1 *Архитектурные принципы фон Неймана. Четыре принципа и сущность концепции вычислительной машины фон Неймана.*

Термин «электронная вычислительная машина – ЭВМ» определяется как совокупности технических средств, служащих для автоматизированной обработки дискретных данных по заданному алгоритму.

В основе архитектуры современных ЭВМ лежит представление алгоритма решения задачи в виде программы последовательных вычислений. Согласно стандарту ISO 2382/1-84, программа для ЭВМ – это «упорядоченная последовательность команд, подлежащая обработке».

ЭВМ, в которой определенным образом закодированные команды программы хранятся в памяти, известна под названием *вычислительной машины (ВМ) с хранимой в памяти программой*, а сама концепция вычислительной машины – *концепции хранимой в памяти программы (концепция машины с хранимой в памяти программой)*.

Сущность фон-неймановской концепции вычислительной машины можно свести к четырем принципам:

1. двоичного кодирования;
2. программного управления;
3. адресности;
4. однородности памяти.

**Принцип двоичного кодирования**

Вся информация в вычислительной машине (как данные, так и команды) кодируется в двоичной форме и разделяется на единицы (элементы) информации, которые называются *словами*. Группа двоичных разрядов, составляющих слово, обрабатываются одновременно. Каждый тип информации имеет свой *формат*. Например, в формате данных с фиксированной запятой обычно выделяют *поле знака* и *поле значащих разрядов*.

**Принцип программного управления**

Порядок выполнения действий вычислительной машиной задается алгоритмом. Все действия, предусмотренные алгоритмом решения задачи, представляются в виде *программы*, состоящей из последовательности управляющих слов – *машинных команд*. Каждая команда предписывает некоторую элементарную операцию из набора операций, реализуемых вычислительной машиной, с помощью которых осуществляется обработка данных. Команды программы хранятся в последовательных ячейках памяти вычислительной машины и выполняются в *естественной последовательности*, т. е. в порядке их положения в программе. При необходимости, с помощью специальных команд, эта последовательность может быть изменена. Решение об изменении порядка выполнения команд программы принимается либо на основании анализа результатов предшествующих действий, либо безусловно.

В формате команды выделяются две части: *операционная часть* и *адресная часть*. В операционной части задается *код операции*. Код операции представляет собой указание, какая операция должна быть выполнена. Адресная часть содержит адреса объектов обработки (*операндов*) и результата.

**Принцип адресности**

Структурно память вычислительной машины состоит из пронумерованных ячеек, причем процессору в произвольный момент доступна любая ячейка. Слова информации (двоичные коды команд и данных) хранятся в ячейках памяти, а для доступа к ним используются номера соответствующих ячеек, которые называются *адресами* слов.

**Принцип однородности памяти**

Команды и данные хранятся в одной и той же памяти и внешне в памяти неразличимы, распознать их можно только по способу использования, т.е. разнотипные слова информации различаются по способу использования, но не способом кодирования. Отсутствует явное различие между командами и данными. Их идентифицируют неявным способом при выполнении операций. Так, объект, адресуемый командой перехода, определяется как команда, а операнды, с которыми имеет дело команда, определяются как данные. В свою очередь, назначение (тип) данных не является их неотъемлемой составной частью, т.е. ЭВМ безразлична к целевому назначению данных – ей все равно, какую логическую нагрузку несут обрабатываемые данные. Нет, например, никаких средств, позволяющих явно отличить набор бит, представляющих число с плавающей запятой, от набора бит, являющихся строкой символов. Назначение данных определяется логикой программы. Если машина извлекает из памяти команду сложения чисел с плавающей запятой, то предполагается, что операнды – числа с плавающей запятой, и над операндами выполняется сложение согласно правилам арифметики чисел с плавающей запятой.

## 2 *Структура фон-неймановской вычислительной машины, пять функциональных блоков и их взаимодействие. Аппаратные и программные средства.*

Фон-неймановская ВМ включает пять функциональных блоков (рис. 1): устройство ввода, память, АЛУ, устройство управления и устройство вывода.

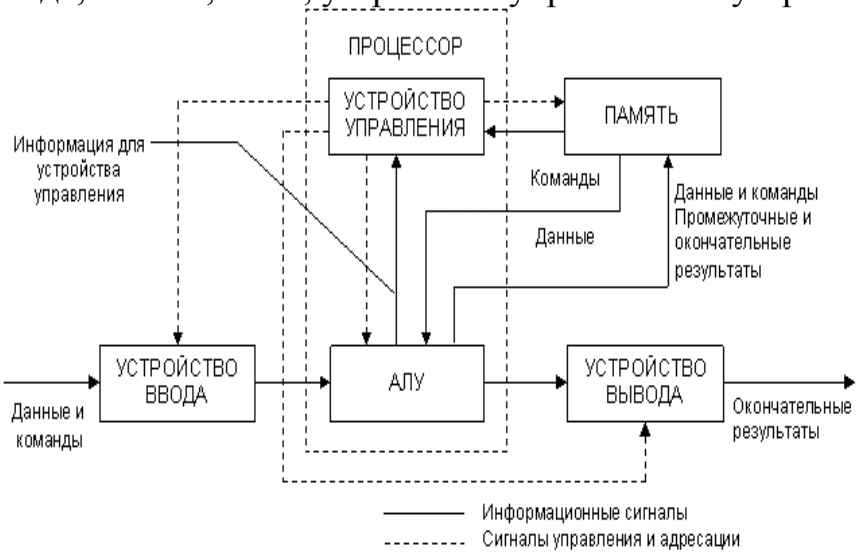


**Рисунок 1 - Структура фон-неймановской вычислительной машины**

В любой ВМ имеются средства для ввода программ и данных. Информация поступает из подсоединенных к ВМ *периферийных устройств* (ПУ) ввода. Затем отдельные команды программы одна за другой автоматически поступают в *устройство управления* (УУ), которое их декодирует и управляет выполнением операции, заданной в команде. Операции обычно выполняются в *арифметико-логическом устройстве* (АЛУ), содержащем все необходимые для обработки данных схемы. При этом данные должны поступить в АЛУ из памяти. Результаты вычислений выводятся на периферийные устройства вывода. Связь и взаимодействие ВМ и ПУ обеспечивают *порты ввода* и *порты вывода*. Термином порт обозначают аппаратуру сопряжения периферийного устройства с ВМ. Совокупность портов ввода и вывода называют *устройством ввода/вывода* (УВВ). АЛУ и устройство управления вместе образуют *центральное процессорное устройство* (ЦПУ), которое обычно называют *центральным процессором* (ЦП) или просто *процессором*.

Чтобы программа могла выполняться, команды и данные должны располагаться в *основной памяти* (ОП), организованной таким образом, что каждое двоичное слово хранится в отдельной ячейке, идентифицируемой адресом, причем соседние ячейки памяти имеют следующие по порядку адреса. Доступ к любым ячейкам *запоминающего устройства* (ЗУ) основной памяти может производиться в произвольной последовательности. Такой вид памяти известен как *память с произвольным доступом*. ОП современных ВМ в основном состоит из полупроводниковых *оперативные запоминающих устройств* (ОЗУ), обеспечивающих как считывание, так и запись информации. Для таких ЗУ характерна энергозависимость – хранимая информация теряется при отключении электропитания. Если необходимо, чтобы часть основной памяти была энергонезависимой, в состав ОП включают *постоянные запоминающие устройства* (ПЗУ), также обеспечивающие произвольный доступ. Хранящаяся в ПЗУ информация в рабочем режиме ВМ может только считываться (но не записываться).

В вычислительной машине может быть *дополнительная* память, известная как *вторичная*. Введенная информация всегда сначала запоминается в основной памяти, а затем может переноситься во вторичную память для длительного хранения. Вторичная память энергонезависима и чаще всего реализуется на базе магнитных дисков. Информация в ней хранится в виде специальных программно поддерживаемых объектов – файлов.

Обработка данных осуществляется главным образом в АЛУ. Встроенные операции, как правило, элементарны: функции АЛУ обычно сводятся к простым арифметическим и логическим операциям, а также операциям сдвига. АЛУ обеспечивает обработку двух входных переменных, в результате которой формируется выходная переменная. Более сложные математические действия должны выполняться с помощью программ, использующих встроенные операции. АЛУ содержит набор регистров, используемых для промежуточного хранения информации в процессе ее обработки и называемых *рабочими* или *регистрами*

*общего назначения* (РОН). Помимо результата операции АЛУ формирует ряд *признаков результата* (*флагов*), характеризующих полученный результат и события, произошедшие в процессе его получения (равенство нулю, знак, четность, перенос, переполнение и т. д.). Флаги могут анализироваться в УУ с целью принятия решения о дальнейшей последовательности выполнения команд программы.

Устройство управления управляет работой ВМ, организуя автоматическое выполнение программ и обеспечивая функционирование ВМ как единой системы. УУ автоматически, последовательно по одной, выбирает команды из памяти, декодирует каждую из них и генерирует необходимые для ее выполнения сигналы. Для того чтобы получить команду из памяти, УУ прежде всего должно знать ее адрес. Обычно команды выбираются из последовательных ячеек памяти, и их адреса указываются специальным *программным счетчиком* (program counter), находящимся в устройстве управления. После выборки текущей команды содержимое программного счетчика автоматически увеличивается с тем, чтобы указывать на следующую по порядку команду. Далее, чтобы иметь возможность декодировать и выполнить текущую команду, она помещается в *регистр команд* (instruction register), который находится в УУ. Код операции однозначно определяет операцию, выполняемую в процессе интерпретации команды. Адресная часть команды (если она присутствует) указывает на ячейки памяти или РОН, к которым нужно обращаться, выполняя команду (например, считывать операнды и записывать результат). Кроме того, УУ синхронизирует работу отдельных блоков ВМ. Эта функция осуществляется с помощью *генератора тактовых импульсов* (ГТИ), или *тактового генератора*.

Таким образом, функционирование ВМ сводится к выполнению последовательности команд программы.

## 3 *Понятия организации и архитектуры. Уровни организации вычислительной машины.*

Под *организацией* понимают состав компонентов (аппаратных или программных средств), связи между ними и их функциональные характеристики.

ВМ имеет многоуровневую иерархическую организацию со своими составными компонентами на каждом уровне:

1. нижний уровень – уровень физических компонентов – *физическая организация* (представляется в виде принципиальной схемы);
2. уровень реализуемых в ВМ функций – *логическая* (*функциональная*) *организация* (представляется в виде функциональной схемы);
3. верхний уровень – уровень аппаратуры (состав, функциональные связи и характеристики аппаратных модулей) – *структурная организация* (представляется в виде структурной схемы).

Определение термина «архитектура компьютера» дал в 1970 г. С.С.Хассон (S.S.Husson): это «характеристики вычислительной системы с точки зрения программиста». Первоначально под термином *архитектура вычислительной машины* подразумевалось описание структуры данных и регистров, необходимое

для уяснения системы команд ВМ и интерпретации этих команд. Иначе говоря, этим понятием охватывались те минимальные знания, которые могли понадобиться программисту для составления программы на машинном языке:

1. программно-доступные регистры (программистская модель) ВМ;
2. форма представления команд для ВМ (коды операций и форматы машинных команд) и правила их интерпретации этой машиной;
3. способы адресации данных в этих командах;
4. форма представления данных (типы и форматы данных).

Архитектура, понимаемая в этом смысле, называется *архитектурой набора команд*. Архитектура набора команд служит границей между аппаратурой и программным обеспечением и представляет ту часть ВМ, которая видна программисту, разрабатывающему программы на машинно-ориентированном языке. Следует отметить, что это наиболее частое употребление этого термина.

В *широком смысле* под архитектурой понимают структурную организацию ВМ в виде совокупности функциональных модулей и определенных связей между ними.

Под *микроархитектурой* понимается структурная организация процессора (микропроцессора), т.е. внутренняя реализация архитектуры набора команд процессора.

## 4 *Основные типы архитектур вычислительных машин. Принстонская и гарвардская архитектуры. Организация пространств памяти и ввода/вывода.*

Фон-неймановская (принстонская) и гарвардская архитектуры

Эти архитектурные варианты были предложены в конце 1940-х годов специалистами соответственно Принстонского и Гарвардского университетов США для разрабатываемых ими моделей компьютеров.

В большинстве современных ВМ для хранения программ и данных используется общая память. Для обращения к этой памяти используется общая системная шина, по которой в процессор поступают и команды, и данные. Такая организация получила название *архитектуры фон Неймана* или *Принстонской архитектуры*, а ВМ с такой архитектурой называют *машинами фон-неймановского* или *принстонского типа*. В них области для хранения программ (Program Space – PS) и данных (Data Space – DS) образуют единое пространство и могут размещаться в любом месте общей памяти. При этом нет никаких признаков, указывающих на тип информации в ячейке памяти. Содержимое ячейки интерпретируется ЦП, и задача программиста – следить за тем, чтобы данные и программа обрабатывались по-разному.

Эта архитектура имеет ряд важных достоинств. Наличие общей памяти позволяет оперативно перераспределять ее объем для хранения отдельных массивов команд и данных в зависимости от решаемых задач. Таким образом, обеспечивается возможность более эффективного использования имеющегося

объема основной памяти в каждом конкретном случае применения микропроцессора. Использование общей шины для передачи команд и данных значительно упрощает отладку, тестирование и текущий контроль функционирования системы, повышает ее надежность.

*Гарвардская архитектура* характеризуется физическим разделением памяти команд (программ) и памяти данных. В ее классическом варианте программы и данные хранятся в двух раздельных памятях, каждая из которых соединяется с процессором отдельной шиной, что позволяет полностью совмещать во времени выборку и исполнение команд, т.е. одновременно с чтением-записью данных при выполнении текущей команды производить выборку и декодирование следующей команды. ЭВМ, спроектированные в соответствии с концепцией разделения памяти на два вида, называют *машинами гарвардского типа*. Такое разделение позволяет повысить быстродействие и упростить схемотехническую реализацию микропроцессорной системы.

Недостатки Гарвардской архитектуры связаны с необходимостью использования большего числа шин, а также с фиксированным объемом памяти, выделенной для команд и данных, назначение которой не может оперативно перераспределяться в соответствии с требованиями решаемой задачи. Поэтому приходится использовать память большего объема, коэффициент использования которой при решении различных задач оказывается более низким, чем в системах с Принстонской архитектурой. Однако развитие микроэлектронной технологии позволяет в значительной степени преодолеть указанные недостатки.

Дальнейшее совершенствование архитектур обоих типов состояло в выделении специального пространства данных небольшого объема, которое представляет собой набор *программно-доступных регистров* (Register Space). В отличие от памяти и портов ввода/вывода регистры располагаются всегда внутри процессора вместе с АЛУ, что обеспечивает быстрый физический доступ к информации, хранящейся в них. В некоторые интервалы времени программа наиболее интенсивно работает лишь с небольшим объемом данных. Для временного хранения этих данных и предназначена *регистровая область* – набор программно доступных регистров. Регистровая область может быть как полностью изолирована от пространства данных DS, так и частично пересекаться с ним, что дает возможность рассматривать отдельные регистры процессора как обычные ячейки памяти данных. Такая организация является целесообразной, если в процессоре поддерживается быстрый доступ ко всей или хотя бы к некоторой части памяти данных.

Пространство ввода/вывода представляет набор адресуемых буферных схем и регистров, которые называются *портами* и через которые осуществляется связь с внешними и внутренними аппаратными средствами ВМ. В вычислительной машине может использоваться два варианта организации доступа к портам ввода/вывода: изолированный и совмещенный ввод/вывод.

При *изолированном* вводе/выводе порты размещены в специальном пространстве ввода/вывода (Input/Output Space – IOS), изолированном от других пространств данных. В этом случае процессор имеет специальный *набор команд ввода/вывода*.

При *совмещенном* вводе/выводе или вводе/выводе *с отображением на память* изолированное пространство ввода/вывода отсутствует, а в пространстве памяти данных DS выделяются области, в которых размещаются порты. Организация доступа к портам в такой ВМ ничем не отличается от процесса обращения к данным в памяти.

На рис. 2 представлены четыре типовых набора областей для хранения программ и данных. Стрелками показан процесс изоляции отдельных областей, приводящий к появлению нового типового набора. Все наборы существуют реально, каждый имеет свои преимущества и недостатки, учет которых позволяет создавать высокоэффективные системы различного назначения.

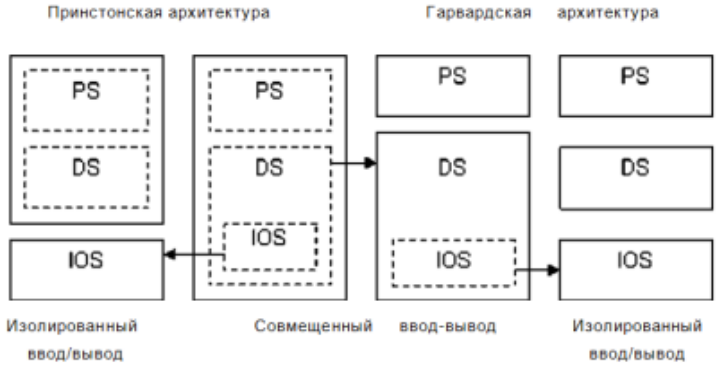



**Рисунок 2 – Типовые наборы областей для хранения программ и данных**

Организация пространств памяти и ввода/вывода

Память представляет собой линейно упорядоченный набор n-разрядных ячеек с произвольным доступом (одномерный массив) – *линейная память*. Все ячейки пронумерованы, таким образом каждой ячейке набора соответствует число, называемое ее *адресом*. Все адреса занимают целочисленный диапазон от 0 до $2^{m-1}$ (m – разрядность адреса), который образует *адресное пространство* памяти. В большинстве случаев процессор может адресоваться к памяти с точностью до одного байта, т.е. наименьшей адресуемой единицей является байт и память имеет *байтовую организацию*.

## 5 *Структура микропроцессорной системы. Магистрально-модульный принцип организации микропроцессорной системы.*

Микропроцессорная система состоит из микропроцессорного модуля, подсистем памяти и ввода/вывода. Эти составные части МПС соединяются между собой множеством сигнальных проводов, называемым *шиной*. При этом МПС можно представить в виде собственно вычислительной машины и периферийных (внешних) устройств, которые разделяются на устройства ввода/вывода и внешнюю память. Эти устройства подсоединяются к шине с помощью соответствующих интерфейсов (рис. 3).


**Рисунок 3 – Структура микропроцессорной системы**

В данном курсе под МПС понимается сама вычислительная машина.

*Микропроцессорная система* – цифровое устройство или цифровая система (система обработки данных, контроля и управления), построенная на базе одного или нескольких микропроцессоров. *Программно-аппаратный принцип* построения МПС – один из основных принципов их организации. Этот принцип заключается в том, что *реализация целевого назначения МПС достигается не только аппаратными средствами, но и с помощью программного обеспечения*.

Магистрально-модульный принцип организации микропроцессорной системы

Второй принцип, лежащий в основе организации микропроцессорных систем, – *магистрально-модульный принцип*. В соответствии с этим принципом основные структурные компоненты МПС выполняются в виде отдельных функционально законченных модулей, которые подключаются к единой внутрисистемной магистрали (шине).

## 6 *Основные классы микропроцессорных средств. Понятия микропроцессор, микропроцессорная схема, микроЭВМ, микропроцессорный комплект, однокристальная микроЭВМ, микроконтроллер, одноплатная микроЭВМ и микропроцессорные средства.*

Микропроцессорный модуль включает *микропроцессор*, *генератор тактовых импульсов* (ГТИ), а также, возможно, и другие устройства, например, *таймер*, *контроллер прерываний* и т.п.

В *подсистеме памяти* выделяют *модули ОЗУ*, предназначенных для хранения переменных и загружаемого извне программ, и *модули ПЗУ*, которые используются для хранения программ и констант.

В составе *подсистемы ввода/вывода* в простейшем случае выделяются адресуемые процессором буферные схемы и регистры – *порты ввода/вывода*. Они предназначены для связи с простыми внешними устройствами, такими как

светодиодные индикаторы, переключатели и т.п. Более сложные модули подсистемы ввода/вывода, предназначенные для управления внешним интерфейсным оборудованием и реализации специальных функций ввода/вывода, строятся на основе портов ввод/вывод и называются *адаптерами* или *контроллерами периферийных устройств*. Наиболее сложными из модулей подсистемы ввода/вывода являются *процессоры* (*сопроцессоры*) ввода/вывода, которые работают по собственным программам, хранящимся в памяти, и по сути дела представляют собой отдельные микропроцессорные системы.

Основные классы микропроцессорных средств

*Микропроцессор* (МП) – функционально законченное программно-управляемое устройство, предназначенное для обработки данных и управления процессом этой обработки, выполненное в виде одной или нескольких микросхем. С точки зрения организации микропроцессорной системы – это ЦП, реализованный в виде одной или нескольких микросхем.

*Микропроцессорная схема* (МП схема) – интегральная микросхема (ИМС), выполняющая функцию МП или его части. По существу – это интегральная схема с процессорной организацией, разработанная для построения микропроцессорных систем. Микропроцессорные схемы относятся к особому классу микросхем, одной из особенностей которых является возможность программного управления ими с помощью определенного набора команд.

*МикроЭВМ* – ЭВМ, содержащая одну или несколько МП схем, микросхемы памяти (ПЗУ и ОЗУ), интерфейсов периферийных устройств, а также некоторые другие схемы (рис. 4).


**Рисунок 4 – Структура микроЭВМ**

*Микропроцессорный комплект* (*семейство, набор*) (МПК) – совокупность МП и других микросхем, совместимых по конструктивно-технологическому исполнению и предназначенных для совместного применения при построении МП, микроЭВМ и микропроцессорных систем.

*Однокристальная микроЭВМ* – микроЭВМ, выполненная в виде одной интегральные схемы. Однокристальные микроЭВМ широко используются для управления различной аппаратурой и оборудованием, например в бытовых приборах.

*Микроконтроллер* – однокристальная микроЭВМ с небольшими вычислительными ресурсами и упрощенной системой команд, ориентированная

на выполнение процедур логического управления различным оборудованием (а не на производство вычислений). Особенностью микроконтроллеров является расширенная реализация периферийных средств на кристалле.

*Одноплатная микроЭВМ* – микроЭВМ, выполненная в виде одной печатной платы и предназначенная для встраивания в различную аппаратуру. В отличие от однокристальной ЭВМ, размещая на одной плате несколько микросхем, можно получить микроЭВМ с достаточно большими вычислительными ресурсами.

*Микропроцессорные средства* – МПК, однокристальные (включая микроконтроллеры) и одноплатные микроЭВМ

## 7 *Типовые структуры микропроцессорных систем. Магистральная, магистрально-каскадная и магистрально-радиальная структуры*

Магистральная, магистрально-каскадная и магистрально-радиальная структуры

В зависимости от способа подключения отдельных модулей микропроцессорной системы к системной магистрали различают три типовые структуры микропроцессорных систем:

1. магистральная;
2. магистрально-каскадная;
3. магистрально-радиальная.

В *магистральной* структуре (рис. 5) все модули подсистем памяти и ввода/вывода подключаются непосредственно к системной магистрали.

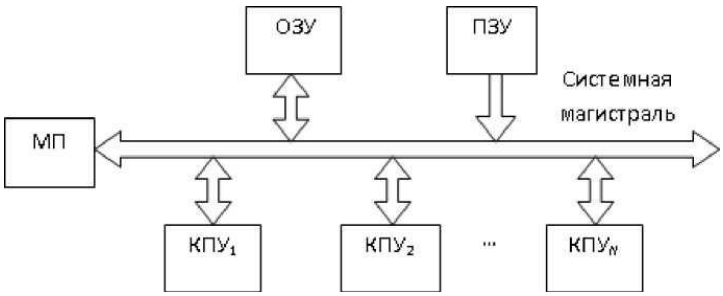



**Рисунок 5 – Магистральная структура**

Это наиболее простая структура. Недостатками магистральной структуры являются:

• все модули должны поддерживать протокол обмена по системной магистрали и содержать средства сопряжения с ней, которые в зависимости от микропроцессора могут быть достаточно сложными;

• небольшое быстродействие, так как медленные периферийные устройства могут надолго занимать системную магистраль.

В *магистрально-каскадной* и *магистрально-радиальной* структурах отдельные модули подключаются с помощью специальных *контроллеров*

(*адаптеров*) *шин*, основное назначение которых – реализовать приоритетные соотношения при использовании магистрали.

В *магистрально-каскадной* структуре (рис. 6) отдельные модули подключаются к контроллеру шины с помощью дополнительного общего канала, например, магистрали или шины ввода/вывода, т.е. по *магистральной схеме*.

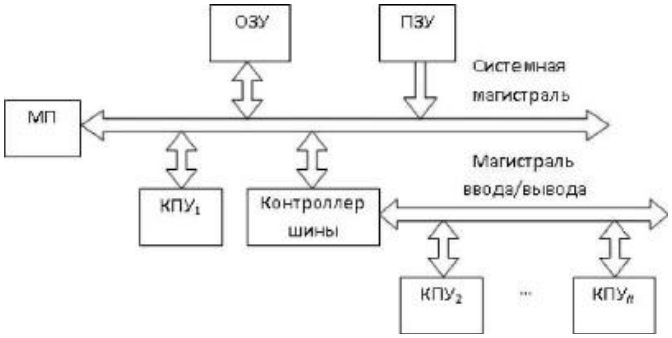



**Рисунок 6 – Магистральная-каскадная структура**

В *магистрально-радиальной* структуре (рис. 7) каждый модуль подключается к контроллеру шины с помощью индивидуального канала, т. е. по *радиальной схеме*.

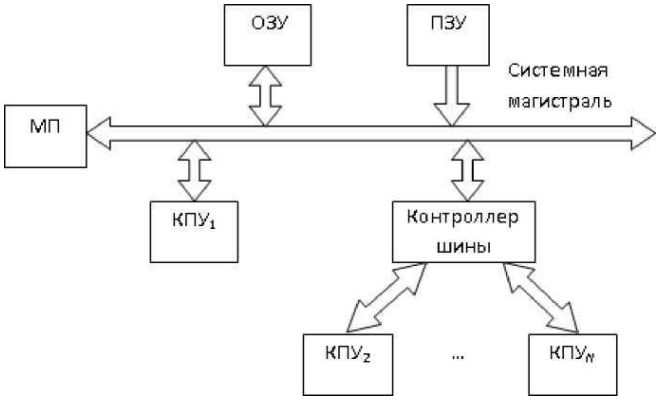



**Рисунок 7 – Магистральная-радиальная структура**

## 8 *Иерархия шин. Структуры с одной, двумя и тремя видами шин.*

В микропроцессорной системе системная магистраль служит единственным трактом для потоков команд, данных и управления. Наличие общей шины существенно упрощает реализацию МПС, позволяет легко менять ее состав и конфигурацию. Вместе с тем, именно с шиной связан и основной недостаток такой архитектуры: в каждый момент передавать информацию по шине может только одно устройство. Основную нагрузку на шину создают обмены между процессором и памятью, связанные с извлечением из памяти команд и данных и записью в память результатов вычислений. На операции ввода/вывода остается лишь часть пропускной способности шины. Практика показывает, что даже при достаточно быстрой шине для 90% приложений этих

остаточных ресурсов обычно не хватает, особенно в случае ввода или вывода больших массивов данных.

Поэтому при сохранении фон-неймановской концепции последовательного выполнения команд программы шинная архитектура в чистом ее виде оказывается недостаточно эффективной. Более распространена *архитектура с иерархией шин*, где помимо системной шины имеется еще несколько дополнительных шин. Они могут обеспечивать непосредственную связь между устройствами с наиболее интенсивным обменом, например процессором и кэш-памятью. Другой вариант использования дополнительных шин – объединение однотипных устройств ввода/вывода с последующим выходом с дополнительной шины на системную. Это позволяет снизить нагрузку на общую шину и более эффективно расходовать ее пропускную способность. Наибольшее распространение получили микропроцессорные системы с одной шиной, с двумя или тремя видами шин.

В структурах с одной шиной имеется одна системная шина, обеспечивающая обмен информацией между процессором и памятью, а также между устройствами ввода/вывода, с одной стороны, и процессором либо памятью – с другой (рис. 8).


**Рисунок 8 – МПС с одной шиной**

Для такого подхода характерны простота и низкая стоимость. Однако одношинная организация не в состоянии обеспечить высокую скорость обмена, причем узким местом является именно шина.

В МПС с двумя видами шин устройства ввода/вывода подключаются к *шинам ввода/вывода*, которые берут на себя основной обмен, не связанный с выходом на процессор или память (рис. 9). Подключение осуществляется с помощью *адаптеров шин*, которые обеспечивают буферизацию данных при их пересылке между системной шиной и контроллерами устройств ввода/вывода. Это позволяет микропроцессорной системе поддерживать работу множества устройств ввода/вывода и одновременно развязать обмен информацией по тракту процессор-память и обмен информацией с устройствами ввода/вывода. Подобная схема существенно снижает нагрузку на скоростную шину процессор-память и способствует повышению общей производительности микропроцессорной системы.


**Рисунок 9 – МПС с двумя видами шин**

*Шина процессор-память* обеспечивает непосредственную связь между процессором и основной памятью. В современных микропроцессорах такую шину часто называют *шиной переднего плана* (передней или первичной) и обозначают аббревиатурой FSB (Front-Side Bus). Интенсивный обмен между процессором и памятью требует, чтобы полоса пропускания шины, т.е. количество информации, проходящей по шине в единицу времени, была наибольшей. В варианте с одной шиной роль этой шины выполняет системная шина, однако в плане эффективности значительно выгоднее, если обмен между процессором и основной памятью ведется по отдельной шине. К рассматриваемому виду можно отнести также шину, связывающую процессор с кэш-памятью второго уровня, известную как *шина заднего плана* (тыльная или вторичная) – BSB (Back-Side Bus). BSB позволяет вести обмен с большей скоростью, чем FSB, и полностью реализовать возможности более скоростной кэш-памяти. Архитектура с использованием шин FSB и BSB известна как *архитектура двойной независимой шины* DIB (Dual Independent Bus). Наличие двух шин позволяет одновременно обращаться к основной памяти и кэш-памятью второго уровня, тем самым увеличивая общую производительность системы.

Для подключения быстродействующих периферийных устройств в систему шин может быть добавлена высокоскоростная шина расширения (рис. 10).

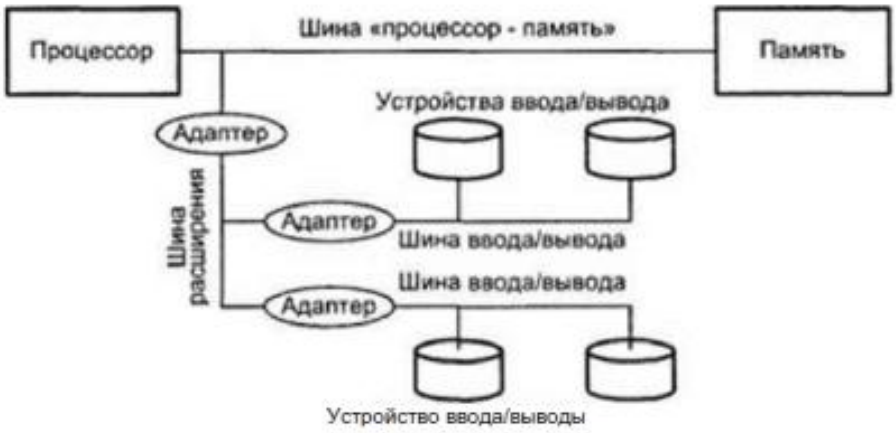



**Рисунок 10 – МПС с тремя видами шин**

Шины ввода/вывода подключаются к шине расширения, а уже с нее через адаптер к шине процессор-память. Схема еще более снижает нагрузку на шину процессор-память. Такую организацию шин называют *архитектурой с «пристройкой»* (mezzanine architecture).

## 9 *Характеристики микропроцессоров. Универсальность, производительность и технологичность. Качества и параметры.*

Архитектура МП ориентирована на достижение универсальности применения, высокой производительности и технологичности.

*Универсальность* (возможность разнообразного применения) МП определяется их широкими функциональными возможностями и обеспечивается:

• программным управлением, позволяющим производить программную настройку МП на выполнение определенных функций;

• гибкой системой команд и разнообразием способов адресации;

• магистрально-модульным принципом построения;

• специальными аппаратно-логическими средствами: регистровая память, виртуальная память, многоуровневая система прерываний, прямой доступ к памяти и т.п.

Относительно *высокая производительность* МП достигается использованием для их построения быстродействующих микросхем и специальных архитектурных решений, таких как регистровая память, кэш-память, конвейерная обработка, суперскалярная архитектура, предсказание переходов, механизм динамического выполнения команд и т.п.

*Технологичность* микропроцессорных средств обеспечивается модульным принципом конструирования, который предполагает реализацию этих средств в виде набора функционально законченных микросхем, просто объединяемых в соответствующие устройства и системы.

Кроме перечисленных выше трех основных особенностей, исключительно широкое применение МП в различных цифровых устройствах и системах обеспечивается также:

• низкой стоимостью;

• небольшими размерами;

• малой мощностью потребления;

• высокой надежностью и большой устойчивостью к неблагоприятным внешним воздействиям.

МП характеризуется очень большим числом параметров и качеств, поскольку он, с одной стороны *функционально* является сложным программно-управляемым цифровым процессором, т.е. устройством ЭВМ, а с другой – интегральной микросхемой, т.е. электронным прибором. Поэтому для МП важны такие качества и параметры, как:

• тип микроэлектронной технологии, проектные нормы и число слоев металлизации (проектные нормы определяют минимальные топологические размеры элементов, что, в свою очередь, определяет количество транзисторов, размещаемых на кристалле МП, и максимальную рабочую частоту МП);

• количество кристаллов, образующих МП;

• площадь кристалла и количество транзисторов на кристалле;

• тип корпуса;

• разрядность МП;

• быстродействие МП (рабочая частота; число одновременно декодируемых инструкций; число команд, запускаемых на выполнение за один такт; время выполнения команд);

• размер адресуемой памяти;

• наличие и размер кэш-памяти;

• наличие арифметического сопроцессора;

• число входящих в микропроцессорный набор дополнительных микросхем и выполняемые ими функции;

• система команд (количество команд, выполняемые операции, способы адресации, наличие команд обработки бит, чисел с плавающей запятой, десятичной арифметики);

• форматы данных;

• типы и число уровней прерывания;

• возможность прямого доступа к памяти;

• пропускная способность интерфейса ввода/вывода (частота и разрядность системной шины);

• количество и уровни питающих напряжений;

• требования к синхронизации;

• параметры используемых сигналов;

• потребляемая мощность;

• помехоустойчивость;

• нагрузочная способность;

• надежность и т.д.

## 10 *Классификация микропроцессоров. По числу кристаллов, назначению, типу набора команд, виду обрабатываемых входных сигналов, характеру временной организации работы.*

Микропроцессоры делятся на отдельные классы в соответствии с их архитектурой, структурой и функциональным назначением.

По *числу кристаллов, образующих МП*, различают микропроцессоры:

• однокристальные с фиксированной разрядностью и системой команд;

• многокристальные с фиксированной разрядностью и системой команд;

• многокристальные с разрядно-модульной организацией (секционные микропрограммируемые).

По *назначению* различают МП:

• универсальные (общего назначения);

• специализированные.

По *типу набора команд* (по *типу архитектуры набора команд*) различают МП:

• реализованные на базе архитектуры, называемой компьютером со сложным (полным, расширенным) набором команд (CISC – Complex Instruction Set Computer) – МП с CISC-архитектурой или CISC-микропроцессоры;

• реализованные на базе архитектуры, называемой компьютером с сокращенным набором команд (RISC – Reduced Instruction Set Computer) – МП с RISC-архитектурой или RISC-микропроцессоры.

По *виду обрабатываемых входных сигналов* различают МП:

• цифровые;

• аналоговые.

По *характеру временной организации работы* различают МП:

• синхронные;

• асинхронные.

## 11 *Эволюция микропроцессоров. Краткая характеристика процессоров фирмы Intel: 4004, 4040, 8080, Pentium, PentiumPro, Pentium 2, Celeron, Pentium 3, Pentium 4, Core / Core 2, Corei3 / Corei5 / Corei7, Atom, Xeon.*

КОНСПЕКТ

## 12 *Организация магистрали микропроцессорной системы и виды передачи данных. Понятие шина. Структура магистрали. Шины данных, адреса, управления и состав их сигналов.*

На физическом уровне МП взаимодействует с памятью и периферийными устройствами через единый набор системных шин – *внутрисистемную магистраль*. В общем случае магистраль обеспечивает три вида передачи данных:

1. процессор – память;

2. процессор – интерфейс периферийного устройства;

3. память – интерфейс периферийного устройства (канал прямого доступа к памяти).

Рассмотрим первые два вида (прямого доступа к памяти рассматривается в отдельной теме). В обоих случаях передачей данных управляет МП. Память и интерфейс по управляющим сигналам от процессора осуществляют передачу

данных. Направление передачи данных определяется МП. Пересылка данных внутрь процессора называется *считыванием* (*вводом*), обратный процесс – *записью* (*выводом*).

В общем случае магистраль микропроцессорной системы состоит из набора шин. *Шиной* системы называют физическую группу линий передачи сигналов, имеющих схожие функции в рамках системы. Стандартная структура магистрали микропроцессорной системы включает три шины:

1. шина данных DB (Data Bus);
2. шина адреса (адресная шина) AB (Address Bus);
3. шина управления CB (Control Bus).

Магистраль такого типа называется *трехшинной с раздельными шинами передачи адреса и данных*.

*Шина данных*. По этой шине производится обмен данными между МП и другими устройствами системы – памятью и периферийными устройствами. Шина данных является *двунаправленной*. Имеется возможность установки выходов в третье отключенное состояние. Хотя передача данных по шине данных может производиться в обоих направлениях, однако в каждый момент времени она осуществляется лишь в одном направлении, причем, по всем разрядам шины в одном и том же, т.е. в любой момент по всем линиям шины данные могут либо только вводиться либо только выводиться. Обычно разрядность шины данных и длину слов, обрабатываемых в МП (разрядность машинного слова МП), выбирают одинаковыми. В любом случае ширину шины данных выбирают кратной целому числу байтов, причем это число, как правило, представляет собой целую степень числа 2.

*Шина адреса*. Используется для передачи физического адреса ячейки памяти или порта ввода/вывода, к которым осуществляется обращение. Эта шина предназначена для того, чтобы выбирать правильный тракт для электрического соединения в пределах микропроцессорной системы. Шина адреса является выходной по отношению к МП. Разрядность адресной шины определяет наибольшее число адресов, к которым может обращаться МП. Если разрядность адресной шины МП равна m, то он способен адресовать пространство физической памяти и пространство ввода/вывода объемом 2m.

*Шина управления*. Служит для передачи сигналов управления обменом данными через магистраль и работой микропроцессорной системы. Как правило, часть этих сигналов является выходными, а другая часть – входными сигналами. Однако некоторые линии шины управления могут быть двунаправленными. Линии шины управления объединяются в группы по функциональному назначению. Конкретный состав сигналов шины управления зависит от типа МП. Отметим наиболее типичные из них:

• *сигналы тактирования и синхронизации*. Обеспечивают тактирование работы микропроцессора. Все события в системе привязываются к какому-либо фронту этих сигналов;

• *четность данных*. Сигналы на этих линиях определяют признаки паритета данных, передаваемых по шине данных, и используются для обнаружения

ошибок, возникших в процессе передачи. Как правило, на каждый байт шины данных отводится отдельный сигнал;

• *сигналы определения цикла магистрали*. Указывают тип выполняемого цикла магистрали. Они разделяют циклы записи и циклы чтения, циклы данных и циклы управления, циклы обращения к памяти и циклы ввода/вывода, а также некоторые другие;

• *сигналы управления магистралью*. Определяют начало цикла магистрали, управляют передачей и разрядностью данных, завершением цикла магистрали;

• *сигналы управления состоянием процессора*. Изменяют состояние МП в ходе выполнения или перед выполнением программы. Используются для распределения функций управления магистралью между несколькими активными устройствами, при обработке прерываний, сбросе и инициализации.

## 13 *Организация обмена по магистрали. Циклы обращения к магистрали, их типы, основные типы сигналов.*

Команды и данные передаются между МП и другими устройствами системы в ходе операции обмена, которая может включать один или несколько магистральных циклов, т.е. физический обмен через магистраль выполняется словами определенной разрядности в виде следующих друг за другом обращений к магистрали. Время осуществления одного считывания, записи, ввода или вывода называется *циклом обращения к магистрали* или просто *циклом магистрали* (*циклом шины*). За один цикл обращения к магистрали между МП, памятью или периферийным устройством передается одно слово. Такие циклы обращения к магистрали называются *простыми*. Таким образом, циклы магистрали обеспечивают доступ к пространству физической памяти и пространству ввода/вывода.

Существует несколько типовых циклов магистрали. Основные циклы магистрали связаны с возможными операциями, выполняемыми в микропроцессорной системе. К ним относятся циклы чтения и записи. При совмещенном вводе/выводе по этим циклам осуществляется как обращение к памяти, так и к портам ввода/вывода. При изолированном вводе/выводе эти циклы разделяются на циклы обращения к памяти и цикла обращения к портам ввода/вывода. Поэтому в микропроцессорной системе с изолированным вводом/выводом выделяется четыре основных цикла:

• цикл чтения из памяти;
• цикл записи в память;
• цикл ввод из порта ввода;
• цикл вывода в порт вывода.

В случае гарвардской архитектуры вводится также цикл чтения памяти программ.

Основными сигналами, связанными с выполнением приведенных выше циклов магистрали, являются сигналы двух типов:

• сигналы управления записью/чтением, связанные с обращением к памяти;

• сигналы управления записью/чтением (вводом/выводом), связанные с обращением к портам ввода/вывода.

Когда применяется изолированный ввод/вывод, используются четыре управляющих сигнала:

• чтение данных из памяти MEMRD;

• запись данных в память MEMWR;

• ввод данных из порта ввода IORD;

• вывод данных в порт вывода IOWR.

Для гарвардской архитектуры добавляется сигнал чтение памяти программ PMEN.

Сигналы MEMRD, MEMWR, IORD, IOWR, PMEN являются как сигналами, определяющими цикл магистрали, так и управляющими синхронизирующими сигналами, показывающими, в какой интервал времени в цикле шины должна осуществляться соответствующая операция, т.е. *стробирующими сигналами*.

В случае ввода/вывода с отображением на память порты ввода/вывода и память не различаются по способу доступа, поэтому можно использовать два стробирующих сигнала RD и WR.

Кроме рассмотренной существуют системы с другим составом управляющих сигналов, например, с тремя управляющими сигналами:

• MEM/IO, RD, WR. Сигнал MEM/IO указывает, к какому из пространств (памяти или ввода-вывода) осуществляется обращение в данном цикле (выбор пространства), т.е. разделяет циклы обращения к памяти и циклы ввода/вывода, RD – строб чтения, WR – строб записи. Оба эти сигнала являются общими как для памяти, так и для портов ввода/вывода;

• MEM/IO, RD/WR, STRB. Сигнал RD/WR указывает, является ли данный цикл циклом записи или циклом чтения (выбор операции чтения или записи), т.е. разделяет циклы чтения и циклы записи, STRB – строб, используемый как для чтения, так и для записи.

14 *Организация обращения к магистрали с синхронным доступом. Протокол обмена и схема соединения памяти с магистралью. Циклы чтения/записи в память.*

МП управляет работой шины синхронно с входной тактовой частотой. Элементарным интервалом времени при реализации протоколов обмена является *такт* магистрали, равный одному периоду синхросигнала. Каждый цикл шины содержит несколько тактов.

Протокол обмена по магистрали предполагает выполнение определенной последовательности действий:

1. адресация памяти или порта ввода/вывода;
2. коммутация направления передачи (задание операции обмена – чтение или запись);
3. передача данных (выполнение операции обмена);
4. фиксация данных.

В стандартном цикле магистрали для реализации каждого из приведенных выше действий отводится по одному такту, т.е. стандартный цикл магистрали содержит четыре обязательных такта Т1-Т4.

Чтение/запись данных в память. Схема соединения памяти с магистралью приведена на рис. 11.

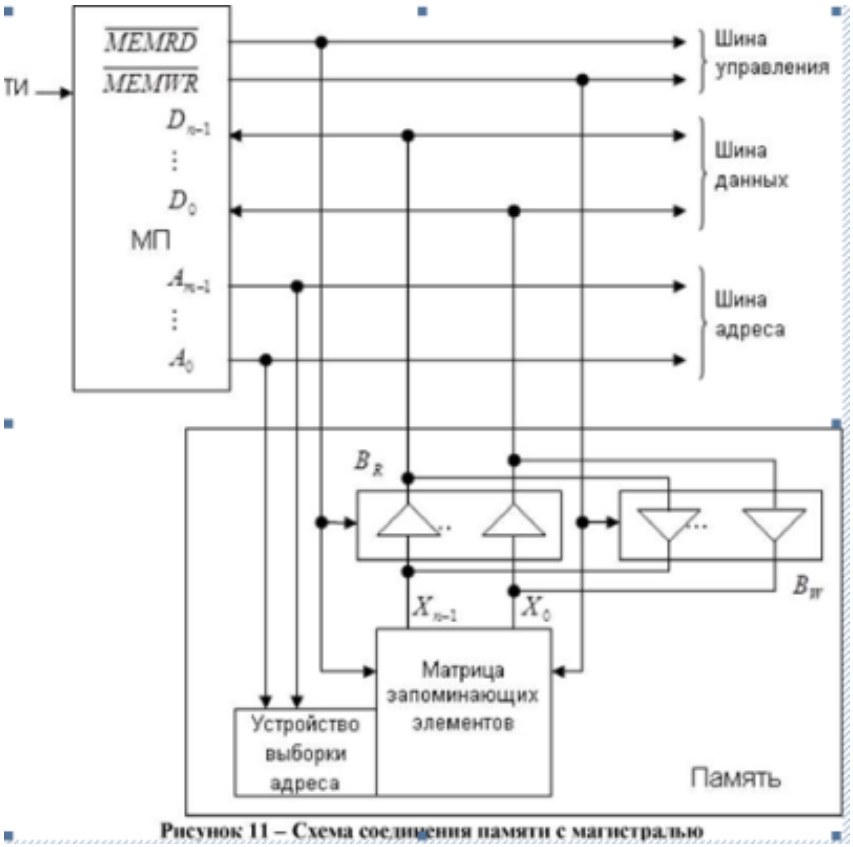



Рисунок 11 – Схема соединения памяти с магистралью

**Рисунок 11 – Схема соединения памяти с магистралью**

В начале *цикла чтения памяти* (рис. 12) МП по адресной шине передает адрес, по которому происходит выборка ячейки в памяти, и указанная ячейка

памяти подключается к линиям Xn-1-X0. В первой половине такта Т2 сигнал MEMRD переходит на низкий уровень, а во второй половине такта Т4 он возвращается на высокий уровень. Низким уровнем этого сигнала открывается тристабильный вентиль чтения BR и линии Xn-1-X0 соединяются с линиями Dn-1-D0 шины данных. При этом вентиль записи BW закрыт. С момента передачи адреса по адресной шине в память до выдачи содержимого указанной ячейки памяти требуется определенное время, которое называют *временем обращения к памяти*. Во время считывания микропроцессором данные на шине данных (выходе памяти) должны поддерживаться в неизменном состоянии. Выполнение этого требования обеспечивается за счет наличия в цикле магистрали такта Т3 – к концу этого такта содержимое указанной ячейки памяти должно находиться на шине данных. По заднему фронту положительного импульса такта Т4 содержимое шины данных заносится в МП (данные считываются МП и фиксируются во внутреннем регистре).

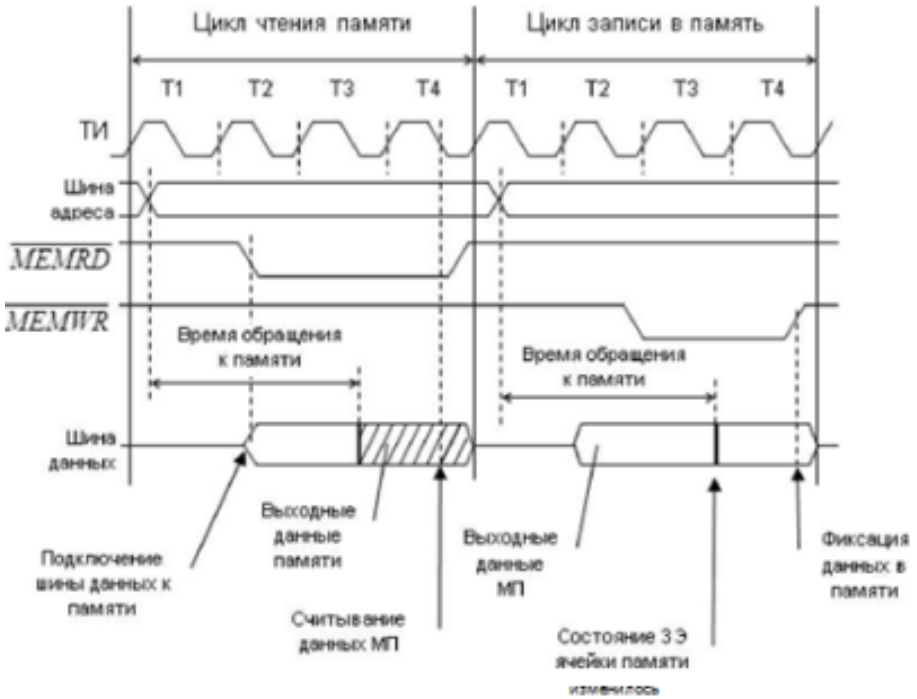



**Рисунок 12 – Циклы чтения/записи в память**

В начале *цикла записи в память* (см. рис. 12) МП передает адрес. С первой половины такта Т2 до окончания такта Т4 МП осуществляет вывод записываемых данных на линии шины данных. Низким уровнем сигнала открывается тристабильный вентиль записи BW, линии соединяются с линиями Xn-1-X0, и начинается процесс записи в выбранную ячейку памяти. Для записи информации в память также требуется некоторое время, в течение которого происходит изменение состояния запоминающих элементов ячейки памяти. Поэтому в течение времени, пока сигнал MEMWR имеет низкий уровень, данные на входе памяти должны поддерживаться в неизменном состоянии. Для этого служит такт Т3. Когда уровень сигнала MEMWR на такте Т4 становится высоким, содержимое линий Xn-1-X0 фиксируется в ячейке памяти, указанной адресом.

15 *Организация обращения к магистрали с синхронным доступом. Схема соединения портов ввода/вывода с магистралью. Циклы обращения к магистрали в системе с тремя управляющими сигналами.*

Ввод/вывод данных. Схема соединения портов ввода/вывода с магистралью приведена на рис. 13.

.

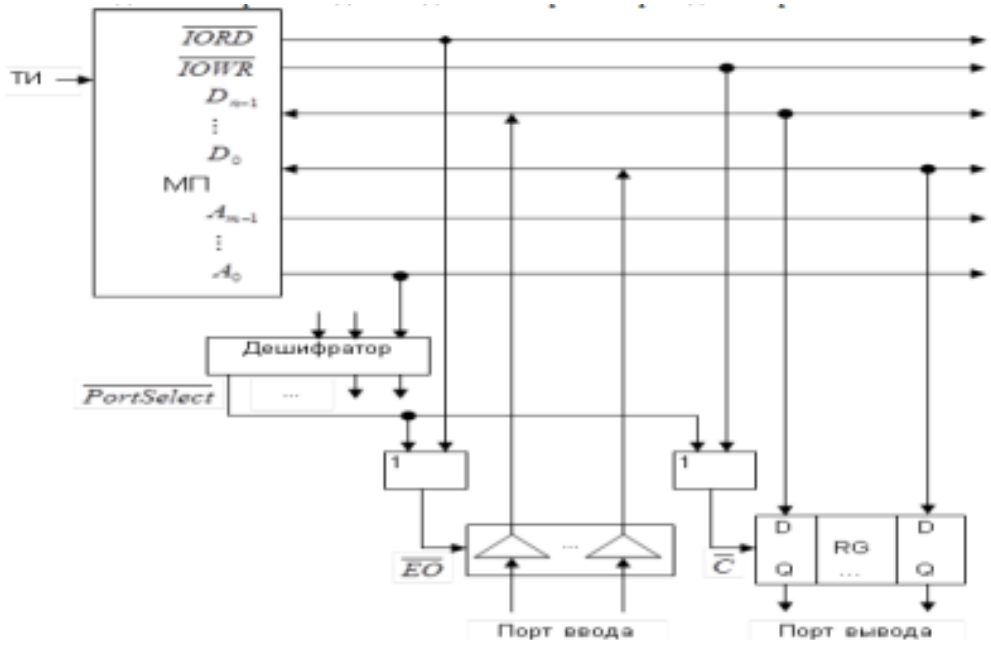



**Рисунок 13 – Схема соединения портов ввода/вывода с магистралью**

*Порт ввода* представляет собой ряд тристабильных вентилей, при открывании которых по сигналу ЕО данные, поступающие в этот порт, передаются на линии шины данных, откуда уже считываются МП. *Порт вывода* представляет собой регистр, работа которого заключается в следующем. МП выводит данные на линии шины данных. Эти данные стробирующим сигналом С заносятся в регистр, который обеспечивает их сохранность до записи новых данных. Для выбора портов ввода/вывода используется дешифратор. Младшими k разрядами адресной шины можно осуществлять выбор 2k портов ввода или вывода.

Временная диаграмма работы в циклах ввода/вывода данных аналогична временной диаграмме в циклах чтения/записи в память. Различие состоит в том, что вместо сигналов MEMRD и MEMWR МП выдает сигналы IORD и IOWR соответственно.

Временные диаграммы работы шины в системе с тремя управляющими сигналами приведены на рис. 14 (на рисунке не показана шина данных).

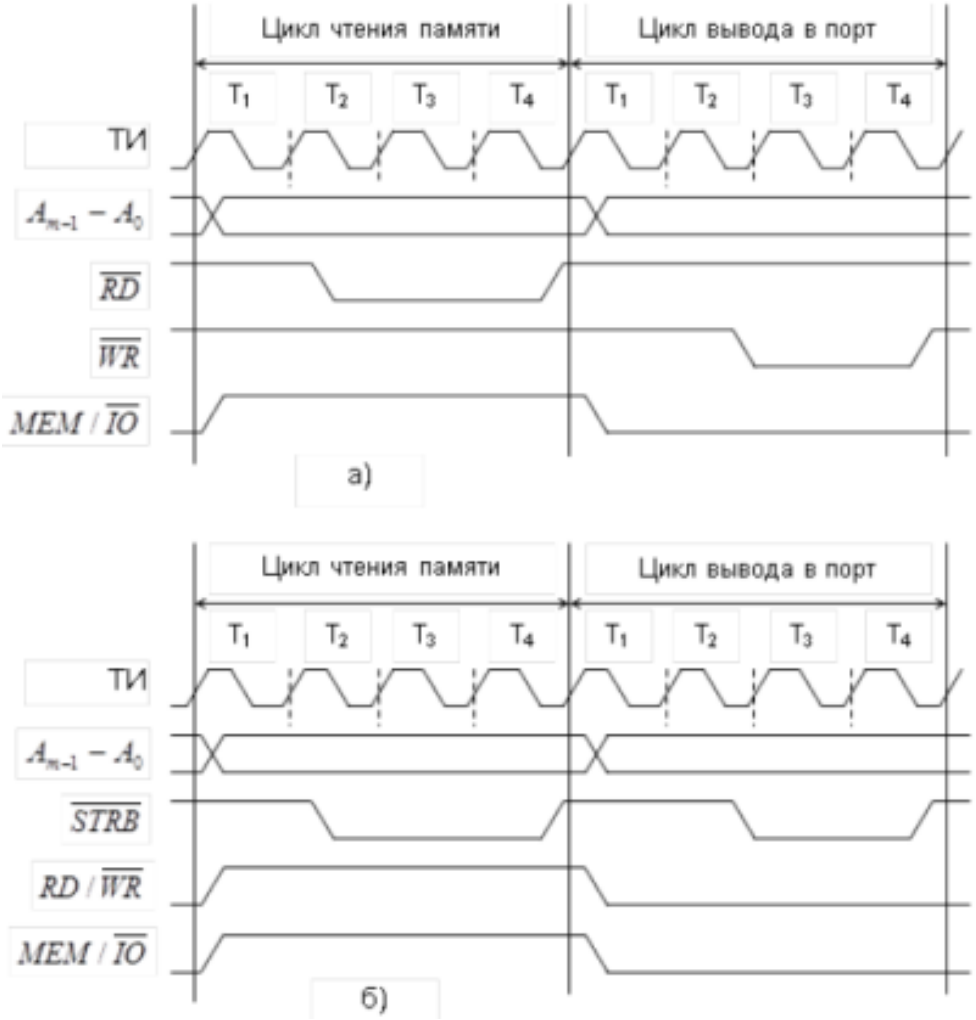



**Рисунок 14 – Циклы обращения к магистрали в системе с тремя управляющими сигналами**

Сигнал MEM/IO формируется в начале цикла одновременно с адресом и поддерживается неизменным в течение всего цикла магистрали. Аналогично формируется сигнал RD/WR, так как направление передачи остается неизменным в течение всего цикла шины. Управляющие сигналы RD и WR стробируют выполнение операций чтение и запись и вырабатываются аналогично сигналам MEMRD (IORD) и MEMWR (IOWR) соответственно. Стробирующий сигнал STRB определяет время выполнения операции чтение или запись и формируется так же, как и сигналы MEMRD, MEMWR, IORD или IOWR.

В рассмотренных циклах обращения к магистрали временные соотношения чтения/записи полностью задаются МП. В этом случае память и порты ввода/вывода должны постоянно находиться в рабочем (готовом) состоянии. Такая организация обмена по магистрали называется *обращением с синхронным доступом*.

16 *Организация обращения к магистрали с асинхронным доступом. Цикл обращения к магистрали с применением сигнала готовности.*

Для медленных устройств система должна позволять произвольно увеличивать длительность циклов магистрали. Для этого из памяти или из портов ввода/вывода передаются управляющие сигналы, задающие время окончания цикла (подтверждающие окончание цикла). Как правило для этой цели используется сигнал READY (ГОТОВНОСТЬ), но могут также использоваться сигналы WAIT (ОЖИДАНИЕ) и TRANSFER_ACKNOWLEDGE (ПОДТВЕРЖДЕНИЕ ПЕРЕДАЧИ).

Временная диаграмма работы магистрали с применением сигнала готовности приведена на рис. 15.

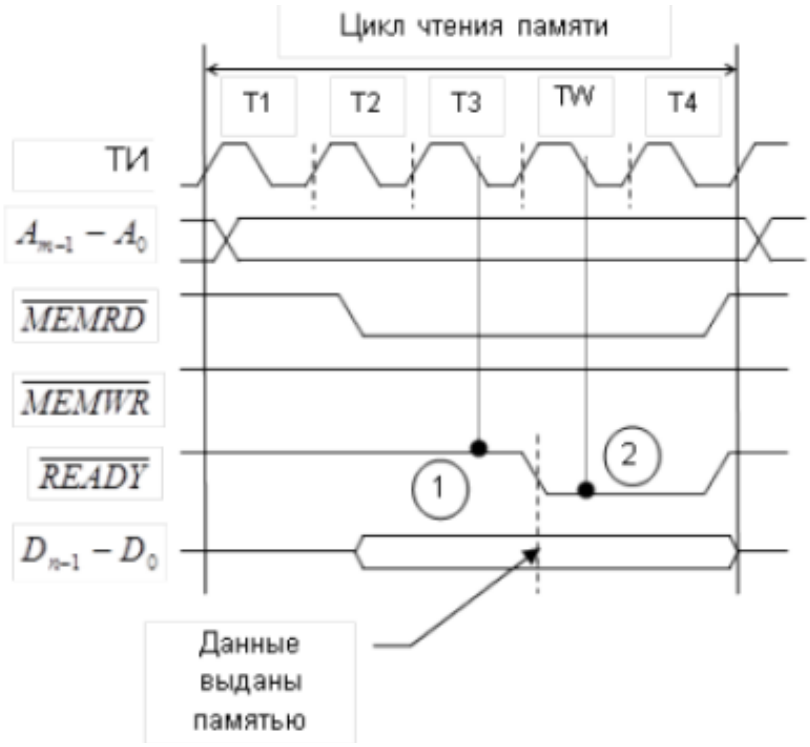



**Рисунок 15 – Цикл обращения к магистрали с применением сигнала готовности**

МП по заднему фронту положительного импульса такта Т3 (момент времени 1) анализирует состояние сигнала READY. Если данный сигнал имеет высокий уровень, цикл дополняется еще одним тактом (ожидания) TW. По заднему фронту положительного импульса такта TW (момент времени 2) опять анализируется состояние сигнала READY. Если уровень этого сигнала низкий, новые дополнительные такты не вводятся, а следующий такт Т4 является последним тактом цикла. Если сигнал READY, анализируемый в такте TW, имеет высокий уровень, цикл дополняется новыми тактами.

Таким образом, длительность цикла можно изменять в зависимости от готовности памяти или порта ввода/вывода. Такая организация обмена по магистрали называется *обращением с асинхронным доступом*.

17 *Совмещение адресной шины и шины данных. Схема с двухшинной магистралью. Цикл магистрали с совмещенными шинами передачи адреса и данных.*

В некоторых МП с целью сокращения ширины физической магистрали используют совмещение адресной шины с шиной данных. В течение первого такта цикла магистрали шина данных не используется, поэтому этот интервал можно использовать для передачи по шине данных адресных сигналов (адреса). Такая шина называется *совмещенной шиной адреса/данных* AD (Address/Data Bus). Этап передачи адресной информации по совмещенной шине адреса/данных отделяется по времени от этапа передачи данных и стробируется специальным сигналом ALE (Address Latch Enable), который включается в состав шины управления. Данную магистраль называют *двухшинной с совмещенными шинами передачи адреса и данных*. Если разрядность данных меньше разрядности адреса, то по совмещенной шине передаются только младшие разряды адреса, а старшие разряды при этом передаются по адресной шине.

Входящий в состав шины управления сигнал ALE используется для разделения функций, выполняемых совмещенной шиной AD. По этому сигналу присутствующая на шине AD адресная информация должна быть принята (зафиксирована) во внешний (по отношению к МП) адресный регистр-фиксатор. Для этой цели обычно служит срез сигнала ALE (переход из высокого уровня в низкий). Обычно каждый модуль микропроцессорной системы с двухшинной магистралью (модуль памяти или интерфейс периферийного устройства) содержит локальный адресный регистр для запоминания адресной информации. Для фиксации адресной информации может быть использован и один общий регистр, в результате МП с двухшинной магистралью преобразуется в МП с тремя раздельными шинами (рис. 16). Когда уровень управляющего сигнала, приходящего на вход С регистра-фиксатора, становится высоким, входная информация без изменения передается на выход. При переходе управляющего сигнала на входе С в низкий уровень информация фиксируется в регистре.

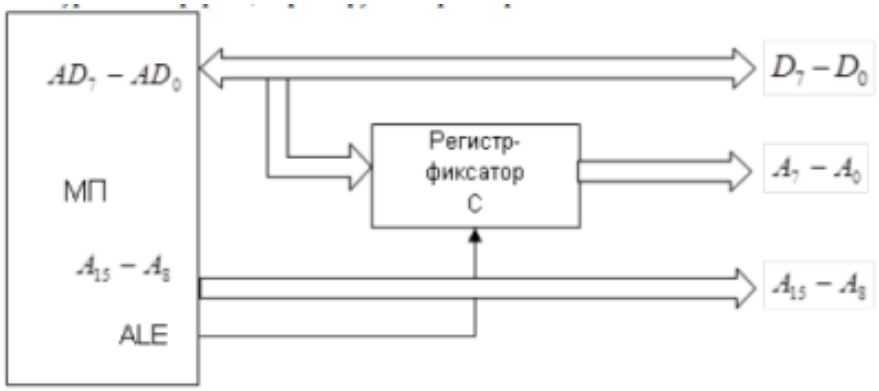



**Рисунок 16 – МПС с двухшинной магистралью**

Временная диаграмма работы МПС с совмещением адресной шины с шиной данных приведена на рис. 17.

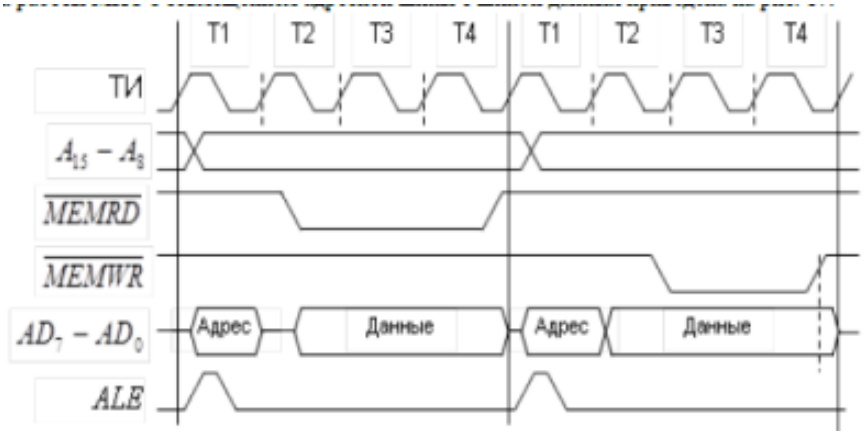



**Рисунок 17 – Цикл магистрали с совмещенными шинами передачи адреса и данных**

В течение первого такта T1 по совмещенной шине AD7-AD0 передаются адресные разряды A7-A0. Эти разряды по сигналу ALE фиксируются в регистре фиксаторе, который находится вне МП.

18 *Способы повышения эффективности системной магистрали. Конвейеризация обмена по магистрали: схема подключения и циклы чтения/записи в память.*

КОНСПЕКТ

19 *Способы повышения эффективности системной магистрали. Последовательности циклов и пакетная передача. Временная диаграмма пакетного цикла чтения из памяти.*

Для обслуживания некоторых внутренних запросов при работе с памятью процессору может понадобиться последовательность циклов обмена, во время которых передаются данные, расположенные в смежной области адресного пространства. Такая ситуация может иметь место при выборке операндов, имеющих разрядность большую, чем разрядность шины данных (например, 32-разрядный процессор может обращаться к 64- или 128-разрядным операндам), или при заполнении строки кэш-памяти (например, если строка кэша имеет длину 32 байта, то для ее пересылки требуется четыре 64-разрядных цикла магистрали). Во всех таких случаях, когда требуется больше одного цикла для передачи данных, микропроцессор может выполнять *пакетные циклы*. Во время пакетного цикла между МП и памятью передается

более одного слова, причем эти слова занимают смежные адреса и направление передачи для всех слов одинаково (т.е. все слова читаются из памяти или записываются в память). Такой блок данных называется *пакетом*, а протокол обмена по магистрали – *режимом пакетной передачи* (пакетный режим – Burst Mode).

Выполнение стандартного цикла магистрали можно разбить на две фазы:

• фаза адресации, которая включает адресацию памяти и коммутацию направления передачи;

• фаза данных, которая включает передачу данных и их фиксацию.

В пакетном режиме одна фаза адресации сопровождается множественными фазами данных (чтения или записи, но не чередующимися). Это означает, что пакет данных передается без указания текущего адреса внутри пакета. При этом передается адрес только первого слова, все последующие адреса генерируются из первого в самой памяти по определенному, заранее известному правилу. Такой протокол обмена возможен, если адрес и сигналы идентификации типа цикла выдавать только в первой фазе пакета, а в каждой из последующих фаз передавать только данные, адреса для которых уже не передаются по адресной шине. В результате на передачу первого слова затрачивается две фазы, а далее данные передаются в каждой фазе, а не через фазу, как в обычных циклах обмена.

На рис. 21 приведена временная диаграмма пакетного цикла чтения из памяти, включающего передачу четырех слов.

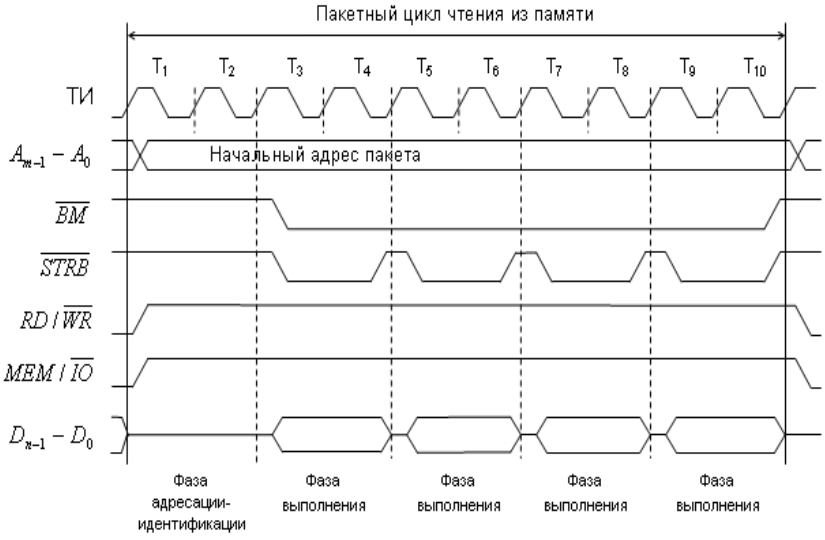



**Рисунок 21 – Пакетный цикл чтения из памяти**

В пакетном цикле фаза 1 занимает такты Т1 и Т2, а фаза 2 – такты Т3 и Т4. Пакетный цикл начинается МП так же, как и обычный: в первой фазе на шине адреса устанавливается адрес первого слова пакета, а на шине управления – сигналы идентификации типа цикла (например, MEM/IO и RD/WR). В следующей фазе передается первое слово данных, и, если оно не единственное, специальный управляющий сигнал BM, который указывает, что данный цикл пакетный. Далее МП продолжает цикл как пакетный, не вводя фазы адресации, а сразу перейдет к передаче следующего слова данных. О

завершении пакетного цикла микропроцессор сообщает памяти снятием сигнала ВМ.

Скорость передачи собственно данных в пакетном режиме увеличивается естественным образом за счет уменьшения числа передаваемых адресов. Из рис. 21 видно, что для передачи четырех слов с помощью пакетного цикла требуется 10 тактов, в то время как передача четырех слов с помощью обычных циклов занимает 4 цикла х 4 такта = 16 тактов.

## 20 *Характеристики подсистемы памяти микропроцессорной системы. Функции и операции в памяти.*

*Подсистема памяти* – совокупность устройств, служащих для *запоминания, хранения и выдачи* информации, представленной в виде цифровых кодов.

Содержимое памяти обычно относится к одной из четырех категорий:

- коды программы;
- данные (постоянные);
- промежуточные результаты обработки (переменные);
- состояние системы.

Можно выделить три основные функции памяти:

- хранение программ и данных;
- запоминание промежуточных результатов, используемых в ходе обработки;
- работа в качестве элемента устройства обработки.

Отдельные устройства, входящие в подсистему памяти микропроцессорной системы, называют запоминающими устройствами (ЗУ) или памятями того или иного типа. Оба эти термина в настоящее время почти синонимы. ЗУ обычно употребляется, когда речь идет о принципе построения некоторого устройства памяти. Например, полупроводниковое ЗУ, ЗУ на магнитных дисках и т.п. Термин память употребляется, когда хотят подчеркнуть выполняемую устройством памяти логическую функцию или место в составе системы. Например, основная память, кэш-память, внешняя память.

Основные операции в памяти:

- занесение информации в память – *запись*;
- выборка информации из памяти – *считывание* или *чтение*.

Обе эти операции называют обращением к памяти или обращением при считывании (чтении) и обращением при записи.

21 *Характеристики запоминающих устройств внутренней памяти. Емкость, единица пересылки, метод доступа к данным, быстродействие, время обращения к памяти, период обращения, скорость передачи данных.*

Основными характеристиками запоминающих устройств являются:
• емкость;
• единица пересылки;
• метод доступа;
• быстродействие;
• стоимость.

*Емкость* (*объем*) ЗУ определяется максимальным количеством информации, которое одновременно может в нем храниться, и выражается в количестве слов определенной разрядности – битов, байтов или n-разрядных слов. Так как эта емкость может быть достаточно велика, то обычно используют более крупные единицы, образованные присоединением приставок К/K (кило/kilo), М/M (мега/mega), Г/G (гига/giga), Т/T (тера/tera), П/P (пета/peta), Е/E (экса/exa), З/Z (зетта/zetta), И/Y (иотта/yotta) к перечисленным выше единицам. В вычислительной технике эти приставки означают умножение основной единицы измерений на целую степень числа 2:
• K – $2^{10}$ = 1024;
• M – $2^{20}$ = 1 048 676;
• G – $2^{30}$ = 1 073 741 824;
• T – $2^{40}$ = 1 099 511 627 776;
• P – $2^{50}$;
• E – $2^{60}$;
• Z – $2^{70}$;
• Y – $2^{80}$.

Важной характеристикой ЗУ является *единица пересылки*. Для основной памяти единица пересылки определяется шириной шины данных, т.е. количеством битов, передаваемых по линиям шины параллельно (одновременно). Обычно единица пересылки равна длине слова. Данные могут передаваться единицами, превышающими размер слова, которые называются *блоками*. Например, блоками передаются данные при пересылке между основной памятью и кэш-памятью.

При оценке быстродействия необходимо учитывать применяемый в данном типе ЗУ *метод доступа* к данным. Различают четыре основных метода доступа: прямой, последовательный, произвольный и ассоциативный. В ЗУ внутренней памяти используются три последних метода:

*Быстродействие* ЗУ является одним из важнейших его показателей. Для количественной оценки быстродействия обычно используют три параметра: время обращения к памяти, цикл обращения и скорость передачи данных.

*Время обращения к памяти*. Время обращения при записи и чтении определяется следующим образом

$t_{обр\ зап} = t_д + t_{зап}$;

$t_{обр\ чт} = t_д + t_{счит}$,

где

• $t_д$ – *время доступа*.

• $t_{зап}$ – *время записи*.

• $t_{счит}$ – *время считывания*.

Практически для большинства ЗУ $t_{обр\ зап} = t_{обр\ чт}$.

*Время цикла обращения к памяти* (цикла памяти) или *период обращения* Тц. Означает минимальное время между двумя последовательными обращениями к памяти. Период обращения включает в себя время доступа плюс некоторое дополнительное время. Дополнительное время может требоваться для приведение в исходное состояние элементов памяти, а в некоторых типах ЗУ, где считывание информации приводит к ее разрушению, – для восстановления (регенерации) считанной информации.

*Скорость передачи данных*. Определяется количеством данных считываемых или записываемых запоминающим устройством в единицу времени.

*Стоимость* ЗУ принято оценивать отношением общей стоимости ЗУ к его емкости в битах, т.е. стоимостью хранения одного бита информации.

## 22 *Многоуровневая организация памяти. Иерархическая память. Принцип локальности по обращению. Характеристики, используемые при оценке эффективности организации памяти.*

Память является «узким местом» МПС из-за ее серьезного отставания по быстродействию от процессоров, причем разрыв этот неуклонно увеличивается. Так, если производительность процессоров ежегодно возрастает вдвое примерно каждые 1,5 года, то для микросхем памяти прирост быстродействия не превышает 9% в год (удвоение за 10 лет), что выражается в увеличении разрыва в быстродействии между процессором и памятью приблизительно на 50% в год.

В отношении устройств памяти, используемых в настоящее время, выявляется следующая закономерность:

• чем меньше время доступа, тем выше стоимость хранения бита;

• чем больше емкость, тем ниже стоимость хранения бита, но больше время доступа.

Основным требованием, которому должно удовлетворять ЗУ, является обеспечение требуемой емкости и высокого быстродействия за приемлемую цену. Как правило, эти две характеристики рассматриваются во взаимосвязи –

желательно, чтобы память обладала как можно большей емкостью и как можно большим быстродействием и при этом была бы экономически выгодной с точки зрения технической реализации (приемлемые стоимость, габариты, масса, потребляемая мощность). Удовлетворить эти противоречащие друг другу требования одновременно в одном устройстве достаточно сложно.

Наиболее распространенным подходом является построение подсистемы памяти по иерархическому принципу. *Иерархическая память* состоит из ЗУ различных типов, которые, в зависимости от характеристик, относят к определенному уровню иерархии:

1. регистры;
2. кэш-память;
3. основная память (ОП);
4. твердотельные и магнитные диски;
5. оптические диски, ЗУ на магнитных лентах.

Три верхних уровня иерархии образуют *внутреннюю память* МПС, а все нижние уровни - это *внешняя* или *вторичная память*.

Более высокий уровень меньше по емкости, быстрее и имеет большую стоимость в пересчете на бит, чем более низкий уровень. Уровни иерархии взаимосвязаны: все данные на одном уровне могут быть также найдены на более низком уровне, и все данные на этом более низком уровне могут быть найдены на следующем нижележащем уровне и т.д. По мере движения вниз по иерархической структуре:

1. уменьшается соотношение «стоимость/бит»;
2. возрастает емкость;
3. растет время доступа;
4. уменьшается частота обращения к памяти со стороны процессора.

Если память организована в соответствии с пунктами 1-3, а характер размещения в ней данных и команд удовлетворяет пункту 4, иерархическая организация ведет к уменьшению общей стоимости при заданном уровне производительности.

На каждом уровне иерархии информация разбивается на *блоки*, выступающие в качестве наименьшей информационной единицы, пересылаемой между двумя соседними уровнями иерархии. Размер блоков может быть фиксированным либо переменным. При фиксированном размере блока емкость памяти обычно кратна его размеру. Размер блоков на каждом уровне иерархии чаще всего различен и увеличивается от верхних уровней к нижним.

При оценке эффективности подобной организации памяти обычно используют следующие характеристики:

• *коэффициент попаданий* (hit rate) – отношение числа обращений к памяти, при которых произошло попадание, к общему числу обращений к ЗУ данного уровня иерархии;

• *коэффициент промахов* (miss rate) – отношение числа обращений к памяти, при которых имел место промах, к общему числу обращений к ЗУ данного уровня иерархии;

• *время обращения при попадании* (hit time) – время, необходимое для поиска нужной информации в памяти верхнего уровня (включая выяснение, является ли обращение попаданием), плюс время на фактическое считывание данных;

• *потери на промах* (miss penalty

Описание некоторого уровня иерархии ЗУ предполагает конкретизацию четырех моментов:

• размещения блока – допустимого места расположения блока на примыкающем сверху уровне иерархии;

• идентификации блока – способа нахождения блока на примыкающем сверху уровне;

• замещения блока – выбора блока, заменяемого при промахе с целью освобождения места для нового блока;

• согласования копий (стратегии записи) – обеспечения согласованности копий одних и тех же блоков, расположенных на разных уровнях, при записи новой информации в копию, находящуюся на более высоком уровне.

## 23 *Классификация устройств памяти: по физическим принципам работы, в зависимости от методов размещения и поиска информации, по характеру работы с памятью, по характеру хранения, по способу обращения процессора.*

По физическим принципам работы ЗУ делятся на

• *магнитные*. ЗУ на магнитных дисках или лентах, ЗУ на цилиндрических магнитных доменах (ЦМД);

• *полупроводниковые*. Используются ЗЭ *с накоплением электрических зарядов* – на этих элементах строится динамическая память и *на основе активных приборов* (например, триггер) – на этих элементах строится статическая память;

• *оптические*. Запоминание информации происходит благодаря оптическому отображению ее на поверхности материала, осуществляемому лучом лазера.

В зависимости от методов размещения и поиска информации память делится на два типа

• *адресная память*. Размещение и поиск информации основан на использовании адреса. Адрес – номер ячейки памяти. Аадресная память строиттся на ЗУ с *произвольным* (непосредственным) *доступом* (выборкой);

• *безадресная память*. Размещение и поиск информации производится не по адресу. По способу доступа к ячейкам памяти безадресная память подразделяется на *ассоциативную* и *стековую*.

По характеру работы с памятью ЗУ делятся на
• ЗУ, *допускающие многократную запись и считывание* (например, ОЗУ);
• ЗУ, *допускающие только считывание после однократной записи* (например, ПЗУ).

По характеру хранения (в зависимости от источника питания) память делится на два типа
• *энергозависимая память*. Содержимое элементов памяти теряется при выключении источника питания – ОЗУ, ЗУ на ПЗС;
• *энергонезависимая память*. Элементы памяти сохраняют содержимое независимо от состояния источника питания – магнитные и оптические ЗУ, ПЗУ.

По способу обращения процессора к ЗУ память делится на два типа
• *внутренняя* память;
• *внешняя* память.

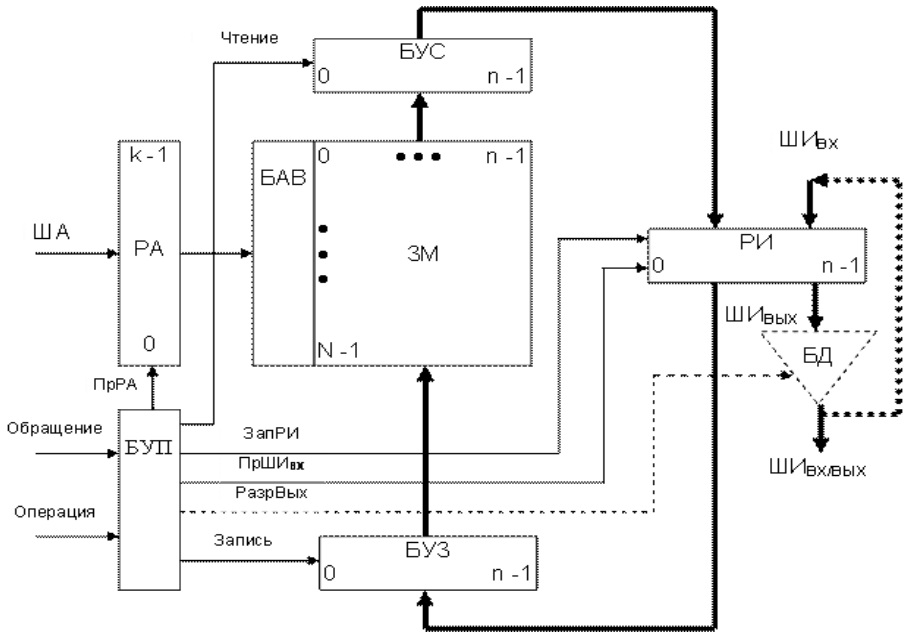
## 24 *Элементы и устройства памяти. Адресная память.*

ЗУ содержит множество одинаковых запоминающих элементов (ЗЭ), образующих запоминающий массив (ЗМ). ЗМ разделен на отдельные ячейки. Каждая ячейка предназначена для хранения двоичного кода, число разрядов в котором определяется шириной выборки памяти, – слова памяти.

В памяти с адресной организацией размещение и поиск информации в ЗМ основаны на использовании адреса хранения слов. Адресом служит номер ячейки ЗМ, в котором это слово размещается. При записи или чтении слова в ЗМ инициирующая эту операцию команда должна указывать адрес (номер ячейки), по которому производится обращение.

Типичная структура ЗУ с произвольным доступом (рис. 22) содержит ЗМ, состоящий из N n-разрядных ячеек, регистр адреса РА, имеющий $k = \log_2 N$ разрядов, регистр информации РИ, блок адресной выборки БАВ, блок устройств считывания БУС, блок устройств записи БУЗ и блок управления памятью БУП.

**Рисунок 22 – Структура ЗУ с произвольным доступом**




По коду адреса в РА БАВ формирует в соответствующей ячейке памяти сигналы, позволяющие произвести в ячейке считывание или запись слова. Блок управления памятью БУП генерирует необходимые последовательности управляющих сигналов, инициирующих работу отдельных блоков ЗУ.

Цикл обращения к памяти инициируется поступлением в БУП сигнала *Обращение*. Общая часть цикла обращения начинается с прием в РА с шины адреса ША адреса ячейки. Далее БАВ дешифрирует адрес и выбирает заданную адресом ячейку ЗМ. Одновременно осуществляется прием в БУП и расшифровка управляющего сигнала *Операция*, указывающего вид запрашиваемой операции (чтение или запись).

Далее при чтении БУП посылает сигналы считывания в заданную адресом ячейку ЗМ. При этом код записанного в ячейке слова считывается устройствами считывания БУС и передается в РИ. Операция чтения завершается выдачей слова из РИ на выходную информационную шину ШИВЫХ. В памяти с разрушающим считыванием (при считывании все ЗЭ ячейки устанавливаются в нулевое состояние) производится регенерация информации в ячейке путем записи в нее из РИ считанного слова.

При записи производится прием записываемого слова с входной информационной шины ШИВХ в РИ. Затем в выбранную БАВ ячейку с помощью устройств записи БУЗ записывается слово из РИ.

Для приема в ЗУ записываемых данных и выдачи из ЗУ считанных данных может использоваться одна двунаправленная информационная шина ШИВХ/ВЫХ. Для этого ШИВХ и ШИВЫХ объединяются с помощью буфера данных БД. Этот буфер подключает выходы РИ к ШИВХ/ВЫХ только при выполнении операции чтения из ЗУ. При выполнении операции записи выходы РИ отключены от ШИВХ/ВЫХ, что обеспечивает прием входных данных в РИ.

## 25 *Элементы и устройства памяти. Ассоциативная память: структура и виды поиска информации.*

В запоминающих устройствах с произвольным доступом для обращения к информации требуется указание адреса ячейки. Однако часто значительно удобнее искать информацию не по адресу, а опираясь либо на какой-нибудь характерный признак, содержащийся в самой информации, либо на положение слова в запоминающем массиве относительно других хранящихся там слов. Такой подход лежит в основе безадресной памяти, к которой относятся ассоциативная и стековая память.

В основе ассоциативной памяти лежит ассоциативная обработка.

*Ассоциативная обработка* основана на извлечении данных из памяти по их содержимому, т.е. обращение к данным происходит не на основании указания физического положения информации – адреса, а по некоторым признакам этих данных. При этом обращение (поиск) производится путем сравнения и сопоставления. В качестве признаков может использоваться различная информация, например, найти числа, равные заданному значению, или найти числа, большие (мень`шие) заданного значения. Указание признаков содержимого памяти не обязательно подразумевает только выявление их полного наличия или отсутствия в отыскиваемых данных, может быть также задана степень совпадения или несовпадения этих признаков с признаками данных и более сложные зависимости, что позволяет реализовать достаточно сложные методы обращения к данным. В ассоциативной обработке отсутствует понятие адресации, и обработка по своей сущности является параллельной, так как поиск по признаку происходит параллельно во времени для всех ячеек памяти.

Таким образом, *ассоциативная память* – память, аппаратные средства которой ориентированы на реализацию ассоциативной обработки. Ассоциативное ЗУ – это устройство, способное хранить информацию, сравнивать ее с некоторым заданным образцом и указывать на их соответствие или несоответствие друг другу. Признак, по которому производится поиск информации, будем называть *ассоциативным признаком*, а кодовую комбинацию, выступающую в роли образца для поиска, – *признаком поиска*. Ассоциативный признак может быть частью искомой информации или дополнительно придаваться ей. В последнем случае его принято называть *тегом* или *ярлыком*.

Типичная структура ассоциативного ЗУ (рис. 23) содержит запоминающий массив ЗМ, регистр ассоциативного признака РАП, регистр маски РМ, регистр информации РИ, блок сравнения БС, регистр совпадений РС и блок формирования результата ассоциативного обращения БР.

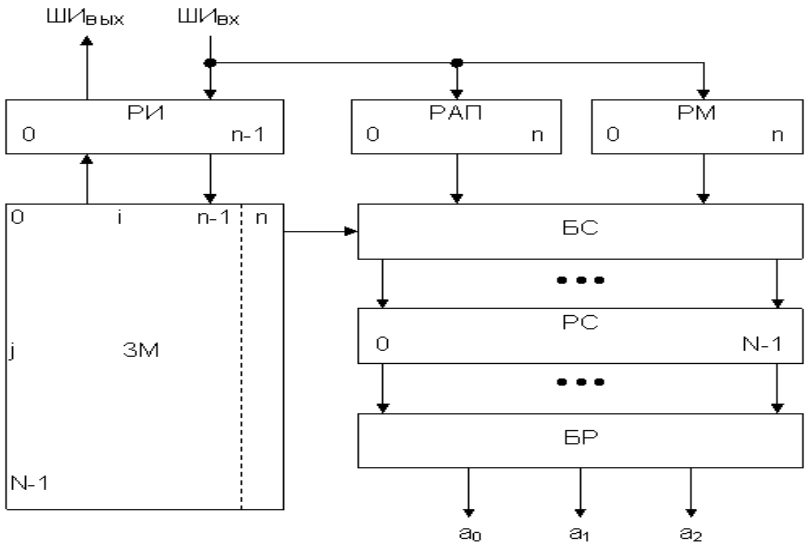



**Рисунок 23 – Структура ассоциативного ЗУ**

ЗМ содержит N (n+1)-разрядных ячеек. Для указания занятости ячейки используется служебный n-й разряд (0 – ячейка свободна, 1 – в ячейке записано слово).

По входной информационной шине ШИВХ в РАП поступает ассоциативный запрос (признак поиска), а в РМ – код маски поиска. Ассоциативный поиск производится только для тех разрядов РАП, которым соответствует 1 в РМ (незамаскированные разряды РАП). Блок сравнения БС состоит из N схем совпадения, которые используются для параллельного сравнения каждого бита всех хранимых слов с соответствующим битом признака поиска и выработки сигналов совпадения. В регистре совпадений РС каждой ячейке запоминающего массива соответствует один разряд. Для слов, в которых разряды совпадают с незамаскированными разрядами РАП, блок сравнения БС устанавливает 1 в соответствующих разрядах РС и 0 в остальных разрядах.

Блок формирования результата ассоциативного обращения БР формирует из слова, образовавшегося в РС, сигналы a0, a1, a2, соответствующие случаям:

• $a_0$ – отсутствие слов в ЗМ, удовлетворяющих ассоциативному признаку;

• $a_1$ – наличие одного слова в ЗМ, удовлетворяющего ассоциативному признаку;

• $a_2$ – наличие нескольких (более чем одного) слов в ЗМ, удовлетворяющих ассоциативному признаку.

Формирование содержимого РС и сигналов a0, a1, a2 по содержимому РАП, РМ и ЗМ называется *операцией контроля ассоциации*. Эта операция является составной частью операций чтения и записи.

При чтении:

• производится контроль ассоциации по признаку поиска в РАП;

• проверяются признаки a0, a1, a2. При a0 = 1 считывание не производится из-за отсутствия искомой информации, при a1 = 1 в РИ считывается найденное слово (слово, на которое указывает 1 в регистре совпадений), при a2 = 1 в РИ считывается слово из ячейки, имеющей наименьший номер среди ячеек,

отмеченных 1 в РС. При этом соответствующий разряд в РС сбрасывается в 0. Повторяя эту операцию, можно последовательно считать все слова.

Запись производится в первую свободную ячейку. Для этого:

• отыскиваются свободные ячейки путем выполнения операции контроля ассоциации при РАП = xxx…x0 и РМ = 000…01. При этом свободные ячейки отмечаются 1 в РС;

• для записи выбирается ячейка с наименьшим номером;

• в выбранную ячейку записывается слово, поступившее в РИ с ШИВХ.

Для отыскания свободной ячейки могут использоваться служебные разряды, показывающие, как давно производилось обращение к данной ячейке. Свободной считается либо пустая ячейка, либо та, которая дольше всего не использовалась.

С помощью операции контроля ассоциации можно, не считывая слов из памяти, определить по содержимому РС, сколько в памяти слов, удовлетворяющих ассоциативному признаку, например, реализовать запросы типа: сколько студентов в группе имеют отличную оценку по данной дисциплине. При использовании соответствующих комбинационных схем в ассоциативной памяти могут выполняться достаточно сложные логические операции, такие, как поиск большего (меньшего) числа, поиск максимального (минимального) числа и др.

Для построения ассоциативной памяти необходимы ЗЭ, допускающие считывание без разрушения записанной в них информации. Это связано с тем, что при ассоциативном поиске считывание производится по всему ЗМ для всех незамаскированных разрядов и негде сохранять временно разрушаемую считыванием информацию.

Главное преимущество ассоциативных ЗУ определяется тем, что время поиска информации зависит только от числа разрядов в признаке поиска и скорости опроса разрядов и не зависит от числа ячеек в запоминающем массиве.

Различают два вида поиска информации в ассоциативных ЗУ:

• простой поиск;

• сложный поиск.

При *простом поиске* требуется полное совпадение всех разрядов признака поиска с одноименными разрядами слов, хранящихся в запоминающем массиве.

К *сложным видам поиска* относятся варианты поиска с частичным совпадением. Можно, например, ставить задачу поиска слов с максимальным или минимальным значением ассоциативного признака. Многократная выборка из ассоциативного ЗУ слова с максимальным или минимальным значением ассоциативного признака с исключением его из дальнейшего поиска, по существу, представляет собой упорядоченную выборку информации. Упорядоченную выборку можно обеспечить и другим способом, если вести поиск слов, ассоциативный признак которых по отношению к признаку опроса является ближайшим большим или меньшим значением.

Реализация сложных методов поиска связана с соответствующими изменениями в архитектуре ассоциативного ЗУ, в частности, с усложнением схемы ЗУ и введением в нее дополнительной логики.

Ассоциативная память имеет большую стоимость, так как она содержит достаточно большое множество логических схем (блок сравнения, блок формирования результата ассоциативного обращения). Поэтому ассоциативная память в основном используется для специальных целей, например, для реализации кэш-памяти и виртуальной памяти.

## 26 *Элементы и устройства памяти. Стековая память, ее основные типы. Понятия стек, вершина стека, аппаратно-программный стек и указатель стека.*

*Стековая память* является безадресной памятью с последовательным доступом.

В стековом ЗУ ячейки образуют одномерный массив, в котором соседние ячейки связаны друг с другом разрядными цепями передачи слов. Слова становятся доступными для чтения и записи только в определенном порядке. Каждое хранящееся слово привязано не к конкретной ячейке, а к своему положению относительно других хранящихся слов. Слова могут перемещаться по ячейкам, но при этом сохраняют свою взаимную упорядоченность. Поэтому достаточно обеспечить средства для чтения только определенной ячейки. Конкретное слово считывается в тот момент, когда в процессе перемещения по памяти оно оказывается в ячейке, из которой может производиться чтение. Аналогично достаточно обеспечить средства для записи только в определенную ячейку ЗУ.

В зависимости от того, как перемещаются слова в ЗМ, стековые ЗУ подразделяются на два основных типа:

- память типа очереди или память типа FIFO (First In First Out – первым вошел, первым вышел);
- память магазинного типа или стек.

*Память типа очереди* или *память типа FIFO*. В ЗУ этого типа слова перемещаются всегда в одном направлении от входа к выходу. Данные становятся доступными для считывания в том порядке, в котором производилась запись.

Возможно два варианта организации памяти типа FIFO. В *первом* случае информация записывается в ячейку в начале цепочки и считывается в ее конце. Запись и чтение информации производятся синхронно. ЗУ этого типа используются в качестве задержки на N тактов, где N – количество ячеек в памяти. Во *втором* случае информация записывается в первую свободную

ячейку. Запись и чтение информации производятся асинхронно. ЗУ этого типа используются в качестве буферов с целью согласования различных скоростей информационных потоков двух компонентов микропроцессорной системы.

*Стек*. В ЗУ этого типа слова считываются в порядке, обратном порядку записи, т.е. по правилу LIFO (Last In First Out – последним вошел, первым вышел). Стек можно представить в виде вертикально расположенного массива ячеек. Доступ осуществляется всегда к верхней ячейке, которая называется *вершиной стека*. При записи в вершину стека все слова сдвигаются вниз на одну ячейку, а содержимое нижней ячейки теряется, т.е. стек опускается и происходит операция вталкивания в стек (PUSH). При чтении из вершины стека происходит обратное действие, т.е. стек поднимается и происходит операция выталкивания из стека (POP). При этом становится доступным слово из второй ячейки.

В универсальных МП стековая память организуется с использованием адресной памяти – памяти с произвольным доступом (моделируется на памяти с произвольным доступом). При этом в качестве стека используется часть основной памяти. Такой стек называется *аппаратно-программным стеком*. Это дает дополнительную гибкость, поскольку емкость стека может меняться при необходимости, и максимальный размер стека оказывается ограниченным только объемом основной памяти. Кроме того, перенос стека в память дает экономию аппаратуры.

Для организации стека в памяти с произвольным доступом используется реверсивный счетчик, который называется *указателем стека*. В стандартном стеке в указателе стека всегда находится адрес ячейки памяти, которая соответствует вершине стека. При записи в стек слова сначала указатель стека уменьшается на 1, а затем слово помещается по адресу, полученному в указателе стека. При чтении слова из стека сначала слово извлекается из вершины стека (по адресу, находящемуся в указателе стека), а затем указатель стека увеличивается на 1. В случае использования памяти с побайтовой адресацией при занесении слова в стек и извлечении слова из стека содержимое указателя стека изменяется на величину, равную количеству байт в слове.

Возможны и другие варианты организации стека в памяти с произвольным доступом, отличающиеся тем, что указатель стека указывает на первую свободную ячейку в стеке (на ячейку, расположенную над вершиной стека), а также тем, что стек продвигается в памяти в направлении увеличения адресов.

В универсальных МП стек и стековая адресация используются для временного хранения данных, при организации переходов к подпрограммам и возврате из них, а также при обработке прерываний.

## 27 *Основная память. Получение требуемой разрядности запоминающего устройства. Схемы распределения адресов по банкам памяти. Блочная структура*.

В процессе выполнения программы МП обращается непосредственно только к основной памяти, т.е. основная память представляет собой единственный вид памяти, к которой процессор может обращаться непосредственно (исключение составляют лишь регистры процессора). Основная память – это реально существующая (физическая) память, в которой с точки зрения процессора находятся выполняемые программы и в которой должны размещаться данные, чтобы программа во время работы могла к ним обращаться. Информация, хранящаяся на внешних ЗУ, становится доступной процессору только после того, как будет переписана в основную память. Адреса, существующие в основной памяти, называются *физическими адресами*, а диапазон физических адресов, к которым может обращаться конкретный процессор, называется *пространством физических адресов этого процессора*.

Обычно адресация основной памяти производится с точностью до байта. С целью ускорения этого процесса при обращении к основной памяти запись и считывание нескольких байтов могут осуществляться за один раз.

Основную память образуют запоминающие устройства с произвольным доступом. Основная память может включать в себя два типа устройств: оперативные запоминающие устройства (ОЗУ) и постоянные запоминающие устройства (ПЗУ).

Особенностью современной ОП является ее *блочная организация*.

Необходимость объединения нескольких микросхем памяти возникает по двум причинам:

- разрядность ячеек в микросхемах памяти, как правило, меньше разрядности шины данных МПС;
- емкость основной памяти современных МПС слишком велика, чтобы ее можно было реализовать на базе единственной интегральной микросхемы памяти.

Получение *требуемой разрядности ЗУ* реализуется за счет объединения адресных входов объединяемых микросхем памяти. Информационные входы и выходы микросхем являются входами и выходами модуля ЗУ увеличенной разрядности (рис. 24).

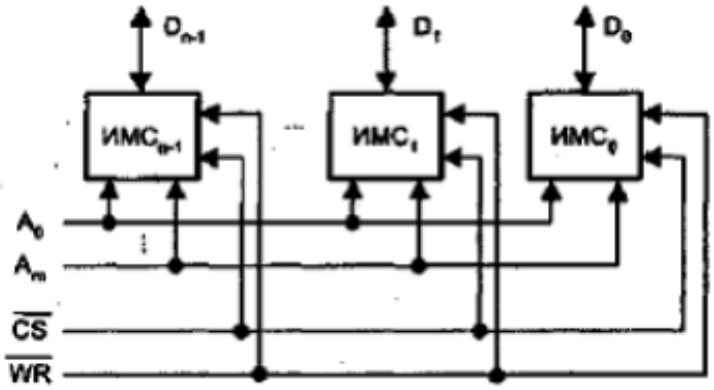



**Рисунок 24 – Получение требуемой разрядности ЗУ**

Полученную совокупность микросхем называют *модулем памяти*. Модулем можно считать и единственную микросхему, если она уже имеет нужную разрядность. Один или несколько модулей образуют *банк памяти*.

Для получения требуемой емкости ЗУ нужно определенным образом объединить несколько банков памяти меньшей емкости. В общем случае основная память МПС практически всегда имеет блочную структуру, т. е. содержит несколько банков.

При использовании блочной памяти, состоящей из B банков, адрес ячейки A преобразуется в пару (b, w), где b – номер банка, w – адрес ячейки внутри банка.

Известны три схемы распределения адресов A по банкам памяти (распределения разрядов адреса A между b и w):

1. блочная – номер банка b определяют старшие разряды адреса A;
2. циклическая – b = A mod B (остаток от деления), w = A div B (частное от деления);
3. блочно-циклическая – комбинация первых двух схем.

Рассмотрение основных структур блочной ОП будем проводить на примере памяти емкостью 512 слов, построенной из четырех банков по 128 слов в каждом.

*Блочная структура (схема)*. Адресное пространство памяти разбивается на группы последовательных адресов, и каждая такая группа обеспечивается отдельным банком памяти (рис. 25). Для обращения к ОП используется 9-разрядный адрес, семь младших разрядов которого (A6-A0) поступают параллельно на все банки памяти и выбирают в каждом из них одну ячейку. Два старших разряда адреса (A8, A7) содержат номер банка. Выбор банка обеспечивается либо с помощью дешифратора номера банка памяти, либо путем мультиплексирования информации (на рис. 25 показаны оба варианта). В функциональном отношении такая ОП может рассматриваться как единое ЗУ, емкость которого равна суммарной емкости составляющих банков, а быстродействие – быстродействию отдельного банка.

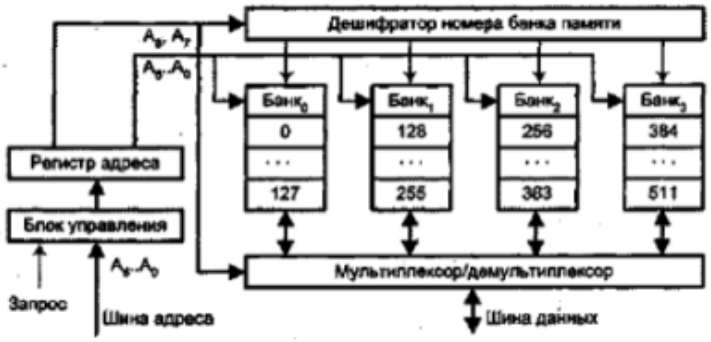



**Рисунок 25 – Структура основной памяти на основе блочной схемы**

28 Основная память. Схемы распределения адресов по банкам памяти. *Циклическая и блочно-циклическая схемы.* Понятие *конфликта по доступу.*

*Циклическая схема*. Кроме возможности наращивания емкости, блочное построение памяти обладает еще одним достоинством – позволяет сократить время доступа к данным. Это возможно благодаря потенциальному параллелизму, присущему блочной организации. Большей скорости доступа можно достичь за счет одновременного доступа к нескольким банкам памяти. Одна из используемых для этого методик называется *расслоением памяти*. В ее основе лежит так называемое *чередование адресов* (address interleaving). Прием чередования адресов базируется на рассмотренном ранее свойстве локальности по обращению, согласно которому последовательный доступ в память обычно производится к ячейкам, имеющим смежные адреса. Суть процедуры чередования адресов состоит в том, что соседние адреса относятся к разным банкам. Чередование адресов обеспечивается за счет использования для распределения адресов между банками памяти циклической схемы. В нашем примере (рис. 26) для выбора банка используются два младших разряда адреса (A1, A0), а для выбора ячейки в банке – 7 старших разрядов (A8-A2).

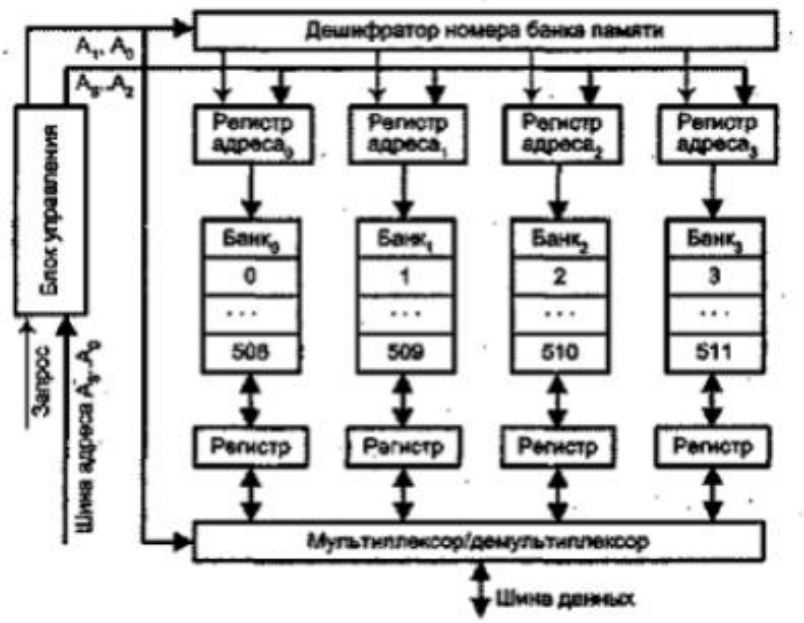



**Рисунок 26 – Структура основной памяти на основе циклической схемы**

Поскольку в каждом такте на шине адреса может присутствовать адрес только одной ячейки, параллельное обращение к нескольким банкам невозможно, однако оно может быть организовано со сдвигом на один такт. Адрес ячейки запоминается в индивидуальном регистре адреса, и дальнейшие операции по доступу к ячейке в каждом банке протекают независимо (рис. 27).


**Рисунок 27 – Организация обращения к основной памяти на основе циклической схемы**

При большом количестве банков среднее время доступа к ОП сокращается почти в B раз (B – количество банков), но при условии, что ячейки, к которым производится последовательное обращение, относятся к разным банкам. Если же запросы к одному и тому же банку следуют друг за другом, каждый следующий запрос должен ожидать завершения обслуживания предыдущего. Такая ситуация называется *конфликтом по доступу*. При частом возникновении конфликтов по доступу метод становится неэффективным.

Обычно B равно 2-16, но в некоторых случаях число банков памяти может достигать 64-128.

Механизм расслоения памяти может использоваться и для повышения надежности памяти. При неисправностях или повреждениях соответствующие банки памяти исключаются из основной памяти с последующим ее перегруппированием, в результате чего работоспособность памяти сохраняется, хотя и с некоторым ухудшением ее параметров

## 29 Кэш-память: определение, структура кэш-памяти, *теги, типы строк и кэш-памяти.*

*Кэш-память* предназначена для повышения быстродействия процесса обращения к основной памяти.

Основная память, как правило, реализуется на относительно медленной и дешевой динамической памяти (DRAM), обращение к которой приводит к простою процессора – появляются такты ожидания. Статическая память (SRAM), построенная, как и процессор, на триггерных ячейках, имеет быстродействие, соизмеримое с быстродействием процессора, и способна сделать ненужными такты ожидания или сократить их количество, но имеет высокую стоимость. Разумным компромиссом для построения экономичных и производительных МПС является иерархический способ организации основной памяти. Идея заключается в сочетании основной памяти большого объема на DRAM с относительно небольшой буферной памятью на основе быстродействующей SRAM, т.е. в использовании двухуровневой памяти, когда между ОП и процессором размещается небольшая, но быстродействующая буферная память. В процессе работы в буферную память копируются те участки ОП, к которым производится обращение со стороны процессора. Производится отображение участков ОП на буферную память и переадресация на нее всех обращений в пределах скопированного участка. Выигрыш в быстродействии достигается за счет ранее рассмотренного свойства локальности.

Для обозначения рассмотренной буферной памяти получил распространение термин *кэш-память* (от английского слова cache – убежище, тайный склад, тайник, заначка), поскольку такая память обычно скрыта от программиста в том смысле, что он не может ее адресовать и может даже вообще не знать о ее существовании.

Кэш является дополнительным быстродействующим хранилищем копий блоков информации из основной памяти, вероятность обращения к которым в ближайшее время велика. Кэш не может хранить копию всей основной памяти, поскольку его объем во много раз меньше основной памяти. Он хранит только ограниченное количество блоков данных. Кроме того, кэшироваться может не вся память, доступная процессору.

Для современных микропроцессоров особенно важно то, что кэш-память (поскольку она небольшого размера) можно разместить внутри кристалла, благодаря чему исчезают потери времени на передачу данных между процессором и памятью.

Кэш-память состоит из (рис. 28): массива данных, справочника или каталога (cache directory) и контроллера (устройства управления).


**Рисунок 28 – Структура кэш-памяти**

В массив данных копируются блоки основной памяти, а их адреса заносятся в каталог. Каталог содержит список текущего соответствия блоков данных областям основной памяти. При каждом обращении к памяти контроллер кэш-памяти по каталогу проверяет, есть ли действительная копия затребованных данных в кэше. Если она там есть, то реализуется *кэш-попадание* (cache hit), и данные берутся из кэш-памяти. Если действительной копии там нет, то реализуется *кэш-промах* (cache miss), и данные берутся из основной памяти и помещаются в кэш-память.

ОП состоит из 2n адресуемых слов, где каждое слово имеет уникальный n-разрядный адрес. При взаимодействии с кэшем эта память рассматривается как M блоков фиксированной длины по k слов в каждом (M = 2n/k). Кэш-память состоит из m блоков аналогичного размера (блоки в кэш-памяти принято называть *строками*), причем их число значительно меньше числа блоков в основной памяти (m << M). При считывании слова из какого-либо блока ОП этот блок копируется в одну из строк кэша. Поскольку число блоков ОП больше числа строк, отдельная строка не может быть выделена постоянно одному и тому же блоку ОП.

С каждой строкой кэша связана информация об адресе скопированного в нее блока основной памяти и ее состоянии. Информация о том, какой именно блок основной памяти занимает данную строку называется *тегом* (tag) и хранится в связанной с данной строкой ячейке *памяти тегов*. В качестве тега обычно используется часть адреса ОП. Здесь же хранится и информация о состоянии строки. Строка может быть *действительной* (valid), если в ней в текущий момент времени хранится (присутствует) копия соответствующего блока основной памяти, или *недействительной*. Строка может *достоверно* отражать соответствующий блок основной памяти или быть *модифицированной* (говорят строка «грязная» – dirty). Таким образом, кроме адресной части тега с каждой строкой кэша связаны биты признаков *действительности* (присутствия) V и *модифицированности* M данных. Память тегов представляет собой каталог или

справочник кэш-памяти. В операциях обмена с основной памятью строка участвует целиком. Такой кэш называется *несекторированным*. Возможен и вариант *секторированного* кэша, при котором одна строка содержит несколько смежных секторов, размер которых соответствует минимальной порции обмена данными кэша с основной памятью. При этом в записи каталога, соответствующей каждой строке, должны храниться биты действительности для каждого сектора данной строки. Секторирование позволяет экономить память, необходимую для хранения каталога при увеличении объема кэша, так как при этом увеличивается количество разрядов каждого элемента каталога, а не количество самих элементов (размер каталога).

30 *Факторы,* влияющие на эффективность применения кэш-памяти: емкость кэш-памяти, размер строки, способ отображения основной памяти на кэш-память.

На эффективность применения кэш-памяти в иерархической системе памяти влияет целый ряд моментов. К наиболее существенным из них можно отнести:

- емкость кэш-памяти;
- размер строки;
- способ отображения основной памяти на кэш-память;
- алгоритм замещения информации в заполненной кэш-памяти;
- алгоритм согласования содержимого основной и кэш-памяти;
- число уровней кэш-памяти.

Емкость кэш-памяти

Выбор емкости кэш-памяти - это всегда определенный компромисс. С одной стороны, кэш-память должна быть достаточно мала, чтобы ее стоимостные показатели были близки к величине, характерной для ОП. С другой – она должна быть достаточно большой, чтобы среднее время доступа в системе, состоящей из основной и кэш-памяти, определялось временем доступа к кэш-памяти. В пользу меньшего размера кэш-памяти имеется больше мотивировок. Так, чем больше емкость кэш-памяти, тем сложнее ее адресация. Как следствие, кэш-память большей емкости работает медленнее по сравнению с кэш-памятью меньшей емкости.

Реальная эффективность использования кэш-памяти зависит от характера решаемых задач, и невозможно заранее определить, какая ее емкость будет действительно оптимальной.

Общая тенденция: по мере увеличения емкости кэш-памяти вероятность промахов сначала существенно снижается, но при достижении определенного значения эффект сглаживается и становится несущественным. Установлено, что

для большинства задач близкой к оптимальной является кэш-память емкостью от 1 до 512 Кбайт.

Размер строки

Еще одним важным фактором, влияющим на эффективность использования кэш-памяти, является размер строки. Когда в кэш-память помещается строка, вместе с требуемым словом туда попадают и соседние слова. По мере увеличения размера строки вероятность промахов сначала падает, так как в кэш, согласно принципу локальности, попадает все больше данных, которые понадобятся в ближайшее время. Однако вероятность промахов начинает расти, когда размер строки становится достаточно большим. Объясняется это тем, что:

• большие размеры строки уменьшают общее количество строк, которые можно загрузить в кэш-память, а малое число строк приводит к необходимости частой их смены;

• по мере увеличения размера строки каждое дополнительное слово оказывается дальше от запрошенного, поэтому такое дополнительное слово менее вероятно понадобится в ближайшем будущем.

Зависимость между размером строки и вероятностью промахов во многом определяется характеристиками конкретной программы, из-за чего трудно рекомендовать определенное значение величины строки. Считается, что наиболее близким к оптимальному является размер строки, равный 4-8 адресуемым единицам (словам или байтам). На практике размер строки обычно выбирают равным ширине шины данных, связывающей кэш-память с основной памятью, или размеру пакета, если процессор поддерживает режим пакетной передачи.

Сущность отображения блока основной памяти на кэш-память состоит в копировании этого блока в какую-то строку кэш-памяти, после чего все обращения к блоку в ОП должны переадресовываться на соответствующую строку кэш-памяти.

Способ отображения должен одновременно отвечает трем требованиям:

1. обеспечивать быструю проверку кэш-памяти на наличие в ней копии блока основной памяти;

2. обеспечивать быстрое преобразование адреса блока ОП в адрес строки кэша;

3. реализовывать достижение первых двух требований наиболее экономичными средствами.

Способы отображения оперативной памяти на кэш-память будем рассматривать на следующем *примере*:

• емкость основной памяти 256 Кслов;

• емкость кэш-памяти 2 Кслова;

• ОП разбивается на блоки по 16 слов в каждом (размер строки кэш-памяти 16 слов).

Для адресации каждого слова основной памяти необходим 18-разрядный адрес (256К = 218). ОП состоит из 256К/16 = 218/24 = 214 = 16384 блоков. При

такой организации 18-разрядный адрес можно условно разделить на две части: младшие 4 разряда определяют адрес слова в пределах блока, а старшие 14 – номер блока. Эти старшие 14 разрядов будем называть *адресом блока* основной памяти.

Известные варианты отображения основной памяти на кэш можно свести к трем видам:

1. прямое отображение;
2. полностью ассоциативное;
3. частично-ассоциативное.

*Прямое отображение*. При прямом отображении адрес строки i кэш-памяти, на которую может быть отображен блок j из ОП, однозначно определяется выражением:

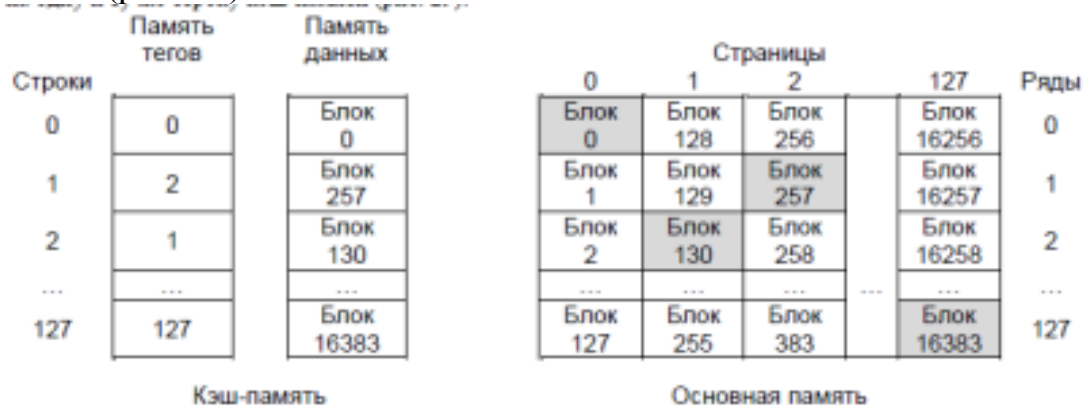
i = j mod m,

где m – общее число строк в кэш-памяти.

В нашем примере i =j mod 128, где адрес строки i может принимать значения от 0 до 127, а адрес блока – от 0 до 16383.

Такое отображение означает, что на строку кэша с номером i отображается каждый m-й блок ОП, если отсчет начинать с блока, номер которого равен i.

В нашем примере на строку кэша с номером i отображается каждый 128-й блок ОП. При этом основная память условно разбивается на 16384/m = 16384/128 = 128 страниц по m = 128 блоков и представляется в виде двухмерного массива блоков, в котором количество рядов равно числу строк в кэш-памяти, и в каждом ряду находятся блоки, претендующие (переадресуемые) на одну и ту же строку кэш-памяти (рис. 29).




При реализации такого отображения 14-разрядный адрес блока основной памяти условно разбивается на два поля: 7-разрядный номер страницы и 7-разрядное поле строки. Поле строки указывает на одну из 128 = $2^7$ строку кэш-памяти, в которую может быть отображен блок с заданным адресом. Номер страницы определяет, какой именно блок из закрепленных за данной строкой кэша, отображается в этой строке. Когда блок фактически заносится в память данных кэша, в память тегов кэш-памяти записывается номер страницы, которой принадлежит этот блок. Таким образом, семь старших разрядов адреса блока ОП служат тегом.

Прямое отображение – простой и недорогой в реализации способ отображения. Основной его недостаток – жесткое закрепление за определенными блоками ОП одной строки в кэш-памяти. Поэтому если программа поочередно обращается к словам из двух различных блоков, отображаемых на одну и тут же строку кэш-памяти, то постоянно будет происходить обновление данной строки и вероятность попадания будет низкой.

*Полностью ассоциативное отображение*. Полностью ассоциативное отображение позволяет преодолеть недостаток прямого отображения, разрешая загрузку любого блока ОП в любую строку кэш-памяти. При этом в адресе ОП выделяются два поля: поле адреса блока и поле слова в блоке. Когда блок фактически заносится в память данных кэша, в память тегов кэш-памяти записывается адрес этого блока (рис. 30). Таким образом, адрес блока ОП служат тегом. Для проверки наличия копии блока в кэш-памяти контроллер кэш-памяти должен одновременно проверить теги всех строк на совпадение с полем адреса блока. Этому требованию наилучшим образом отвечает ассоциативная память.

**Рисунок 30 – Организация кэш-памяти с полностью ассоциативным отображением** Ассоциативное отображение обеспечивает гибкость при выборе строки для вновь записываемого блока. Принципиальный недостаток этого способа – необходимость использования дорогостоящей ассоциативной памяти.

*Множественно-ассоциативное отображение*. Множественно-ассоциативное отображение относится к группе методов частично-ассоциативного отображения. Оно является одним из возможных компромиссов, сочетающим достоинства прямого и ассоциативного способов отображения и, в известной мере, свободным от их недостатков.

Кэш-память (как тегов, так и данных) разбивается на v подмножеств (наборов), каждое из которых содержит k строк (принято говорить, что набор имеет k входов). Зависимость между набором и блоками ОП такая же, как и при прямом отображении: на строки, входящие в набор i, могут быть отображены только вполне определенные блоки основной памяти, в соответствии с соотношением $i = j \bmod v$, где j – адрес блока ОП. В то же время размещение блоков по строкам набора – произвольное, и для поиска нужной строки в пределах набора используется ассоциативный принцип.

Рассмотрим пример 4-входовой кэш-памяти с множественно-ассоциативным отображением (рис. 31). Память данных кэш-памяти разбита на 32 набора по 4 строки в каждом. Память тегов также содержит 32 набора, в каждом из которых может храниться 4 значения тегов по одному на каждую строку набора. 14-разрядный адрес блока ОП представляется в виде двух полей: 9- разрядного поля тега и 5-разрядного поля номера набора. Номер набора однозначно указывает на один из наборов кэш-памяти.

Он также позволяет определить номера тех блоков ОП, которые можно отображать на этот набор. Такими являются блоки ОП, номера которых при делении на $2^5 = 32$ дают в остатке число, совпадающее с номером данного набора кэш-памяти. Так, блоки 0, 32, 64, 96 и т. д. основной памяти отображаются на

набор с номером 0; блоки 1, 33, 65, 97 и т. д. отображаются на набор 1 и т. д. Любой из блоков в последовательности может быть загружен в любую из четырех строк соответствующего набора.


**Рисунок 31 – Организация кэш-памяти с четырехканальным наборно-ассоциативным отображением**

Роль тега выполняют 9 старших разрядов адреса блока ОП, в которых содержится порядковый номер блока в последовательности блоков, отображаемых на один и тот же набор кэш-памяти. Например, блок 65 в последовательности блоков, отображаемых на набор 1, имеет порядковый номер 2 (отсчет ведется от 0).

При обращении к кэш-памяти 5-разрядный номер набора указывает на конкретный набор памяти тегов (это соответствует прямому отображению). Далее производится параллельное сравнение каждого из четырех тегов, хранящихся в этом наборе, с полем тега поступившего адреса, т.е. поиск нужного тега среди четырех возможных осуществляется ассоциативно.

В предельных случаях, когда v = m, k = 1, множественно-ассоциативное отображение сводится к прямому, а при v = 1, k = m – к ассоциативному.

Упрощенно можно считать, что кэш с множественно-ассоциативным отображением представляет собой несколько параллельно и согласовано работающих каналов прямого отображения, в которых строки с одинаковыми номерами образуют соответствующий набор.

В зависимости от способа отображения основной памяти на кэш-память различают три архитектуры кэш-памяти:

1. 1.кэш прямого отображения (direct-mapped cache);
2. 2.полностью ассоциативный кэш (fully associative cache);
3. 3.наборно- (частично- или множественно-) ассоциативный кэш (set associative cache).

*Кэш прямого отображения* имеет самую простую аппаратную реализацию, так как кэш-память имеет структуру обычной прямо адресуемой памяти и необходимо всего одно устройство сравнения. Поэтому такой кэш может иметь большой объем. Кэш-память этого типа в основном применяется во внешнем вторичном кэше, который подключается к системной шине процессора.

Реализация *полностью ассоциативного кэша* является сложной аппаратной задачей, которая решается только для небольших объемов, т.е. полностью ассоциативный кэш из-за своей сложности не может иметь большой объем и

используется, как правило, для вспомогательных целей. Например, в процессорах Intel Pentium MMX полностью ассоциативный кэш используется в блоке страничной переадресации (осуществляет трансляцию линейного адреса в физический страницами размером 4 Кбайт или 4 Мбайт) для построения буфера ассоциативной трансляции TLB (Translation Look aside Buffer), предназначенного для ускорения доступа к интенсивно используемым страницам размером 4 Кбайт: TLB команд – 32 вхождения, TLB данных – 64 вхождения.

Промежуточным между полностью ассоциативным кэшем и кэшем прямого отображения является *наборно-ассоциативный кэш*, который в основном и используется в современных микропроцессорах. Например, в процессоре Intel Core 2 Duo E6400: L1 D- Cache, L1 I-Cache – 32 KB × 2 8-way set associative (8WSA – 8-канальный наборно-ассоциативный кэш), L2 Cache – 2048 KB 8WSA.

31 *Факторы,* влияющие на эффективность применения кэш-памяти: алгоритм замещения информации в заполненной кэш-памяти, алгоритм согласования содержимого основной и кэш-памяти, число уровней кэш-памяти.

Когда кэш-память заполнена, занесение в нее нового блока связано с замещением содержимого одной из строк. При *прямом* отображении каждому блоку основной памяти соответствует только одна определенная строка в кэш-памяти, и никакой иной выбор удаляемой строки здесь невозможен. При *полностью* и *частично ассоциативных* способах отображения требуется какой-либо алгоритм замещения (выбора удаляемой из кэш-памяти строки).

Основная цель стратегии замещения – *удерживать* в кэш-памяти строки, к которым наиболее вероятны обращения в ближайшем будущем, и *заменять* строки, доступ к которым произойдет в более отдаленном времени или вообще не случится. Оптимальным будет алгоритм, который замещает ту строку, обращение к которой в будущем произойдет позже, чем к любой другой строке кэша. Такое предсказание практически нереализуемо, поэтому используются алгоритмы, уступающие оптимальному. В любом случае для достижения высокой скорости алгоритм замещения должен быть реализован аппаратными средствами.

Наиболее распространенными являются четыре алгоритма замещения, рассматриваемые в порядке уменьшения их относительной эффективности.

1. *Алгоритм замещения на основе наиболее давнего использования* (LRU – Least Recently Used). Является наиболее эффективным алгоритм замещения. В соответствии с этим алгоритмом замещается та строка кэш-памяти, к которой дольше всего не было обращения. Проводившиеся исследования показали, что алгоритм LRU работает достаточно хорошо в сравнении с оптимальным алгоритмом.

Наиболее известны два способа аппаратной реализации этого алгоритма.

В *первом* из них с каждой строкой кэш-памяти связывается счетчик. К содержимому всех счетчиков через определенные интервалы времени

добавляется единица. При обращении к строке ее счетчик обнуляется. Таким образом, наибольшее число будет в счетчике той строки, к которой дольше всего не было обращений, и эта строка – первый кандидат на замещение.

*Второй* способ реализуется с помощью очереди, куда в порядке заполнения строк кэш-памяти заносятся ссылки на эти строки. При каждом обращении к строке ссылка на нее перемещается в конец очереди. В итоге первой в очереди каждый раз оказывается ссылка на строку, к которой дольше всего не было обращений. Именно эта строка, прежде всего и заменяется.

2. *Алгоритм, работающий по принципу FIFO* (первый вошел, первый вышел – First In First Out). В соответствии с этим алгоритмом заменяется строка, дольше всего находившаяся в кэш-памяти. Алгоритм легко реализуется с помощью рассмотренной очереди, с той лишь разницей, что после обращения к строке положение соответствующей ссылки в очереди не меняется.
3. *Алгоритм замены наименее часто использовавшейся строки* (LFU – Least Frequently Used). В соответствии с этим алгоритмом заменяется та строка в кэш-памяти, к которой было меньше всего обращений. Аппаратная реализация алгоритма: каждая строка связывается со счетчиком обращений, к содержимому которого после каждого обращения добавляется единица. Главным претендентом на замещение является строка, счетчик которой содержит наименьшее число.
4. *Произвольный выбор строки для замены*. Простейший алгоритм, в соответствие с которым замещаемая строка выбирается случайным образом. Реализовано это может быть, например, с помощью счетчика, содержимое которого увеличивается на единицу с каждым тактовым импульсом, вне зависимости от того, имело место попадание или промах. Значение в счетчике определяет заменяемую строку. Данный алгоритм используется крайне редко.

В процессе вычислений процессор может не только считывать имеющуюся информацию, но и записывать новую, обновляя тем самым содержимое кэш-памяти. С другой стороны, многие устройства ввода/вывода могут напрямую обмениваться информацией с основной памятью (прямой доступ к памяти). В обоих вариантах возникает ситуация, когда содержимое строки кэша и соответствующего блока ОП перестают совпадать. В результате на связанное с основной памятью устройство вывода может быть выдана устаревшая информация, поскольку все изменения в ней, сделанные процессором, фиксируются только в кэш-памяти, а процессор будет использовать старое содержимое кэш-памяти вместо новых данных, загруженных в ОП из устройства ввода.

Для разрешения первой из рассмотренных ситуаций, когда процессор выполняет операцию записи, в системах с кэш-памятью предусмотрены методы обновления основной памяти (политики записи), которые можно разбить на две большие группы:

- метод сквозной записи WT (write through);
- метод обратной записи WB (write back).

По методу *сквозной записи*, прежде всего, обновляется слово, хранящееся в основной памяти. Если в кэш-памяти существует копия этого слова, то она также

обновляется. Если же в кэш-памяти отсутствует нужная копия, то возможны два варианта:

1. *сквозная запись с отображением* – из основной памяти в кэш-память пересылается блок, содержащий обновленное слово;
2. *сквозная запись без отображения* – пересылка блока в кэш-память не производится.

Метод достаточно прост в реализации и легко обеспечивает целостность данных за счет постоянного совпадения копий данных в кэше и основной памяти. Основное достоинство метода сквозной записи состоит в том, что когда строка в кэш-памяти назначается для хранения другого блока, то удаляемый блок можно не возвращать в основную память, поскольку его копия там уже имеется. При этом можно обойтись без признака модифицированности. Недостаток метода состоит в том, что эффект от использования кэш-памяти (сокращение времени доступа) в отношении к операциям записи отсутствует. Данный метод применен в микропроцессорах i486 фирмы Intel.

Определенный выигрыш дает его модификация, известная как метод *отложенной буферизированной сквозной записи*. Информация сначала записывается в кэш-память и в специальный буфер, работающий по схеме FIFO. Запись в основную память производится уже из буфера, а процессор, не дожидаясь ее окончания, может сразу же продолжать свою работу.

Соответствующая логика управления должна заботиться о том, чтобы своевременно переписывать заполненный буфер в ОП (например, во время свободных тактов шины). При использовании буферизации процессор полностью освобождается от работы с ОП.

Согласно методу *обратной записи*, слово заносится только в кэш-память. Если соответствующей строки в кэш-памяти нет, то нужный блок сначала пересылается из ОП, после чего запись все равно выполняется только в кэш-память. При этом строка кэша, в которую произведена запись, помечается как грязная или модифицированная, т.е. требующая выгрузки в основную память.

Только после этой выгрузки (записи в основную память) строка становится чистой, и ее можно будет использовать для кэширования других блоков без потери целостности данных. В основную память данные переписываются только целой строкой. Выгрузка (запись) в основную память откладывается до наступления крайней необходимости (например, обращение к кэшированной памяти другим устройством, замещение строки в кэше новыми данными) или выполняется в свободное время после модификации всей строки. Данный метод сложнее в реализации, но существенно эффективнее (в среднем на 10%), чем сквозная запись, так как позволяет уменьшить количество операций записи в основную память.

Предотвратить несогласованность в ситуации, когда в основную память из устройства ввода, минуя процессор, заносится новая информация и неверной становится копия, хранящаяся в кэш-памяти, позволяют два приема. В первом случае ввод любой информации в ОП автоматически сопровождается соответствующими изменениями в кэш-памяти. Во втором случае ввод любой информации в ОП допускается только через кэш-память.

## 32 Концепция виртуальной памяти. *Механизм динамического преобразования адресов*. Отображение виртуальных адресов в физические. Двухуровневая схема построения памяти.

Термин виртуальная память ассоциируется с возможностью адресовать пространство памяти, большее, чем емкость основной (реальной, физической) памяти конкретного процессора.

При выполнении программы предполагается, что все команды программы и данные содержатся в основной памяти, так как обращение к не существующим в основной памяти командам и данным невозможно. Следовательно, общее пространство памяти, к которому может обращаться программа, ограничивается емкостью основной памяти, т.е. аппаратными средствами. Однако

было бы удобнее составлять программу, не учитывая емкость основной памяти. Суть концепции виртуальной памяти заключается в том, что адреса, к которым обращается выполняющаяся программа (задача, процесс), отделяются от адресов, реально существующих в основной памяти. Те адреса, на которые делает ссылки выполняющаяся программа, называются *виртуальными (логическими) адресами*, а те адреса, которые существуют в основной памяти, называются *реальными* или *физическими*. Диапазон виртуальных адресов, к которым может обращаться выполняющаяся программа, называется *пространством виртуальных*

*адресов этой программы*. Диапазон физических адресов, реально существующих в основной памяти, что определяется конкретным процессором, называется пространством *физических адресов этого процессора*. Для программы создается *единое виртуальное адресное пространство* – виртуальная (логическая) память, в которой физическая ограниченность емкости основной памяти скрыта от нее, т.е. создается видимость произвольной адресации с отсутствием ограничения на емкость используемой памяти, что значительно облегчает программирование.

Несмотря на то, что программы обращаются к виртуальной памяти (виртуальным адресам), процессор в действительности должен работать с реальной памятью. Поэтому во время выполнения программы виртуальные адреса необходимо преобразовывать в реальные (физические), причем это нужно делать быстро, так как в противном случае производительность системы будет снижаться до неприемлемых уровней и тем самым практически сведутся на нет те преимущества, которые призвана обеспечить концепция виртуальной памяти. Соответствие между физическими и виртуальными адресами устанавливается совместно аппаратными средствами и операционной системой, причем это делается прозрачно (невидимо) для программиста.

Для установления соответствия между виртуальными и физическими адресами на практике используется *механизм динамического преобразования адресов* DAT, который обеспечивает преобразование адресов во время выполнения программы. Этот механизм обладает следующим свойством (рис. 32): смежные адреса виртуального адресного пространства не обязательно будут

смежными в физической памяти. Это свойство называют *искусственной смежностью*. Оно позволяет устранять *фрагментацию* физической памяти.

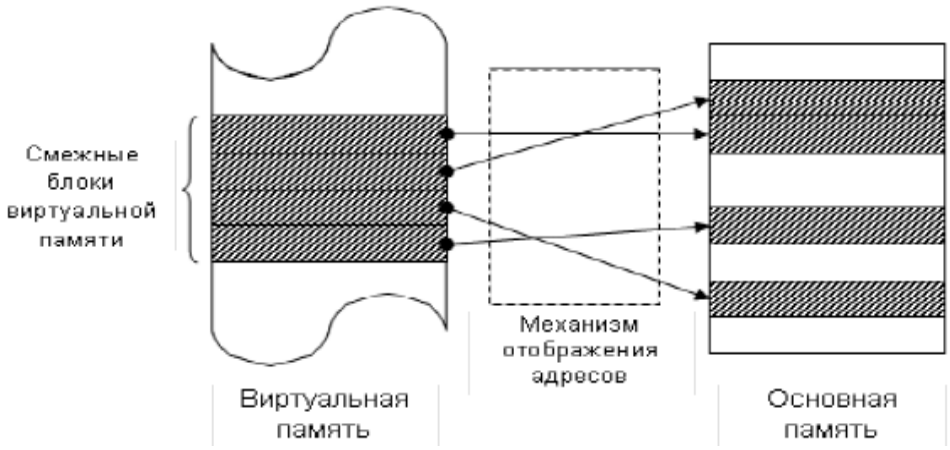



**Рисунок 32 – Отображение виртуальных адресов в физические**

Таким образом программист освобождается от необходимости учитывать размещение своих программ и данных в физической памяти. Он получает возможность писать программы наиболее естественным образом, прорабатывая только детали алгоритма и структуры программы и игнорируя конкретные особенности структуры аппаратных средств, служащих для выполнения программы. При этом микропроцессорная система может рассматриваться только как логическое средство, обеспечивающее реализацию необходимых алгоритмов, а не как физическая система с уникальными характеристиками, часть которых может лишь затруднить процесс проектирования программ.

Концепция виртуальной памяти требует применения двухуровневой схемы построения памяти (рис. 33).


**Рисунок 33 – Двухуровневая схема построения памяти**

*Первый уровень* – это основная память, в которой находятся выполняемые программы и в которой должны размещаться данные, чтобы программа во время работы могла к ним обращаться. С точки зрения виртуальной памяти это реальная или физическая память.

*Второй уровень* – это внешняя память большой емкости, способная хранить программы и данные, которые не могут все сразу уместиться в основной (реальной, физической) памяти ограниченной емкости. С точки зрения виртуальной памяти это вспомогательная память.

Только небольшая часть процедур и данных каждой выполняемой программы, как правило, размещается в первичной памяти одновременно. Остальная часть хранится на устройствах внешней памяти с быстрым прямым доступом. Таким образом, виртуальное адресное пространство размещается во внешней памяти, например на магнитных дисках. Часть этого пространства, необходимая для выполнения программ в данный момент, копируется в основную память.

*Виртуальная память – это способ организации основной памяти микропроцессорной системы большой емкости с помощью внешней памяти* или *метод расширения адресного пространства основной памяти за счет ее совместного использования с внешней памятью*, при котором достигается гибкое динамическое распределение памяти, устраняется ее фрагментация и создаются значительные удобства для работы программистов. Это удается достигнуть без заметного снижения производительности системы ценой усложнения аппаратных средств и операционной системы и процессов их функционирования. Принцип виртуальной памяти предполагает, что пользователь при подготовке своей программы имеет дело не с физической основной памятью, действительно (реально) работающей в составе микропроцессорной системы, а с виртуальной (т.е. кажущейся) одноуровневой памятью, объем которой равен всему адресному пространству, определяемому принятым в процессоре способом адресации памяти (размером адресных полей в форматах команд и адресных регистров).

Например, в 32-разрядных процессора фирмы Intel размер виртуального адресного пространства для одной задачи составляет 64 Тбайт.

## 33 Реализация виртуальной памяти: *страницы, сегменты и комбинированная память.* Схема преобразования виртуального адреса в физический.

Виртуальный адрес при страничной организации – это упорядоченная пара v = (p, d), где p – номер страницы, содержащей адрес v, a d – смещение адреса v относительно начала страницы p. Основная память разделяется на страничные кадры того же самого размера, что и виртуальные страницы. Поступающая в основную память страница может быть помещена в любой свободный страничный кадр.

Динамическое преобразование адресов при страничной организации памяти предусматривает отображение номера виртуальной страницы p на номер страничного кадра основной памяти p’, т.е. номер каждой страницы виртуального пространства ставится в соответствие номеру страницы физического адресного пространства. Взаимосвязь между номерами страниц обоих типов устанавливается *таблицей страниц* (рис. 35). При этом полная таблица страниц размещается в основной памяти. Прежде чем программа начинает выполняться,

операционная система загружает физический адрес таблицы отображения страниц в регистр начального адреса этой таблицы. Этот базовый адрес a таблицы прибавляется к номеру страницы p, образуя адрес основной памяти a + p для строки страницы p в таблице. В этой строке указано, что виртуальной странице p соответствует страничный кадр p'. Затем к значению p' пристыковывается (путем конкатенации) смещение d, так что образуется физический адрес r. Такой подход называется *способом прямого отображения*, так как таблица отображения страниц содержит отдельную строку для каждой страницы виртуальной памяти


**Рисунок 35 – Схема преобразования виртуального адреса в физический при страничной организации**

34 Страничная организация виртуальной памяти. Схема преобразования виртуального адреса в физический при страничной организации. Преобразование адресов страниц на основе ассоциативно-прямого отображения.

Так как таблица страниц ведется в основной памяти, то обращение к ней при преобразовании виртуального адреса в физический занимает относительно много времени. Для ускорения этой процедуры используется метод преобразования адресов страниц на основе *ассоциативно-прямого отображения* (рис. 36). Суть этого метода состоит в использовании специального буфера – ассоциативной памяти, способной хранить небольшую часть полной таблицы отображения страниц. В этот буфер записываются номера наиболее часто используемых в данное время страниц и номера соответствующих им страничных кадров в основной памяти. В ходе преобразования адресов вначале проверяется ассоциативная память, и если в ней обнаруживаются сведения о необходимой странице, преобразование осуществляется без использования полной таблицы

страниц в основной памяти в соответствии с рассмотренным выше механизмом. Такой буфер для высокоскоростного преобразования адресов называется *буфером динамической трансляции* или *буфером ассоциативной трансляции* адресов виртуальной памяти TLB – Translation Look aside Buffer.

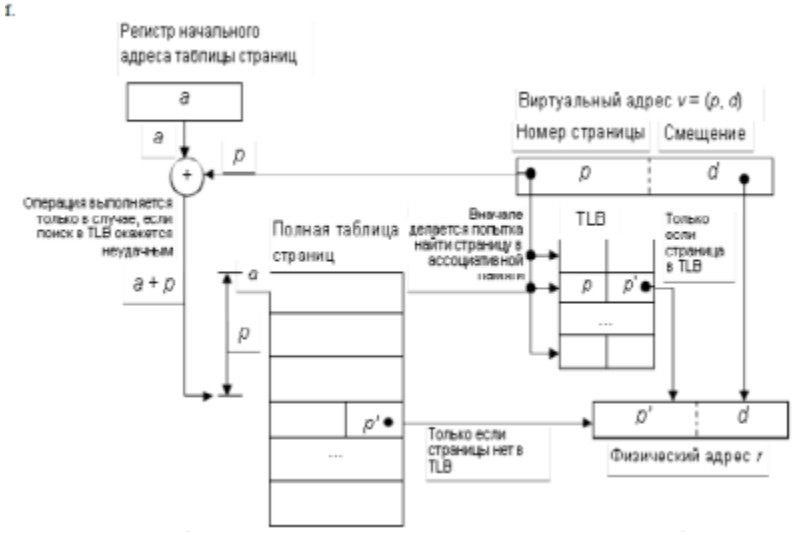



**Рисунок 36 – Преобразование адресов страниц на основе ассоциативно-прямого отображения**

35 Страничная организация виртуальной памяти. *Буфер динамической трансляции, многоуровневое разбиение на страницы*, преобразование адресов страниц на основе двухуровневой схемы.

Расширение виртуального пространства влечет за собой увеличение таблицы страниц. Одним из способов решения этой проблемы является *многоуровневое разбиение на страницы*. Суть этого разбиения состоит в том, что одномерное виртуальное пространство подразделяется на два уровня – разделов и страниц, а преобразование виртуального адреса производится по

двухуровневой таблице (рис. 37). На первом уровне находится таблица разделов, которая содержит ссылки на таблицы страниц разделов. На втором уровне находятся таблицы страниц разделов. Такой подход позволяет экономить объем памяти, выделяемый для информации отображения адресов, так как можно вести таблицы страниц не для всех разделов, в отличие от одноуровневого подхода, который требует хранения в основной памяти полной таблицы всех страниц виртуального пространства.


**Рисунок 37 – Преобразование адресов страниц на основе двухуровневой схемы**

36 Сегментная организация виртуальной памяти. Комбинированная сегментно-страничная организация виртуальной памяти. Преобразование адресов.

В этом случае виртуальное пространство разбивается на блоки по логическим признакам, устанавливаемым программистом. Такое разбиение называется *сегментацией*, а блоки – *сегментами*. Сегменты используются для размещения *логических объектов*, например, программы (подпрограммы) или структуры данных и в отличие от страниц имеют переменную длину (размер).

Каждый сегмент может иметь именно тот размер, который необходим для размещения логического объекта. Не обязательно, чтобы все сегменты выполняющейся программы одновременно находились в основной памяти или были в ней смежными.

Программа может выполняться, если в основной памяти находится, как минимум, ее текущий сегмент.

Виртуальный адрес при сегментной организации представляет собой упорядоченную пару v = (s, d), где s – номер сегмента, содержащего адрес v, а d – смещение адреса v относительно начала сегмента s. Динамическое преобразование адресов при сегментной организации памяти предусматривает отображение номера сегмента s на начальный (базовый) адрес сегмента $A_S$ в основной памяти, т.е. номер каждого сегмента виртуального пространства ставится в соответствие положению сегмента в физическом адресном пространстве. Взаимосвязь между номерами сегментов и их базовыми адресами устанавливается *таблицей сегментов*. При этом полная таблица сегментов размещается в основной памяти. Преобразование адресов может выполняться практически точно так же, как и при страничной организации, за исключением того, что физический адрес получается в результате сложения базового адреса сегмента и смещения. Поскольку

сегменты имеют переменную длину, смещение d необходимо контролировать, чтобы оно не выходило за пределы сегмента.

Так как сегменты являются логическими объектами, можно организовать защиту информации и управление для коллективного использования сегментированной информации, что реализуется более естественным образом по сравнению со страничной организацией. Для этого выполняемым программам предоставляются различные сочетания прав доступа для чтения, записи и выполнения при обращении к различным сегментам.

Комбинированная сегментно-страничная организация виртуальной памяти

Виртуальный адрес представляет собой упорядоченную тройку v = (s, p, d), где s – сегмент, в котором находится виртуальный адрес v, p – страница в сегменте s, содержащая адрес v, d – смещение адреса v в странице p. Преобразование адресов осуществляется в два этапа:

- строка таблицы сегментов указывает на таблицу страниц;
- строка таблицы страниц в свою очередь указывает на страничный кадр.

Преобразование адреса при комбинированной сегментно-страничной организации подобно на преобразование адреса при двухуровневой страничной организации. Однако эти два способа организации виртуальной памяти принципиально отличаются друг от друга. В первом случае сегменты являются самостоятельными логическими объектами. Во втором случае разбиение страниц на разделы является условным и используется для уменьшения объема памяти, требуемой для хранения информации отображения адресов.

Кроме комбинированной сегментно-страничной организации виртуальной памяти в современных микропроцессорах используется и совместное применение сегментной и страничной организаций памяти. В этом случае сегментный и страничный механизмы работают независимо, что позволяет в свою очередь независимо управлять памятью на логическом и физическом уровнях. С помощью сегментного механизма виртуальное пространство, состоящее из сегментов, преобразуется в линейное пространство, которое затем с помощью страничного механизма путем разбиения на страницы преобразуется в физическое пространство.

## 37 Архитектура подсистемы ввода/вывода микропроцессорной системы**.** *Интерфейсы: системный, ввода/вывода. Пространство доступа, протокол обмена, типы регистров.*

Подсистема ввода/вывода отвечает за связь с устройствами ввода/вывода.

Обычно употребляется термин *периферийные устройства* (ПУ), которые подразделяются на *устройства ввода/вывода* (УВВ – клавиатура, мышь, принтер, монитор и др.) и *внешние запоминающие устройства* (ВЗУ). Внешние запоминающие устройства занимают в микропроцессорной системе особое

положение. С точки зрения выполняемой функции они относятся к подсистеме памяти (внешняя память, предназначенная для хранения информации, совместно с основной памятью может использоваться для организации виртуальной памяти), а связь с ними осуществляется так же, как и с устройствами ввода/вывода, т.е. с помощью подсистемы ввода/вывода. Поэтому при рассмотрении подсистемы ввода/вывода все периферийные устройства будем называть устройствами ввода/вывода.

Связь устройств микропроцессорной системы друг с другом осуществляется с помощью специальных совокупностей средств и правил, которые называются *интерфейсами*. При любой форме взаимодействия с микропроцессорной системой аппаратура ввода/вывода обычно состоит из собственно устройства ввода/вывода и устройства управления этим устройством ввода/вывода – контроллера устройства ввода/вывода. Интерфейс между процессором и контроллером устройства ввода/вывода называется *системным интерфейсом*, а интерфейс между контроллером и устройством ввода/вывода – *интерфейсом ввода/вывода*.

Системный интерфейс, как правило, является общим для всех видов устройств ввода/вывода, а интерфейсы ввода/вывода специализированы для конкретных видов устройств ввода/вывода.

С точки зрения программиста, работающего на уровне машинных команд, подсистему ввода/вывода можно представить в виде пространства ввода/вывода IOS и набора команд ввода/вывода, обеспечивающих к нему доступ. Организация пространства ввода/вывода подобна организации пространства памяти: IOS организовано в виде набора n-разрядных ячеек – портов, каждый из которых может быть адресован независимо от других.

Между микропроцессором и периферийными устройствами происходит обмен информацией двух типов:

1. служебной;
2. собственно данными.

Служебная информация от МП инициирует действия, связанные с обменом данными, и передается с помощью *управляющих слов* CW (Control Word). Служебные сообщения от периферийных устройств информируют МП о их текущем состоянии и называются *словами состояния* SW (Status Word). В отличие от них данные передаются с помощью *слов данных* DW (Data Word).

Объем служебной информации, которой обмениваются периферийные устройства и микропроцессор, а также ее интерпретация зависят от типа периферийного устройства. Для наиболее простых устройств, таких как переключатели (кнопки) или светодиодные индикаторы, служебная информация не нужна. В других случаях управляющая информация и данные о состоянии устройства могут иметь большой объем. При этом каждое ПУ воспринимает определенный, присущий только ему набор команд управления. Поэтому передача служебной информации осуществляется путем мультиплексирования шины данных: в одни моменты времени она используется для передачи данных, в другие моменты – для передачи служебной информации. При этом для связи с периферийным устройством отводится ряд портов ввода/вывода, через которые и проходит вся информация: управляющая, слова состояния и непосредственно

данные. С точки зрения программиста множество портов ввода/вывода, связанных с данным периферийным устройством, образует *пространство доступа к этому периферийному устройству*.

Основу контроллера ПУ составляют регистры, которые служат для хранения передаваемой информации. Взаимодействие микропроцессора с этими регистрами осуществляется через порты ввода/вывода из пространства доступа к ПУ. Регистры и порты тесно связаны, иногда их трудно отделить друг от друга. Зачастую их отождествляют друг с другом. В этом смысле каждый регистр имеет свой адрес. Под адресом регистра понимается адрес порта, через который осуществляется доступ к этому регистру. В контроллере ПУ используются регистры четырех типов в зависимости от типа информации, для хранения которой они предназначены:

1. регистр входных данных или входной регистр (доступен микропроцессору только по чтению);
2. регистр выходных данных или выходной регистр (доступен микропроцессору только по записи);
3. регистр состояния (доступен микропроцессору только по чтению). Содержит информацию о текущем состоянии ПУ (включено/выключено, готово/не готово к обмену данными, ошибка и т.п.);
4. регистр управления (доступен микропроцессору только по записи). Служит для приема из МП команд для ПУ.

В контроллерах сложных ПУ регистров каждого типа может быть несколько. В контроллерах, предназначенных для подключения простых ПУ, удается совместить в один регистры состояния и управления, что позволяет сократить количество используемых в контроллере портов ввода/вывода, а следовательно, и адресов, выделенных для данного ПУ.

## 38 Способы организации передачи данных между системой и устройствами ввода/вывода. Программно-управляемый обмен: прямой, условный, обмен с прерыванием программы.

В микропроцессорных системах используются два основных способа организации передачи данных между системой и устройствами ввода/вывода (рис. 38):

1. программно-управляемый обмен;
2. прямой доступ к памяти (ПДП).
3. Программно-управляемый обмен


**Рисунок 38 – Способы организации передачи данных**

*Программно-управляемый обмен* – это обмен, управляемый программой, т.е. когда процедуры обмена информацией с периферийным устройством инициируются и выполняются непосредственно программой, реализуемой процессором через его регистры. Программно-управляемый обмен осуществляется при непосредственном участии и под управлением процессора (см. рис. 38).

С точки зрения аппаратных затрат программно-управляемый обмен является наиболее эффективным типом обмена, поэтому он находит самое широкое применение в различных микропроцессорных системах.

Программно-управляемый обмен может осуществляться одним из трех способов:

1. прямой (безусловный, синхронный) обмен;
2. условный (асинхронный) обмен (обмен по условию);
3. обмен с прерыванием программы.

В зависимости от используемого способа обмена программно-управляемый ввод/вывод называется соответственно

прямым вводом/выводом;

условным вводом/выводом;

вводом/выводом по прерыванию.

*Прямой ввод/вывод*. Использ-ся только для процессов, строго фиксированных во времени и полностью определенных. Процедуры ввода или вывода выпол-тся независимо от состояния периферийного устройства. Процедуры прямого ввода/вывода в чистом виде возможны только при работе с всегда готовыми к обмену простейшими устройствами, для которых точно известно время выполнения операции (например, индикатор). Они являются составными элементами более сложных процедур программно-управляемого обмена – условного ввода/вывода и ввода/вывода по прерыванию.

Прямой ввод/вывод реализ-ся с помощью портов пространства IOS. Порты ввода/вывода явл-ся простейшими схемными элементами, на основе которых реализуется связь с периферийными устройствами. Они образуют первый, наиболее близкий к МП уровень аппаратных средств подсистемы ввода/вывода. В простейшем случае этот уровень явл-ся единственным.

*Условный ввод/вывод*. Прямой ввод/вывод явл-ся наиб. простым видом обмена, требующим минимальных затрат аппаратных и программных средств.

Однако, как правило, скорость работы периферийных устройств во много раз ниже скорости работы МП, что приводит к проблеме синхронизации обмена. Поэтому прежде чем приступить к чтению новых данных из порта ввода, необходимо удостовериться, что ПУ готово предоставить или уже предоставило эти данные. Иначе операция сведется к вводу недействительных или старых данных. Аналогичная ситуация складывается и при выводе данных, когда требуется проверка готовности ПУ к приему новых данных. В противном случае неразрешенный со стороны ПУ вывод может привести к потере данных.

Типичное решение проблемы синхронизации обмена:использование условного ввода/вывода. При условном вводе/выводе операции обмена сопровождаются специальным сигналом готовности RDY, вырабатываемом периферийным устройством и входящим в состав его слова состояния SW. Этот сигнал служит для информирования МП о готовности периферийного устройства принять или передать новые данные. После завершения операции ввода/вывода сигнал готовности (соответствующий разряд слова состояния SW) должен быть снят и выставлен заново только при новой готовности к обмену. С этой целью периферийное устройство следует проинформировать об окончании операции, для чего используется включенный в управляющее слово CW сигнал подтверждения ACK. Протокол обмена служебной информацией такого типа называется *квитированием*. Он обеспечивает надежную асинхронную передачу данных со скоростями, определяемыми периферийным устройством. Состояние сигнала готовности RDY может быть определено микропроцессором путем чтения регистра состояния, в котором находится слово состояния ПУ, через соответствующий порт ввода. Сигнал подтверждения ACK может формироваться микропроцессором путем записи управляющего слова в регистр управления через соответствующий ему порт вывода. Такой вариант формирования сигналов квитирования называется *программным квитированием*. В сравнении с прямым условный ввод/вывод с программным квитированием связан с увеличением аппаратных затрат, а также с потерями времени МП на ожидание готовности периферийного устройства. Однако это наиболее распространенный вид обмена с периферийными устройствами. Он используется в системах, где эффективность не связана с ожиданиями.

Признаком окончания операции служит само обращение к порту данных. Это упрощает как схему порта, так и процедуру обмена, освобождая пользователя от работы с управляющим словом. При этом на входе порта ввода предусматривается регистр- защелка, фиксирующий входные данные по стробу STB, генерируемому периферийным устройством. Запись во входной буфер устанавливает флажок готовности *входной буфер полный* IBF (Input Buffer Full), инициируя операцию ввода. Флажок сбрасывается автоматически при чтении МП содержимого входного буфера. При выводе роль флажка готовности выполняет флажок *выходной буфер пустой* OBE (Output Buffer Empty). Сигналом окончания операции вывода и установки флажка OBE служит сигнал подтверждения ACK, генерируемый периферийным устройством.

*Ввод/вывод по прерыванию*. Эффективность работы микропроцессорной системы может быть существенно повышена, если отказаться от

малопроизводительного ожидания готовности ПУ на программном уровне и передать эту функцию специальным аппаратным средствам. В это время процессор может выполнять некоторую полезную работу, связанную с обработкой данных или обслуживанием других ПУ. При готовности приступить к очередной операции ввода/вывода ПУ посылает в микропроцессор запрос на прерывание, по получении которого МП временно приостанавливает (прерывает) выполнение текущей программы и передает управление специальной подпрограмме, организующей нужный вид обмена данными. После обслуживания ПУ микропроцессор возвращается к прерванной программе, продолжая ее с момента приостановки. Обслуживание прерываний осуществляется в незаметном для основной программы режиме, поэтому их наличие прямо не влияет на работу последней, за исключением времени ее исполнения.

## 39 Организация прерываний в микропроцессорной системе. Последовательность действий при обработке прерывания.

*Системой прерываний* называется совокупность специальных аппаратных средств, осуществляющих прием запроса на прерывание и переход к подпрограмме обработки этого запроса, команд и программ обслуживания запросов прерывания. Кроме обслуживания ПУ (выполнения обмена) в микропроцессорной системе существуют и другие события, которые могут вызвать прерывание процессора. Типы прерываний зависят от конкретной микропроцессорной системы. Прерывания распадаются на два основных класса:

внешние;

внутренние.

*Внешние прерывания*. Вызываются асинхронными событиями, которые происходят вне прерываемой программы. Например, прерывания от таймера или подсистемы ввода/вывода.

*Внутренние прерывания*. Вызываются событиями, которые связаны с работой процессора и являются синхронными с его операциями. Например:

арифметическое переполнение, попытка деления на 0, переполнение или исчезновение порядка при выполнении операций с плавающей запятой;

внутренние прерывания происходят при обращении к защищенным или несуществующим ячейкам памяти, а также к отсутствующему сегменту или странице;

прерывания вызывает попытка использовать незадействованный код операции, а также попытка использовать привилегированные команды в пользовательском режиме;

внутренние прерывания могут вызываться также сбоями системы, например, ошибкой четности;

при выполнении специальных команд.

Прерывание программы не должно оказывать на прерванную программу никакого влияния кроме увеличения времени ее выполнения за счет

приостановки на время выполнения подпрограммы обработки прерывания. Для этого после приема запроса на прерывание состояние МП необходимо сохранить. Для того чтобы прерванная программа могла быть продолжена после обслуживания очередного запроса на прерывание с того места, на котором она была приостановлена, состояние процессора должно быть восстановлено. Всякий раз, когда процессор воспринимает запрос на прерывание, он активизирует процедуру обслуживания, передавая ее стартовый адрес в программный счетчик. Чтобы не потерялось старое содержимое программного счетчика, которое является адресом возврата в прерванную программу, оно должно быть автоматически сохранено. Лучше всего для этой цели использовать системный стек, тогда возврат к прерванной программе будет заключаться в передаче управления по адресу, находящемуся на вершине стека. Обычно аппаратными средствами обработки прерывания автоматически сохраняется не только содержимое программного счетчика, но и регистра состояния процессора, а также его программно-доступных регистров.

Таким образом, хотя существует несколько различных способов обработки прерывания, следующая последовательность действий присуща большинству микропроцессорных систем:

фиксируются характеристики произошедшего прерывания (тип прерывания);

сохраняется состояние прерванной программы – состояние процессора;

анализируется тип прерывания и передается управление соответствующей подпрограмме обработки этого прерывания;

обрабатывается прерывание – выполняется соответствующая подпрограмма;

восстанавливается состояние процессора, что приводит к возобновлению выполнения прерванной программы.

Во всех системах прерываний предусмотрен механизм программно-управляемой блокировки запросов, который реализуется с помощью набора флажков, разрешающих или запрещающих восприятие запросов на прерывание процессором. Эти флажки образуют маску прерываний и либо упаковываются в отдельный регистр маски прерываний, либо входят в состав регистра флагов (признаков) процессора.

Существуют две системы прерываний:

радиальная система прерываний;

векторная система прерываний.

## 40 Радиальная система прерываний. Формирование запроса прерывания в контроллере периферийного устройства. *Поллинг.* Одно- и многоуровневые системы прерываний.

Физический интерфейс простейшей системы прерываний может быть представлен единственной линией запроса на прерывание IRQ (Interrupt ReQuest).

Для программиста такая система прерываний представляется в виде отдельной точки входа в процедуру обслуживания.

Формирование запросов на прерывание – запросов ПУ на обслуживание – происходит в контроллерах соответствующих ПУ (рис. 39).

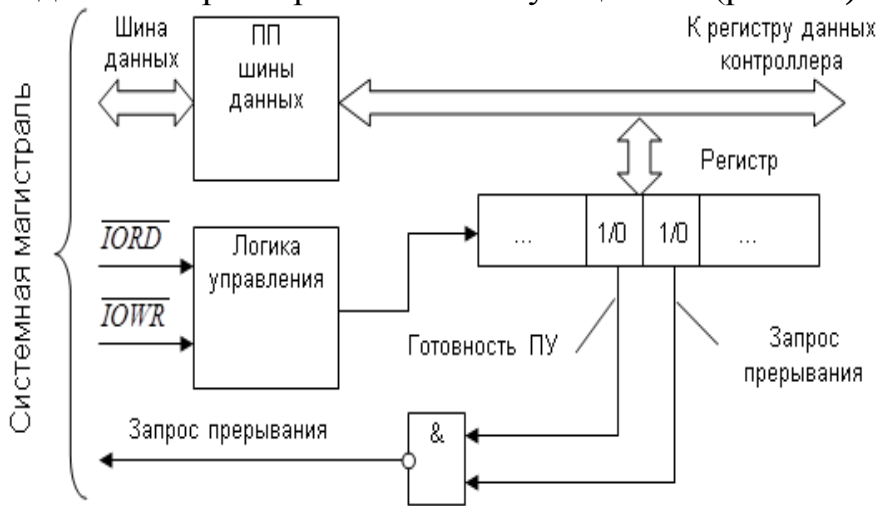



Рисунок 39 – Формирование запроса прерывания в контроллере ПУ

В простейших случаях в качестве сигнала запроса прерывания может использоваться *сигнал готовности* ПУ, поступающий из контроллера ПУ с выхода соответствующего разряда регистра состояния. Однако такое простое решение обладает существенным недостатком – процессор не имеет возможности управлять прерываниями, т.е. разрешать или запрещать их для отдельных ПУ. В результате организация обмена данными в режиме прерывания с несколькими ПУ существенно усложняется. Поэтому регистр состояния контроллера ПУ дополняют еще одним разрядом – *разрешение прерывания*. Запись 1 или 0 в этот разряд регистра состояния производится программным путем (прямой вывод) по одной из линий шины данных системной магистрали.

Управляющий сигнал *запрос прерывания* от ПУ формируется с помощью схемы совпадения только при наличии единиц в

*разрядах готовность* ПУ и *разрешение прерывания* регистра состояния контроллера ПУ.

При необходимости обслуживания нескольких ПУ в такой системе сигналы запросов на прерывание от всех ПУ поступают на один вход процессора, при этом запросы объединяются по схеме «монтажное ИЛИ». В этом случае возникает проблема идентификации ПУ, запросившего обслуживания. Данная проблема решается с помощью специальной процедуры, называемой *поллингом*. Функция поллинга состоит в последовательном опросе состояния всех ПУ и выявлении готовых к обслуживанию. В данном случае поллинг реализуется *программным способом* путем анализа разряда готовности регистров состояния контроллеров ПУ. Организация прерываний с программным опросом готовности предполагает наличие в памяти микропроцессорной системы *единой подпрограммы обслуживания прерываний от всех ПУ*. Обслуживание ПУ с помощью единой подпрограммы обработки прерываний производится следующим образом. После выполнения очередной команды основной программы процессор

проверяет наличие запроса на прерывание от ПУ. Если запрос прерывания есть и в процессоре прерывание разрешено, то МП переключается на выполнение подпрограммы обработки прерываний. После сохранения содержимого регистров процессора, используемых в подпрограмме, начинается последовательный опрос регистров состояния контроллеров всех ПУ, работающих в режиме прерывания. Как только подпрограмма обнаружит готовое к обмену ПУ, сразу выполняются действия по его обслуживанию. Завершается подпрограмма обработки прерывания после опроса готовности всех ПУ восстановлением содержимого регистров процессора.

Приоритет ПУ в системе с программным опросом готовности ПУ однозначно определяется порядком их опроса в подпрограмме обработки прерывания. Чем раньше в подпрограмме опрашивается готовность ПУ, тем меньше время реакции на его запрос и выше приоритет. *Приоритет запроса* отражает важность и срочность его обслуживания. При назначении приоритетов учитываются частота появления запроса, длительность процесса обслуживания, последствия задержки обслуживания и др.

Необходимость проверки готовности всех ПУ существенно увеличивает время обслуживания тех ПУ, которые опрашиваются последними. Это является основным недостатком рассматриваемого способа организации прерываний. Поэтому обслуживание прерываний с программным опросом готовности ПУ используется только в тех случаях, когда отсутствуют жесткие требования на время обработки запросов прерываний ПУ.

Для увеличения числа одновременно обслуживаемых источников прерываний в процессоре вводится несколько линий с фиксированными стартовыми адресами подпрограмм обслуживания. Такие линии называются *радиальными,* а система прерываний – *радиальной системой прерываний*. В радиальной системе прерываний проблема идентификации ПУ решается путем выделения каждому ПУ своей линии для передачи запроса на прерывание. В зависимости от того, по какой линии приходит запрос на прерывание, управление передается подпрограмме обслуживания соответствующего ПУ.

Обычно часть радиальных линий отводится для приема внутренних прерываний процессора, отражающих его критические состояния и требующих немедленного обслуживания. Остальные отводятся для приема внешних (по отношению к процессору) запросов.

В зависимости от числа запросов, одновременно находящихся на обслуживании, различают одно- и многоуровневые системы прерываний.

В *одноуровневой* системе прерываний в каждый момент времени допускается только один подтвержденный запрос, т.е. на обслуживании одновременно находится только один запрос. При этом обработка всех других запросов откладывается до окончания текущего обслуживания. Если несколько устройств одновременно запросили обслуживание, процессор выбирает только одно из них на основании приоритета каждого из запросов. Наиболее практичной и естественной является система прерываний с *фиксированными линейно упорядоченными приоритетами*. В такой системе все приоритеты строго упорядочены, что обеспечивает однозначный выбор одного из них. Приоритеты

запросов могут динамически изменяться по заданномуалгоритму (аппаратно или программно), однако в каждый момент времени все приоритеты остаются строго упорядоченными. Широко применяемым алгоритмом изменения приоритетов является *циклический*. В соответствии с ним после каждого очередного обслуживания запроса происходит циклический сдвиг приоритетов с присвоением минимального приоритета только что обработанному запросу. Такая схема приводит к равномерному распределению внимания процессора между всеми запросами. Она может быть использована при обслуживании группы одинаковых устройств, когда выделение какого либо из них нежелательно или их нельзя однозначно упорядочить по приоритетам.

*Многоуровневая* система разрешает многократные (по числу уровней) прерывания одних процедур обслуживания другими. Каждый уровень связывается с определенной радиальной линией и строго упорядоченным приоритетом. Процедура обслуживания некоторого уровня может быть прервана только запросами более высокого уровня. Фоновую работу процессора связывают с минимальным приоритетом, ее может прервать любой запрос. Для работы многоуровневой системы прерываний необходимо знать *приоритет текущей процедуры*. Для этого вводится понятие приоритет процессора. *Приоритет процессора* всегда равен приоритету текущей выполняемой процедуры: при выполнении фоновой задачи процессор имеет минимальный приоритет, при выполнении процедуры обслуживания запроса некоторого уровня приоритет процессора принимает значение, равное приоритету этого уровня. При поступлении запроса на прерывание его приоритет сравнивается с приоритетом процессора и, если приоритет запроса выше, то такой запрос прерывает работу процессора. При этом приоритет процессора повышается, принимая значение, равное приоритету запроса.

## 41 Векторная система прерываний. Вектор прерывания. Методы формирования вектора прерывания. Формирование вектора прерывания средствами периферийного устройства. Векторная система прерываний на основе контроллера прерываний.

Повышение эффективности системы прерываний связано с передачей функции идентификации ПУ, запросившего обслуживания, внешним по отношению к процессору средствам. В *векторной системе прерываний* ПУ, запросившее обслуживание, само идентифицирует себя с помощью *вектора прерывания*, который принимается МП. В общем случае для передачи вектора прерывания необходима специальная шина. Однако она, как правило, физически совмещается с шиной данных системной магистрали, при этом ввод вектора прерывания осуществляется в специальном цикле магистрали, который

называется *циклом подтверждения прерывания*. Такое совмещение требует включения в шину управления линии подтверждения прерывания INTA, по которой передается сигнал от процессора, разрешающий выдачу вектора прерывания в ответ на запрос прерывания от ПУ. МП, получив вектор прерывания, сразу переключается на выполнение требуемой подпрограммы обработки прерывания. Так же как и радиальная система, векторная система прерываний предполагает наличие для каждого ПУ собственной подпрограммы обработки прерывания. При этом вектор прерывания определяет, какой подпрограмме обработки прерывания процессор должен передать управление.

Вектор прерывания может представлять собой:

полную команду вызова подпрограммы вместе с адресом подпрограммы обработки прерывания;

адрес подпрограммы обработки прерывания;

указатель на адрес подпрограммы обработки прерывания. В качестве указатель может использоваться либо адрес, по которому в памяти хранится адрес подпрограммы обработки прерывания (иногда такой указатель называют адресом вектора прерывания), либо тип прерывания.

Существует два подхода к построению векторной системы прерываний, которые различаются используемым методом формирования вектора прерывания. Первый подход использует метод децентрализованного управления – определение запроса с наибольшим приоритетом и формирование вектора прерывания осуществляется непосредственно ПУ. Второй подход использует метод централизованного управления и состоит в передаче функции формирования вектора прерывания специальному устройству – контроллеру прерывания.

*При формировании вектора прерывания средствами ПУ* логика работы программного поллинга переносится на аппаратные средства – определение наиболее приоритетного запроса осуществляется с помощью аппаратного опроса готовности ПУ. Такой подход называется *аппаратным поллингом*. Линии запросов от всех ПУ объединяются по схеме «монтажное ИЛИ» и подключаются к общей линии запроса прерывания IRQ процессора (рис. 40).

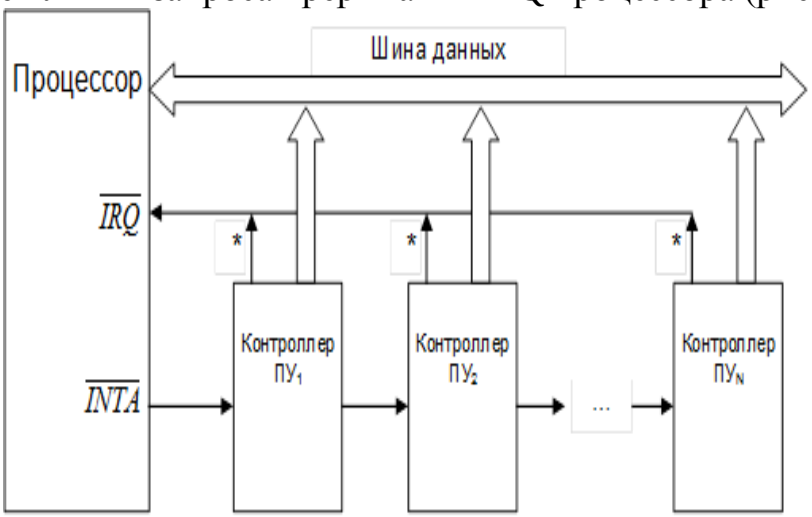



**Рисунок 40 – Формирование вектора прерывания средствами ПУ**

Процессор при поступлении в него по линии IRQ запроса прерывания формирует управляющий сигнал подтверждения прерывания INTA, который

поступает сначала в контроллер ближайшего к процессору ПУ. Если это ПУ не требовало обслуживания, то его контроллер пропускает сигнал подтверждения прерывания на следующий контроллер, иначе дальнейшее распространение сигнала прекращается и контроллер выдает в шину данных вектор прерывания. Такая схема носит ярко выраженный шлейфовый характер. Одна линия подтверждения прерывания проходит последовательно через контроллеры ПУ и образует последовательную приоритетную структуру, называемую *дейзи-цепочкой*. Приоритет определяется физическим положением каждого ПУ. Ближайшее к процессору ПУ имеет наибольший приоритет.

Дейзи-цепочка имеет два преимущества. Во-первых, в системной магистрали нужна только одна линия запроса прерывания (одна линия запроса используется и в системе с программным опросом готовности ПУ, однако аппаратный опрос готовности ПУ производится гораздо быстрее). Во-вторых, в систему можно ввести новое ПУ с любым требуемым приоритетом, просто подключая его в нужную физическую позицию. Количество ПУ в системе ограничивается только числом векторов прерываний. Однако дейзи-цепочка медленнее параллельного способа, реализуемого в контроллере прерываний, так как сигнал подтверждения прерывания последовательно распространяется через каждое ПУ. Еще один недостаток шлейфовой структуры – трудность управления приоритетами. ПУ, стоящие в дейзи-цепочке ближе к процессору, обладают более высоким приоритетом, поэтому изменение приоритетов требует изменения последовательности включения ПУ, что во многих случаях затруднительно. Наиболее эффективно векторная система прерываний реализуется с помощью контроллера прерываний (рис. 41)

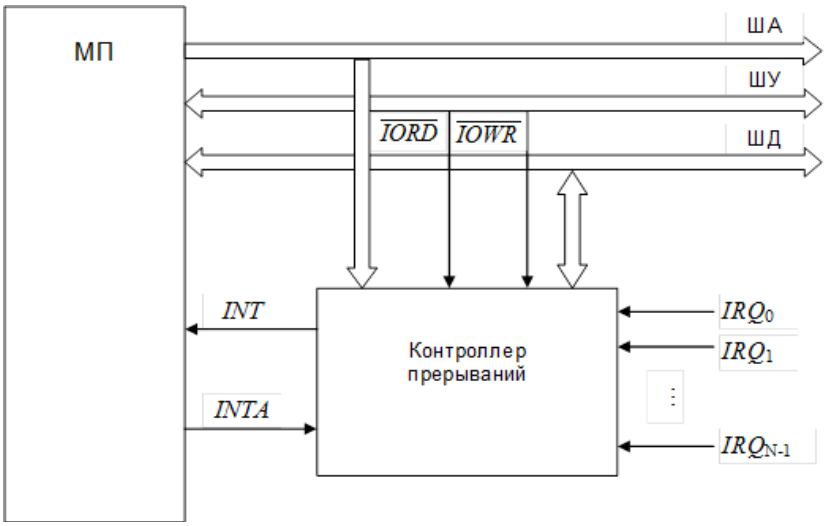



**Рисунок 41 – Векторная система прерываний на основе контроллера прерываний**

*Контроллер прерываний* (КПР) может рассматриваться как расширение процессора, по этой причине его иногда называют сопроцессором обработки прерываний. На основе КПР формируется многоуровневая приоритетная система векторных прерываний. КПР обеспечивает прием и обработку N запросов на прерывание. Приоритетная логика КПР выбирает из числа поступивших запросов на прерывание запрос с наивысшим приоритетом и сравнивает его с текущим

приоритетом запроса, находящегося на обслуживании. При превышении текущего приоритета КПР генерирует сигнал запроса прерывания INT, который поступает в процессор. МП подтверждает прием запроса INT генерацией сигнала подтверждение прерывания INTA, в ответ на который КПР выдает на шину данных системной магистрали соответствующий вектор прерывания. До тех пор, пока некоторый запрос находится в обслуживании, все запросы с равным или меньшим приоритетом игнорируются. В то же время запросы с более высоким приоритетом приводят к генерации сигнала INT, инициируя вложенные прерывания. Для оперативного управления работой контроллера предусматривается возможность его программирования, что позволяет динамически изменять приоритеты запросов, формируемые вектора прерываний и т.п.

## 42 Организация прямого доступа к памяти в микропроцессорной системе. Режим прямого доступа к памяти с захватом цикла. Передача блока данных с использованием прямого доступа к памяти. Режим прямого доступа к памяти с блокировкой процессора.

*Прямой доступ к памяти* (ПДП) является одним из способов обмена данными с ПУ. В этом режиме обмен данными между ПУ и памятью микропроцессорной системы происходит без участия процессора. Обменом в режиме ПДП управляет не программа, выполняемая процессором, а внешнее по отношению к процессору специальное устройство, называемое *контроллером ПДП* (КПДП). ПДП используется для быстрого ввода/вывода блоков данных и разгрузки процессора от управления операциями ввода/вывода. Обмен блоками данных с помощью программно-управляемого обмена осуществляется относительно медленно, так как на обмен каждым байтом затрачивается несколько команд процессора. ПДП освобождает процессор от управления операциями ввода/вывода, позволяя тем самым

осуществлять параллельно во времени выполнение процессором программы и обмен данными между ПУ и памятью;

производить этот обмен со скоростью, ограниченной только пропускной способностью памяти или ПУ.

Таким образом, ПДП, разгружая процессор от обслуживания операций ввода/вывода, способствует возрастанию общей производительности микропроцессорной системы.

Для реализации режима ПДП необходимо обеспечить непосредственную связь контроллера ПДП и памяти микропроцессорной системы, т.е. специальный информационный канал, по которому осуществляется обмен в режиме ПДП, –

*канал ПДП*. Для этой цели можно использовать специально выделенную магистраль, связывающую контроллер ПДП с памятью. Однако это приведет к значительному усложнению микропроцессорной системы в целом, особенно при подключении нескольких ПУ. Поэтому с целью сокращения количества линий в шинах микропроцессорной системы контроллер ПДП подключается к памяти посредством шин системной магистрали. При этом возникает проблема совместного использования шин системной магистрали процессором и контроллером ПДП. Можно выделить два основных способа ее решения:

реализация обмена в режиме ПДП с захватом цикла;

реализация обмена в режиме ПДП с блокировкой процессора. Режим ПДП с захватом цикла

Существует две разновидности прямого доступа к памяти с захватом цикла. Наиболее простой способ организации ПДП состоит в том, что для обмена используются те циклы процессора, в которых он не обменивается данными с памятью. В такие циклы контроллер ПДП может обмениваться данными с памятью, не мешая работе процессора. Однако возникает необходимость выделения таких циклов, чтобы не произошло временного перекрытия обмена ПДП с операциями обмена, инициируемыми процессором. В некоторых процессорах формируется специальный управляющий сигнал, указывающий циклы, в которых процессор не использует память. Если процессор не формирует такого сигнала, то для выделения свободных циклов необходимо применение в контроллере ПДП специальной схемы, что приводит к его усложнению. Применение этого способа организации ПДП не снижает производительности системы, но при этом обмен в режиме ПДП возможен только в случайные моменты времени одиночными словами.

Наиболее распространенным является *ПДП с захватом цикла и принудительным отключением процессора от шин системной магистрали*. Для реализации такого режима ПДП системная магистраль дополняется двумя управляющими сигналами – *требование прямого доступа к памяти* HOLD и *предоставление прямого доступа к памяти* HLDA.

Управляющий сигнал HOLD формируется контроллером ПДП. Процессор, получив этот сигнал, приостанавливает выполнение текущей команды, не дожидаясь ее завершения, отключается от шин системной магистрали и выдает контроллеру ПДП управляющий сигнал HLDA. С этого момента все шины системной магистрали управляются контроллером ПДП. Контроллер ПДП, используя шины системной магистрали, осуществляет обмен одним словом данных с памятью и затем, сняв сигнал HOLD, возвращает управление системной магистралью процессору. Как только контроллер ПДП будет готов к обмену следующим словом данных, он вновь захватывает цикл процессора и т.д. В промежутках между захватами циклов контроллером ПДП процессор продолжает выполнять команды программы. Тем самым выполнение программы замедляется, но значительно в меньшей степени, чем при обмене в режиме прерывания.

Передача блока данных с использованием ПДП предполагает выполнение определенной последовательности действий (рис. 42):

1. начальная установка (предварительная подготовка) контроллера ПДП;

2. запуск контроллера ПДП;
3. многократное занятие цикла процессора;
4. завершение обмена.
5. 

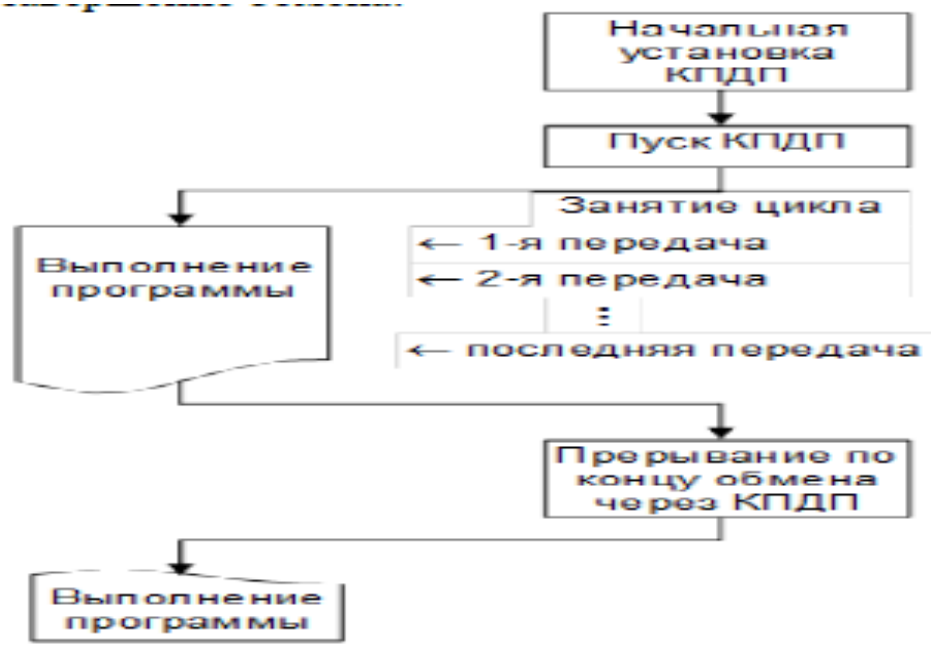



**Рисунок 42 – Передача блока данных с использованием ПДП**

Программа используется только для начальной установки и пуска обмена через канал ПДП. После этого процессор может выполнять основную программу, которая не связана с обменом. Во время выполнения этой программы каждый раз при поступлении запроса на ПДП контроллер ПДП будет занимать цикл процессора и осуществлять передачу. После окончания обмена для передачи управления программе завершения обмена в режиме ПДП используется прерывание. Затем основная программа может быть продолжена.

Начальная подготовка к обмену в режиме ПДП состоит в выделении ПУ области памяти, используемой при обмене, и указании ее размера, т.е. количества записываемых в память или читаемых из памяти слов информации. Для этого контроллер ПДП имеет в своем составе регистр адреса и счетчик слов. Перед началом обмена с ПУ в режиме ПДП процессор должен выполнить

программу загрузки, которая обеспечивает запись в указанные регистры контроллера ПДП начального адреса выделенной ПУ области памяти и ее размера в словах заданной разрядности.

Запуск контроллера ПДП осуществляется командой вывода, по которой ПУ подключается к контроллеру ПДП. После команды пуска контроллера ПДП должна быть команда разрешения прерывания. В дальнейшем ввод блока данных через канал ПДП осуществляется без участия команд программы.

Когда ПУ подготовит слово данных, оно посылается в регистр данных контроллера. При этом каждое слово сопровождается управляющим сигналом *ввод данных* из ПУ, который обеспечивает запись слова данных в регистр данных контроллера и формирование сигнала *требование прямого доступа к памяти* HOLD. В ответ процессор формирует сигнал *предоставление прямого доступа к*

*памяти* HLDA, после чего следующий машинный цикл занимается под обмен. При этом осуществляется одна пересылка слова данных в ячейку памяти по адресу, находящемуся в регистре адреса контроллера. По сигналу HLDA контроллер выставляет на шины адреса и данных системной магистрали содержимое своих регистров адреса и данных соответственно.

Формируя управляющий сигнал MEMRW, контроллер ПДП обеспечивает запись слова данных из своего регистра данных в память.

После передачи каждого слова данных из содержимого счетчика слов контроллера вычитается 1, и когда оно становится равным 0, устанавливается запрос на прерывание, который поступает на соответствующий вход процессора. Процессор прерывает выполнение программы и передает управление подпрограмме обработки прерывания для завершения обмена.

Завершение обмена осуществляется путем отключения ПУ от контроллера ПДП командой вывода. По окончании обработки прерывания управление возвращается основной программе.

Если нет необходимости продолжать выполнение некоторой программы параллельно с передачей в режиме ПДП, то прерывание не используется. В течение обмена через канал ПДП процессор находится в цикле ожидания окончания передачи, опрашивая соответствующий разряд готовности регистра состояния контроллера ПДП по команде ввода. Как только процессор обнаружит готовность, он переходит к процедуре завершения обмена (шаг 4 рассмотренной выше последовательности), после чего выполнение программы продолжается.

Выше были рассмотрены только процесс подготовки контроллера ПДП и непосредственно передача данных в режиме ПДП. На практике любой сеанс обмена данными с ПУ в режиме ПДП всегда включает также и этап подготовки ПУ к обмену. На этом этапе процессор в режиме программно-управляемого обмена опрашивает состояние ПУ, проверяя его готовность к обмену, и посылает в ПУ команды, обеспечивающие его подготовку к обмену данными по каналу ПДП. Такая подготовка может сводиться, например, к перемещению головок на требуемую дорожку в НМД. Затем выполняется загрузка регистров контроллера ПДП, после чего начинается обмен данными в режиме ПДП.

Следует отметить, что использование в микропроцессорной системе обмена в режиме ПДП с захватом цикла требует от программиста очень ясного понимания процессов, происходящих в системе при выполнении программы, и четкой синхронизации процесса выполнения программы и ввода/вывода по каналу ПДП.

Режим ПДП с блокировкой процессора

Прямой доступ к памяти с блокировкой процессора отличается от режима ПДП с захватом цикла тем, что управление системной магистралью передается контроллеру ПДП не на время передачи одного слова, а на время обмена блоком данных. Такой режим ПДП необходим в тех случаях, когда время между двумя сигналами *требование прямого доступа к памяти* HOLD сопоставимо с циклом процессора. В этом случае процессор не успевает выполнить хотя бы одну команду между очередными операциями обмена в режиме ПДП.

В микропроцессорной системе можно использовать несколько ПУ, работающих в режиме ПДП. Предоставление таким ПУ шин системной магистрали для обмена данными производится на приоритетной основе. В этом случае приоритеты ПУ реализуются так же, как и при обмене данными в режиме прерывания. Как правило, для каждого ПУ используется своя пара управляющих сигналов *требование прямого доступа к памяти* HOLD и *предоставление прямого доступа к памяти* HLDA и отдельный канал в контроллере ПДП

## 46 Структурная организация микропроцессоров. Аккумуляторная архитектура. Формат одноадресной команды.

Архитектура на базе аккумулятора исторически возникла одной из первых. В ней для хранения одного из операндов арифметической или логической операции в процессоре имеется выделенный регистр — *аккумулятор*. В этот же регистр заносится и результат операции. Изначально оба операнда хранятся в основной памяти, и до выполнения операции один из них нужно загрузить в аккумулятор. После выполнения команды обработки результат находится в аккумуляторе и, если он не является операндом для последующей команды, его требуется сохранить в ячейке памяти.

Типичная архитектура микропроцессора на базе аккумулятора показана на рис. 47.

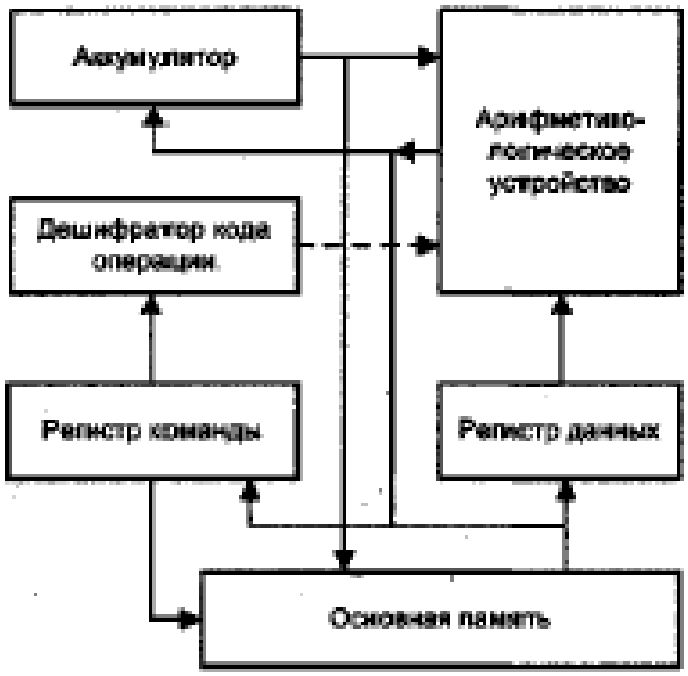



**Рисунок 47 – Архитектура микропроцессора на базе аккумулятора**

Для загрузки в аккумулятор содержимого ячейки x предусмотрена команда загрузки *load x*. По этой команде информация считывается из ячейки памяти x, выход памяти подключается ко входу аккумулятора и происходит занесение

считанных данных в аккумулятор. Запись содержимого аккумулятора в ячейку x осуществляется командой сохранения *store x*, при выполнении которой выход аккумулятора подключается к шине, после чего информация с шины записывается в память.

Таким образом, один из операндов всегда находится в аккумуляторе. Для выполнения операции в АЛУ производится считывание второго операнда из памяти в регистр данных. Выходы регистра данных и аккумулятора подключаются к соответствующим входам АЛУ. По окончании предписанной операции результат с выхода АЛУ заносится в аккумулятор. В процессорах аккумуляторного типа возможен следующий способ передачи данных:

(Акк) * (Память) --> Акк;

Поскольку положение одного из операндов предопределено, в командах обработки достаточно явно указать местоположение только второго операнда, т.е. команды являются *одноадресными* и имеют следующий формат (рис. 48): поле *кода операции* и поле *адреса ячейки памяти*.


**Рисунок 48 – Формат одноадресной команды**

Достоинствами аккумуляторной архитектуры можно считать короткие команды и простоту декодирования команд. Однако наличие всего одного регистра порождает многократные обращения к основной памяти.

Рассмотренная архитектура в основном используется в 8-разрядных микропроцессорах, микроконтроллерах и специализированных микропроцессорах.

## 47 Структурная организация микропроцессоров. Архитектура микропроцессора на базе регистров общего назначения, выполнение операции. Формат двухадресной команды.

Микропроцессор с архитектурой данного типа включает в себя массив регистров (регистровый файл), известных как *регистры общего назначения* (РОН). В микропроцессоре отсутствует специальный регистр данных, выполняющий функции аккумулятора. Функции аккумулятора выполняет один из регистров общего назначения (рис. 49).

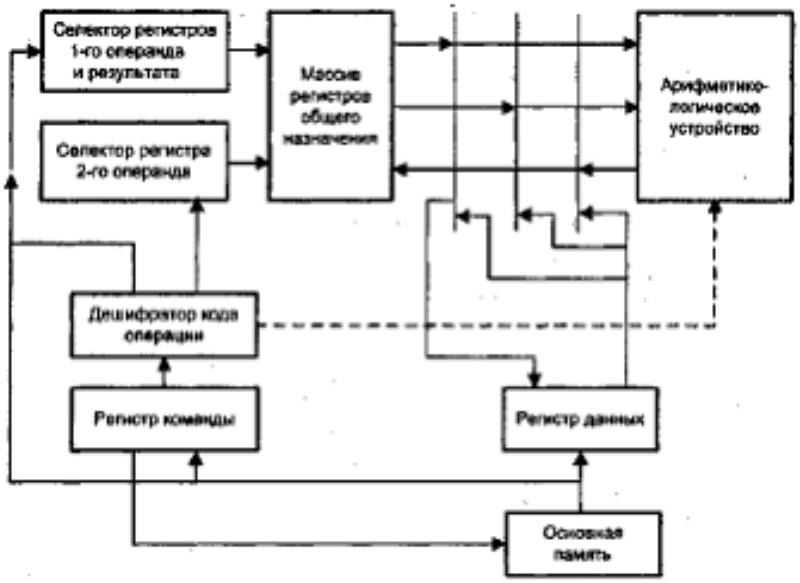



**Рисунок 49 – Архитектура микропроцессора на базе регистров общего назначения**

Размер регистров обычно фиксирован и совпадает с размером машинного слова. К любому регистру можно обратиться, указав его номер. Количество РОН в архитектурах типа CISC обычно невелико (от 8 до 32), и для представления номера конкретного регистра необходимо не более пяти разрядов, благодаря чему в адресной части команд обработки допустимо одновременно указать номера двух регистров или номер регистра и адрес памяти. RISC-архитектура предполагает использование существенно большего числа РОН (до нескольких сотен). Однако типичная для таких микропроцессоров длина команды (обычно 32 разряда) позволяет определить в команде до трех регистров (двух регистров операндов и регистра результата).

Регистровая архитектура допускает расположение операндов в одном из двух пространств: основной памяти или регистрах. С учетом возможного размещения операндов в рамках регистровых архитектур выделяют три типа команд обработки:

-регистр-регистр;

-регистр-память;

-память-память.

В варианте «регистр-регистр» операнды могут находиться только в регистрах. В них же засылается и результат. Вариант

«регистр-память» предполагает, что один из операндов размещается в регистре, а второй в основной памяти. Результат обычно замещает один из операндов. В командах типа «память-память» оба операнда хранятся в основной памяти. Результат заносится в память. Вариант «регистр-регистр» является основным в микропроцессорах типа RISC. Команды типа «регистр-память» характерны для CISC-микропроцессоров. Вариант «память-память» считается неэффективным, и в микропроцессорах не используется.

Операции загрузки регистров из памяти и сохранения содержимого регистров в памяти идентичны таким же операциям с аккумулятором. Отличие состоит в этапе выбора нужного регистра, обеспечиваемого соответствующими

селекторами. При этом между АЛУ и регистровым файлом должны быть, по крайней мере, три шины.

Выполнение операции в АЛУ включает в себя:

-выбор регистра первого операнда;

-определение расположения второго операнда (память или регистр);

-подачу на вход АЛУ операндов и выполнение операции;

-выбор расположения результата и занесение в него результата операции из АЛУ.

В качестве двух операндов, поступающих в АЛУ, могут использоваться операнды, хранящиеся в любых РОН, или одним из операндов может быть операнд, считываемый из основной памяти. Результат операции, выданный АЛУ, записывается в РОН или в основную память.

При адресации в команде, выполняющей операцию в АЛУ, указываются два адреса операндов, т.е. команды являются

*двухадресными* (рис. 50).

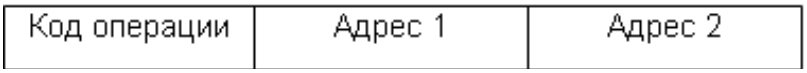



**Рисунок 50 – Формат двухадресной команды**

Обычно первый адрес указывает только на регистр общего назначения, а второй адрес может указывать как на регистр общего назначения, так и на основную память. Результат операции обычно записывается по первому адресу, однако более удобной считается система, позволяющая выборочно производить запись как по первому, так и по второму адресу.

В этом случае возможны четыре способа передачи данных:

- (РОН1) * (РОН2) --> РОН1;
- (РОН1) * (РОН2) --> РОН2;

(РОН) * (Память) --> РОН;

(РОН) * (Память) --> Память.

К достоинствам регистровой архитектуры следует отнести: компактность получаемого кода, высокую скорость вычислений за счет замены обращений к основной памяти на обращения к быстрым регистрам. С другой стороны, данная архитектура требует

более длинных инструкций по сравнению с аккумуляторной архитектурой. В целом регистровая архитектура имеет более высокую эффективность, поэтому именно она в основном используется в современных универсальных микропроцессорах.

48 Структурная организация микропроцессоров. Архитектура микропроцессора с выделенным доступом к памяти.

В архитектуре с выделенным доступом к памяти обращение к основной памяти возможно только с помощью двух специальных команд: load и store. В английской транскрипции данную архитектуру называют *Load/Store architecture*. Команда *load* (загрузка) обеспечивает считывание значения из основной памяти и занесение его в регистр процессора (в команде обычно указывается адрес ячейки памяти и номер регистра). Пересылка информации в противоположном направлении производится командой *store* (сохранение). Операнды во всех командах обработки информации в АЛУ могут находиться только в регистрах процессора (чаще всего в регистрах общего назначения). Результат операции также заносится в регистр. В архитектуре отсутствуют команды обработки, допускающие прямое обращение к основной памяти. Допускается наличие в наборе команд ограниченного числа команд, где операнд является частью кода команды.

Состав и информационные тракты микропроцессора с выделенным доступом к памяти показаны на рис. 51. Две из трех шин, расположенных между массивом РОН и АЛУ, обеспечивают передачу в арифметико-логическое устройство операндов, хранящихся в двух регистрах общего назначения. Третья служит для занесения результата в выделенный для этого регистр. Эти же шины позволяют загрузить в регистры содержимое ячеек основной памяти и сохранить в основной памяти информацию, находящуюся в РОН.

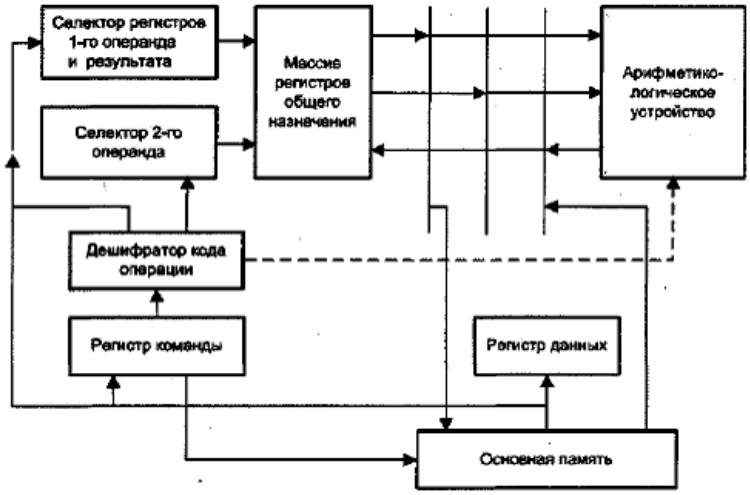



**Рисунок 51 – Архитектура микропроцессора с выделенным доступом к памяти**

К достоинствам архитектуры с выделенным доступом к памяти следует отнести простоту декодирования и исполнения команды. Архитектура с выделенным доступом к памяти характерна для всех микропроцессоров с RISC-архитектурой

## 49 Структурная организация микропроцессоров. Архитектура микропроцессора на базе стека. Последовательность вычислений в микропроцессоре со стековой архитектурой.

Данная архитектура основана на базе стека (стековой памяти). При соответствующем расположении операндов в стеке можно вычислять выражения полностью безадресными командами, указывающими только вид операции. Операнды перед обработкой помещаются в две верхних ячейки стековой памяти. Команда извлекает из стека в соответствии с кодом операции один или два операнда, выполняет над ними предписанную операцию и заносит результат обратно в стек. Микропроцессоры, в которых реализована архитектура на базе стека, обычно называют *стековыми микропроцессорами*.

При описании вычислений с использованием стека обычно используется форма записи математических выражений, известная как обратная польская запись (обратная польская нотация), которую предложил польский математик Я. Лукашевич. Особенность ее в том, что в выражении отсутствуют скобки, а знак операции располагается не между операндами, а следует за ними (постфиксная форма). Запись математических выражений с использованием обратной польской нотации производится по следующему правилу. Исходное выражение просматривается слева направо и последовательно друг за другом выписываются встречающиеся операнды. Как только все операнды некоторой операции выписаны, записывается знак этой операции и продолжается запись операндов.

Если операндом некоторой операции является результат предыдущей операции и ее знак выписан, считается что этот операнд выписан.

Принцип действия стекового процессора поясним на примере вычисления выражения a = a + b + a × c. Это выражение в польской записи имеет вид: a = ab+acx+. Данная форма записи не содержит скобок и однозначно определяет порядок загрузки операндов в стек и порядок выполнения операций (рис. 52), т.е. может рассматриваться как программа вычисления исходного математического выражения, если под буквами понимать команды загрузки соответствующих операндов в стек (такие команды содержат только адрес операнда в основной памяти), а под знаками операций — безадресные команды, содержащие только коды операций. Эти безадресные команды инициируют извлечение из стека двух (или одного) операндов, выполнение над ними указанной в команде операции и засылку результата в вершину стека.

| push a | push b | add | push a | push c | mul | add | pop a |
|---|---|---|---|---|---|---|---|
| a | b | a+b | a | c | a*c | a+b+a*c | |
| | a | | a+b | a | a+b | | |
| | | | | a+b | | | |
| | | | | | | | |

Рисунок 52 – Последовательность вычислений в микропроцессоре со стековой архитектурой

Основные узлы и информационные тракты одного из возможных вариантов микропроцессора на основе стековой архитектуры показаны на рис. 53.


**Рисунок 53 – Архитектура микропроцессора на базе стека**

Информация может быть занесена в вершину стека из памяти или из АЛУ. Для записи в стек содержимого ячейки памяти с адресом x выполняется команда *push x*, по которой информация считывается из ячейки памяти, заносится в регистр данных, а затем проталкивается в стек. Результат операции из АЛУ заносится в вершину стека автоматически. Сохранение содержимого вершины стека в ячейке памяти с адресом x производится командой *pop x*. По этой команде содержимое верхней ячейки стека подается на шину, с которой и производится запись в ячейку x, после чего вся находящаяся в стеке информация проталкивается на одну позицию вверх.

Для выполнения арифметической или логической операции на вход АЛУ подается информация, считанная из двух верхних ячеек стека (при этом содержимое стека продвигается на две позиции вверх, т.е. операнды из стека удаляются). Результат операции заталкивается в вершину стека. Возможен вариант, когда результат сразу же переписывается в память с помощью автоматически выполняемой операции pop x.

Верхние ячейки стековой памяти, где хранятся операнды и куда заносится результат операции, как правило, делаются более быстродействующими и размещаются в процессоре, в то время как остальная часть стека может располагаться в основной памяти.

К достоинствам стековой архитектуры следует отнести возможность предельного сокращения адресной части команд, поскольку все операции производятся через вершину стека, т. е. адреса операндов и результата в командах арифметической и логической обработки информации указывать не нужно. Поэтому код программы получается компактным, что экономит память. Достаточно просто реализуется декодирование команд, что способствуют повышению производительности микропроцессора. Однако при такой структуре команд возникают осложнения с построением команд передачи управления и работы с портами ввода/вывода.

Кроме того, стековая архитектура по определению не предполагает произвольного доступа к памяти, из-за чего компилятору

трудно создать эффективный программный код, хотя создание самих компиляторов упрощается. Поэтому организация вычислений с использованием стековой памяти нашла применение только в специализированных микропроцессорах

## 50 Классификация команд микропроцессоров. Команды пересылки данных, команды арифметической и логической обработки, команды ввода/вывода, команды управления микропроцессором, команды передачи управления.

Несмотря на различие в системах команд разных МП, некоторые основные типы команд могут быть найдены в любом из них. К основным типам команд МП относятся следующие:

команды пересылки данных;

команды арифметической и логической обработки;

команды ввода/вывода;

команды управления микропроцессором;

команды передачи управления.

*Команды пересылки данных*. Это наиболее распространенный тип машинных команд. Команды этого типа обеспечивают передачу информации между процессором и ОП, внутри процессора и между ячейками памяти.

*Команды арифметической и логической обработки*. В данную группу входят команды, обеспечивающие арифметическую и логическую обработку информации в различных формах ее представления. Для каждой формы представления чисел обычно предусматривается некоторый стандартный набор операций.

К стандартному набору операций над целыми числами, представленными в форме с фиксированной запятой, относятся:

двухместные арифметические операции (операции с двумя операндами): сложение, вычитание, умножение и деление;

одноместные арифметические операции (операции с одним операндом): вычисление абсолютного значения (модуля) операнда, изменение знака операнда;

сложение с учетом переноса, вычитание с учетом заема, увеличение значения операнда на единицу (инкремент), уменьшение значения операнда на единицу (декремент);

операции сравнения, обеспечивающие сравнение двух целых чисел и выработку признаков, характеризующих соотношение между сопоставляемыми величинами: =, <>, >, <, <=, >=.

Для работы с числами, представленными в форме с плавающей запятой, обычно предусматриваются следующие операции:

основные арифметические операции: сложение, вычитание, умножение и деление;

операции сравнения, обеспечивающие сравнение двух вещественных чисел с выработкой признаков: =, <>, >, <, <=, >=;

операции преобразования: формы представления (между фиксированной и плавающей запятой), формата представления (с одинарной и двойной точностью).

Стандартная система команд МП содержит команды для выполнения различных логических операций над отдельными битами слов или других адресуемых единиц. Такие команды предназначены для обработки символьных и логических данных.

Минимальный набор поддерживаемых логических операций – НЕ, И, ИЛИ и сложение по модулю 2.

В дополнение к побитовым логическим операциям, практически во всех МП предусмотрены команды для реализации операций логического, арифметического и циклического сдвигов.

При *логическом сдвиге* влево или вправо сдвигаются все разряды слова. Биты, вышедшие за пределы разрядной сетки, теряются, а освободившиеся позиции заполняются нулями.

При *арифметическом сдвиге* данные трактуются как целые числа со знаком, причем бит знака не изменяет положения. При сдвиге вправо освободившиеся позиции заполняются значением знакового разряда, а при сдвиге влево – нулями.

Арифметические сдвиги позволяют ускорить выполнение некоторых арифметических операций. Так, если числа представлены двоичным дополнительным кодом, то сдвиги влево и вправо эквивалентны соответственно умножению и делению на 2.

При *циклическом сдвиге* сдвигаются все разряды слова, причем значение разряда, выходящего за пределы слова, заносится в позицию, освободившуюся с противоположной стороны, т.е. потери информации не происходит. Одно из возможных применений циклических сдвигов – это перемещение интересующего бита в крайнюю левую (знаковую) позицию, где он может быть проанализирован как знак числа.

Десятичные числа представляются в МП в двоично-кодированной форме. В вычислительных машинах первых поколений для обработки таких чисел предусматривались специальные команды, обеспечивавшие выполнение основных арифметических операций (сложение, вычитание, умножение и деление). В современных МП подобных команд обычно нет, а соответствующие вычисления имитируются с помощью команд целочисленной арифметики.

Помимо вычисления результата выполнение арифметических и логических операций сопровождается формированием в АЛУ признаков (флагов), характеризующих этот результат. Наиболее часто фиксируются такие признаки, как: Z (Zero) – нулевой результат; N (Negative) – отрицательный результат; V (oVerflow) – переполнение разрядной сетки; C (Carry) — наличие переноса.

*Команды ввода/вывода*. Команды этой группы могут быть подразделены на команды управления периферийным устройством, проверки его состояния, ввода и вывода.

Команды управления периферийным устройством служат для запуска ПУ и указания ему требуемого действия. Например, накопителю на магнитной ленте может быть предписано на необходимость перемотки ленты или ее продвижения вперед на одну запись. Трактовка подобных инструкций зависит от типа ПУ.

Команды проверки состояния ввода/вывода применяются для тестирования различных признаков, характеризующих состояние ПУ. Благодаря этим командам процессор может выяснить, включено ли питание ПУ, завершена ли предыдущая операция ввода/вывода, возникли ли в процессе ввода/вывода какие-либо ошибки и т.п.

Собственно обмен информацией с ПУ обеспечивают команды ввода и вывода. Команды ввода предписывают получить элемент данных (байт или слово) от ПУ и поместить его на шину данных, а команды вывода – принять элемент данных с шины данных и переслать его на ПУ.

*Команды управления микропроцессором*. Команды, входящие в эту группу, являются привилегированными и могут выполняться, только когда процессор находится в привилегированном состоянии или выполняет программу, находящуюся в привилегированной области памяти (обычно привилегированный режим используется лишь операционной системой). Так, только эти команды способны считывать и изменять состояние системных регистров МП.

*Команды передачи управления*. Команды этого типа позволяют изменить естественный порядок следования команд программы и передать управление в другую точку программы. В адресной части таких команд содержится адрес точки перехода (адрес той

команды, которая должна быть выполнена следующей). Переход реализуется путем загрузки адреса точки перехода в счетчик команд (вместо увеличения содержимого этого счетчика на длину команды).

В системе команд можно выделить три типа команд, способных изменить последовательность вычислений:

безусловные переходы;

условные переходы (ветвления);

вызовы процедур и возвраты из процедур.

Среди команд рассматриваемой группы наиболее часто используются условные переходы (81%).

Команда *безусловного перехода* обеспечивает переход по заданному адресу без проверки каких-либо условий. Присутствие в программе большого числа команд безусловного перехода считается признаком плохого стиля программирования.

*Условный переход* происходит только при соблюдении определенного условия, в противном случае выполняется следующая по порядку команда программы. Условием, на основании которого осуществляется переход, чаще всего выступают признаки результата предшествующей арифметической или логической операции, которые фиксируется в регистре флагов процессора. Возможен и иной подход, когда решение о. переходе принимается в зависимости от состояния одного из регистров общего назначения, куда предварительно

помещается результат операции сравнения. Третий вариант – это объединение операций сравнения и перехода в одной команде.

Для всех языков программирования характерно интенсивное использование механизма процедур. Процедура может быть вызвана в любой точке программы. Для МП такой вызов означает, что в этой точке необходимо выполнить процедуру, после чего вернуться в точку, непосредственно следующую за местом вызова. Процедурный механизм базируется на командах вызова процедуры, обеспечивающих переход из текущей точки программы к начальной команде процедуры, и командах возврата из процедуры, для возврата в точку, непосредственно расположенную за командой вызова. Такой режим предполагает наличие средств для сохранения текущего состояния содержимого счетчика команд в момент вызова (запоминание адреса точки возврата) и его восстановления при выходе из процедуры

## 51 Структура команд микропроцессоров. Трехадресный, двухадресный, *полутораадресный,* одноадресный, нульадресный форматы команды.

Структура команды определяется ее *форматом*, т.е. количеством двоичных разрядов, отводимых под всю команду, а также количеством и расположением отдельных полей команды. *Полем* называется совокупность двоичных разрядов, кодирующих составную часть команды.

Типовая команда, в общем случае, должна указывать:

подлежащую выполнению операцию;

адреса исходных данных (операндов), над которыми выполняется операция;

адрес, по которому должен быть помещен результат операции.

В соответствии с этим команда состоит из двух частей: операционной и адресной. *Операционная часть* команды представляет собой поле кода операции, с помощью которого задается подлежащая выполнению операция. В *адресной части* команды содержится информация о местонахождении исходных данных и месте сохранения результата операции. Обычно местонахождение каждого из операндов и результата задается в команде путем указания адреса соответствующей ячейки основной памяти или номера регистра процессора. Принципы использования информации из адресной части команды определяет *система адресации*. Система адресации задает число адресов в команде команды и принятые *способы адресации*.

Для определения количества адресов, включаемых в адресную часть, используется термин *адресность*. В максимальном варианте необходимо указать три компонента: адрес первого операнда, адрес второго операнда и адрес результата операции. Такой формат команды называется *трехадресным* (рис. 54).

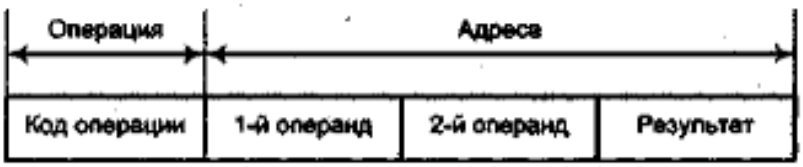



**Рисунок 54 – Трехадресный формат команды**

В трехадресном формате длина команды может оказаться достаточно большой, если все три поля будут задавать адреса в основной памяти.

Если по умолчанию взять в качестве адреса результата адрес одного из операндов (обычно второго), то можно обойтись без третьего адреса, и в итоге получаем *двухадресный* формат команды (рис. 55). Естественно, что в этом случае соответствующий операнд после выполнения операции теряется.

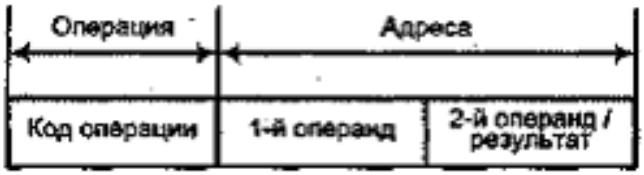



**Рисунок 55 – Двухадресный формат команды**

Длину команды можно сократить, если в качестве первого операнда использовать только регистры. Так как число регистров в МП невелико, для указания одного из них в команде достаточно иметь сравнительно короткое адресное поле. Соответствующий формат носит название *полутораадресного* или *регистрового* формата (рис. 56).

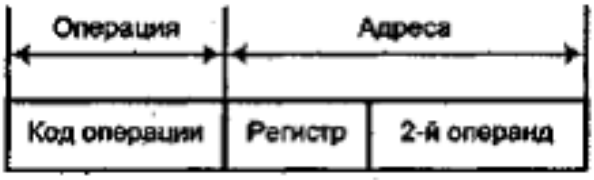



**Рисунок 56 – Регистровый формат команды**

Если выделить определенное стандартное место для хранения первого операнда и результата, то можно еще больше сократить команду, перейдя к *одноадресному* формату (рис. 57). Обычно для этой цели используется аккумулятор.


**Рисунок 57 – Одноадресный формат команды**

Если для обоих операндов выделяется четко заданное местоположение, а также в случае команд, не требующих операнда, используется *безадресный* (нульадресный) формат команды (рис. 58).


**Рисунок 58 – Безадресный формат команды**

В таком варианте адресная часть команды вообще отсутствует или не задействуется.

52 Регистровые структуры микропроцессоров. Функциональная неоднородность регистров. Адресные регистры: *регистр косвенного адреса, базовый регистр, индексный регистр, регистры автоинкрементной и автодекрементной адресации, регистр расширения адресного пространства, указатели сегментов и страниц*.

Функциональная неоднородность регистров

Количество и назначение регистров зависит от архитектуры микропроцессора. Часть регистров микропроцессора не используется в качестве средств программирования. Это объясняется не их физическим отсутствием, а тем, что программисту не предоставляются средства изменения содержимого этих регистров. Такие регистры называются *программно недоступными*. Для другой части регистров микропроцессора программисту предоставляются средства изменения их содержимого. Такие регистры называются *программно доступными*. Они образуют регистровую область микропроцессора.

Регистровую область или набор программно доступных регистров можно рассматривать как скоростное ОЗУ малой емкости, входящее в состав микропроцессора. Этот набор регистров используется для временного хранения данных, адресной информации, информации о состоянии микропроцессора и управляющей информации, контролируемых программистом.

Короткая адресация регистровой области и быстрый доступ к ней обеспечивают создание эффективно исполняемых программ. Регистры микропроцессора *функционально неоднородны*: одни служат для хранения данных и/или адресной информации, другие

для управления работой микропроцессора.

В соответствии с этим все регистры микропроцессора можно разделить на 3 группы:

регистры данных;

указатели или адресные регистры;

регистры специального назначения или специальные регистры.

*Регистры данных* участвуют в арифметических и логических операциях в качестве источников операндов и приемников результата, *адресные регистры* используются для вычисления данных и команд, расположенных в основной памяти.

*Специальные регистры* служат для индикации текущего состояния микропроцессора и управления режимами его работы. Часть регистров может использоваться для хранения как операндов, так и адресов. Их называют *регистрами общего назначения* (РОН). Функциональная неоднородность регистров микропроцессора связана с широким использованием *неявной* (подразумеваемой) адресации регистров, которая, в свою очередь, определяется стремлением к созданию коротких программ. В то же время функциональная специализация регистров затрудняет программирование, так как требует учета особенностей организации регистрового набора, присущих данному микропроцессору. Однако в результате программа выполняется быстрее и для ее хранения требуется меньше памяти.

На уровне символической записи (мнемоники) команд для прямой ссылки на конкретные регистры микропроцессора им присваиваются имена, например, A, B, C, D, SP, X или R0, R1, R2 и т.д. Обычно эти имена отражают функциональное назначение регистра и способствуют пониманию мнемоники команд.

Адресные регистры

Адресные регистры применяются для реализации различных методов *непрямой* (вычисляемой) адресации данных (рис. 59).

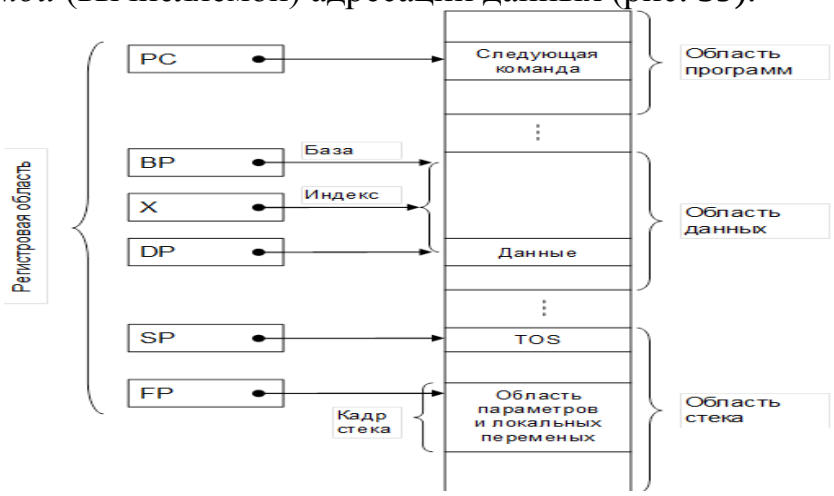



**Рисунок 59 – Адресные регистры**

К адресным регистрам относятся:

1. *регистр косвенного адреса* DP (Data Pointer). Содержит непосредственно адрес операнда;
2. *базовый регистр* (регистр базы) BP (Base Pointer). Используется для хранения начального адреса массива;
3. *индексный регистр* X. Содержит относительный (по отношению к базе) адрес (индекс) операнда (элемента массива);

4. *регистры автоинкрементной и автодекрементной адресации*. Автоматически увеличивают или уменьшают свое содержимое до или после выполнения операции доступа в соответствии с длиной адресуемого ими операнда;
5. *регистр расширения адресного пространства*. Обеспечивает расширение адресного пространства путем переключения между несколькими банками основной памяти. Содержит номер текущего банка основной памяти;
6. *указатели сегментов и страниц*

К классу адресных регистров с автоинкрементированием относится *программный счетчик* (*счетчик команд*, *указатель команд*) PC (Program Counter), содержимое которого после очередной выборки элемента командной последовательности увеличивается на длину этого элемента. За счет использования такого регистра в микропроцессоре процесс адресации следующего элемента командной последовательности в основном осуществляется неявно и автоматически. Изменение последовательности процесса выборки команд осуществляется специальными командами передачи управления, связанными с загрузкой в PC адреса, отличного от адреса следующей команды.

Частным случаем регистра с автомодификацией является *указатель стека* SP (Stack Pointer), который указывает на положение вершины стека.

При использовании стека для хранения локальных переменных и обмена параметрами между вызываемой и вызывающей процедурами может оказаться полезным специальный адресный регистр, указывающий на начало области локальных переменных и/или параметров в стеке, – *указатель кадра* FP (Frame Pointer). Значение указателя стека SP непрерывно меняется, поэтому применять его в качестве точки отсчета при доступе к данным в стеке неудобно. Процедуру доступа можно значительно упростить, если функцию точки отсчета отдать специально зарезервированному для этой цели регистру – указателю кадра FP. Указатель кадра относится к классу базовых регистров.

## 53 Регистровые структуры микропроцессоров. Специальные регистры. *Регистр и слово состояния программы. Поле кода условия. Системные регистры.* Регистры данных.

**Специальные регистры**

При выполнении операций АЛУ генерирует ряд признаков, характеризующих их результат. Функцию хранения этих признаков, а также некоторых других выполняет специальный *регистр состояния* SR (Status Register) или *слова состояния программы* PSW (Program Status Word). С каждым признаком связывается одноразрядная переменная – *флажок*. Флажки, связанные с признаками результата операции, группируются в *поле кода условия* CC (Condition Code). Типовой состав флажков-признаков результата операции:

- CF (Carry Flag) – флажок переноса из старшего разряда АЛУ при выполнении арифметических операций. При сдвигах CF равен выдвинутому значению младшего или старшего разряда;
- ZF (Zero Flag) – флажок признака нуля;
- SF (Sign Flag) – флажок знака результата. SF равен значению старшего разряда результата;
- AF (Auxiliary carry Flag) – флажок дополнительного переноса (переноса из младшей тетрады);
- OF (Overflow Flag) – флажок арифметического переполнения;
- PF (Parity Flag) – флажок четности количества единичных разрядов в результате.

Удобно в поле кода условия CC иметь один или несколько *флажков пользователя,* функциональное назначение которых определяет он сам. Обычно эти флажки служат для связи между отдельными частями программы.

Состояние поля кода условия CC тестируется *командами условного типа*.

В состав PSW входит также ряд специальных флажков, *управляющих работой микропроцессора,* например:

- IF (Interrupt Flag) – маски и приоритеты прерываний, а также условия реакции на прерывания;
- TF (Trace Flag) – флажок пошаговой трассировки, маска специального прерывания.

Регистр PSW включает также различные *модификаторы команд*, изменяющие реакцию МП на отдельные команды (например, флажок направления DF).

Для обеспечения *особых условий выполнения программ* в PSW вводят специальный флажок, определяющий эти условия. Так, например, флажок H/U (Halt/User) служит для разрешения особых условий выполнения программ и реализации директив пультового терминала, а флажок U/S (User/System) используется для перевода микропроцессора из специального системного режима, в котором выполняются программы ОС, в пользовательский.

Упаковка всех флажков в одно слово (и соответственно хранение в одном регистре) дает возможность организовать их быструю запись в память с последующим восстановлением. В некоторых микропроцессорах поле кода условия, а также ряд других флажков (например, флажки, управляющие работой МП, и модификаторы команд) выделяются в отдельный *регистр признаков (флагов)* F.

В сложных микропроцессорах состав специальных регистров более широкий. В этих микропроцессорах имеются специальные регистры, используемые преимущественно ОС, – *системные регистры*. К системным регистрам относятся:

- *управляющие регистры* (разрешают защищенный режим, страничную организацию памяти, управляют работой с сопроцессором, задают информацию для управления памятью, например, адреса системных таблиц, и т.п.);

- *отладочные регистры*;
- *тестовые регистры*. Регистры данных

Наиболее типичным представителем регистров данных является *аккумулятор* A (Accumulator), который используется для временного хранения операндов и промежуточных результатов. С аккумулятором связано большинство команд арифметической и логической обработки. Ссылка на него, как правило, производится неявно с помощью кода операции. В состав микропроцессора может входить один или несколько аккумуляторов.

Другим примером регистров данных являются *рабочие регистры*. В отличие от аккумулятора они адресуются явно и могут рассматриваться как сверхскоростное регистровое ОЗУ данных. Рабочие регистры могут совмещать свою функцию хранения данных с функцией их адресации. В этом случае они приобретают функции регистров общего назначения (РОН).

Как аккумуляторы, так и РОН могут при необходимости объединяться в регистровые пары.

Микропроцессор может содержать несколько (например, два или четыре) наборов рабочих регистров. При этом один из них используется для системных целей или обработки прерываний, в все остальные – для прикладных программ пользователя. В каждый момент времени доступен только один набор рабочих регистров, выбираемый специальным регистром – *указателем рабочего набора* WP (Work Pointer). Переключение доступного рабочего набора связано с перезагрузкой малоразрядного указателя WP.

Расширение разрядности указателя WP до полного размера адресного регистра и отображение набора рабочих регистров на основную память данных с базой в WP приводит к типовой *архитектуре с рабочими областями*. Однако при этом теряется

быстрый доступ к данным, хранящимся в рабочих регистрах. Решение этой проблемы связано с реализацией части основной памяти данных на одном кристалле с микропроцессором и размещением рабочей области в этой внутренней части ОЗУ

## 54 Адресация данных в микропроцессорах. *Адресный код.* Способы адресации. Непосредственная адресация. *Прямая адресация.*

Представление адресной информации

*Исполнительным* или *эффективным адресом* операнда (EA) называется двоичный код номера ячейки памяти, служащей источником или приемником операнда. По нему происходит фактическое обращение к указанной ячейке. Если операнд хранится не в основной памяти, а в регистре процессора, его исполнительным адресом будет номер регистра.

*Адресный код* команды – это двоичный код в адресном поле команды, из которого необходимо сформировать исполнительный адрес операнда.

В современных микропроцессорах исполнительный адрес и адресный код, как правило, не совпадают, и для доступа к данным требуется соответствующее преобразование. *Способ адресации* – это способ формирования исполнительного адреса операнда по адресному коду команды. Способ адресации существенно влияет на параметры процесса обработки информации. Одни способы позволяют увеличить емкость адресуемой памяти без удлинения команды, но снижают скорость выполнения операции, другие – ускоряют операции над массивами данных, третьи – упрощают работу с подпрограммами и т.д. В современных МП обычно имеется возможность применения нескольких различных способов адресации операндов к одной и той же операции.

Чтобы МП мог определить, какой именно способ адресации задан в данной команде, используются различные подходы. При первом подходе разным способам адресации соответствуют разные коды операции. Другой подход – это добавление в состав команды специального поля способа адресации, содержимое которого определяет, какой из способов адресации должен быть применен. Возможен также вариант, когда в команде вообще отсутствует адресная информация, т.е. имеет место *неявная* адресация. При неявной адресации адресного поля либо просто нет, либо оно содержит не все необходимые адреса – отсутствующий адрес подразумевается кодом операции. Так, при исключении из команды адреса результата подразумевается, что результат помещается на место одного из операндов. Неявная адресация применяется достаточно широко, поскольку позволяет сократить длину команды.

Способы адресации

В настоящее время используются различные способы адресации, наиболее распространенные из которых рассматриваются ниже. *Непосредственная адресация*. При непосредственной адресации в адресном поле команды вместо адреса содержится непосредственно сам операнд (рис. 60). Этот способ может применяться при выполнении арифметических операций, операций сравнения, а также для загрузки констант в регистры. Непосредственная адресация сокращает время выполнения команды, так как не требуется обращение к памяти за операндом. Кроме того, экономится память, поскольку отпадает необходимость в ячейке для хранения операнда. Однако в адресном поле могут быть указаны только константы.

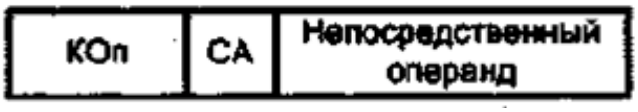



**Рисунок 60 – Непосредственная адресация**

*Прямая адресация*. При прямой или абсолютной адресации адресный код прямо указывает номер ячейки памяти, к которой производится обращение, т.е. адресный код совпадает с исполнительным (эффективным) адресом (рис. 61). При всей простоте использования способ имеет существенный недостаток – ограниченный размер адресного пространства, так как для адресации к памяти большой емкости нужно длинное адресное поле. Более существенным

недостатком можно считать то, что адрес, указанный в команде, не может быть изменен в процессе вычислений (во всяком случае, такое изменение не рекомендуется).

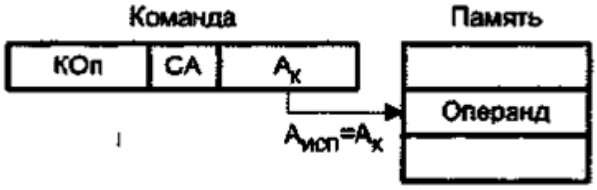



**Рисунок 61 – Прямая адресация**

55 Адресация данных в микропроцессорах. Регистровая адресация. *Косвенная регистровая адресация.*

*Регистровая адресация*. Регистровая адресация напоминает прямую адресацию. Различие состоит в том, что адресное поле инструкции указывает не на ячейку памяти, а на регистр процессора (рис. 62). Обычно размер адресного поля в данном случае составляет три или четыре бита, что позволяет указать соответственно на один из 8 или 16 регистров общего назначения (РОН).

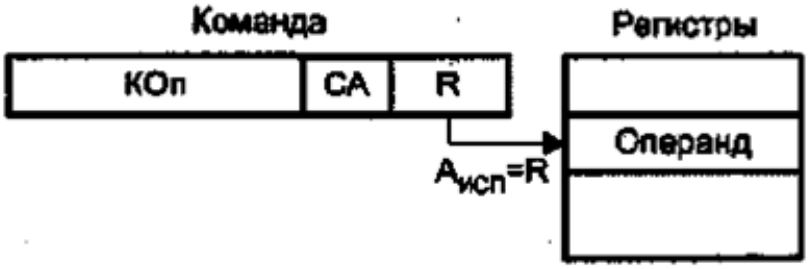



**Рисунок 62 – Регистровая адресация**

Двумя основными преимуществами регистровой адресации являются: короткое адресное поле в команде и исключение обращений к памяти. Малое число РОН позволяет сократить длину адресного поля команды. К сожалению, возможности по использованию регистровой адресации ограничены малым числом РОН в составе процессора.

*Косвенная регистровая адресация.* Косвенная адресация – это такой прием, когда адресное поля команды указывает на местоположение эффективного адреса операнда. При косвенной адресации содержимое адресного поля команды остается

неизменным, в то время как косвенный адрес в процессе выполнения программы можно изменять. Это позволяет проводить вычисления, когда адреса операндов заранее неизвестны и появляются лишь в процессе решения задачи. Дополнительно такой прием упрощает обработку массивов и списков, а также передачу параметров подпрограммам.

Косвенная регистровая адресация представляет собой вариант косвенной адресации, в котором исполнительный адрес операнда задается в регистре процессора. При этом адресное поле команды указывает на этот регистр (рис. 63).

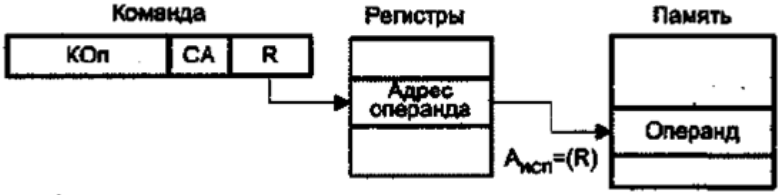



**Рисунок 63 – Косвенная регистровая адресация**

56 Адресация данных в микропроцессорах. *Адресация со смещением. Относительная адресация.*

*со смещением*. При адресации со смещением исполнительный адрес формируется в результате суммирования содержимого адресного поля команды с содержимым одного или нескольких регистров процессора.

В адресном поле команды содержится константа, смысл которой в разных вариантах адресации со смещением может меняться. Константа может представлять собой некий базовый адрес, к которому добавляется хранящееся в регистре смещение. Допустим и прямо противоположный подход: базовый адрес находится в регистре процессора, а в поле команды указывается смещение относительно этого адреса. В некоторых процессорах для реализации определенных вариантов адресации со смещением предусмотрены специальные регистры, например базовый или индексный. Использование таких регистров предполагается по умолчанию, поэтому адресная часть команды содержит только поле константы. Существуют способы, в которых смещение перед вычислением исполнительного адреса умножается на масштабный коэффициент. В наиболее общем случае адресация со смещением подразумевает наличие двух адресных полей: АК и R.

Ниже рассматриваются основные способы адресации со смещением, каждый из которых имеет собственное название.

*Относительная адресация*. При относительной адресации для получения исполнительного адреса операнда содержимое подполя АК команды складывается с содержимым счетчика команд (рис. 64). Таким образом, адресный код в команде представляет собой смещение относительно адреса текущей команды. Следует отметить, что в момент вычисления исполнительного адреса операнда в счетчике команд уже сформирован адрес следующей команды, что нужно учитывать при выборе величины смещения. Обычно поле АК трактуется как двоичное число со знаком в дополнительном коде.

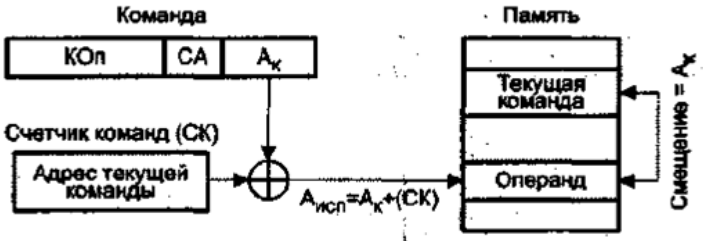
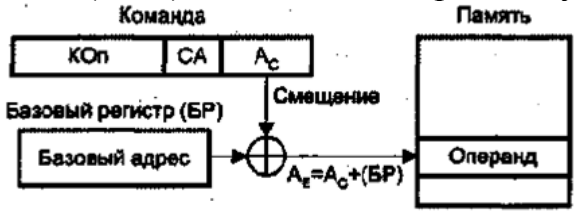



**Рисунок 64 – Относительная адресация**

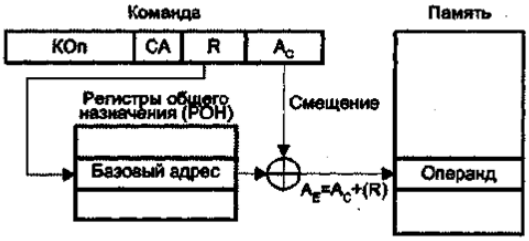
Адресация относительно счетчика команд базируется на свойстве локальности. Это позволяет сэкономить на длине адресной части команды, поскольку разрядность поля АК может быть небольшой. Главное достоинство данного способа адресации состоит в том, что он делает программу перемещаемой в памяти: независимо от текущего расположения программы в адресном пространстве взаимное положение команды и операнда остается неизменным, поэтому адресация операнда остается корректной.

57 Адресация данных в микропроцессорах. *Базовая адресация* с базовым регистром и с использованием регистра общего назначения. *Автоиндексирование, автоинкрементная, поставтоинкрементная, преавтоинкрементная* адресация.

*Базовая адресация*. В случае базовой адресации регистр, называемый *базовым*, содержит полноразрядный адрес, а поле АК – смещение относительно этого адреса (рис. 65). Ссылка на базовый регистр может быть явной или неявной. В некоторых МП имеется специальный базовый регистр и его использование является неявным, т.е. адресное поле R в команде отсутствует. Более типичен случай, когда в роли базового регистра выступает один из регистров общего назначения (РОН), тогда его номер явно указывается в поле R команды.




**а – с базовым регистром**




**б – с использованием РОН**

**Рисунок 65 – Базовая адресация**

Базовую адресацию обычно используют для доступа к элементам массива, положение которого в памяти в процессе вычислений может меняться. В базовый регистр заносится начальный адрес массива, а адрес элемента массива указывается в подполе АК команды в виде смещения относительно начального адреса массива. Достоинство данного способа адресации в том, что смещение может иметь меньшую длину, чем полный адрес, что позволяет сократить длину адресного поля команды.

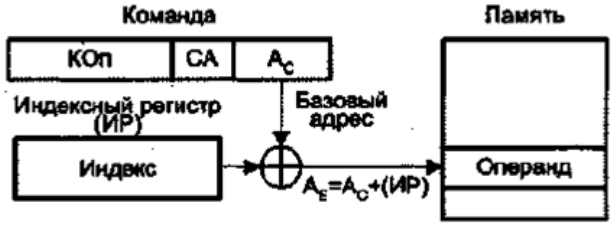
в большинстве МП увеличение или уменьшение содержимого индексного регистра до или после обращения к нему осуществляется автоматически как часть машинного цикла. Такой прием называется *автоиндексированием*.

Если для индексной адресации используются специально выделенные регистры, автоиндексирование может производиться неявно и автоматически. При использовании для хранения индексов регистров общего назначения необходимость операции автоиндексирования должна указываться в команде специальным битом.

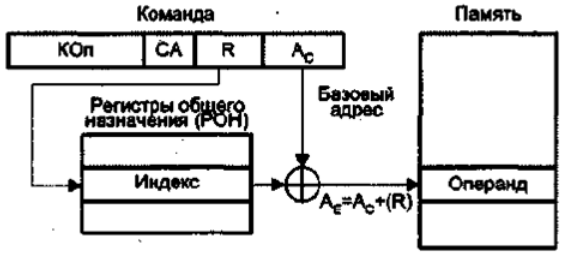
Автоиндексирование с увеличением содержимого индексного регистра носит название *автоинкрементной* адресации. В первом варианте увеличение содержимого индексного регистра происходит после формирования исполнительного адреса, и этот способ называется *поставтоинкрементной* адресацией. Во втором случае сначала производится увеличение содержимого индексного регистра, и уже новое значение используется для формирования исполнительного адреса. Такой способ называется *преавтоинкрементной* адресацией.

58 Адресация данных в микропроцессорах. *Индексная адресация* с индексным регистром и с использованием регистра общего назначения. *Автодекрементная, поставтодекрементная* и *преавтодекрементная* адресация, *индексная адресация с масштабированием и смещением.*

*Индексная адресация*. При индексной адресации поле АК содержит адрес памяти, а регистр, называемый *индексным* (указанный явно или неявно), – смещение относительно этого адреса (рис. 66).




**а – с индексным регистром**




**б – с использованием РОН**

**Рисунок 66 – Индексная адресация**

Этот способ адресации похож на базовую адресацию. Поскольку при индексной адресации в поле АК находится полноразрядный адрес памяти, играющий роль базы, длина этого поля больше, чем при базовой адресации. Тем не менее, вычисление исполнительного адреса операнда производится идентично.

Индексная адресация предоставляет удобный механизм для организации итеративных вычислений и работы с массивами

Автоиндексирование с уменьшением содержимого индексного регистра носит название *автодекрементной* адресации. Здесь также возможны два варианта, отличающиеся последовательностью выполнения операций уменьшения содержимого индексного регистра и вычисления исполнительного адреса: *поставтодекрементная* и *преавтодекрементная* адресация.

Интересным и весьма полезным является еще один вариант индексной адресации – *индексная адресация с масштабированием и смещением*: содержимое индексного регистра умножается на масштабный коэффициент и суммируется с АК. Масштабный коэффициент может принимать значения 1, 2, 4 или 8, для чего в адресной части команды выделяется дополнительное поле.

Описанный способ адресации реализован, например, в микропроцессорах фирмы Intel.

Следует особо отметить, что система команд многих МП предоставляет возможность различным образом сочетать базовую и индексную адресации в качестве дополнительных способов адресации.