было рассмотрено выше. Следовательно, в нашем примере *длина* записи должна быть равной 32 - 12 + 8 = 28 двоичным разрядам, т.е. с округлением до целого числа *байт* – 4 *байт*. Таким образом, для каждого выполняющегося в компьютере процесса ОС должна создать страничную таблицу размером 4 \* Nв байт = 5 \* 220 = 4 Мбайт.

Процедура преобразования виртуального адреса в физический без принятия специальных мер (*кэширование* активных страниц) занимает один цикл оперативной памяти, который затрачивается на считывание номера физической страницы из таблицы страниц. Поэтому любое обращение к ОП будет занимать 2 *цикла* вместо одного при работе без виртуальной памяти. Другим фактором, влияющим на *производительность* систем, являются *затраты* времени на обработку страничных прерываний. При неправильно выбранной *стратегии замещения* страниц может возникнуть ситуация, когда система тратит большую часть времени впустую на подкачку страниц из оперативной памяти на *диск* и обратно.