АРХИТЕКТУРА ЭВМ И СИСТЕМ

конспект лекций

Основные характеристики ЭВМ. Общие принципы построения современных ЭВМ. Общие сведения и классификация устройств памяти. Архитектурная организация процессора ЭВМ. Структура машинной команды. Способы адресации. Особенности архитектур микропроцессоров. Архитектура суперскалярных микропроцессоров. Принципы организации системы прерывания программ. Классификация вычислительных систем.

Лекция 1. ПРИНЦИПЫ ПОСТРОЕНИЯ И АРХИТЕКТУРА ЭВМ

1.1. Основные характеристики ЭВМ

Электронная вычислительная машина - комплекс технических и программных средств, предназначенный для автоматизации подготовки и решения задач пользователей.

Структура - совокупность элементов и их связей. Различают структуры технических, программных и аппаратурно-программных средств.

Архитектура ЭВМ - это многоуровневая иерархия аппаратурно-программных средств, из которых строится ЭВМ. Каждый из уровней допускает многовариантное построение и применение. Конкретная реализация уровней определяет особенности структурного построения ЭВМ.

Одной из важнейших характеристик ЭВМ является ее *быстродействие,* которое характеризуется числом команд, выполняемых ЭВМ за одну секунду. Поскольку в состав команд ЭВМ включаются операции, различные по длительности выполнения и по вероятности их использования, то имеет смысл характеризовать его или средним быстродействием ЭВМ, или предельным (для самых «коротких» операций типа «регистр-регистр»). Современные вычислительные машины имеют очень высокие характеристики по быстродействию, измеряемые сотнями миллионов операций в секунду. Например, новейший микропроцессор Merced, совместного производства фирм Intel и Hewlett-Packard, обладает пиковой производительностью более миллиарда операций в секунду.

Другой важнейшей характеристикой ЭВМ является *емкость запоминающих устройств.* Этот показатель позволяет определить, какой набор программ и данных может быть одновременно размещен в памяти. В настоящее время персональные ЭВМ теоретически могут иметь емкость оперативной памяти 768Мбайт (chipset BX). Этот показатель очень важен для определения, какие программные пакеты и их приложения могут одновременно обрабатываться в машине.

Надежность - это способность ЭВМ при определенных условиях выполнять требуемые функции в течение заданного периода времени. Например, у современных HDD среднее время наработки на отказ достигает 500 тыс.ч. (около 60 лет).

Точность - возможность различать почти равные значения. Точность получения результатов обработки в основном определяется разрядностью ЭВМ, а также используемыми структурными единицами представления информации (байтом, словом, двойным словом). С помощью средств программирования языков высокого уровня этот диапазон может быть увеличен в несколько раз, что позволяет достигать очень высокой точности.

Достоверность - свойство информации быть правильно воспринятой. Достоверность характеризуется вероятностью получения безошибочных результатов. Заданный уровень

достоверности обеспечивается аппаратурно-программными средствами контроля самой ЭВМ. Возможны методы контроля достоверности путем решения эталонных задач и повторных расчетов. В особо ответственных случаях проводятся контрольные решения на других ЭВМ и сравнение результатов.

Лекция 2

1.2.Классификация средств ЭВТ

Традиционно электронную вычислительную технику (ЭВТ) подразделяют на аналоговую и цифровую. Редкие образцы аналоговой ЭВТ используются в основном в проектных и научно-исследовательских учреждениях в составе различных стендов по отработке сложных образцов техники. По своему назначению их можно рассматривать как специализированные вычислительные машины.

То, что 10-15 лет назад считалось современной большой ЭВМ. в настоящее время является устаревшей техникой с очень скромными возможностями. В этих условиях любая предложенная классификация ЭВМ очень быстро устаревает и нуждается в корректировке. Например, в классификациях десятилетней давности широко использовались названия мини-, миди- и микроЭВМ, которые почти исчезли из обихода.

Академик В.М. Глушков указывал, что существуют три глобальные сферы деятельности человека, которые требуют использования качественно различных типов ЭВМ.

Первое направление является традиционным - применение ЭВМ для автоматизации вычислений.. Отличительной особенностью этого направления является наличие хорошей математической основы, заложенной развитием математических наук и их приложений. Первые, а затем и последующие вычислительные машины классической структуры в первую очередь и создавались для автоматизации вычислений.

Вторая сфера применения ЭВМ связана с использованием их в системах управления. Она родилась в 60-е годы, когда ЭВМ стали внедряться в контуры управления автоматических и автоматизированных систем. Математическая база этой сферы была создана в течение последующих 15-20 лет. Новое применение вычислительных машин потребовало видоизменения их структуры. ЭВМ, используемые в управлении, должны были не только обеспечивать вычисления, но и автоматизировать сбор данных и распределение результатов обработки.

Третье направление связано с применением ЭВМ для решения задач искусственного интеллекта. Напомним, что задачи искусственного интеллекта предполагают получение не точною результата, а чаще всего осредненного в статистическом, вероятностном смысле. Примеров подобных задач много: задачи робототехники, доказательства теорем, машинного перевода текстов с одного языка на другой, планирования с учетом неполной информации, составления прогнозов, моделирования сложных процессов и явлений и т.д. Это направление все больше набирает силу. Во многих областях науки и техники создаются и совершенствуются базы данных и базы знаний, экспертные системы. Доя технического обеспечения этого направления нужны качественно новые структуры ЭВМ с большим количеством вычислителей (ЭВМ или процессорных элементов), обеспечивающих параллелизм в вычислениях. По существу, ЭВМ уступают место сложнейшим вычислительным системам.

Еще один класс наиболее массовых средств ЭВТ - встраиваемые микропроцессоры. Успехи микроэлектроники позволяют создавать миниатюрные вычислительные устройства, вплоть до однокристальных ЭВМ. Эти устройства, универсальные по характеру применения, могут встраиваться в отдельные машины, объекты, системы. Они находят все большее применение в бытовой технике (телефонах, телевизорах, электронных часах, микроволновых печах и т.д.), в городском хозяйстве (энерго-, тепло- , водоснабжении, регулировке движения транспорта и

т.д.), па производстве (робототехнике, управлении технологическими процессами). Постепенно они входят в нашу жизнь, все больше изменяя среду обитания человека.

Таким образом, можно предложить следующую классификацию средств вычислительной техники, в основу которой положено их разделение по быстродействию,

• СуперЭВМ для решения крупномасштабных вычислительных задач. для обслуживания крупнейших информационных банков данных.

• Большие ЭВМ для комплектования ведомственных, территориальных и региональных вычислительных центров.

• Средние ЭВМ широкого назначения для управления сложными технологическими производственными процессами. ЭВМ этого типа могут использоваться и для управления распределенной обработкой информации в качестве сетевых серверов.

• Персональные и профессиональные ЭВМ**,** позволяющие удовлетворять индивидуальные потребности пользователей. На базе этого класса ЭВМ строятся автоматизированные рабочие места (АРМ) для специалистов различного уровня.

• Встраиваемые микропроцессоры, осуществляющие автоматизацию управления отдельными устройствами и механизмами.

Лекция 3

1.3. Общие принципы построения современных ЭВМ

Основным принципом построения всех современных ЭВМ является программное управление. В основе его лежит представление алгоритма решения любой задачи в виде программы вычислений. Стандартом для построения практически всех ЭВМ стал способ, описанный Дж. фон Нейманом в 1945 г. при построении еще первых образцов ЭВМ. Суть его заключается в следующем.

Все вычисления, предписанные алгоритмом решения задачи, должны быть представлены в виде программы, состоящей из последовательности управляющих слов-команд. Каждая команда содержит указания на конкретную выполняемую операцию, место нахождения операндов (адреса операндов) и ряд служебных признаков. *Операнды* - переменные, значения которых участвуют в операциях преобразования данных. Список (массив) всех переменных (входных данных, промежуточных значении и результатов вычислений) является еще одним неотъемлемым элементом любой программы.

Для доступа к программам, командам и операндам используются их адреса. В качестве адресов выступают номера ячеек памяти ЭВМ, предназначенных для хранения объектов. Различные типы объектов, размещенные в памяти ЭВМ, идентифицируются по контексту.

Последовательность битов в формате, имеющая определенный смысл, называется *полем.* Например, в каждой команде программы различают поле кода операций, поле адресов операндов. Применительно к числовой информации выделяют знаковые разряды, поле значащих разрядов чисел, старшие и младшие разряды.

Последовательность, состоящая из определенного принятого для данной ЭВМ числа байтов, называется *словом.*

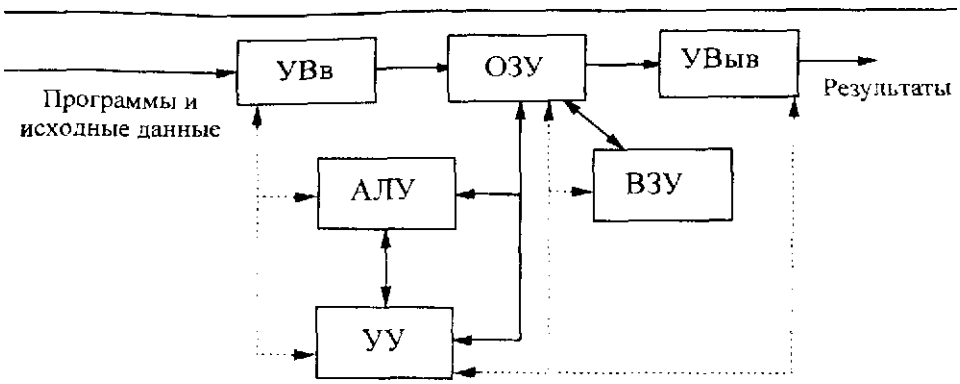


Рис. 1.1. Структурная схема ЭВМ первого и второго поколений

В любой ЭВМ имеются устройства ввода информации (УВв), с помощью которых пользователи вводят в ЭВМ программы решаемых задач и данные к ним. Введенная информация полностью или частично сначала запоминается в оперативном запоминающем устройстве (ОЗУ), а затем переносится во внешнее запоминающее устройство (ВЗУ), предназначенное для длительного хранения информации, где преобразуется в файл. При использовании файла в вычислительном процессе его содержимое переносится в ОЗУ. Затем программная информация команда за командой считывается в устройство управления (УУ).

Устройство управления предназначается для автоматического выполнения программ путем принудительной координации всех остальных устройств ЭВМ. Цепи сигналов управления показаны на рис. 1.1 штриховыми линиями. Вызываемые из ОЗУ команды дешифрируются устройством управления: определяются код операции, которую необходимо выполнить следующей, и адреса операндов, принимающих участие в данной операции.

В зависимости от количества используемых в команде операндов различаются одно-, двух-, трех-, четырех- адресные и безадресные команды. В одноадресных командах указывается, где находится один из двух обрабатываемых операндов. Второй операнд должен быть помещен заранее в арифметическое устройство.

Двухадресные команды содержат указания о двух операндах, размещаемых в памяти (или в регистрах и памяти). После выполнения команды в один из этих адресов засылается результат, а находившийся там операнд теряется.

В трехадресных командах обычно два адреса указывают, где находятся исходные операнды, а третий - куда необходимо поместить результат.

В безадресных командах обычно обрабатывается один операнд, который до и после операции находится на одном из регистров арифметико-логического устройства (АЛУ). Кроме того, безадресные команды используются для выполнения служебных операций (запрет прерывания, выход из подпрограммы и др.).

Все команды программы выполняются последовательно, команда за командой, в том порядке, как они записаны в памяти ЭВМ (естественный порядок следования команд) или если команда четырех- адресная (характерно для первых ЭВМ) адрес следующей команды находится в поле четвертого операнда. Этот порядок характерен для линейных программ, т.е. программ, не содержащих разветвлений. Для организации ветвлений используются команды, нарушающие естественный порядок следования команд. Отдельные признаки результатов r ($r = 0$, $r < 0$, $r > 0$ и др.) устройство управления использует для изменения порядка выполнения команд программы.

АЛУ выполняет арифметические и логические операции над данными. Основной частью АЛУ является операционный автомат, в состав которого входят сумматоры, счетчики, регистры, логические преобразователи и др. Оно каждый раз перенастраивается на выполнение очередной операции. Результаты выполнения отдельных операций сохраняются для последующего использования на одном из регистров АЛУ или записываются в память. Результаты, полученные после выполнения всей программы вычислений, передаются на

устройства вывода (УВыв) информации. В качестве УВыв могут использоваться экран дисплея, принтер, графопостроитель и др.

Современные ЭВМ имеют достаточно развитые системы машинных операций. Например, ЭВМ типа IBM PC имеют около 200 различных операций (170 - 300 в зависимости от типа микропроцессора). Любая операция в ЭВМ выполняется по определенной микропрограмме, реализуемой в схемах АЛУ соответствующей последовательностью сигналов управления (микрокоманд). Каждая отдельная микрокоманда - это простейшее элементарное преобразование данных типа алгебраического сложения, сдвига, перезаписи информации и т.п.

Уже в первых ЭВМ для увеличения их производительности широко применялось совмещение операций. При этом последовательные фазы выполнения отдельных команд программы (формирование адресов операндов, выборка операндов, выполнение операции, отсылка результата) выполнялись отдельными функциональными блоками. В своей работе они образовывали конвейер, а их параллельная работа позволяла обрабатывать различные фазы целого блока команд. Этот принцип получил дальнейшее развитие в ЭВМ следующих поколений. Но все же первые ЭВМ имели очень сильную централизацию управления, единые стандарты форматов команд и данных, «жесткое» построение циклов выполнения отдельных операций, что во многом объясняется ограниченными возможностями используемой в них элементной базы. Центральное УУ обслуживало не только вычислительные операции, но и операции ввода-вывода, пересылок данных между ЗУ и др. Все это позволяло в какой-то степени упростить аппаратуру ЭВМ, но сильно сдерживало рост их производительности.

В ЭВМ третьего поколения произошло усложнение структуры за счет разделения процессов ввода-вывода информации и ее обработки (рис. 1.2).

Сильносвязанные устройства АЛУ и УУ получили название *процессор,* г.е. устройство, предназначенное для обработки данных. В схеме ЭВМ появились также дополнительные устройства, которые имели названия: процессоры ввода-вывода, устройства управления обменом информацией, каналы ввода-вывода (КВВ). Последнее название получило наибольшее распространение применительно к большим ЭВМ. Здесь наметилась тенденция к децентрализации управления и параллельной работе отдельных устройств. что позволило резко повысить быстродействие ЭВМ в целом.

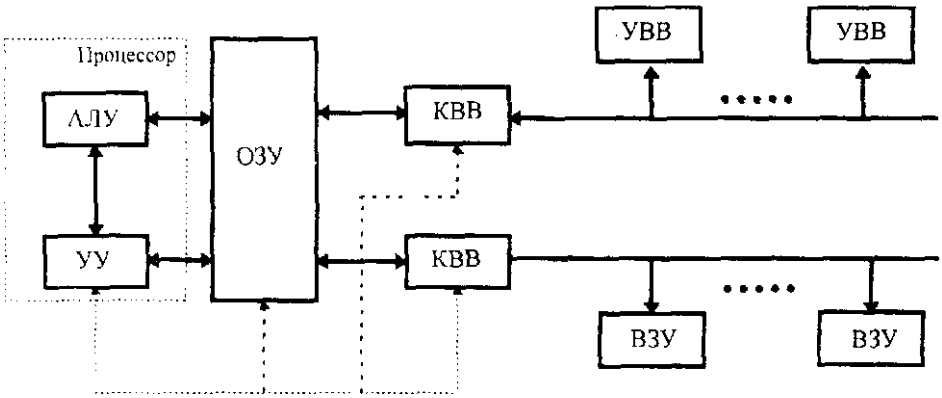


Рис. 1.2. Структурная схема ЭВМ третьего поколения

Среди каналов ввода-вывода выделяли мультиплексные каналы, способные обслуживать большое количество медленно работающих устройств ввода-вывода (УВВ). и селекторные каналы, обслуживающие в многоканальных режимах скоростные внешние запоминающие устройства (ВЗУ).

В персональных ЭВМ, относящихся к ЭВМ четвертого поколения, произошло дальнейшее изменение структуры (рис. 1.3). Они унаследовали ее от мини-ЭВМ.

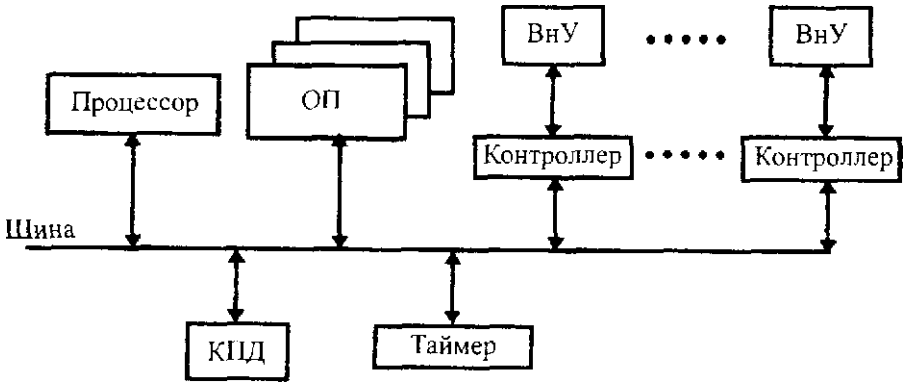


Рис. 1.3. Структурная схема ПЭВМ

Соединение всех устройств в единую машину обеспечивается с помощью общей шины, представляющей собой линии передачи данных, адресов, сигналов управления и питания. Единая система аппаратурных соединений значительно упростила структуру, сделав ее еще более децентрализованной. Все передачи данных по шине осуществляются под управлением сервисных программ.

Ядро ПЭВМ образуют процессор и основная память (ОП), состоящая из оперативной памяти и постоянного запоминающего устройства (ПЗУ). ПЗУ предназначается для постоянного хранения программ первоначального тестирования ПЭВМ (POST) и загрузки ОС. Подключение всех внешних устройств (ВнУ), дисплея, клавиатуры, внешних ЗУ и других обеспечивается через соответствующие адаптеры - согласователи скоростей работы сопрягаемых устройств или контроллеры - специальные устройства управления периферийной аппаратурой. Контроллеры в ПЭВМ играют роль каналов ввода-вывода. В качестве особых устройств следует выделить таймер - устройство измерения времени и контроллер прямого доступа к памяти (КПД) - устройство, обеспечивающее доступ к ОП, минуя процессор.

Децентрализация построения и управления вызвала к жизни такие элементы, которые являются общим стандартом структур современных ЭВМ:

модульность построения, магистральность, иерархия управления.

Модульность построения предполагает выделение в структуре ЭВМ достаточно автономных, функционально и конструктивно законченных устройств (процессор, модуль памяти, накопитель на жестком или гибком Mai-нитном диске).

Модульная конструкция ЭВМ делает ее открытой системой, способной к адаптации и совершенствованию. К ЭВМ можно подключать дополнительные устройства, улучшая ее технические и экономические показатели. Появляется возможность увеличения вычислительной мощности, улучшения структуры путем замены отдельных устройств на более совершенные, изменения и управления конфигурацией системы, приспособления ее к конкретным условиям применения в соответствии с требованиями пользователей.

В современных ЭВМ принцип децентрализации и параллельной работы распространен как на периферийные устройства, так и на сами ЭВМ (процессоры). Появились вычислительные системы, содержащие несколько *вычислителей* (ЭВМ или процессоры), работающие согласованно и параллельно. Внутри самой ЭВМ произошло еще более резкое разделение функций между средствами обработки. Появились отдельные специализированные процессоры, например сопроцессоры, выполняющие обработку чисел с плавающей точкой, матричные процессоры и др.

Все существующие типы ЭВМ выпускаются *семействами,* в которых различают старшие и младшие модели. Всегда имеется возможность замены более слабой модели на более мощную. Это обеспечивается информационной, аппаратурной и программной совместимостью. Программная совместимость в семействах устанавливается по принципу снизу-вверх, т.е. программы, разработанные для ранних и младших моделей, могут обрабатываться и на старших,

но не обязательно наоборот.

Модульность структуры ЭВМ требует стандартизации и унификации оборудования, номенклатуры технических и программных средств, средств сопряжения - интерфейсов, конструктивных решений, унификации типовых элементов замены, элементной базы и нормативно-технической документации. Все это способствует улучшению технических и эксплуатационных характеристик ЭВМ, росту технологичности их производства.

Децентрализация управления предполагает *иерархическую организацию структуры ЭВМ.* Централизованное управление осуществляет устройство управления главного, или центрального, процессора. Подключаемые к центральному процессору модули (контроллеры и КВВ) могут, в свою очередь, использовать специальные *шины или магистрали* для обмена управляющими сигналами, адресами и данными. Инициализация работы модулей обеспечивается по командам центральных устройств, после чего они продолжают работу по собственным программам управления. Результаты выполнения требуемых операций представляются ими «вверх по иерархии» для правильной координации всех работ.

По иерархическому принципу строится система памяти ЭВМ. Так, с точки зрения пользователя желательно иметь в ЭВМ оперативную память большой информационной емкости и высокого быстродействия. Однако одноуровневое построение памяти не позволяет одновременно удовлетворять этим двум противоречивым требованиям. Поэтому память современных ЭВМ строится по многоуровневому, пирамидальному принципу.

В состав процессоров может входить сверхоперативное запоминающее устройство небольшой емкости, образованное несколькими десятками регистров с быстрым временем доступа (единицы нс). Здесь обычно хранятся данные, непосредственно используемые в обработке.

Следующий уровень образует кэш-память. Она представляет собой буферное запоминающее устройство, предназначенное для хранения активных страниц объемом десятки и сотни Кбайтов. Время обращения к данным составляет 2-10 нс, при этом может использоваться ассоциативная выборка данных. Кэш-память, как более быстродействующая ЗУ, предназначается для ускорения выборки команд программы и обрабатываемых данных. Сами же программы пользователей и данные к ним размещаются в оперативном запоминающем устройстве (емкость - миллионы машинных слов, время выборки 10-70 нс).

Часть машинных программ, обеспечивающих автоматическое управление вычислениями и используемых наиболее часто, может размещаться в постоянном запоминающем устройстве (ПЗУ). На более низких уровнях иерархии находятся внешние запоминающие устройства на магнитных носителях: на жестких и гибких магнитных дисках, магнитных лентах, магнитооптических дисках и др. Их отличает более низкое быстродействие и очень большая емкость.

Организация заблаговременного обмена информационными потоками между ЗУ различных уровней при децентрализованном управлении ими позволяет рассматривать иерархию памяти как единую абстрактную виртуальную память. Согласованная работа всех уровней обеспечивается под управлением программ операционной системы. Пользователь имеет возможность работать с памятью, намного превышающей емкость ОЗУ.

Децентрализация управления и структуры ЭВМ позволила перейти к более сложным *многопрограммным (мультипрограммным)* режимам. При этом в ЭВМ одновременно может обрабатываться несколько программ пользователей.

В ЭВМ, имеющих один процессор, многопрограммная обработка является кажущейся. Она предполагает параллельную работу отдельных устройств, задействованных в вычислениях по различным задачам пользователей. Например, компьютер может производить распечатку каких-либо документов и принимать сообщения, поступающие по каналам связи. Процессор при этом может производить обработку данных по третьей программе, а пользователь - вводить данные или программу для новой задачи, слушать музыку и т.п.

В ЭВМ иди вычислительных системах, имеющих несколько процессоров обработки, многопрограммная работа может быть более глубокой. Автоматическое управление вычислениями предполагает усложнение структуры за счет включения в ее состав систем и блоков, разделяющих различные вычислительные процессы друг от друга, исключающие возможность возникновения взаимных помех и ошибок (системы прерываний и приоритетов, защиты памяти). Самостоятельного значения в вычислениях они не имеют, но являются необходимым элементом структуры для обеспечения этих вычислений.

Как видно, полувековая история развития ЭВТ дала не очень широкий спектр основных структур ЭВМ. Все приведенные структуры не выходят за пределы классической структуры фон Неймана. Их объединяют следующие традиционные признаки [53]:

• ядро ЭВМ образует процессор - единственный вычислитель в структуре, дополненный каналами обмена информацией и памятью-

• линейная организация ячеек всех видов памяти фиксированного размера;

• одноуровневая адреса11ия ячеек памяти, стирающая различия между всеми типами информации:

• внутренний машинный язык низкого уровня, при котором команды содержат элементарные операции преобразования простых операндов;

• последовательное централизованное управление вычислениями;

• достаточно примитивные возможности устройств ввода-вывода.

Несмотря на все достигнутые успехи, классическая структура ЭВМ не обеспечивает возможностей дальнейшего увеличения производительности. Наметился кризис, обусловленный рядом существенных недостатков:

• плохо развитые средства обработки нечисловых данных (структуры, символы, предложения, графические образы, звук, очень большие массивы данных и др.);

• несоответствие машинных операций операторам языков высокого уровня;

• примитивная организация памяти ЭВМ;

• низкая эффективность ЭВМ при решении задач, допускающих параллельную обработку и т.п.

Все эти недостатки приводят к чрезмерному усложнению комплекса программных средств, используемого для подготовки и решения задач пользователей.

В ЭВМ будущих поколений, с использованием в них «встроенного искусственного интеллекта», предполагается дальнейшее усложнение структуры. В первую очередь это касается совершенствования процессов общения пользователей с ЭВМ (использование аудио-, видеоинформации, систем мультимедиа и др.) , обеспечения доступа к базам данных и базам знаний, организации параллельных вычислений. Несомненно, что этому должны соответствовать новые параллельные структуры с новыми принципами их построения. В качестве примера укажем, что самая быстрая ЭВМ фирмы IBM в настоящее время обеспечивает быстродействие 600 MIPS (миллионов команд в секунду), самая же большая гиперкубическая система nCube дает быстродействие 123.10^3 MIPS. Расчеты показывают, что стоимость одной машинной операции в гиперсистеме примерно в тысячу раз меньше. Вероятно, подобными системами будут обслуживаться большие информационные хранилища.

Лекция 4

1.4 Понятие о состоянии процессора (программы). Вектор (слово) состояния

При выполнении процессором программы после каждого рабочего такта, а тем более в результате завершения выполнения очередной команды, изменяется содержимое регистров, счетчиков, состояния отдельных управляющих триггеров. Можно говорить, что изменяется состояние процессора, или, употребляя другую терминологию, *состояние программы*

Понятие *состояния процессора (состояния программы)* занимает важное место в организации вычислительного процесса в ЭВМ.

Информация о состоянии процессора (программы) лежит в основе многих процедур управления вычислительным процессом, например при анализе ситуаций при отказах и сбоях, при возобновлении выполнения программы после перерывов, вызванных отказами, сбоями, прерываниями, для фиксации состояния процессора (программы) в момент перехода в мультипрограммном режиме от обработки данной программы к другой и т. п.

Состоянием процессора (программы) после данного такта или после выполнения данной команды, строго говоря, следует считать совокупность состояний в соответствующий момент времени всех запоминающих элементов устройства — триггеров, регистров, ячеек памяти.

Однако не вся эта информация исчезает или искажается при переходе к очередной команде или другой программе. Поэтому из всего многообразия информации о состоянии процессора (программы) отбираются наиболее существенные ее элементы, как правило, подверженные изменениям при переходе к другой команде или программе.

Совокупность значений этих информационных элементов получила название *вектора состояния* или *слова состояния процессора (программы).*

Вектор состояния в каждый момент времени должен содержать информацию, достаточную для продолжения выполнения программы или повторного пуска программы с точки, соответствующей моменту формирования данного вектора состояния. При этом предполагается, что остальная информация, характеризующая состояние процессора, например содержимое регистров, или сохраняется, или может быть восстановлена программным путем по копии, сохраненной в памяти.

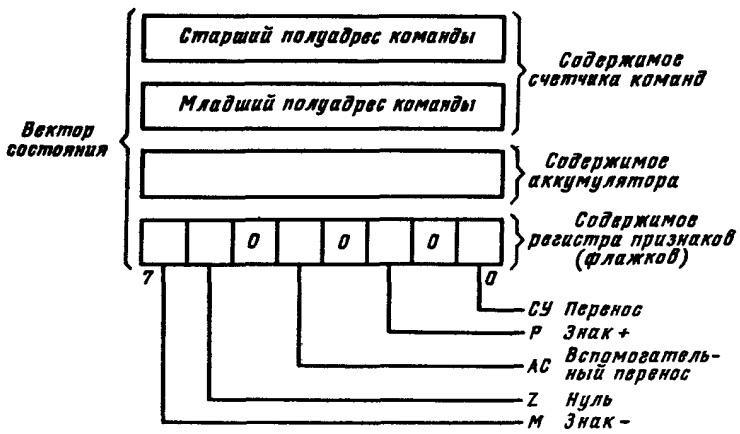


Рис. 1.4. Вектор состояния 8-разрядного микропроцессора К580 (четыре 8-разрядных слова)

Вектор состояния формируется в соответствующем регистре (регистрах) процессора, претерпевая изменения после выполнения каждой команды.

Наборы информационных элементов, образующих векторы состояния, отличаются у ЭВМ разных типов. Наиболее просто он выглядит у микропроцессоров. Например, вектор состояния микропроцессора К580, как это показано на рис. 9.22, включает в себя содержимое 16-разрядного счетчика команд (адрес очередной команды); содержимое 8-разрядного регистра признаков, называемое в документации на этот микропроцессор *словом состояния процессора,* и содержимое 8-разрядного аккумулятора АЛУ.

Современные ЭВМ имеют более сложные структуры вектора состояния, или, иначе говоря, слова состояния программы.

Использование слова (вектора) состояния — распространенный прием построения управления устройствами вычислительной техники. Во многих устройствах ЭВМ для организации их функционирования формируются свои, специфические слова состояния (или *байты состояния*), фиксирующие в виде некоторого кода состояние устройства, например готовность его к выполнению задаваемой операции, успешное или неуспешное завершение операции и т. д.

Контрольные вопросы

1. Каково понятие архитектуры ЭВМ?

2. По каким техническим характеристикам осуществляются оценка и выбор ЭВМ?

3. Какова связь областей применения ЭВМ и их структур?

4. Каковы основные тенденции развития ЭВМ?

5. Охарактеризуйте понятие машинного парка.

6. Каковы основные принципы построения ЭВМ?

7. Поясните место и роль программного обеспечения ЭВМ.

8. Что представляет собой класс персональных ЭВМ?

9. Назначение и отличительные особенности построения сетевых компьютеров.

Лекция 5

2. Определение архитектуры ЭВМ

Архитектурой ЭВМ принято считать совокупность принципов системной,функциональной, логической и физической организации аппаратных и программных средств ЭВМ.

В этом, достаточно четком определении есть общеупотребительное слово "ЭВМ". Определяя ЭВМ как цифровую вычислительную машину дискретного действия (полное имя ее ЭЦВМ) следует помнить, что буква Э служила для исключения от рассмотрения класса ЦВМ, сконструированных на механических (машина Бэббиджа) и электромеханических (Марк-1) "элементах".Только совокупность электронных триггеров, выполненных на радиолампах и транзисторах, давала право на "Э".

Усложнение периферии ЭВМ и появление многопроцессорных систем породили осторожный термин "вычислительные системы", "платформы", "вычислительные среды" и т.д. Вероятно, в настоящее время, интуитивное понятие ЭВМ наиболее полное - это ЦВМ, выполненная как единый конструктив.

С другой стороны, в понятие ЭВМ можно включить и специализированный микропроцессор управления "рукой" робота, и персональный компьютер, и суперкомпьютер, который, как элемент сети может именоваться также майнфреймом. Общим при этом является функциональное назначение ЭВМ - обеспечение потребностей прикладной системной области. Рассматривается при этом использование ЭВМ, в основном, как универсальные вычислительные системы (платформы).

2.1. Классификация ЭВМ по областям применения

По идеологии открытых систем, все вычислительные платформы должны удовлетворять любые запросы пользователей. Но реализация общего ядра для всех приложений, как и всякая универсализация, ведет к большим накладным расходам. Пользователей интересует не только

интерфейс с системой, но время ответа и стоимость услуги.

Вычислительные платформы, как комплекс программно-аппаратного оборудования, операционного и сетевого окружения можно классифицировать по спектру информационных услуг, предоставляемых пользователям.

Грубая классификация пользователей /Л.Н. Королев/ открытых систем и аппаратного оборудования такова:

1. Самым массовым пользователем открытых систем можно считать владельцев ПК, используемых для бытовых нужд, число таких пользователей приближается к сотне миллионов. Современные платформы - ПК таких пользователей предоставляют им доступ к сетям по телефонным каналам. Оборудование - 90% платформы IBM PC (Intei x86) и программное обеспечение, созданное для этой архитектуры.
2. Пользователи, работающие в сфере бизнеса: банковская сфера,маркетинг, складской учет и т.д. Им требуется доступ к более производительной вычислительной технике, к глобальным сетям, требуются услуги по созданию и изменению СУБД и другие средства для разработки своих приложений. Использует ПК, но переходит на рабочие станции, хост-машины, до многопроцессорных супер-ЭВМ.
3. Пользователи этого инженерного класса занимаются разработкой приложений для промышленного производства. Для них характерен доступ к пакетам прикладных программ. Ориентируется на рабочие станции фирм DEC,HP и другие.
4. Пользователи, проводящие научные расчеты. Для этого класса пользователей необходим доступ высокопроизводительным вычислителям. Мощные платформы: Amdal(IBM), Cray(CDC),SPP(HP).
5. Пользователи - студенты, требующие услуги по освоению новых информационных технологий. Высшая школа и университетская наука, главным образом, ориентируется на рабочии станции SUN.

Основные характеристики, области применения ЭВМ различных классов

Понятие архитектуры ЭВМ

Сложность современных вычислительных машин закономерно привела к понятию архитектуры вычислительной машины, охватывающей комплекс общих вопросов ее построения, существенных в первую очередь для пользователя, интересующегося главным образом возможностями машины, а не деталями ее технического исполнения.

Круг вопросов, подлежащих рассмотрению при изучении архитектуры ЭВМ, можно условно разделить на вопросы общей структуры, организации вычислительного процесса и общения пользователя с машиной, вопросы логической организации представления, хранения и преобразования информации и вопросы логической организации совместной работы различных устройств, а также аппаратных и программных средств машины.

Основные характеристики ЭВМ

Важнейшими эксплуатационными характеристиками ЭВМ являются ее *производительность P и общий коэффициент эффективности машины:*

$$Э = P / (C_{ЭВМ} + C_{ЭКС}),$$

представляющий собой отношение ее производительности к сумме стоимости самой машины $C_{ЭВМ}$ и затрат на ее эксплуатацию за определенный период времени (например, период окупаемости капитальных затрат) $C_{ЭКС.}$

Так как часто трудно оценить затраты на эксплуатацию данной ЭВМ, а создатели новых машин стремятся приравнять эти затраты к нулю, то оценивают эффективность машины по упрощенной формуле

$$Э' = P / C_{ЭВМ} .$$

К наиболее распространенным характеристикам ЭВМ относятся:

- число разрядов в машинном слове (влияет на точность вычислений и диапазон представляемых в машине чисел);
- скорость выполнения основных видов команд;
- емкость оперативной памяти;
- максимальная скорость передачи информации между ядром ЭВМ (процессор или память) и внешним периферийным оборудованием;
- эксплуатационная надежность машины.

При создании новых ЭВМ обеспечивается значительное возрастание отношений производительность/стоимость и надежность/стоимость.

Лекция 5

СуперЭВМ

В настоящее время к сверх производительным машинам (системам) относят машины с производительностью в сотни и более GFLOP/s. Подобные машины используются для решения особенно сложных научно-технических задач, задач обработки больших объемов данных в реальном масштабе времени, поиска оптимальных в задачах экономического планирования и автоматического проектирования сложных объектов.

Самым ярким примером служит деятельность Cray Research. Эта фирма долго лидировала на рынке суперЭВМ. Но с разрушением «железного занавеса» спрос на ее компьютеры упал, что привело к распаду корпорации. В прошлом году в автокатастрофе погиб и ее основатель – Симур Крей.

Долгое время лидером в области суперкомпьютеров оставалась Cray Research,. По данным на начало 1997 года она занимала 43% всего рынка. Cray Research, приобретенная корпорацией Silicon Graphics в начале 1996 г, продает широкий спектр систем, начиная со старых моделей семейства J90 до машин новой серии Origin, в которых используется архитектура коммутации, построенная на базе процессора MIPS R10000.

Hewlett-Packard, владеет 7% этого сегмента рынка. Другими американскими производителями мощных компьютеров являются IBM, которая строит свои суперкомпьютеры SP на многокристальной версии PowerPC (14% рынка), а также Digital Equipment, предлагающая кластеры SMP-систем на базе процессора Alpha (13% рынка).

И наконец, японские фирмы Fujitsu и NEC занимают твердые позиции на рынке суперкомпьютеров, имея доли в 8 и 4% соответственно.

Сегодня самые быстрые суперЭВМ принадлежат Intel. В настоящее время Intel выполняет заказ министерства энергетики США.

В архитектуре суперЭВМ обнаруживается ряд принципиальных отличий от классической фоннеймонавской модели ЭВМ. Различные архитектуры суперЭВМ будут рассмотрены в теме «архитектурные особенности организации ЭВМ различных классов»

Малые и микроЭВМ.

Имеется большое число, условно говоря, «малых» применений вычислительных машин, таких, как автоматизация производственного контроля изделий, обработка данных при экспериментах, прием и обработка данных с линии связи, управление технологическими процессами, управление станками и разнообразными цифровыми терминалами, малые расчетные инженерные задачи.

В настоящее время малые и микроЭВМ встраивают в различные «умные» приборы (электросчетчики, микроволновки, стиральные машины, модемы, датчики и т.д.).

МинисуперЭВМ и суперминиЭВМ.

В классификации отсутствуют четкие границы между рассмотренными типами ЭВМ. В последнее время стали выделять два промежуточных типа.

К суперминиЭВМ относят высокопроизводительные ЭВМ содержащих один или несколько слабосвязанных процессоров, объединенных с общей магистралью (общей шиной). Для суперминиЭВМ характерно, что скорость выполнения его арифметических операций над числами с плавающей точкой существенно ниже скорости работы, определяемой по смеси команд, соответствующей информационно-логическим запросам. К этому типу можно отнести IBM-овский шахматный компьютер Deep Blue.

МинисуперЭВМ – это упрощенные (в частности за счет более короткого слова) многопроцессорные ЭВМ, чаще всего со средствами векторной и конвейерной обработки, с высокой скоростью выполнения операций над числами с плавающей точкой. К этому типу можно отнести ЭВМ с SMP(Symmetric multiprocessor) архитектурой.

Лекция 6. Организация памяти ЭВМ

Запоминающие устройства можно классифицировать по следующим критериям:

- по типу запоминающих элементов
- по функциональному назначению
- по типу способу организации обращения
- по характеру считывания
- по способу хранения
- по способу организации

По типу запоминающих элементов

Полупроводниковые
Магнитные
Конденсаторные
Оптоэлектронные
Голографические
Криогенные

По функциональному назначению

ОЗУ

По способу организации обращения

С последовательным поиском
С прямым доступом
С непосредственным доступом или Адресные
Ассоциативные
Стековые
Магазинные

По характеру считывания

С разрушением информации
Без разрушения информации

По способу хранения

Статические
Динамические

По способу организации

СОЗУ	Однокоординатные
ВЗУ	Двухкоординатные
ПЗУ	Трехкоординатные
ППЗУ	Двух- трехкоординатные

2.1. Общие сведения и классификация устройств памяти

Памятью ЭВМ называется совокупность устройств, служащих для запоминания, хранения и выдачи информации. Отдельные устройства, входящие в эту совокупность, называют *запоминающими устройствами* или памятями того или иного типа.

Производительность и вычислительные возможности ЭВМ в значительной степени определяются составом и характеристиками ее ЗУ. В составе ЭВМ используется одновременно несколько типов ЗУ, отличающихся принципом действия, характеристиками и назначением.

Основными операциями в памяти являются занесение информации в память — *запись* и выборка информации из памяти — *считывание*. Обе эти операции называются *обращением к памяти*.

При обращении к памяти производится считывание или запись некоторой единицы данных — различной для устройств разного типа. Такой единицей может быть, например, байт, машинное слово или блок данных.

Важнейшими характеристиками отдельных устройств памяти (запоминающих устройств) являются емкость памяти, удельная емкость, быстродействие.

Емкость памяти определяется максимальным количеством данных, которые могут в ней храниться.

Удельная емкость есть отношение емкости ЗУ к его физическому объему.

Плотность записи есть отношение емкости ЗУ к площади носителя. Например, у HDD емкостью до 10 Гб плотность записи составляет 2 Гбит на кв. дюйм.

Быстродействие памяти определяется продолжительностью операции обращения, т. е. временем, затрачиваемым на поиск нужной единицы информации в памяти и на ее считывание (*время обращения при считывании*), или временем на поиск места в памяти, предназначаемого для хранения данной единицы информации, и на ее запись в *память (время обращения при записи*).

Продолжительность обращения к памяти (время цикла памяти) при считывании

$$t_{обр}^{счит} = t_{дост}^{счит} + t_{счит} + t_{реген}$$

где $t_{дост}^{счит}$ — время доступа, определяющееся промежутком времени между моментом начала операции обращения при считывании до момента, когда становится возможным доступ к данной единице информации; $t_{счит}$ — продолжительность самого физического процесса считывания, т. е. процесса обнаружения и фиксации состояний соответствующих запоминающих элементов или участков поверхности носителя информации.

В некоторых устройствах памяти считывание информации сопровождается ее разрушением (стиранием). В таком случае цикл обращения должен содержать операцию восстановления (регенерации) считанной информации на прежнем месте в памяти.

Продолжительность обращения (время цикла) при записи

$$t_{обр}^{зап} = t_{дост}^{зап} + t_{подг} + t_{зап}$$

где $t_{дост}^{зап}$ — время доступа при записи, т. е. время от момента начала обращения при записи до момента, когда становится возможным доступ к запоминающим элементам (или участкам поверхности носителя), в которые производится запись; $t_{подг}$ — время подготовки, расходуемое на приведение в исходное состояние запоминающих элементов или участков

поверхности носителя информации для записи определенной единицы информации (например, байта или слова); $t_{зап}$ — время занесения информации, т. е. изменения состояния запоминающих элементов (участков поверхности носителя). Большей частью

$$t_{обр}^{счит} = t_{дост}^{зап} = t_{дост}$$

В качестве продолжительности цикла обращения к памяти принимается величина

$$t_{обр} = \max(t_{обр}^{счит}, t_{дост}^{зап}).$$

В зависимости от реализуемых в памяти операций обращения различают: а) память с произвольным обращением (возможны считывание и запись данных в память); б) память только для считывания информации («постоянная» или «односторонняя»). Запись информации в постоянную память производится в процессе ее изготовления или настройки.

Эти типы памяти соответствуют терминам RAM (random access memory — память с произвольным обращением) и ROM (read only memory — память только для считывания).

По способу организации доступа различают устройства памяти с непосредственным (произвольным), с прямым (циклическим) и последовательным доступами.

В памяти с *непосредственным (произвольным)* доступом время доступа, а поэтому и цикл обращения не зависят от места расположения участка памяти, с которого производится считывание или в который записывается информация. В большинстве случаев непосредственный доступ реализуется при помощи электронных (полупроводниковых) ЗУ. В подобных памятях цикл обращения обычно составляет 70 и менее наносекунд. Количество разрядов, считываемых или записываемых в памяти с непосредственным доступом параллельно во времени за одну операцию обращения, называется *шириной выборки*.

В двух других типах памяти используются более медленные электромеханические процессы. В устройствах *памяти с прямым доступом*, к которым относятся дисковые устройства, благодаря непрерывному вращению носителя информации возможность обращения к некоторому участку носителя для считывания или записи циклически повторяется. В такой памяти время доступа составляет обычно от нескольких долей секунды до нескольких десятков миллисекунд.

В памяти *с последовательным доступом* производится последовательный просмотр участков носителя информации, пока нужный участок носителя не займет некоторое исходное положение. Характерным примером является ЗУ на магнитных лентах, т.н. стримеры (*streamer*). Время доступа может в неблагоприятных случаях расположения информации достигнуть нескольких минут.

Хорошим примером ленточного накопителя является применение адаптера АРВИД с VHS видеомагнитофоном. Емкость этого накопителя составляет 4ГБ/180мин.

Запоминающие устройства различаются также по выполняемым в ЭВМ функциям, зависящим в частности, от места расположения ЗУ в структуре ЭВМ.

Требования к емкости и быстродействию памяти являются противоречивыми. Чем больше быстродействие, тем технически труднее достигается и дороже обходится увеличение емкости памяти. Стоимость памяти составляет значительную часть общей стоимости ЭВМ. Поэтому память ЭВМ организуется в виде иерархической структуры запоминающих устройств, обладающих различными быстродействием и емкостью. В общем случае ЭВМ содержит следующие типы памяти, в порядке убывания быстродействия и возрастания емкости.

Иерархическая структура памяти позволяет экономически эффективно сочетать хранение больших объемов информации с быстрым доступом к информации в процессе обработки.

Таблица 2.1.

Оперативной или основной памятью (ОП) называют устройство, которое служит для хранения информации (данных программ, промежуточных и конечных результатов обработки), непосредственно используемой в процессе выполнения операций в арифметико-логическом

устройстве (АЛУ) и устройстве управления (УУ) процессора.

В процессе обработки информации осуществляется тесное взаимодействие процессора и ОП. Из ОП в процессор поступают команды программы и операнды, над которыми производятся предусмотренные командой операции, а из процессора в ОП направляются для хранения промежуточные и конечные результаты обработки.

Характеристики ОП непосредственно влияют на основные показатели ЭВМ и в первую очередь на скорость ее работы. На текущий момент оперативная память имеет емкость от нескольких МБ до нескольких ГБ и цикл обращения около 60 нс и менее. Запоминающие устройства ОП изготовляются на интегральных микросхемах с большой степенью интеграции (полупроводниковые ЗУ).

В последнее время ряд фирм заявили о начале серийного выпуска чипов динамической памяти емкостью 1Гб. Признанным лидером является Samsung. Самым массовым изделием на сегодняшний день можно считать 64 Мб чипы. В ближайший год предполагается широкое применение 128Мб и 256Мб чипов.

В ряде случаев быстродействие ОП оказывается недостаточным, и в состав машины приходится включать СОП (буферную или кэш-память на несколько сотен или тысяч килобайт с циклом обращения, составляющим несколько наносекунд. Такие СОП выполняются на чипах статической памяти. Быстродействие КЭШа должно соответствовать скорости работы арифметико-логических и управляющих устройств процессора. Сверхоперативная (буферная) память используется для промежуточного хранения считываемых процессором из ОП участков программы и групп данных, в качестве рабочих ячеек программы, индексных регистров, для хранения служебной информации, используемой при управлении вычислительным процессом. Она выполняет роль согласующего звена между быстродействующими логическими устройствами процессора и более медленной ОП.

В качестве ОП и СОП используются быстродействующие ЗУ с произвольным обращением и непосредственным доступом.

Обычно емкость ОП оказывается недостаточной для хранения всех необходимых данных в ЭВМ. Поэтому ЭВМ содержит в своем составе несколько ЗУ с прямым доступом на дисках (емкость одного ЗУ на HDD дисках 1 - 30 Гбайт) и несколько ЗУ с последовательным доступом на магнитных лентах (емкость одного ЗУ 4 – 35 Гбайт).

Оперативная память вместе с СОП и некоторыми другими специализированными памятями процессора образуют *внутреннюю память* ЭВМ (рис. 4.1). Электромеханические ЗУ образуют *внешнюю память* ЭВМ, а сами они поэтому называются *внешними запоминающими устройствами* (ВЗУ).

Запоминающее устройство любого типа состоит из запоминающего массива, хранящего информацию, и блоков, служащих для поиска в массиве, записи и считывания (а в ряде случаев и для регенерации) информации.

Лекция 7

2.2 Адресная, ассоциативная и стековая организации памяти

Запоминающее устройство с произвольным обращением, как правило, содержит множество одинаковых запоминающих элементов, образующих запоминающий массив (ЗМ). Массив разделен на отдельные ячейки; каждая из них предназначена для хранения двоичного кода, число разрядов в котором определяется шириной выборки памяти (в частности, это может быть одно, половина или несколько машинных слов). Способ организации памяти зависит от методов размещения и поиска информации в запоминающем массиве. По этому признаку различают адресную, ассоциативную и стековую (магазинную) памяти.

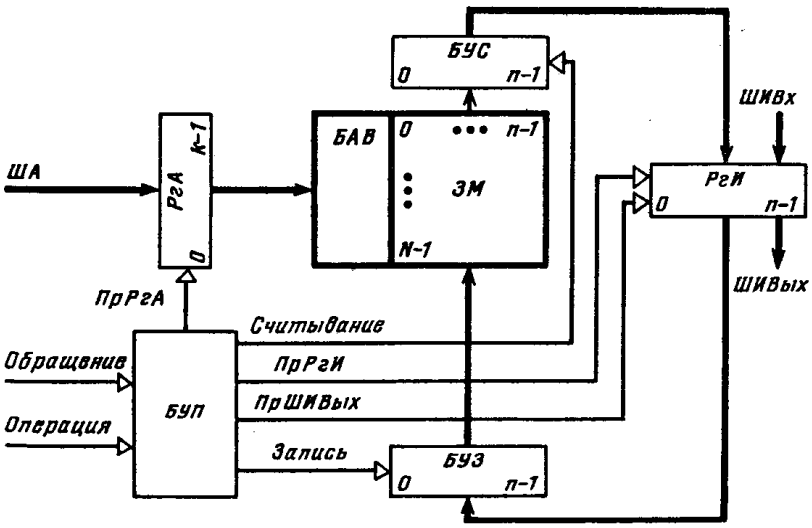
Адресная память. В памяти с адресной организацией размещение и поиск информации в ЗМ основаны на использовании адреса хранения слова (числа, команды и т. п.). Адресом служит номер ячейки ЗМ, в которой это слово размещается.

При записи (или считывании) слова в ЗМ инициирующая эту операцию команда должна указывать адрес (номер ячейки), по которому производится запись (считывание).

Типичная структура адресной памяти, содержит запоминающий массив из N-разрядных ячеек и его аппаратное обрамление, включающее в себя регистр адреса *РгА*, имеющий k ($k \approx \log$ N) разрядов, информационный регистр *РгИ*, блок адресной выборки *БАВ*, блок усилителей считывания *БУС*, блок разрядных усилителей-формирователей сигналов записи *БУЗ* и блок управления памятью *БУП*.

По коду адреса в *РгА БАВ* формирует в соответствующей ячейке памяти сигналы, позволяющие произвести в ячейке считывание или запись слова.

Цикл обращения к памяти инициируется поступлением в *БУП* извне сигнала *Обращение*. Общая часть цикла обращения включает в себя прием в *РгА* с шины адреса *ША* адреса обращения и прием в *БУП* и расшифровку управляющего сигнала *Операция*, указывающего вид запрашиваемой операции (считывание или запись).



Далее при считывании *БАВ* дешифрирует адрес, посылает сигналы считывания в заданную адресом ячейку ЗМ, при этом код записанного в ячейке слова считывается усилителями считывания БУС и передается в *РгИ*. Операция считывания завершается выдачей слова из *РгИ* на выходную информационную шину *ШИВых*.

При записи помимо выполнения указанной выше общей части цикла обращения производится прием записываемого слова с входной информационной шины *ШИВх* и *РгИ*. Затем в выбранную *БАВ* ячейку записывается слово из *РгИ*.

Блок управления *БУП* генерирует необходимые последовательности управляющих сигналов, инициирующих работу отдельных узлов памяти.

Ассоциативная память. В памяти этого типа поиск нужной информации производится не по адресу, а по ее содержанию (по ассоциативному признаку). При этом поиск по ассоциативному признаку (или последовательно по отдельным разрядам этого признака) происходит параллельно во времени для всех ячеек запоминающего массива. Во многих случаях ассоциативный поиск позволяет существенно упростить и ускорить обработку данных. Это достигается за счет того, что в памяти этого типа операция считывания информации совмещена с выполнением ряда логических операций.

Типичная структура ассоциативной памяти представлена на рис. 4.3. Запоминающий массив 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 регистр ассоциативного признака *РгАП* в разряды 0..n-1 поступает n-разрядный ассоциативный запрос, а в регистр маски *РгМ* — код маски поиска, при этом n-разряд *РгМ* устанавливается в 0. Ассоциативный поиск производится лишь для совокупности разрядов *РгАП*, которым соответствуют 1 в *РгМ* (незамаскированные разряды *РгАП*). Для слов, в которых цифры в разрядах совпали с незамаскированными разрядами *РгАП*, комбинационная схема *КС* устанавливает в 1 соответствующие разряды регистра совпадения *РгСв* и 0 в остальные разряды. Таким образом, значение *j*-го разряда в *РгСв* определяется выражением

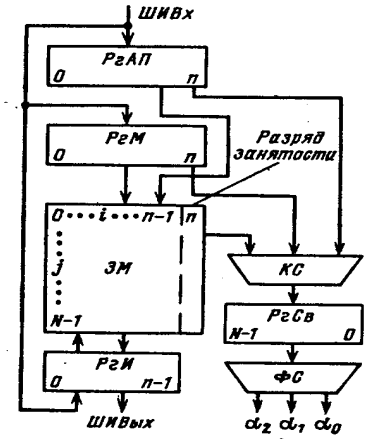


Рис. 2.2. Структура ассоциативной памяти

$$РгСв(j)=\bigwedge_{i=0}^{i=n-1}\{\overline{РгАП[i]\oplus ЗМ[j,i]}\vee\overline{РгМ[i]}\}\quad(0\le j\le N-1)$$

где *РгАП*[*i*], *РгМ*[*i*] и *ЗМ*[*j*, *i*] — значения *i*-го разряда соответственно *РгАП, РгМ* и *j*-и ячейки *ЗМ*.

Комбинационная схема формирования результата ассоциативного обращения *ФС* формирует из слова, образовавшегося в *РгСв*, сигналы α_0, α_1, α_2, соответствующие случаям отсутствия слов в *ЗМ*, удовлетворяющих ассоциативному признаку, и наличия одного (и более) такого слова.

Формирование содержимого *РгСв* и сигналов α_0, α_1, α_2 по содержимому *РгАП, РгМ* и *ЗМ* называется *операцией контроля ассоциации*. Эта операция является составной частью операций считывания и записи, хотя она имеет и самостоятельное значение.

При считывании сначала производится контроль ассоциации по ассоциативному признаку в *РгАП*. Затем при $\alpha_0 = 1$ считывание отменяется из-за отсутствия искомой информации, при $\alpha_1 = 1$ считывается в *РгИ* найденное слово, при $\alpha_2 = 1$ в *РгИ* считывается слово из ячейки, имеющей наименьший номер среди ячеек, отмеченных 1 у *РгСв*. Из *РгИ* считанное слово выдается на *ШИВых*.

При записи сначала отыскивается свободная ячейка. Для этого выполняется операция контроля ассоциации при *РгАП* = 111...10 и *РгМ* = 00...01, при этом свободные ячейки отмечаются 1 в *РгСв*. Для записи выбирается свободная ячейка с наименьшим номером. В нее записывается слово, поступившее с *ШИВх* в *РгИ*.

С помощью операции контроля ассоциации можно, не считывая слов из памяти, определить по содержимому *РгСв*, 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 слов, заключенных в определенных границах, поиск максимального (минимального) числа и др. Ассоциативная память применяется, например, в аппаратуре динамического распределения ОП.

Отметим, что для ассоциативной памяти необходимы запоминающие элементы, 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.

Стековая память, так же как и ассоциативная, является безадресной. Стековую память можно рассматривать как совокупность ячеек, образующих одномерный массив, в котором соседние ячейки связаны друг с другом разрядными цепями передачи слов. Запись нового слова производится в верхнюю ячейку (ячейку 0), при этом все ранее записанные слова (включая слово, находившееся в ячейке 0), сдвигаются вниз, в соседние ячейки с большими на 1 номерами. Считывание возможно

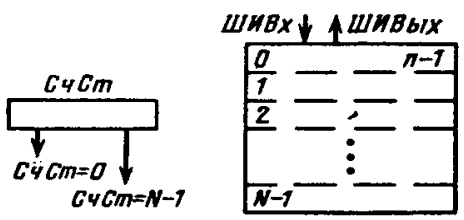


Рис.2.3. Стековая память.

только из верхней (нулевой) ячейки памяти, при этом, если производится считывание с удалением, все остальные слова в памяти сдвигаются вверх, в соседние ячейки с большими номерами. В этой памяти порядок считывания слов соответствует правилу: последним поступил — первым обслуживается. В ряде устройств рассматриваемого типа предусматривается также операция простого считывания слова из нулевой ячейки (без его удаления и сдвига слова в памяти). Иногда стековая память снабжается счетчиком стека *СчСт,* показывающим количество занесенных в память слов. Сигнал *СчСт* = 0 соответствует пустому стеку, *СчСт* = *N* - 1 — заполненному стеку.

Обычно стековую память организуют, используя адресную память. В этом случае счетчик стека, как правило, отсутствует, так как количество слов в памяти можно выявить по указателю стека. Широкое применение стековая память находит при обработке вложенных структур данных, при выполнении безадресных команд и прерываний.

Лекция 8

Архитектурная организация процессора ЭВМ

Процессор занимает в архитектуре ЭВМ центральное место, осуществляя управление взаимодействием всех основных компонент, входящих в состав ЭВМ Он непосредственно осуществляет обработку информации, и программное управление данным процессом дешифрирует и выполняет команды программ, организует обращения к оперативной памяти (ОП), в нужных случаях инициирует операции ввода/вывода и работу периферийных устройств, воспринимает и обрабатывает запросы, поступающие как от устройств ЭВМ, так и из внешней среды (организация системы прерываний). Выполнение каждой команды состоит из выполнения более мелких операций - микрокоманд, выполняющих определенные элементарные действия. Набор микрокоманд определяется системой команд и логической структурой конкретной ЭВМ. Таким образом, каждая команда ЭВМ реализуется соответствующей микропрограммой, хранящейся в постоянном запоминающем устройстве (ПЗУ). В некоторых ЭВМ (в первую очередь, специализированных) все или часть команд реализуются аппаратно, что позволяет повышать их производительность за счет потери определенной части гибкости системы команд машины. Как один, так и второй способ реализации команд ЭВМ имеет свои плюсы и минусы.

Язык микропрограммирования предназначен для описания цифровых устройств, функционирующих на уровне регистров. Он имеет простые и наглядные средства описания машинных слов, регистров, шин и других базовых элементов ЭВМ. С учетом сказанного, иерархию языков описания вычислительного процесса на ЭВМ можно представить, в общем случае, на четырех уровнях: (1) булева операция (функционирование комбинационных ЛС) => (2) микрокоманда (функционирование узлов ЭВМ) => (3) команда (функционирование ЭВМ) => (4) оператор ЯВУ (описание алгоритма решаемой задачи). Для определения временных соотношений между микрокомандами устанавливается единица времени (такт), в течение которой выполняется самая продолжительная микрокоманда. Поэтому выполнение одной команды ЭВМ синхроимпульсами, генерируемыми специальным устройством процессора - тактовым генератором, тактовая частота (измеряемая в МГц) в значительной степени определяет быстродействие ЭВМ. Естественно, для других классов ЭВМ данный показатель иным образом связывается

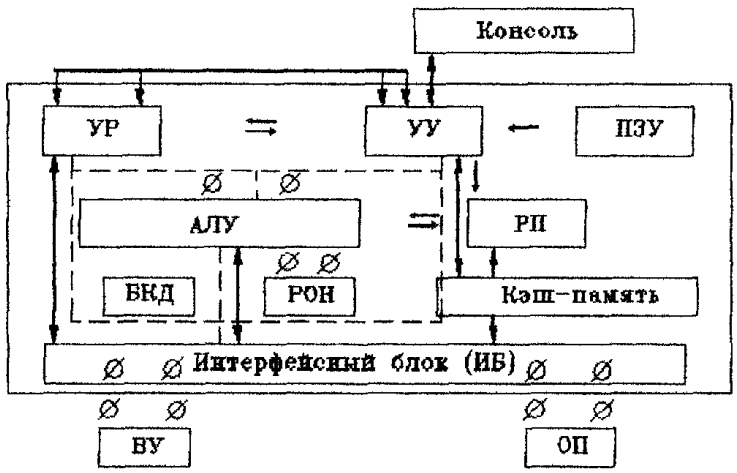


Рис 1 Функциональная схема центрального процессора ЭВМ

с производительностью, определяемой такими дополнительными факторами, как.

- ширина доступа в память,
- время выборки,
- разрядность,
- архитектура процессора и его сопроцессоров,

Укрупненная схема центрального процессора (ЦП) некоторой формальной ЭВМ представлена на рисунке, где изображены только основные его блоки управляющие регистры (УР), устройство управления (УУ), ПЗУ, арифметико-логическое устройство (АЛУ), регистровая память (РП), кэш-память и интерфейсный блок (ИБ). Наряду с перечисленными ЦП содержит ряд других блоков (прерывания, защиты ОП, контроля и диагностики и др.), структура и назначение которых здесь не рассматриваются. Блок УУ вырабатывает последовательность управляющих сигналов, инициирующих выполнение соответствующей последовательности микрокоманд (находящихся в ПЗУ), реализующей текущую команду. Наряду с этим УУ координирует функционирование всех устройств ЭВМ посредством посылки управляющих сигналов обмен данными ЦП <-> ОП, хранение и обработка информации, интерфейс с пользователем, тестирование и диагностика и др. Поэтому УУ целесообразно рассматривать как отдельный блок ЦП; однако на практике большинство управляющих схем распределены по всей ЭВМ. Они связаны большим числом управляющих линий, передающих сигналы для синхронизации операции во всех устройствах ЭВМ и принимающих сигналы об их состоянии. Блок УР предназначен для временного хранения управляющей информации и содержит регистры и счетчики, участвующие совместно с УУ в управлении вычислительным процессом регистр состояния ЦП, программы (ССП), счетчик команд (СК) представляет собой регистр, хранящий в ОП адрес выполняемой команды (в период выполнения текущей команды его содержимое обновляется на адрес следующей команды), регистр команд (РК) содержит выполняемую команду (его выходы связаны с управляющими схемами, генерирующими распределенные во времени сигналы, необходимые для выполнения команд)

Блок РП содержит регистры сверхоперативной памяти (более высокого быстродействия, чем ОП) небольшого объема, позволяющие повысить быстродействие и логические возможности ЦП. Эти регистры используются в командах путем сокращенной регистровой адресации (указываются только номера регистров) и служат для хранения операндов, результатов операций, в качестве базовых и индексных регистров, указателей стэка и др. В некоторых ЦП базовые и индексные регистры входят в состав блока УТ, как правило, РП выполняется в виде быстродействующих полупроводниковых интегральных запоминающих устройств

Блок АЛУ служит для выполнения арифметических и логических операций над данными, поступающими из ОП и хранящимися в РП, и работает под управлением УУ. АЛУ выполняет арифметические операции над бинарными числами с фиксированной и плавающей точками, над десятичными числами, производит обработку символьной информации над словами фиксированной и переменной длины. Логические операции производятся над отдельными битами, группами битов, байтами и их последовательностями. Тип выполняемой АЛУ операции определяется текущей командой функционирующей в данный момент программы, точнее, АЛУ служит для выполнения любой операции, задаваемой ему УУ. В общем случае обрабатываемая ЭВМ информация состоит из слов, содержащих фиксированное число n битов (например n =8. 16. 32, 64, 128 бит). В этом случае АЛУ должно иметь возможность производить операции над n-битными словами операнды поступают из ОП на регистры АЛУ, а УУ указывает операцию, которую необходимо над ними произвести, результат каждой арифметико-логической операции сохраняется в специальном регистре-сумматоре, являющимся основным регистром для арифметико-логических операций.

Сумматор соединен с вентильными схемами для выполнения необходимых операций над его содержимым и содержимым других регистров. Некоторые ЭВМ имеют несколько сумматоров, при количестве, большем 4, они выделяются в специальную группу регистров общего назначения (РОН). Конструктивно АЛУ выполняется на одном или нескольких БИС/СБИС, при этом ЦП может иметь одно АЛУ универсального назначения или несколько

специализированных для отдельных видов операции. В последнем случае увеличивается структурная сложность ЦП, но повышается его быстродействие за счет специализации и упрощения схем вычисления отдельных операций. Такой подход широко используется в современных ЭВМ общего назначения и супер-ЭВМ для повышения их производительности Несмотря на различные классы ЭВМ, их АЛУ используют общие принципы выполнения арифметико-логических операции. Различия касаются схемотехнических решений организации АЛУ и принципов реализации операции, обеспечивающих ускорение их выполнения.

Интерфейсный блок (ИБ) обеспечивает обмен информацией ЦП с ОП и защиту участков ОП от несанкционированного для текущей программы доступа, а также связь ЦП с периферийными устройствами и другими внешними по отношению к нему устройствами (ВУ), в качестве которых могут выступать другие процессоры и ЭВМ. В частности, ИБ содержит два регистра, обеспечивающие связь с ОП - регистр адреса памяти (РАП) и регистр данных памяти (РДП). Первый регистр используется для хранения адреса ячейки ОП, с которой производится обмен данными, а второй содержит собственно данные обмена. Блок контроля и диагностики (БКД) предназначен для обнаружения сбоев и отказов узлов ЦП, восстановления работы текущей программы после сбоев и локализации неисправностей при отказах.

С учетом сказанного представим общую схему выполнения программ процессором. Выполнение программы, находящейся в ОП, начинается с того, что в СК засылается адрес первой ее команды, содержимое СК пересылается в РАП и в ОП посылается сигнал управления считыванием. Через некоторое время (соответствующее времени доступа к ОП) адресуемое слово (в данном случае первая команда программы) извлекается из ОП и загружается в РДП, затем содержимое РДП пересылается в СК. На этой стадии команда готова для декодирования ее УУ и выполнения. Если команда содержит операцию, которая должна быть выполнена АЛУ, то необходимо получить требуемые операнды. Если операнд находится в ОП (а он может быть также в УР), его необходимо выбрать из памяти. Для этого в РАП пересылается адрес операнда и начинается цикл чтения Операнд, выбранный из памяти в РДП, может быть передан в АЛУ. Выбрав таким образом один или несколько операндов, АЛУ может выполнить требуемую операцию, сохранив ее результат в одном из РОН. Если результат операции необходимо запомнить в ОП, он должен быть послан в РДП Адрес ячейки, в которую необходимо поместить результат, пересылается в РАП и начинается цикл записи. Между тем содержимое СК увеличивается, указывая следующую команду, которая должна выполняться. Таким образом, как только завершится выполнение текущей команды, может сразу же начаться выборка на выполнение следующей команды программы.

Помимо передачи данных между ОП и ЦП необходимо обеспечить обмен данными с ВУ, что делают машинные команды, управляющие вводом/выводом. Естественный порядок выполнения программ может нарушаться при поступлении сигнала прерывания. Прерывание является требованием на обслуживание, которое осуществляется ЦП, выполняющим соответствующую программу обработки прерывания (ПОП). Так как прерывание и его обработка могут изменить внутреннее состояние ЦП, то оно сохраняется в ОП перед началом работы ПОП. Сохранение состояния достигается пересылкой содержимого РК, УР и некоторой управляющей информации в ОП. После завершения ПОП состояние ЦП восстанавливается, позволяя продолжить выполнение прерванной программы.

Лекция 9

Структура машинной команды

Машинная команда представляет собой закодированное по определенным правилам указание микропроцессору на выполнение некоторой операции или действия. Приведенный на рис. 1 формат машинной команды является самым общим. Максимальная длина машинной команды - 15 байт. Реальная команда может содержать гораздо меньшее количество полей,

Префиксы. Необязательные элементы машинной команды, каждый из которых состоит из одного байта или может отсутствовать. В памяти префиксы предшествуют команде. Назначение префиксов - модифицировать операцию, выполняемую командой. Прикладная программа может использовать следующие типы префиксов:

- *Префикс замены сегмента*. В явной форме указывает, какой сегментный регистр используется в данной команде для адресации стека или данных. Префикс отменяет выбор сегментного регистра по умолчанию. Префиксы замены сегмента имеют следующие значения:
  - 2eh - замена сегмента cs;
  - 36h - замена сегмента ss;
  - 3eh - замена сегмента ds;
  - 26h - замена сегмента es;
  - 64h - замена сегмента fs;
  - 65h - замена сегмента gs.
- *Префикс разрядности адреса* уточняет разрядность адреса (32 или 16-разрядный).
  Каждой команде, в которой используется адресный операнд, ставится в соответствие разрядность адреса этого операнда. Этот адрес может иметь разрядность 16 или 32 бит. Если разрядность адреса для данной команды 16 бит, это означает, что команда содержит 16-разрядное смещение (см. *рис. 1*), оно соответствует 16-разрядному смещению адресного операнда относительно начала некоторого сегмента. В контексте *рис. 2* это смещение называется эффективный адрес. Если разрядность адреса 32 бит, это означает, что команда содержит 32-разрядное смещение (см. *рис. 1*), оно соответствует 32-разрядному смещению адресного операнда относительно начала сегмента и по его значению формируется 32-битное смещение в сегменте. С помощью префикса разрядности адреса можно изменить действующее по умолчанию значение разрядности адреса. Это изменение будет касаться только той команды, которой предшествует префикс.
- *Префикс разрядности операнда* аналогичен префиксу разрядности адреса, но указывает на разрядность операндов (32 или 16-разрядные), с которыми работает команда. В соответствии с какими правилами устанавливаются значения атрибутов разрядности адреса и операндов по умолчанию?
  В реальном режиме и режиме виртуального i8086 значения этих атрибутов - 16 бит.
  В защищенном режиме значения атрибутов зависят от состояния бита *D* в дескрипторах исполняемых сегментов (см. урок 16). Если *D* = 0, то значения атрибутов, действующие по умолчанию, равны 16 бит; если *D* = 1, то 32 бит. Значения префиксов разрядности операнда **66h** и разрядности адреса **67h**. Вы можете с помощью префикса разрядности адреса в реальном режиме использовать 32-разрядную адресацию, но при этом необходимо помнить об ограниченности размера сегмента величиной 64 Кбайт. Аналогично префиксу разрядности адреса вы можете использовать префикс разрядности операнда в реальном режиме для работы с 32-разрядными операндами (к примеру, в арифметических командах).
- *Префикс повторения* используется с цепочечными командами (командами обработки строк). Этот префикс "зацикливает" команду для обработки всех элементов цепочки. Система команд поддерживает два типа префиксов:
  - *безусловные* (rep - 0f3h), заставляющие повторяться цепочечную команду некоторое количество раз;
  - *условные* (repe/repz - 0f3h, repne/repnz - 0f2h), которые при зацикливании проверяют некоторые флаги, и в результате проверки возможен досрочный выход из цикла.

- **Код операции**. Обязательный элемент, описывающий операцию, выполняемую командой. Многим командам соответствует несколько кодов операций, каждый из которых определяет нюансы выполнения операции.

  Последующие поля машинной команды определяют местоположение операндов, участвующих в операции, и особенности их использования. Рассмотрение этих полей связано со способами задания операндов в машинной команде и потому будет выполнено позже.

- **Байт режима адресации *modr/m***. Значения этого байта определяет используемую форму адреса операндов. Операнды могут находиться в памяти в одном или двух регистрах. Если операнд находится в памяти, то байт *modr/m* определяет компоненты (смещение, базовый и индексный регистры), используемые для вычисления его эффективного адреса (см. *рис. 2*). В защищенном режиме для определения местоположения операнда в памяти может дополнительно использоваться байт *sib* (Scale-Index-Base - масштаб-индекс-база). Байт *modr/m* состоит из трех полей (см. *рис. 1*):
  - поле *mod* определяет количество байт, занимаемых в команде адресом операнда (см. *рис. 1*, поле *смещение в команде*).
    Поле *mod* используется совместно с полем *r/m*, которое указывает способ модификации адреса операнда смещение в команде.
    К примеру, если *mod* = 00, это означает, что поле *смещение в команде* отсутствует, и адрес операнда определяется содержимым базового и (или) индексного регистра. Какие именно регистры будут использоваться для вычисления эффективного адреса, определяется значением этого байта.
    Если *mod* = 01, это означает, что поле *смещение в команде* присутствует, занимает один байт и модифицируется содержимым базового и (или) индексного регистра.
    Если *mod* = 10, это означает, что поле *смещение в команде* присутствует, занимает два или четыре байта (в зависимости от действующего по умолчанию или определяемого префиксом размера адреса) и модифицируется содержимым базового и (или) индексного регистра.
    Если *mod* = 11, это означает, что операндов в памяти нет: они находятся в регистрах. Это же значение байта *mod* используется в случае, когда в команде применяется непосредственный операнд;
  - поле *reg/кол* определяет либо регистр, находящийся в команде на месте первого операнда, либо возможное расширение кода операции;
  - поле *r/m* используется совместно с полем *mod* и определяет либо регистр, находящийся в команде на месте первого операнда (если *mod* = 11), либо используемые для вычисления эффективного адреса (совместно с полем смещение в команде) базовые и индексные регистры.
- **Байт масштаб-индекс-база (байт *sib*)** используется для расширения возможностей адресации операндов.
  На наличие байта *sib* в машинной команде указывает сочетание одного из значений 01 или 10 поля *mod* и значения поля *r/m*= 100. Байт *sib* состоит из трех полей:
  - *поля масштаба* ***ss***. В этом поле размещается масштабный множитель для индексного компонента *index*, занимающего следующие три бита байта *sib*.
    В поле *ss* может содержаться одно из следующих значений: 1, 2, 4, 8.
    При вычислении эффективного адреса на это значение будет умножаться содержимое индексного регистра. Более подробно с практической точки зрения эта расширенная возможность индексации рассматривается на уроке 12 при обсуждении вопросов работы с массивами;
  - *поля* ***index*** - используется для хранения номера индексного регистра, который применяется для вычисления эффективного адреса операнда;

- *поля **base*** - используется для хранения номера базового регистра, который также применяется для вычисления эффективного адреса операнда. Напомню, что в качестве базового и индексного регистров могут использоваться практически все регистры общего назначения.

- **Поле смещения в команде**. 8, 16 или 32-разрядное целое число со знаком, представляющее собой, полностью или частично (с учетом вышеприведенных рассуждений), значение эффективного адреса операнда.
- **Поле непосредственного операнда**. Необязательное поле, представляющее собой 8, 16 или 32-разрядный непосредственный операнд. Наличие этого поля, конечно, отражается на значении байта *modr/m*.

Лекция 10

Способы задания операндов команды

В ходе предыдущего изложения мы поневоле касались вопроса о том, где располагаются операнды, с которыми работает машинная команда, и как это отражается на содержимом ее полей.
В этой части занятия мы рассмотрим этот вопрос более систематизировано и в полном объеме. Это позволит нам уже со следующего занятия перейти непосредственно к практическим вопросам программирования на языке ассемблера.

Операнд задается неявно на микропрограммном уровне. В этом случае команда явно не содержит операндов. Алгоритм выполнения команды использует некоторые объекты по умолчанию (регистры, флаги в eflags и т. д.).
Например, команды cli и sti неявно работают с флагом прерывания if в регистре eflags, а команда xlat неявно обращается к регистру al и строке в памяти по адресу, определяемому парой регистров ds:bx.

Операнд задается в самой команде (непосредственный операнд). Операнд находится в коде команды, то есть является ее частью. Для хранения такого операнда в команде выделяется поле длиной до 32 бит (см. *рис. 1*). Непосредственный операнд может быть только вторым операндом (источником). Операнд получатель может находиться либо в памяти, либо в регистре.
Например: mov ax,0ffffh пересылает в регистр ax шестнадцатеричную константу ffff. Команда add sum,2 складывает содержимое поля по адресу sum с целым числом 2 и записывает результат по месту первого операнда, то есть в память.

Операнд находится в одном из регистров. Регистровые операнды указываются именами регистров. В качестве регистров могут использоваться:

- 32-разрядные регистры EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP;
- 16-разрядные регистры AX, BX, CX, DX, SI, DI, SP, BP;
- 8-разрядные регистры AH, AL, BH, BL, CH, CL, DH, DL;
- сегментные регистры CS, DS, SS, ES, FS, GS.

Например, команда add ax,bx складывает содержимое регистров ax и bx и записывает результат в bx. Команда dec si уменьшает содержимое si на 1.

Операнд располагается в памяти. Это наиболее сложный и в то же время наиболее гибкий способ задания операндов. Он позволяет реализовать следующие два основных вида адресации: *прямую* и *косвенную*.

В свою очередь, *косвенная* адресация имеет следующие разновидности:

- косвенная базовая адресация; другое ее название - регистровая косвенная адресация;
- косвенная базовая адресация со смещением;
- косвенная индексная адресация со смещением;
- косвенная базовая индексная адресация;
- косвенная базовая индексная адресация со смещением.

Операндом является порт ввода/вывода. Как мы уже отмечали, помимо адресного пространства оперативной памяти микропроцессор поддерживает адресное пространство ввода-вывода, которое используется для доступа к устройствам ввода-вывода. Объем адресного пространства ввода-вывода составляет 64 Кбайт. Для любого устройства компьютера в этом пространстве выделяются адреса. Конкретное значение адреса в пределах этого пространства называется портом ввода-вывода. Физически порту ввода-вывода соответствует аппаратный регистр (не путать с регистром микропроцессора), доступ к которому осуществляется с помощью специальных команд ассемблера in и out. Например,
in al,60h ;ввести байт из порта 60h
Регистры, адресуемые с помощью порта ввода-вывода, могут иметь разрядность 8, 16 или 32 бит, но для конкретного порта разрядность регистра фиксирована. Команды **in** и **out** работают с фиксированной номенклатурой объектов. В качестве источника информации или получателя применяются так называемые регистры-аккумуляторы eax, ax, al. Выбор регистра определяется разрядностью порта. Номер порта может задаваться непосредственным операндом в командах in и out или значением в регистре dx. Последний способ позволяет динамически определить номер порта в программе. Например:

```
mov  dx,20h    ;записать номер порта 20h в регистр dx
mov  al,20h    ;записать значение 20h в регистр al
   out  dx,al     ;вывести значение 20h в порт 20H
```

Операнд находится в стеке.

Команды могут совсем не иметь операндов, иметь один или два операнда. Большинство команд требуют двух операндов, один из которых является операндом-источником, а второй - операндом назначения. Важно то, что один операнд может располагаться в регистре или памяти, а второй операнд обязательно должен находиться в регистре или непосредственно в команде. Непосредственный операнд может быть только операндом-источником. В двухоперандной машинной команде возможны следующие сочетания операндов:

- регистр-регистр;
- регистр-память;
- память-регистр;
- непосредственный операнд-регистр;
- непосредственный операнд-память.

У данного правила есть исключения, которые касаются:

- команд работы с цепочками, которые могут перемещать данные из памяти в память;
- команд работы со стеком, которые могут переносить данные из памяти в стек, также находящийся в памяти;

- команд типа умножения, которые кроме операнда, указанного в команде, используют еще и второй, неявный операнд.

Из перечисленных сочетаний операндов наиболее часто употребляются *регистр-память* и *память-регистр*.
Ввиду их важности рассмотрим их подробнее. Обсуждение мы будем сопровождать примерами команд ассемблера, которые будут показывать, как изменяется формат команды ассемблера при применении того или иного вида адресации. В связи с этим посмотрите еще раз на *рис. 2*, на котором показан принцип формирования физического адреса на адресной шине микропроцессора. Видно, что адрес операнда формируется как сумма двух составляющих - сдвинутого на 4 бит содержимого сегментного регистра и 16-битного эффективного адреса, который в общем случае вычисляется как сумма трех компонентов: базы, смещения и индекса.

Перечислим и затем рассмотрим особенности основных видов адресации операндов в памяти:

- Прямая адресация
- Косвенная базовая (регистровая) адресация
- Косвенная базовая (регистровая) адресация со смещением
- Косвенная индексная адресация со смещением
- Косвенная базовая индексная адресация
- Косвенная базовая индексная адресация со смещением

Лекция 11. Адресации

Прямая адресация

Это простейший вид адресации операнда в памяти, так как эффективный адрес содержится в самой команде и для его формирования не используется никаких дополнительных источников или регистров. Эффективный адрес берется непосредственно из поля смещения машинной команды (см. *рис. 1*), которое может иметь размер 8, 16, 32 бит. Это значение однозначно определяет байт, слово или двойное слово, расположенные в сегменте данных.

Прямая адресация может быть двух типов:

- *Относительная прямая адресация*. Используется для команд условных переходов, для указания относительного адреса перехода. Относительность такого перехода заключается в том, что в поле смещения машинной команды содержится 8, 16 или 32-битное значение, которое в результате работы команды будет складываться с содержимым регистра указателя команд ip/eip. В результате такого сложения получается адрес, по которому и осуществляется переход.
  Например:

```
jc m1     ;переход на метку m1, если флаг cf = 1
mov  al,2
...

m1:
```

- Несмотря на то, что в команде указана некоторая метка в программе, ассемблер вычисляет смещение этой метки относительно следующей команды (в нашем случае это mov al,2) и подставляет его в формируемую машинную команду jc.
- *Абсолютная прямая адресация*. В этом случае эффективный адрес является частью машинной команды, но формируется этот адрес только из значения поля смещения в

команде. Для формирования физического адреса операнда в памяти микропроцессор складывает это поле со сдвинутым на 4 бит значением сегментного регистра. В команде ассемблера можно использовать несколько форм такой адресации. Например:

```
mov     ax,dword ptr [0000]     ;записать слово по адресу
   ;ds:0000 в регистр ax
```

- Но такая адресация применяется редко - обычно используемым ячейкам в программе присваиваются символические имена. В процессе трансляции ассемблер вычисляет и подставляет значения смещений этих имен в формируемую им машинную команду в поле смещение в команде (см. *рис. 1*). В итоге получается так, что машинная команда прямо адресует свой операнд, имея, фактически, в одном из своих полей значение эффективного адреса.
  Например:

```
data    segment
per1    dw      5
...
data    ends
code    segment
        mov     ax,data
        mov     ds,ax
...

  mov     ax,per1  ;записать слово per1 (его физический адрес
ds:0000) в ax
```

- Мы получим тот же результат, что и при использовании команды mov ax,dword ptr [0000]

Остальные виды адресации относятся к *косвенным*. Слово "косвенный" в названии этих видов адресации означает то, что в самой команде может находиться лишь часть эффективного адреса, а остальные его компоненты находятся в регистрах, на которые указывают своим содержимым байт modr/m и, возможно, байт sib.

Косвенная базовая (регистровая) адресация

При такой адресации эффективный адрес операнда может находиться в любом из регистров общего назначения, кроме *sp/esp* и *bp/ebp* (это специфические регистры для работы с сегментом стека). Синтаксически в команде этот режим адресации выражается заключением имени регистра в квадратные скобки [ ]. К примеру, команда **mov ax,[ecx]** помещает в регистр ax содержимое слова по адресу из сегмента данных со смещением, хранящимся в регистре ecx. Так как содержимое регистра легко изменить в ходе работы программы, данный способ адресации позволяет динамически назначить адрес операнда для некоторой машинной команды. Это свойство очень полезно, например, для организации циклических вычислений и для работы с различными структурами данных типа таблиц или массивов.

Косвенная базовая (регистровая) адресация со смещением

Этот вид адресации является дополнением предыдущего и предназначен для доступа к данным с известным смещением относительно некоторого базового адреса. Этот вид адресации удобно использовать для доступа к элементам структур данных, когда смещение элементов известно заранее, на стадии разработки программы, а базовый (начальный) адрес структуры

должен вычисляться динамически, на стадии выполнения программы. Модификация содержимого базового регистра позволяет обратиться к одноименным элементам различных экземпляров однотипных структур данных.
К примеру, команда **mov ax,[edx+3h]** пересылает в регистр ax слова из области памяти по адресу: содержимое edx + 3h.
Команда **mov ax,mas[dx]** пересылает в регистр ax слово по адресу: содержимое dx плюс значение идентификатора mas (не забывайте, что транслятор присваивает каждому идентификатору значение, равное смещению этого идентификатора относительно начала сегмента данных).

Косвенная индексная адресация со смещением

Этот вид адресации очень похож на косвенную базовую адресацию со смещением. Здесь также для формирования эффективного адреса используется один из регистров общего назначения. Но индексная адресация обладает одной интересной особенностью, которая очень удобна для работы с массивами. Она связана с возможностью так называемого масштабирования содержимого индексного регистра. Что это такое?
Посмотрите на *рис. 1*. Нас интересует байт *sib*. При обсуждении структуры этого байта мы отмечали, что он состоит из трех полей. Одно из этих полей - поле масштаба ss, на значение которого умножается содержимое индексного регистра.
К примеру, в команде **mov ax,mas[si*2]** значение эффективного адреса второго операнда вычисляется выражением mas+(si)*2. В связи с тем, что в ассемблере нет средств для организации индексации массивов, то программисту своими силами приходится ее организовывать.
Наличие возможности масштабирования существенно помогает в решении этой проблемы, но при условии, что размер элементов массива составляет 1, 2, 4 или 8 байт.

Косвенная базовая индексная адресация

При этом виде адресации эффективный адрес формируется как сумма содержимого двух регистров общего назначения: базового и индексного. В качестве этих регистров могут применяться любые регистры общего назначения, при этом часто используется масштабирование содержимого индексного регистра.
Например:

```
mov     eax,[esi][edx]
```

В данном примере эффективный адрес второго операнда формируется из двух компонентов (esi)+(edx).

Косвенная базовая индексная адресация со смещением

Этот вид адресации является дополнением косвенной индексной адресации. Эффективный адрес формируется как сумма трех составляющих: содержимого базового регистра, содержимого индексного регистра и значения поля смещения в команде.
К примеру, команда **mov eax,[esi+5][edx]** пересылает в регистр eax двойное слово по адресу: (esi) + 5 + (edx).
Команда **add ax,array[esi][ebx]** производит сложение содержимого регистра ax с содержимым слова по адресу: значение идентификатора array + (esi) + (ebx).

Лекция 12

Способы адресации

Следует различать понятия *адресный код* в команде $A_К$ и *исполнительный адрес* $A_И$. Адресный код - это информация об адресе операнда, содержащаяся в команде. Исполнительный адрес - это номер ячейки памяти, к которой производится фактическое обращение. В современных ЭВМ адресный код, как правило, не совпадает с исполнительным адресом.

Выбор способов адресации, формирования исполнительного адреса и преобразования адресов является одним из важнейших вопросов разработки ЭВМ. Рассмотрим способы адресации, используемые в современных ЭВМ.

Подразумеваемый операнд. В команде не содержится явных указаний об адресе операнда; операнд подразумевается и фактически задается кодом операции команды. Данный способ используется не часто, однако имеется несколько важных случаев его применения. В качестве примера можно привести команды подсчета, в которых к некоторому числу, (содержимому счетчика) прибавляется фиксированное приращение, чаще единица младшею разряда. Один из операндов - число в счетчике - обычно адресуется явным методом, второй операнд - приращение - не адресуется, в памяти машины не содержится и является подразумеваемым.

Подразумеваемый адрес. В команде не содержится явных указаний об адресе участвующего в операции операнда или адреса, по которому помещается результат операции, но этот адрес подразумевается. Например, команда может содержать адреса обоих операндов, участвующих в операции, при этом подразумевается, что результат операции помещается по адресу одного из операндов, или команда указывает только адрес одного операнда, а адрес второго, которым является содержимое специального регистра (называемого регистром результата или аккумулятором), подразумевается.

Непосредственная адресации. В команде содержится не адрес операнда, а непосредственно сам операнд. При непосредственной адресации не требуется обращения к памяти для выборки операнда и ячейки для его хранения Это способствует уменьшению времени выполнения программы и занимаемого ею объема памяти. Непосредственная адресация удобна для хранения различного рода констант, однако следует иметь в виду, что при этом способе адресации длина операнда короче кода команды, поскольку часть разрядов команды занята под код операции.

Прямая адресация. Исполнительный адрес совпадает с адресной частью команды. Этот способ адресации был общепринятым в первых вычислительных машинах и продолжает применяться в настоящее время в комбинации с другими способами.

В указанной форме непосредственная адресация реализуется в ЭВМ со сравнительно длинным машинным словом (32 разряда и более).

Относительная адресация или базирование. Исполнительный адрес определяется суммой адресного кода команды $A_К$ и некоторого числа $A_Б$, называемого базовым адресом:

$$A_И = A_К + A_Б.$$

Для хранения базовых адресов в машине могут быть предусмотрены регистры или специально выделенные для этой цели ячейки памяти (базовые регистры). В команде выделяется поле *B* для указания номера базового регистра.

Относительная адресация позволяет при меньшей длине адресного кода команды обеспечить доступ к любой ячейке памяти. Для этого число разрядов в базовом адресе выбирают таким, чтобы можно было адресовать любую ячейку ОП, а адресный код $A_К$, самой команды используют для представления лишь сравнительно короткого «смещения» (обозначают буквой D). Смещение D определяет положение операнда относительно начала массива, задаваемого

базовым адресом $A_Б$. Рисунок 3.5 поясняет процесс формирования исполнительного адреса.

Большей частью исполнительный адрес при базировании образуется с помощью сумматора согласно выражению (рис. 3.5)

$$A_И = \begin{cases} (B) + D, если\ B \neq 0 \\ D, если\ B = 0 \end{cases}$$

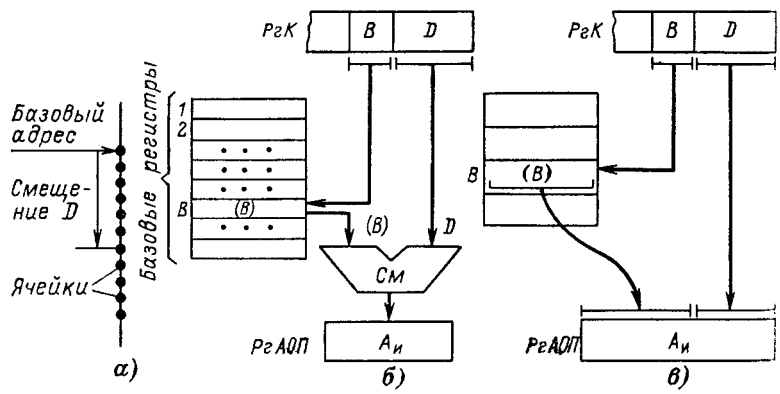


Рис. 3.5. Базирование (относительная адресация): а - образование адреса элемента одномерного массива; б -

где B и D - коды (числа), оящие в соответствующих полях оманды: (B) -содержимое егистра с номером B. При B = 0 гносительная адресация локируется.

Более подробно обращение к ОП при относительной адресации ожно представить в следующем виде:

если *РгК*[*B*] = 0

то *РгАОП* := *РгК*[0]

иначе *РгАОП* :=

РгВ + *РгК*[*D*];

Счит: *РгИОП* :=

ОП [*РгАОП*}:

Суммирование при образовании $A_И$ связано с потерей времени. Поэтому применяют также формирование исполнительного адреса методом совмещения (рис. 3.5,*в*). В этом случае базовый адрес содержит старшие, а смещение - младшие разряды исполнительного адреса, которые объединяются в *РгАОП* согласно операции конкатенации слова:

РгАОП := *РгВ*|*РгК*[*D*];

Однако при совмещении базовый адрес может задавать не любую ячейку, а только те ячейки, адреса которых содержат 0 в младших разрядах, соответствующих смещению.

Относительная адресация обеспечивает так называемую перемещаемость программ, т. е. возможность передвижения программ в памяти без изменений внутри самой программы.

Укороченная адресация. Для уменьшения длины кода команды часто применяется так называемая укороченная адресация. Суть ее сводится к тому, что в команде задаются только младшие разряды адресов, старшие разряды при этом подразумеваются нулевыми. Такая адресация позволяет использовать только небольшую группу фиксированных ячеек с начальными (короткими) адресами и поэтому может применяться лишь совместно с другими способами адресации.

Регистровая адресация есть частный случай укороченной, когда в качестве фиксированных ячеек с короткими адресами используются регистры (ячейки сверхоперативной или местной памяти) процессора. Например, если таких регистров 16, то для адреса достаточно четырех двоичных разрядов. Регистровая адресация наряду с сокращением длины адресов операндов позволяет увеличить скорость выполнения операций, так как уменьшается число обращений к ОП.

Косвенная адресация. Адресный код команды указывает адрес ячейки памяти, в которой

находится адрес операнда или команды. Таким образом, косвенная адресация может быть иначе определена как «адресация адреса».

На косвенную адресацию указывает код операции команды, а в некоторых ЭВМ в команде отводится специальный разряд (указатель адресации - УА), и цифра 0 или 1 в нем указывает, является адресная часть команды прямым адресом или косвенным. Обращение к ОП за операндом при косвенной адресации представляет собой следующую процедуру:

РгАОП := *РгК* [*А*];

Счит: *РгИОП* := *ОП*[*РгАОП*];

если *УА*=0 **то идти к** *М* **иначе** *РгАОП* := *РгИОП*;

Счит: *РгИОП* := *ОП* [*РгАОП*];

М: *РгАЛУ* := *РгИОП*;

В некоторых ЭВМ используется многоступенчатая косвенная адресация. В этом случае ячейки памяти содержат также разряд-указатель косвенной адресации (УА). Если этот разряд указывает на продолжение косвенной адресации, то машина последовательно выбирает из памяти адреса до тех пор, пока не будет найдена ячейка, в которой разряд-указатель определит прямую адресацию. Адрес из этой последней ячейки и является искомым исполнительным адресом.

Косвенная адресация широко используется в малых и микроЭВМ, имеющих короткое машинное слово, для преодоления ограничений короткого формата команды.

Рассмотрим широко применяемое в микропроцессорах, малых и микроЭВМ совместное использование укороченной (регистровой) и косвенной адресаций (рис.3.6). Пусть необходимо передать число 4527 из *Рг5* в ячейку ОП *1765*. Длина адресных полей команды достаточна лишь для указания коротких номеров регистров, она не позволяет задать в команде полный адрес ячейки памяти. Поэтому операнд *4527* указывается регистровой косвенной адресацией (УА=0), а для задания адреса *1765* приходится воспользоваться регистровой косвенной адресацией (УА=1), при которой в команде указывается номер регистра (в рассматриваемом примере *Рг3*), предварительно загруженного полным адресом ячейки, в которою производится передача.

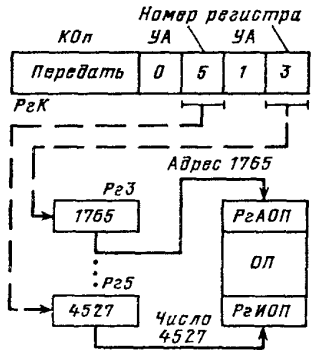


Рис. 3.6 Совместное использование регистровой прямой и регистровой косвенной адресации для преодоления ограничений короткого слова команды.

Автоинкрементная и автодекрементная адресации. Поскольку регистровая косвенная адресация требует предварительной загрузки регистра из ОП косвенным адресом, что связано с потерей времени, такой тип адресации особенно эффективен при обработке массива данных, если имеется механизм автоматического приращения или уменьшения содержимого регистра при каждом обращении к нему, называемый соответственно авто инкрементной и автодекрементной адресацией. В этом случае достаточно 1 раз загрузить в регистр адрес первого обрабатываемого элемента массива, а затем при каждом обращении к регистру в нем в результате инкрементной (декрементной) процедуры формируется адрес следующего элемента массива.

При автоинкрементной адресации по содержимому регистра сначала содержимое регистра используется как адрес операнда, а затем получает приращение, равное числу байт в элементе массива. При автодекрементной адресации сначала содержимое указанного в команде регистра уменьшается на число, равное числу байт в элементе массива, а затем используется как адрес операнда.

Автоинкрементная и автодекрементная адресации могут рассматриваться как упрощенный

вариант индексации - весьма важного механизма преобразования адресных частей команд и организации вычислительных циклов, поэтому их часто называют автоиндексацией.

Адресация слов переменной длины. Эффективность вычислительных систем, предназначенных для обработки данных (экономических, плановых и др.), повышается, если имеется возможность выполнять операции со словами переменной длины. В этом случае в машине должна быть предусмотрена адресация слов переменной длины, которая обычно реализуется путем указания в команде местоположения в памяти начала слова и его длины.

Обычно в ЭВМ одновременно используется несколько типов адресации Тип адресации указывается либо неявно кодом операции, либо в явной форме в специальном поле адресной части команды.

Стековая адресация

Стековая память, реализующая безадресное задание операндов, является эффективным элементом современной архитектуры ЭВМ, особенно широко используемым в микропроцессорах, малых и микроЭВМ, а также в некоторых суперЭВМ. Учитывая своеобразие стековой адресации, ее рассмотрение выделено в отдельный параграф.

Стек представляет собой группу последовательно пронумерованных регистров (*аппаратурный стек*) или ячеек памяти, снабженных указателем стека (обычно регистром) (УС), в котором автоматически при записи и считывании устанавливается номер (адрес) последней занятой ячейки стека (вершины стека}. При операции записи заносимое в стек слово помещается в следующую по порядку свободную ячейку стека, а при считывании из стека извлекается последнее поступившее в него слово. Таким образом, в стеке реализуется правило *«последний пришел - первый ушел».*

Указанное правило при обращении к стеку реализуется автоматически, и поэтому при операциях со стеком возможно безадресное задание операнда — команда не содержит адреса ячейки стека, но содержит адрес (или он подразумевается) ячейки памяти или регистра, откуда слово передается в стек или куда помещается из стека.

Механизм стековой адресации поясняется на рис.3.7. При выполнении команды передачи в стек слова из регистра или ячейки ОП сначала указатель стека увеличивается на 1 (в *перевернутом стеке* уменьшается на 1), а затем слово помещается в ячейку стека, указываемую УС. При команде загрузки из стека регистра или ячейки памяти сначала слово извлекается из вершины стека, а затем указатель стека уменьшается на 1 (в перевернутом стеке увеличивается на 1). Как это ни кажется на первый взгляд удивительным, но при соответствующем расположении операндов в стеке можно вычислять выражения полностью безадресными командами, указывающими только вид операции. Такая команда извлекает из стека в соответствии с кодом операции один или два операнда, выполняет над ними предписанную операцию и заносит результат в стек.

Вычисления с использованием стековой памяти удобно описывать и программировать с помощью польской инверсной (бес скобочной) записи арифметических выражений ПОЛИЗ. Эта запись производится по следующему правилу: читаем арифметическое выражение слева направо и последовательно друг за другом выписываем встречающиеся операнды. Как только окажется, что все операнды некоторой операции выписаны, записываем знак этой операции и продолжаем выписывать операнды. Если операция имеет операндом результат некоторой предыдущей операции и знак последней выписан, то считаем этот операнд

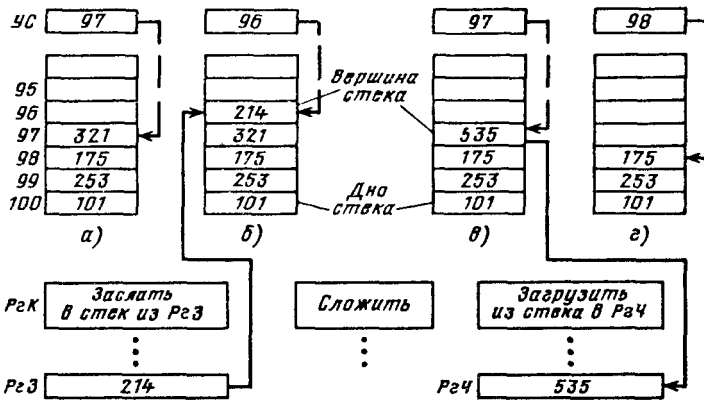


Рис 9.7 Стековая адресация в «перевернутом стеке»:

а - исходное состояние стека; *б* - стек после выполнения команды «Заслать в стек из *Рг3»; в* - стек после выполнения команды «Сложить», *г* — стек после выполнения команды «Загрузить из стека в *Рг4»*

выписанным.

Например, выражение

$(k + l - m)\ (p - s)$

в ПОЛИЗ имеет вид

$k\ l + m - p\ s - *.$

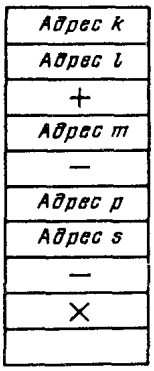


Рис. 3.8. Программа вычисления

Выражение в ПОЛИЗ не содержит скобок, но порядок действий определяет однозначно. При использовании стековой памяти последовательность символов в выражении ПОЛИЗ, может рассматриваться как программа вычисления исходного арифметического выражения (рис. 3.8), если под буквами понимать команды засылки, содержащие только адреса в ОП соответствующих операндов, засылаемых в стек, а под знаками операции безадресные команды, содержащие только коды операций. Команда второго типа инициирует извлечение из стека двух (или одного) слов, выполнение над ними указанной в команде операции и засылку результата в вершину стека

Безадресные команды на основе стековой адресации предельно сокращают формат команд, экономят память и способствуют повышению производительности ЭВМ.

Однако при такой структуре команд возникают осложнения с построением команд передачи управления и работы с периферийными устройствами.

В современной архитектуре процессоров и микропроцессоров стек и стековая адресация широко ИСПОЛЬЗУЮТСЯ при организации переходов к подпрограммам и возврате от них, а также в системах прерывания. Весьма широко стеки и безадресные команды используются в вычислительном комплексе «Эльбрус».

Лекция 13

Команды, процедуры и микропрограммы передачи управления в программах

Рассмотрим команды, управляющие порядком исполнения команд.

При естественном порядке после выполнения очередной команды выбирается команда, расположенная в следующей по порядку ячейке памяти. Обычно адрес команд хранится в специальном регистре, называемом счетчиком команды (СчК), содержимое которого после выполнения каждой команды автоматически увеличивается на 1, а если память имеет побайтную адресацию, то оно увеличивается на столько, сколько байт содержит текущая команда (*приращение адреса команды* L_K).

Выборка очередной и формирование адреса следующей команды (АСК), часто называемого «продвинутым адресом», происходит (при естественном порядке) следующим образом (рис. 9.9):

РгАОП := *СчК*;

Счит:

РгИОП := *ОП*[*РгАОП*];

РгК := *РгИОП*;

. . . . . .

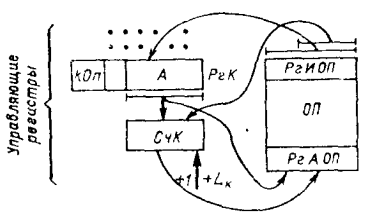


Рис. 3.9. Формирование адреса следующей команды при естественном порядке выборки команд и при его нарушении командами перехода и командой

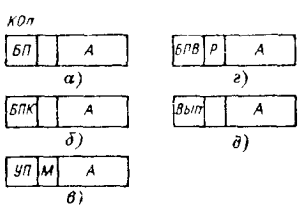


Рис. 3.10. Упрощенные структуры команд перехода безусловною (*а*), безусловного по косвенному адресу (*б*), условного (*в*), безусловного с возвратом (*г*) и

. . . .

(Выполнение текущей команды)

. . . . . .

$АСК_{ЕП}$: СчК := СчК + $L_К$;

Естественный порядок выполнения команд может быть нарушен: 1) командами перехода (командами передачи управления); 2) командами замещения (*Выполнение*); 3) сменой состояния программы; 4) запросами прерывания программы.

Команды перехода (передачи управление). Известны многочисленные варианты команд перехода, однако общий принцип состоит в том, что адресная часть команды перехода непосредственно или после суммирования с содержимым базового регистра передается в счетчик команд, и, следовательно, после данной команды может быть выполнена команда из произвольной ячейки памяти, номер которой определяется адресной частью команды перехода.

Для упрощения изложения материала рассмотрим процедуры выполнения команд переходов без относительной адресации.

Различают два основных вида команд перехода: *безусловный* переход (БП) и *условный* (УП). Упрощенные структуры этих команд показаны на рис. 3.10.

Команда БП (рис. 3.10, *а*) совершает переход всегда независимо от каких-либо условий. Обыкновенно следующей за командой БП выполняется команда, адрес которой указан в адресной части команды БП, т. е.

$АСК_{ЕП}$: СчК := РгК [A};

Другим вариантом команды БП является *безусловный переход по косвенному адресу* (БПК) (рис. 3.10,*б*). На косвенную адресацию указывает код операции команды или специальное поле в формате команды, определяющее тип адресации. Управление передается команде, расположенной в ячейке, адрес которой указан в адресной части команды БПК. В этом случае выполняется следующая процедура:

РгАОП := РгК[A];

Счит : РгИОП := ОП[РгАОП];

$АСК_{БПК}$: СчК := РгИОП[A];

При *условном переходе* адрес следующем команды зависит от выполнения некоторого условия. Обычно, если условие выполняется, происходит переход к команде по адреса, указанному в адресной части команды УП. В противном случае выбирается следующая по порядку команда, адрес которой определяется, как обычно, содержимым СчК, увеличенным *на приращение адреса команды* $L_К$. Так же как и при БП, в команде УП могут использоваться относительная и косвенная адресации.

Условия перехода (например, результат предыдущей команды равен, больше или меньше 0, переполнение разрядной сетки и др.) задаются кодом операции УП или в виде отдельного поля *маска условия М* в команде УП (рис. 3.10, *в*). В этом случае следующая команда определяется по правилу

если Ус [M] = 1

то $АСК_{УП}$: СчК := РгК [A]

иначе $АСК_{ЕП}$: СчК := СчК + $L_К$;

Здесь Ус[M] —логическая функция (условие), заданная маской *М*. Если условие перехода задается кодом операции команды, то в приведенном выражении Ус[M] надо заменить на Ус[К0п].

Обычно выполнение машинной команды сопровождается выработкой кода признака результата *ПР*, формируемого в специальном регистре *РгПР*. Команда УП анализирует сформированный предыдущей командой *ПР*. Смысл кодов *ПР* может быть установлен различным для разных операций. Например, в ЕС ЭВМ двухразрядный признак результата *ПР* при выполнении арифметических операций принимает значения, приведенные в табл. 3.1 (*ПР*[0] и *ПР*[1] — нулевой и первый *разряды ПР*).

С помощью двухразрядной маски можно задать в качестве условия УП любое значение ПР.

Таблица 9.1

Результат операции	Код *ПР*		Маска условия (ЕС ЭВМ)			
	ПР[0]	*ПР*[1]	*М*[0]	*М*[1]	*М*[2]	*М*[3]
Равен 0	0	0	1	0	0	0
Меньше 0	0	1	0	1	0	0
Больше 0	1	0	0	0	1	0
Переполнение	1	1	0	0	0	1

В машинах ЕС ЭВМ используется четырехразрядная маска, и это дает возможность задавать в качестве условия перехода выполнение не только любого из *ПР*, но и их дизъюнкции (если в маске *М* несколько 1).

В общем случае имеем

$$Ус[M] = M[0]\overline{ПР[0]}\ \overline{ПР[1]} \vee M[1]\overline{ПР[0]}\ ПР[1] \vee M[2]ПР[0]\ \overline{ПР[1]} \vee M[3]ПР[0]\ ПР[1]$$

Отметим, что при *М*=1111 команда УП выполняет безусловный переход, а при *М*=0000 (пустая команда) действует естественный порядок выборки команд. Таким образом, исключается необходимость в отдельной команде безусловного перехода.

Команды условных переходов позволяют реализовать программы с разветвлениями в зависимости от промежуточных результатов вычислений или состояния машины.

Важным случаем передачи управления являются *безусловные переходы к подпрограммам*. Их особенность состоит в том, что помимо перехода они должны обеспечить по окончании подпрограммы возврат к исходной программе, к той точке ее, откуда был совершен переход. Обычно для переходов к под программам используется специальная команда *Безусловный перевод с возвратом* (БПВ). По этой команде (рис. 3.11) сначала адрес возврата $A_{ВОЗ}$, т.е. содержимое *СчК* (увеличенное па «приращение адреса команды» $L_К$), запоминается по адресу *Р*, указанному в команде БПВ. затем в счетчик команд заносится содержимое поля *А* команды БПВ, т.е. адрес *А* начала подпрограммы. В конце подпрограммы размещается команда возврата, которая представляет собой команду БПК, указывающую путем косвенной адресации адрес ячейки (или регистра), в которой находится адрес $A_{ВОЗ}$.

Формально всю эту процедуру после приема в *РгК* команды БПВ можно представить в следующем виде (см. рис. 3.9 и 3.11):

```
        СчК := СчК  + Lк;          образование адреса возврата A_ВОЗ
        РгИОП := СчК;
        РгАОП := РгК [Р];
Запись: ОП[РгАОП] = РгИОП;         запоминание адреса возврата A_ВОЗ, в ячейке Р
        СчК := РгК[А];             передача в СчК адреса начала подпрограммы
        . . . . . .
;                                  Выполнение подпрограммы
```

. . . . . .

; Считывание и передача в *РгК* заключающей подпрограммы команды БПК, содержащей в поле *А* косвенный адрес *Р* *)

РгАОП := *РгК*[*А*];

Счит: РгИОП := *ОП*[*РгАОП*]; извлечение адреса возврата $А_{ВОЗ}$

$АСК_{БПВ}$: *СчК* := *РгИОП* [*А*]; возврат к основной программе

- команде в ячейке $А_{ВОЗ}$

Операция замещения, реализуемая командой *Выполнение* (*Вып*), состоит в том, что вместо очередной команды, соответствующей естественному порядку выборки команд, исполняется замещающая команда, указываемая адресной частью команды *Вып*, а затем, если только замещающая команда не оказалась командой перехода, восстанавливается приостановленный на время выполнения команды *Вып* естественный порядок выборки команд. Команда *Вып* должна сохранять неизменным содержимое *СчК*. Поэтому адрес замещающей команды берется не из *СчК*, а из *РгК*. (см. рис. 9.10). Соответствующую процедур после приема в *РгК* команды *Вып* можно представить в виде

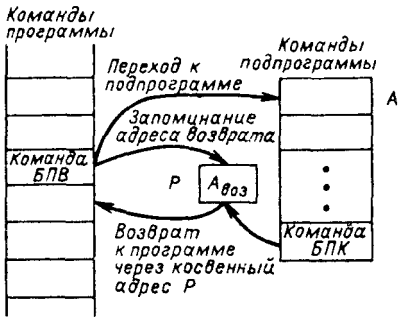


Рис. 3.11. Обращение к подпрограмме с помощью команды безусловного перехода с

РгАОП := *РгК*[*А*];

Счит:РгИОП := *ОП*[*РгАОП*];

РгК := *РгИОП;* выборка замещающей команды

. . . . . .

; Выполнение замещающей команды

. . . . . .

СчК := *СчК* + L_K восстановление естественного порядка выборки команд.

Лекция 14

Особенности RISC архитектуры

RISC-архитектура предполагает реализацию в ЭВМ сокращенного набора простейших, но часто употребляемых команд, что позволяет упростить аппаратурные средства процессора и благодаря этому получить возможность повысить его быстродействие.

При использовании RISC-архитектуры выбор набора команд и структуры процессора (микропроцессора) направлены на то, чтобы команды набора выполнялись за один машинный цикл процессора. Выполнение более сложных, но редко встречающихся операций обеспечивают подпрограммы..

В ЭВМ с RISC *машинным циклом называют время, в течение которого производится выборка двух операндов из регистров, выполнение операции в АЛУ и запоминание результата в регистре.* Большинство команд в RISC являются быстрыми командами типа «регистр - регистр» и выполняются без обращений к ОП. Обращения к ОП сохраняются лишь в командах загрузки регистров из памяти и запоминания в ОП. Чтобы это было возможным, процессор должен содержать достаточно большое число общих регистров.

Благодаря характерным для RISC-архитектуры особенностям - сокращенному набору команд (обычно не более 50-100), небольшому числу (обычно 2-3) простых способов адресации (в основном регистровой), небольшому числу простых форматов команд с фиксированными размерами и функциональным назначением их полей - упрощается управляющее устройство процессора, который в этом случае обходится без микропрограммного уровня управления и управляющей памяти, и его УУ может быть выполнено на «схемной логике».

Уменьшение количества выполняемых команд и другие отмеченные выше особенности RISC-архитектуры приводят к столь значительному упрощению структуры процессора, что становится возможной его реализация на одном кристалле вместе с большим регистровым файлом и кэшем.

Большое число регистров, особенно при наличии обеспечивающего их эффективное использование «оптимизирующего компилятора», позволяет до предела сократить обращение к ОП путем сохранения на регистрах промежуточных результатов, передачи через регистры операндов из одних программ в другие программы или подпрограммы, отказа от передач на сохранение в ОП содержимого регистров при прерываниях.

Особенностью RISC-архитектуры является механизм *перекрывающихся регистровых окон,* предназначенный для уменьшения числа обращений к ОП и межрегистровых передач, что способствует повышению производительности ЭВМ.

Процедурам динамически выделяются небольшие группы регистров фиксированной длины (регистровые окна). Окна последовательно выполняемых процедур перекрываются, благодаря чему возможна передача параметров от одной процедуры к другой При вызове процедуры процессор переключается на работу с другим регистровым окном, при этом не возникает необходимости в передаче содержимого регистров в память.

Окно состоит из трех подгрупп регистров (рис. 9.21). Первая подгруппа содержит параметры, переданные данной процедуре от ее вызвавшей, и результаты для вызывающей процедуры при возврате в нее. Вторая подгруппа содержит локальные переменные процедуры. Третья, являясь буфером для двустороннего обмена между данной и ею вызываемой следующей процедурами, передает последней параметры от данной, которая, в свою очередь, получает через этот буфер результаты от ею вызванной процедуры. Таким образом, одна и та же подгруппа для данной процедуры является регистрами временного хранения, а для следующей — регистрами параметров. Отдельное окно, доступное всем процедурам программы, выделяется для ее глобальных переменных.

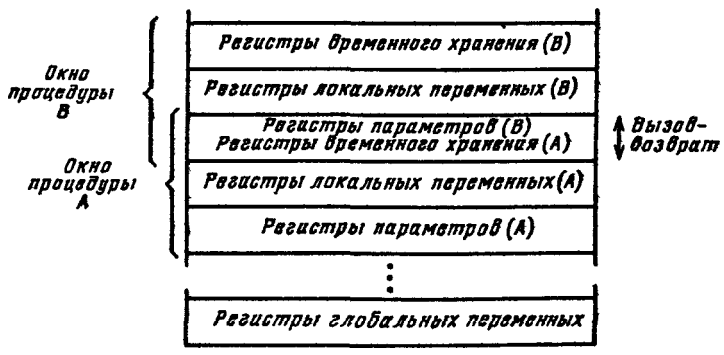


Рис 9.21. СНК-архитектура ЭВМ. Перекрывающиеся регистровые окна.

В настоящее время за рубежом пущен ряд микропроцессоров с RISC-хитектурой. Примером являются ысокопроизводительные станции на зе микропроцессора Alpha 21264, икропроцессор для WindowsCE Intel rongARM.

Несмотря на начавшееся пользование RISC-архитектуры в выпускаемой промышленностью ЭВМ, продолжаются дискуссии вокруг достоинств и недостатков этой архитектуры. К последним, в частности, относят большую длину кода программы после компиляции по сравнению с длиной кода машин обычной архитектуры. Например, при эмуляции команд x86 в среднем на каждую его команду требуется пять-шесть команд машины с RISC-архитектурой. Однако, как показали исследования, выигрыш в скорости выполнения команд перекрывает проигрыш от удлинения объектного кода программы.

В 1989 фирме Intel удалось на основе RISC-архитектуры создать однокристальный микропроцессор 80860, который практически представляет собой кремниевый эквивалент суперЭВМ Gray-1.

Классификация архитектур микропроцессоров

Развитие архитектуры ЭВМ, направленное на повышение их производительности, во многих случаях идет по пути усложнения процессоров путем расширения системы (набора) команд, введения сложных команд, выполняющих процедуры, приближающиеся к примитивам языков высокого уровня, увеличения числа используемых способов адресации и т. д.

Однако расширение и усложнение набора команд порождают и ряд нежелательных побочных эффектов.

Расширение набора команд, увеличение числа способов адресации, введение сложных команд сопровождаются увеличением длины кода команды, в первую очередь, кода операции, что может приводить к использованию «расширяющегося кода операции», увеличению числа форматов команд. Это вызывает усложнение и замедление процесса дешифрации кода операции и других процедур обработки команд. Возрастающая сложность процедур обработки команд заставляет прибегать к микропрограммным управляющим устройствам с управляющей памятью вместо более быстродействующих УУ с «жесткой» («схемной») логикой.

Усложнение процессора делает более трудным или даже невыполнимым реализацию его на одном кристалле интегральной микросхемы, что благодаря сокращению длин межсоединений могло бы облегчить достижение высокой производительности.

Анализ кода программ, генерируемого компиляторами языков высокого уровня, показал, что практически используется только ограниченный набор простых команд форматов "регистр, регистр -> регистр" и "регистр <-> память". Компиляторы не в состоянии эффективно использовать сложные команды. Именно это наблюдение способствовало формированию концепции процессоров с сокращенным набором команд, так называемых RISC-процессоров.

Другим обстоятельством, фактически приведшим к появлению RISC-процессоров, было развитие архитектуры конвейерных процессоров типа Cray. В этих процессорах используются отдельные наборы команд для работы с памятью и отдельные наборы команд для преобразования информации в регистрах процессора. Каждая такая команда единообразно разбивается на небольшое количество этапов с одинаковым временем исполнения (выборка команды, дешифрация команды, исполнение, запись результата), что позволяет построить эффективный конвейер процессора, способный каждый такт выдавать результат исполнения очередной команды.

Однако конвейерность исполнения команд породила проблемы, связанные с зависимостями по данным и управлению между последовательно запускаемыми в конвейер командами. Например, если очередная команда использует результат предыдущей, то ее исполнение невозможно в течение нескольких тактов, необходимых для получения этого результата. Аналогичные проблемы возникают при исполнении команд перехода по условию, когда данные, по которым производится переход, к моменту дешифрации команды условного перехода еще не готовы.

Эти проблемы решаются либо компилятором, устанавливающим очередность запуска команд в конвейере и вставляющим команды "Нет операции" при невозможности запуска очередной команды, либо специальной аппаратурой процессора, отслеживающей зависимости между командами и устраняющей конфликты.

После обособления RISC-процессоров в отдельный класс, процессоры с традиционными наборами команд стали называться CISC-процессорами с полным набором команд. Как правило, в этих процессорах команды имеют много разных форматов и требуют для своего представления различного числа ячеек памяти. Это обусловливает определение типа команды в ходе ее дешифрации при исполнении, что усложняет устройство управления процессора и препятствует повышению тактовой частоты до уровня, достижимого в RISC-процессорах на той же элементной базе.

Очевидно, что RISC-процессоры эффективны в тех областях применения, в которых можно продуктивно использовать структурные способы уменьшения времени доступа к оперативной

памяти. Если программа генерирует произвольные последовательности адресов обращения к памяти и каждая единица данных используется только для выполнения одной команды, то фактически производительность процессора определяется временем обращения к основной памяти. В этом случае использование сокращенного набора команд только ухудшает эффективность, так как требует пересылки операндов между памятью и регистром вместо выполнения команд "память, память - память". Программист должен учитывать необходимость локального размещения обрабатываемых данных, чтобы при пересылках между уровнями памяти по возможности все данные пересылаемых блоков данных принимали участие в обработке. Если программа будет написана так, что данные будут размещены хаотично и из каждого пересылаемого блока данных будет использоваться только небольшая их часть, то скорость обработки замедлится в несколько раз до скорости работы основной памяти. В качестве примера приведем в таблице 1.1 результаты замеров производительности микропроцессора Alpha 21066 233 Мгц при реализации преобразования Адамара при n = 8 - 20.

Таблица 1.1 Производительность микропроцессора Alpha 2I066 при выполнении преобразования Адамара

n	Производительность в условных алгоритмических операциях, млн.оп/с
8	150
10	133
11	73
>=12	20

Пример показывает, что, пока данные размещаются во внутрикристальной кэш-памяти, производительность высока. Как только объем данных превышает размер кэш-памяти и обращения в память идут в "равномерно" распределенные по объему адреса, производительность падает более, чем в 7 раз.

Развитие микропроцессоров происходит при постоянном стремлении сохранения преемственности программного обеспечения (ПО) и повышения производительности за счет совершенствования архитектуры и увеличения тактовой частоты. Сохранение преемственности ПО и повышение производительности, вообще говоря, противоречат друг другу. Процессоры с системой команд x86, относящиеся к классу CISC-процессоров, имеют более низкие тактовые частоты по сравнению с микропроцессорами ведущих компаний-изготовителей RISC-процессоров. Существуют приложения, на которых производительность x86 микропроцессоров значительно ниже, чем у RISC-процессоров, реализованных на той же элементной базе. Однако возможность использования совместимого ПО для различных поколений x86 процессоров, выпущенных в течение последнего десятилетия, обеспечивает им устойчивое доминирующее положение на рынке.

В настоящее время на основе пионерских разработок компаний NexGen и AMD, подхваченных компанией Intel, предпринята попытка решить проблему повышения производительности в рамках архитектуры x86. Эти компании в последних разработках, сохраняя преемственность по системе команд с CISC-микропроцессорами семейства x86, создают новые устройства с использованием элементов RISC-архитектуры. Примером такого подхода могут служить микропроцессоры Nx586 (NexGen), K5, K6 (AMD), Pentium PRO, Pentium II (Intel), использующие концепцию разделенной (decoupled) архитектуры и RISC ядра. В микропроцессор встраивается аппаратный транслятор, превращающий команды x86, в команды RISC-процессора. При этом одна команда x86 может порождать до четырех команд RISC-процессора. Исполнение команд происходит как в развитом суперскалярном процессоре. Компания Intel использовала этот подход в своем микропроцессоре Pentium Pro, что весьма укрепило ее позиции на фоне достижений RISC-архитектур.

ЛЕКЦИЯ 15

Суперскалярные процессоры
Архитектура суперскалярных процессоров

Есть два крайних подхода, при возможных промежуточных, к отображению присущего микропроцессору внутреннего параллелизма обработки данных на архитектурном уровне в системе команд. Первый подход более консервативен и состоит в том, что никакого указания на параллельную обработку внутри процессора система команд не содержит. Такие процессоры относятся к классу суперскалярных.

Второй подход - напротив полностью открывает все возможности параллельной обработки. В специально отведенных полях команды каждому из параллельно работающих обрабатывающих устройств предписывается действие, которое устройство должно совершить. Такие процессоры называются процессорами с длинным командным словом (VLIW или EPIC). Предполагается, что существуют компиляторы с языков высокого уровня, которые готовят программы для загрузки их в микропроцессоры.

Основная идея, определяющая развитие суперскалярных микропроцессоров, состоит в построении возможно большего количества параллельных структур при сохранении традиционных последовательных программ. Это означает, что компиляторы и аппаратура микропроцессора сами, без вмешательства программиста, обеспечивают загрузку параллельно работающих функциональных устройств микропроцессора.

В соответствии с моделью последовательного программирования, программы пишутся в предположении, что команды будут выполнены в том же порядке, в каком они представлены в программе. Однако с целью достижения большей эффективности современные процессоры пытаются выполнять несколько команд одновременно и в некоторых случаях в порядке, отличном от их исходной последовательности в программе. Это переупорядочение может быть выполнено в трансляторе и (или) в аппаратных средствах во время выполнения. Суперскалярные и VLIW-процессоры принадлежат классу архитектур, которые используют параллельность уровня команды (ILP).

ILP-процессоры и компиляторы обычно преобразуют полностью упорядоченное множество команд исходной программы в частично упорядоченное множество, структурированное зависимостями по данным и управлению. Зависимости по управлению (которые проявляются как переходы по условию) представляют главное препятствие высокопараллельному выполнению потому, что эти зависимости должны быть установлены прежде, чем будут выполнены все последующие команды.

Текст последовательной программы, представленной на языке высокого уровня, компилируется в машинный код, отражающий статическую структуру программы, т. е. упорядоченное множество команд (инструкций) в памяти компьютера. Процесс выполнения программы с конкретными наборами входных данных может быть представлен динамической структурой программы, т. е. множеством последовательностей инструкций в порядке их исполнения.

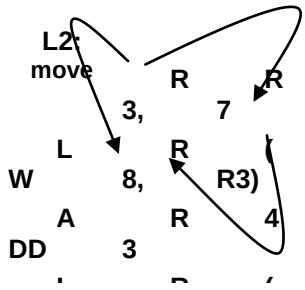
Повысить степень параллелизма программы можно изменяя соответствующим образом ее статическую или динамическую структуру. Поскольку статическая структура программы однозначно соответствует ее исходному тексту (в предположении неизменности компилятора), то изменение статической структуры сводится к изменению исходного кода, что, в общем случае, не всегда возможно. Динамическая же структура программы может быть изменена при неизменной статической структуре. И главной целью такого изменения должно быть повышение степени параллельного исполнения команд.

Допустимые границы преобразования динамической структуры программы задают существующие на множестве инструкций отношения: зависимость по управлению и

зависимость по данным. При описании архитектур суперскалярных процессоров часто используется модель окна исполнения. При исполнении программы микропроцессор как бы продвигает по статической структуре программы окно исполнения. Команды в окне могут исполняться параллельно, если между ними нет зависимости.

Для устранения зависимостей, вызванных командами переходов, используется метод предсказания, позволяющий извлекать и условно исполнять команды предсказанного перехода. Если позднее обнаруживается, что предсказание было сделано верно, то результаты условно исполненных команд принимаются. Если предсказание было ошибочным, состояние процессора восстанавливается на момент принятия решения о выполнении перехода.

Команды, помещенные в окно исполнения, могут быть зависимы по данным. Эти зависимости обусловлены использованием одних и тех же ресурсов памяти (регистров, ячеек памяти) в разных командах. Поэтому для правильного исполнения программы необходимо использование этих ресурсов в предписываемом программой порядке.



Все виды зависимостей по данным могут быть классифицированы по типу ассоциаций: RAR - "чтение после чтения", WAR - "запись после чтения" и WAW - "запись после записи", RAW - "чтение после записи".

Некоторые из зависимостей по данным могут быть устранены. **RAR,** по сути дела, соответствует отсутствию зависимостей, поскольку в данном случае порядок выполнения команд не имеет значения. Действительной зависимостью является только "чтение после записи" (RAW), так как необходимо прочитать предварительно записанные новые данные, а не старые.

Лишние зависимости по данным появляются в результате "записи после чтения" (WAR) и "записи после записи" (WAW). Зависимость WAR состоит в том, что команда должна записать новое значение в ячейку памяти или регистр, из которых должно быть произведено чтение. Лишние зависимости появляются по нескольким причинам: не оптимизированный программный код, ограничение количества регистров, стремление к экономии памяти, наличие программных циклов. Важно отметить, что запись может быть произведена в любой свободный ресурс, а не только тот, который указан в программе.

После удаления лишних зависимостей по управлению и данным команды могут исполняться параллельно. Формирование расписания параллельного выполнения команд возлагается на аппаратные средства микропроцессора. Это расписание учитывает существующие зависимости между командами и имеющиеся функциональные модули процессора.

В современных микропроцессорах широко используется принцип конвейерного выполнения отдельных элементарных операций. Конвейеризация внутренних процессов позволяет выполнять команду за каждый процессорный цикл.

Дальнейшее внедрение принципов конвейеризации привело к появлению класса суперскалярных микропроцессоров. Их отличительной особенностью является возможность выполнения нескольких команд за один процессорный цикл. Такой режим выполнения программы стал возможным благодаря наличию в процессорах нескольких исполнительных устройств.

В число основных блоков суперскалярного микропроцессора входят блок выборки команд и предсказания переходов, блок декодирования команд, анализа зависимостей между командами, переименования и диспетчеризации, блоки регистров и обрабатывающих устройств с плавающей и фиксированной точками, блок управления памятью, а также блок упорядочения выполненных команд.

Ниже рассмотрены основные приемы повышения быстродействия в суперскалярных микропроцессорах.

Поскольку при суперскалярной обработке необходимо извлекать из памяти несколько команд за один такт для загрузки параллельно работающих функциональных модулей, повышенные требования предъявляются к пропускной способности интерфейса микропроцессор-память. В современных микропроцессорах применяются многоуровневые раздельные кэш-памяти данных и команд.

Для уменьшения потерь процессорных циклов, связанных с промахами при обращении к кэш-памяти в случае выполнения команд ветвления, в состав системы кэширования введены средства предсказания переходов, основное назначение которых - повысить вероятность наличия в кэш-памяти требуемой команды.

Исполнение условных ветвлений состоит из следующих этапов:

• распознание команды условного ветвления;

• проверка выполнения условия перехода;

• вычисление адреса перехода;

• передача управления, в случае перехода. На каждом этапе используются специальные приемы повышения производительности.

1. Для быстрого декодирования используются либо дополнительные биты в поле команды, либо преддекодирование команд при выборе из кэш-памяти команд.

2. Часто, когда команда уже выбрана в кэш, условие перехода еще не вычислено. Чтобы не задерживать поток команд в данном случае используется предсказание перехода по одной из нескольких возможных схем. Некоторые предсказатели используют статическую информацию из двоичного кода программы или специально выработанную компилятором. Например, определенные коды операций чаще вырабатывают ветвление, чем другие коды, или ветвление более вероятно (при организации циклов), или компилятор может устанавливать флаг, указывающий направления перехода. Может также использоваться статистическая информация, полученная при трассировке программы.

Другие предсказатели используют динамически формируемую информацию в процессе исполнения программы. Обычно это информация, касающаяся истории выполнения данного ветвления, сохраняемая в таблице ветвлений или в таблице предсказаний ветвлений. Таблица предсказания ветвлений организуется по ассоциативному принципу, подобно кэш-памяти, ее элементы доступны по адресу команды, ветвление которой предсказывается. В некоторых реализациях элемент таблицы предсказания ветвления является счетчиком, значение которого увеличивается при правильном предсказании и уменьшается при неправильном. При этом значение счетчика определяет преобладающее направление ветвлений.

В момент определения действительного значения условия ветвления, вносится изменение в историю ветвления. Если предсказание было неверным, то должна инициироваться выборка правильных команд. Результаты команд, которые были условно выполнены, должны быть аннулированы.

3. Для определения адреса ветвления обычно требуется выполнить целочисленное сложение, прибавляющее к текущему значению счетчика команд смещение, заданное в поле команды ветвления. И хотя это не требует дополнительных циклов для обращения к регистрам, ускорение вычисления адреса может быть достигнуто благодаря использованию буфера, содержащего ранее использованные адреса переходов.

Если требуется осуществить смену значения счетчика команд, то необходим, по крайней мере, один такт для распознания команды ветвления, модификации счетчика команд и выборки команды по заданному значению счетчика команд. Эти задержки вызывают пустые такты в конвейерах процессора. Более сложные решения используют буферы, содержащие наборы команд для двух возможных результатов ветвлений.

Возможно также использование "отложенных переходов", когда одна или несколько команд после команды ветвления выполняются безусловно.

Декодирование команд, переименование ресурсов и диспетчеризация

На этой фазе определяются существенные зависимости (RAW) по данным между командами и преодолеваются несущественные (WAW, WAR), производится распределение команд по буферам команд функциональных устройств.

При декодировании команды создается одна или несколько упорядоченных троек, каждая из которых включает: 1) исполняемую операцию, 2) указатели на операнды, 3) указатель на место помещения результата.

Для преодоления лишних **WAR и WAW** зависимостей, возникающих в результате ограниченности логических ресурсов (ячеек памяти, регистров), используется механизм динамического отображения определяемых текстом программы логических ресурсов на физические ресурсы микропроцессора. При данном подходе с одним логическим ресурсом может быть связано несколько значений в различных физических ресурсах, каждое из которых соответствует значению логической величины в один из моментов времени последовательного выполнения программы.

Когда команда создает новое значение для логического регистра, физический ресурс, в который помещается это значение, получает имя. Последующие команды, использующие это значение, снабжаются именем физического ресурса. Данная процедура называется переименованием регистров. Используются два основных способа переименования.

В первом, физический файл регистров больше логического. При необходимости переименования из списка свободных физических регистров берется один и ему сопоставляется соответствующее логическое имя. Если список свободных регистров пуст, диспетчеризация команд приостанавливается до момента появления свободных физических регистров.

Рассмотрим пример реализации данного способа переименования. Пусть требуется выполнить команду sub r3, r3, 5 (из значения регистра r3 вычесть константу 5 и поместить результат в регистр r3). Логические имена регистров начинаются со строчной буквы, а физические — с прописной. Пусть также в момент исполнения команды в таблице регистру r3 соответствует R1. Первым регистром в списке свободных пусть является R2. Поэтому в поле результата команды sub r3, r3, 5 регистр r3 заменяется на R2. Исполнимая команда приобретает вид sub R2, R1, 5. Любая следующая за sub команда, использующая ее результат, должна использовать в качестве операнда R2.

Остается вопрос о возвращении физических регистров в список свободных после того, как из них считаны данные в последний раз. Один из способов связывает счетчик с каждым физическим регистром. Счетчик увеличивается при каждом переименовании операнда в командах, использующих этот физический регистр. Соответственно при использовании операнда значения счетчика уменьшается на 1. При достижении счетчиком нуля физический ресурс должен быть переведен в список свободных.

Второй способ переименования использует одинаковое число логических и физических регистров и поддерживает их однозначное соответствие. В дополнение имеется буфер с одним вхождением для каждой инициированной на исполнение команды. Этот буфер называется переупорядочивающим, так как он используется также для установления порядка команд при прерываниях. Данный буфер можно рассматривать как FIFO очередь, выполненную в виде кольцевого буфера с указателями "начало" и "конец".

Команды помещаются в конец буфера. По завершению команды ее результат заносится в заранее предписанный ей элемент очереди, независимо от места в очереди, занимаемого этим элементом. К моменту достижения командой начала буфера, если она была исполнена, ее результат помещается в регистровый файл, а сама команда удаляется. Команда, находящаяся в буфере и не исполненная в виду отсутствия значения операнда, остается в нем вплоть до получения этого значения. Одновременно может выбираться из очереди или помещаться в нее несколько команд, однако всегда соблюдается дисциплина FIFO.

Значение логического регистра может быть размещено либо в физическом регистре, либо в переупорядочивающем буфере. В момент декодирования команды значению ее результата

сопоставляется соответствующая результату позиция упорядоченной тройки команды в элементе переупорядочивающего буфера, в котором размещается рассматриваемая декодированная команда, и делается отметка в таблице соответствия значений, которая указывает, что значение результата может быть найдено в соответствующем элементе буфера. Поля источников и результата команды используются для доступа к полям таблицы. Таблица показывает, что соответствующий регистр содержит требуемую величину либо она может быть найдена в переупорядочивающем буфере. Когда переупорядочивающий буфер полон, диспетчеризация команд приостанавливается.

Рассмотрим выполнение переименования на примере команды sub r3,r3,5. Пусть значение г3 находится или будет находиться в переупорядочивающем буфере в элементе 6. Регистр г3 как источник заменяется на соответствующее поле результата элемента 6 буфера. Команда sub помещается в конец переупорядочивающего буфера, например, в элемент 7. Этот номер затем записывается в таблицу для использования командами-потребителями результата. Следует заметить, что переупорядочивающий буфер фактически вводит потоковую модель вычислений по готовности операндов.

Независимо от способа переименования в суперскалярном процессоре устраняются лишние зависимости по данным.

Исполнение команд

После формирования для каждой команды упорядоченных троек, состоящих из кода операции, физических операндов - источника и результата, и размещения их в буферах, наступает фаза динамической проверки готовности значений операндов для исполнения команды.

В идеале команда готова к исполнению, как только готовы ее входные операнды. Однако есть ряд ограничений, связанных с доступностью физических ресурсов, таких как исполнительные устройства, коммутаторы и порты регистровых файлов (или переупорядочивающего буфера). Для организации окна исполнения используются различные методы: одной очереди, многих очередей или метод резервирующей станции.

Если имеется одна очередь, то переименование регистров не требуется, так как доступность значений операндов может отмечаться битом резервирования, сопоставленным каждому регистру. Регистр резервируется, когда модифицирующая его команда назначается на исполнение. И регистр освобождается, когда заканчивается исполнение команды. Если для команды ресурсы не были зарезервированы, то она приостанавливает свое исполнение.

В методе многих очередей каждая очередь организуется для команд одного типа. Например, очередь команд с плавающей точкой или очередь команд работы с памятью.

Третий метод предполагает использование резервирующей станции, состоящей из совокупности элементов, каждый из которых содержит позиции для размещения кода операции, наименования первого операнда, самого первого операнда, признака доступности первого операнда, наименования второго операнда, самого второго операнда, признака доступности второго операнда и наименования регистра результата. Когда команда завершает исполнение и вырабатывает результат, то наименование результата сравнивается с наименованиями операндов в резервирующей станции.

Если в резервирующей станции обнаруживается команда, ждущая этого результата, то данные записываются в соответствующую позицию и устанавливается признак их доступности. Когда у команды доступны все операнды, инициируется ее исполнение. Резервирующая станция следит за доступностью операндов. Когда команда при диспетчеризации попадает в резервирующую станцию, все готовые операнды из регистрового файла переписываются в поля этой команды. Когда все операнды готовы, команда исполняется. Иногда резервирующая станция содержит не сами операнды, а указатели на них в регистровом файле или переупорядочивающем буфере.

Работа с памятью

Для вычисления адреса памяти, как правило, требуется, по крайней мере, одно сложение. После вычисления адреса может понадобиться его преобразование в физический адрес, осуществляемое буфером трансляции адресов (TLB).

Проблемы конфликтов при доступе к разделяемому ресурсу - ячейкам памяти, по сути те же, что и при доступе к регистрам.

Завершение выполнения команды

Завершающей фазой исполнения команды является фаза изменения состояния процессора в соответствии с выполненной командой. Назначение этой фазы - сохранение последовательной модели исполнения программы, при реальном параллельном выполнении отдельных команд и условном выполнении команд ветвления. Для изменения состояния процессора применяются два основных способа, причем оба основаны на использовании двух состояний: состояния, измененного в результате операции, и состояния, требуемого для восстановления.

При первом способе сохраняется состояние процессора в наборе контрольных точек или в буфере истории вычислений, которые, в случае необходимости, используются для восстановления состояния.

Второй способ предполагает рассмотрение логического (архитектурного) и физического состояния процессора. Физическое состояние изменяется немедленно по завершении очередной команды. Архитектурное состояние изменяется тогда, когда ясен результат условно выполненных команд. Для реализации этого способа используется переупорядочивающий буфер: результаты из буфера отправляются в файл архитектурных регистров и память.

В переупорядочивающем буфере для каждой команды содержится соответствующее ей значение счетчика команд и значения других регистров, которые необходимы для корректного обслуживания прерываний.

Основные компоненты суперскалярного микропроцессора: функциональные модули - выполнения операций с плавающей (FPU) и фиксированной (ALU) точкой, устройство загрузки/сохранения, файлы регистров, раздельная кэш-память команд и данных, а также вспомогательные модули, обеспечивающие динамическое планирование вычислительного процесса - устройство связи с кэш-памятью 2-го уровня, блок переупорядочивания команд и блок предварительной дешифрации.

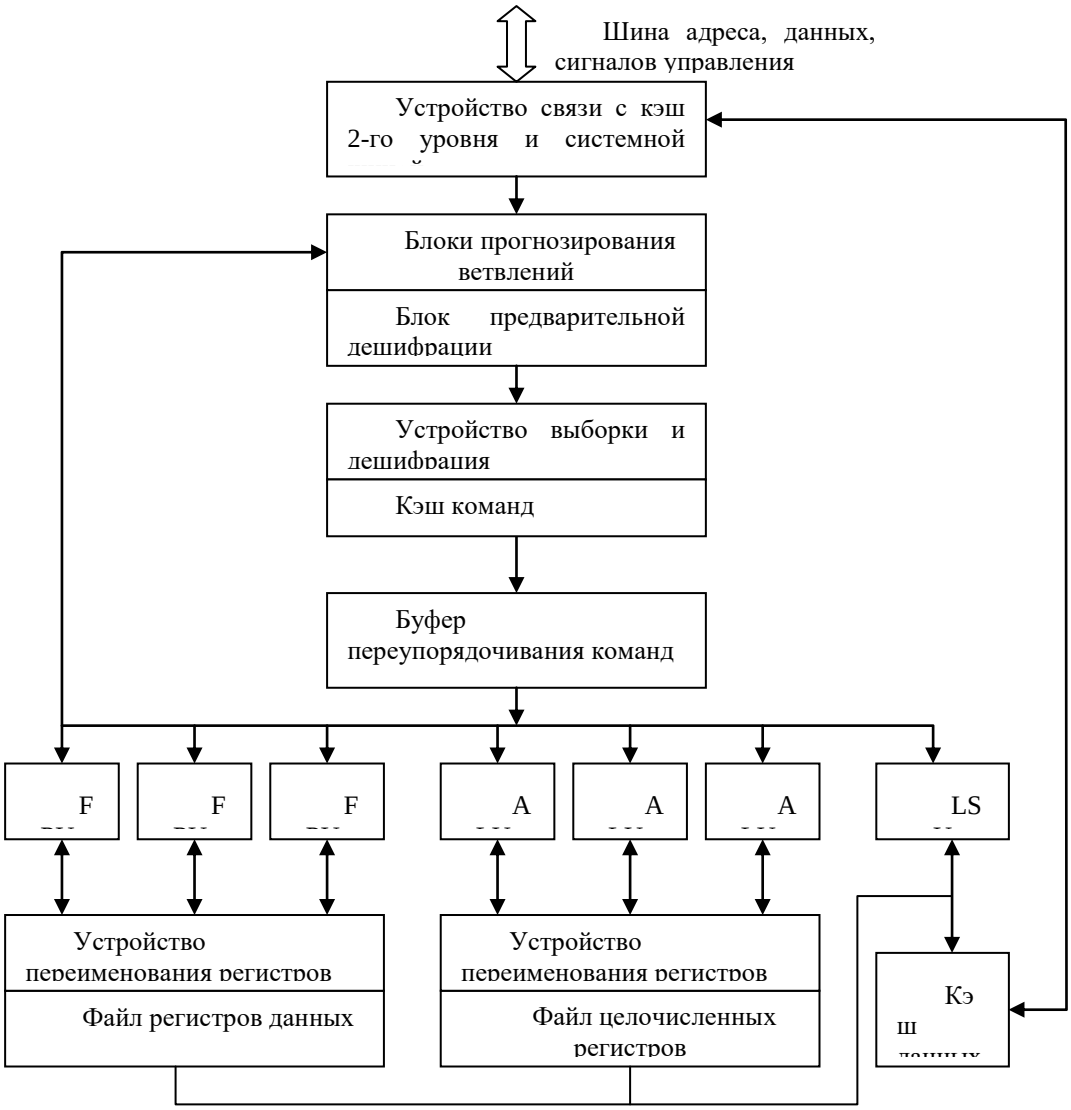
Направления развития суперскалярной архитектуры

Как уже отмечалось ранее, в суперскалярных процессорах предпринимается попытка в рамках модели последовательных программ реализовать параллельное исполнение команд этих программ. После извлечения последовательного потока команд между командами устанавливаются только действительно необходимые зависимости по данным. При этом сохраняется достаточно информации о порядке следования команд в исходной программе, чтобы сохранить их порядок при наступлении прерывания.

Типичный суперскалярный процессор выбирает команды и исследует их по мере выполнения. Исследование проводится с целью выявления и обработки команд перехода, идентификации типа команды для ее дальнейшего направления на соответствующий исполнительный блок или в буфер памяти. Выполняются также некоторые действия для смягчения зависимостей по данным, например переименование регистров. VLIW процессор возлагает на компилятор статическую реализацию тех функций, которые в суперскалярном процессоре выполняются динамически.

По крайней мере два обстоятельства ограничивают эффективность использования суперскалярных архитектур. Во-первых, есть ограничения на степень параллелизма на уровне команд, даже если применяется самая совершенная техника суперскалярных вычислений. Первое ограничение проистекает из условных переходов. Другое следует из того, что размер окна исполнения (число активных команд, могущих исполняться параллельно) ограничивает возможный присущий программе параллелизм, так как не рассматривается параллельное

исполнение команд, находящихся на расстоянии, превышающем размер окна.



Структура суперскалярного микропроцессора

Во-вторых, сложность суперскалярного процессора возрастает как количество параллельно исполняемых команд и даже быстрее.

Вероятнее всего, что пределом распараллеливания при суперскалярной обработке является запуск одновременно на исполнение в каждом такте 7-8 команд.

Альтернатива суперскалярной обработке - длинное командное слово (VLIW). Использование этого метода предполагает задание в командном слове совокупности параллельно выполняемых команд. Подготовкой таких программ занимается компилятор.

Достоинства VLIW заключаются в следующем. Во-первых, компилятор может более эффектно исследовать зависимости между командами и выбирать параллельно исполняемые команды, чем это делает аппаратура суперскалярного процессора, ограниченная размером окна исполнения.

Во-вторых, VLIW процессор имеет более простое устройство управления и потенциально может иметь более высокую тактовую частоту.

Однако у VLIW процессоров есть серьезный фактор, снижающий их производительность. Это команды ветвления, зависящие от данных, значения которых становятся известны только в динамике вычислений. Окно исполнения VLIW-процессора, не может быть очень большим в виду отсутствия у компилятора информации о зависимостях, формируемых динамически, в процессе выполнения. Этот недостаток препятствует возможности переупорядочивания операций в VLIW процессоре. Например, статически не может быть гарантировано правильное выполнение операции загрузки в вызываемой функции параллельно с операцией запоминания в вызывающей функции (особенно, если вызываемая функция определена динамически). Кроме того, VLIW реализация требует большого размера памяти имен, многовходовых регистровых файлов, большого числа перекрестных связей. Возможен также останов, когда во время выполнения возникла ситуация, отличающаяся от состояния в момент генерации плана выполнения (например, во время выполнения произошло неудачное обращение в кэш).

Другим возможным подходом служит переход к мультипроцессорному исполнению, когда вводится несколько счетчиков команд. В этом случае речь идет о распараллеливающих компиляторах с языков высокого уровня.

Таким образом, суперскалярные микропроцессоры являются лидирующим продуктом микроэлектроники, и их производительность постоянно растет, но при использовании этих процессоров необходимо тщательно исследовать архитектурные приемы получения высокой производительности и проверять адекватность этих приемов проблемной области, для решения задач которой создается вычислительная система.

Дальнейшее повышение производительности микропроцессоров связывается в настоящее время со статическим и динамическим анализом кода с целью выявления резервов параллелизма уровня отдельных команд и программных сегментов с использованием информации, предоставляемой компилятором языка высокого уровня Исследования в данном направлении привели к разработке мультискалярной архитектуры процессоров, которые являются дальнейшим развитием суперскалярной архитектуры

В настоящее время работы в данном направлении находятся на стадии теоретического исследования и имитационного моделирования, однако, по видимому, уже в скором времени следует ожидать появления первых микропроцессоров, в полной мере использующих все преимущества, предоставляемые мультискалярной архитектурой. Поэтому основные моменты, связанные с данной архитектурой, будут рассмотрены ниже достаточно подробно.

Лекция 16

Принципы организации системы прерывания программ

Во время выполнения ЭВМ текущей программы внутри машины и в связанной с ней внешней среде (например, в технологическом процессе, управляемом ЭВМ) могут возникать события, требующие немедленной реакции на них со стороны машины.

Реакция состоит в том, что машина прерывает обработку текущей программы и переходит к выполнению некоторой другой программы, специально предназначенной для данного события. По завершении этой программы ЭВМ возвращается к выполнению прерванной программы.

Рассматриваемый процесс, называемый *прерыванием программ,* поясняется на рис. 9.23. Принципиально важным является то, что моменты возникновения событий, требующих прерывания программ, заранее неизвестны и поэтому не могут быть учтены при программировании.

Каждое событие, требующее прерывания, сопровождается сигналом, оповещающим ЭВМ. Назовем эти сигналы *запросами прерывания.* Программу, затребованную запросом прерывания, назовем *прерывающей программой,* противопоставляя ее *прерываемой программе,* выполнявшейся машиной до появления запроса.

Запросы на прерывания могут возникать внутри самой ЭВМ и в ее внешней среде. К первым относятся, например, запросы при возникновении в ЭВМ таких событий, как появление ошибки в работе ее аппаратуры, переполнение разрядной сетки, попытка деления на 0, выход из установленной для данной программы области памяти, затребование периферийным устройством операции ввода-вывода, завершение операции ввода-вывода периферийным устройством или возникновение при этой операции особой ситуации и др.. Хотя некоторые из указанных событий порождаются самой программой, моменты их появления, как правило, невозможно предусмотреть. Запросы во внешней среде могут возникать от других ЭВМ, от аварийных и некоторых других датчиков технологического процесса и т. п.

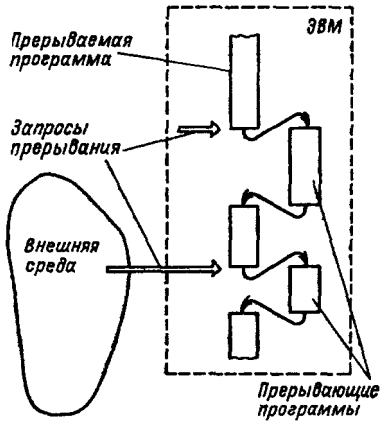


Рис. 9.23. Прерывание программы

В сущности, запросы прерывания генерируются несколькими развивающимися параллельно во времени процессами, которые в некоторые моменты требуют вмешательства процессора. К этим процессам, в частности, относятся процесс выполнения самой программы, процесс контроля правильности работы ЭВМ, операции ввода-вывода, технологический процесс в управляемом машиной объекте и др.

Возможность прерывания программ - важное архитектурное свойство ЭВМ, позволяющее эффективно использовать производительность процессора при наличии нескольких протекающих параллельно во времени процессов, требующих в произвольные моменты времени управления и обслуживания со стороны процессора. В первую очередь это относится к организации параллельной во времени работы процессора и периферийных устройств машины, а также к использованию ЭВМ для управления в реальном времени технологическими процессами.

В некоторых машинах наряду или вместо прерывания с переключением управления на другую программу используется примитивное прерывание - так называемая приостановка, когда по соответствующему запросу приостанавливается выполнение программы и выполняется аппаратурными средствами некоторая процедура без изменения содержания счетчика команд, а по ее окончании продолжается выполнение приостановленной программы.

Чтобы ЭВМ могла, не требуя больших усилии от программиста, реализовывать с высоким быстродействием прерывания программ, машине необходимо придать соответствующие аппаратурные и программные средства, совокупность которых получила название *системы прерывания программ* или *контроллера прерывания.*

Основными функциями системы прерывания являются:

- запоминание состояния прерываемой программы и осуществление перехода к прерывающей программе,
- восстановление состояния прерванной программы и возврат к ней.

При наличии нескольких источников запросов прерывания должен быть установлен определенный порядок (дисциплина) в обслуживании поступающих запросов. Другими словами, между запросами (и соответствующими прерывающими программами) должны быть установлены *приоритетные соотношения,* определяющие, какой из нескольких поступивших запросов подлежит обработке в первую ередь, и устанавливающие, имеет право и не имеет данный запрос (прерывающая ограмма) прерывать ту или иную ограмму. Приоритетный выбор запроса я исполнения входит в процедуру перехода

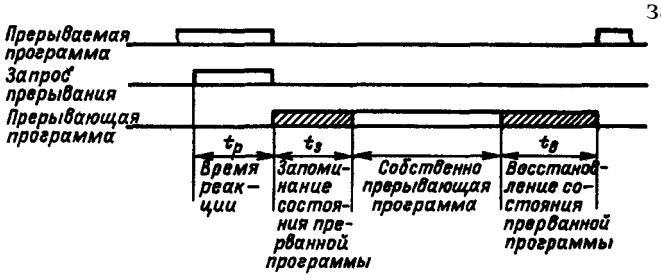


Рис. 924. Упрощенная временная диаграмма процесса прерывания

к прерывающей программе.

Характеристики системы прерывания. Для оценки эффективности систем прерывания могут быть использованы следующие характеристики.

Общее число запросов прерывания (входов в систему прерывания).

Время реакции - время между появлением запроса прерывания и началом выполнения прерывающей программы

На рис. 9.24 приведена упрощенная временная диаграмма процесса прерывания в предположении, что управление запоминанием состояния и возвратом возложено на саму прерывающую программу, которая в этом случае состоит из трех частей: подготовительной и заключительной, обеспечивающих переключение программ, и собственно прерывающей программы, выполняющей затребованную запросом работу.

Для одного и того же запроса задержки в исполнении прерывающей программы зависят от того, сколько программ со старшим приоритетом ждут обслуживания. Поэтому время реакции определяют для запроса с наивысшим приоритетом.

Время реакции зависит от того, в какой момент допустимо прерывание. Большей частью прерывание допускается после окончания текущей команды. В этом случае время реакции определяется в основном длительностью выполнения команды.

Это время реакции может оказаться недопустимо большим для ЭВМ, предназначенных для работы в реальном масштабе времени. В таких машинах часто допускается прерывание после любого такта выполнения команды. Однако при этом возрастает количество информации, подлежащей запоминанию и восстановлению при переключении программ, так как в этом случае необходимо сохранять также и состояния в момент прерывания счетчика тактов, регистра кода операции и некоторых других. Поэтому такая организация прерывания возможна только в машинах с быстродействующей сверхоперативной памятью.

Имеются ситуации, в которых желательно немедленное прерывание. Если аппаратура контроля обнаружила ошибку, то целесообразно сразу же прервать операцию, пока ошибка не оказала влияния на следующие такты работы машины.

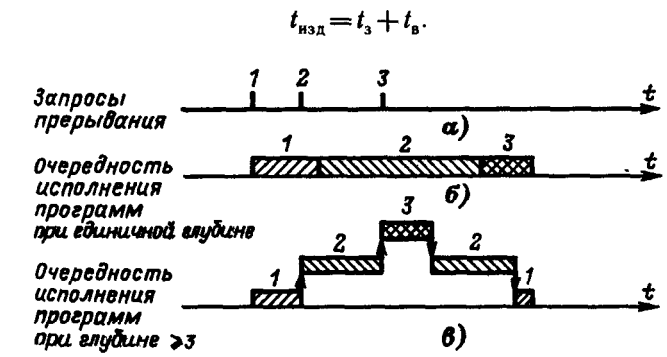


Рис 9 25 Прерывание в системах с различной глубиной прерывания

Затраты времени на переключение программ (издержки прерывания) равны суммарному расходу времени на запоминание и восстановление состояния программы:

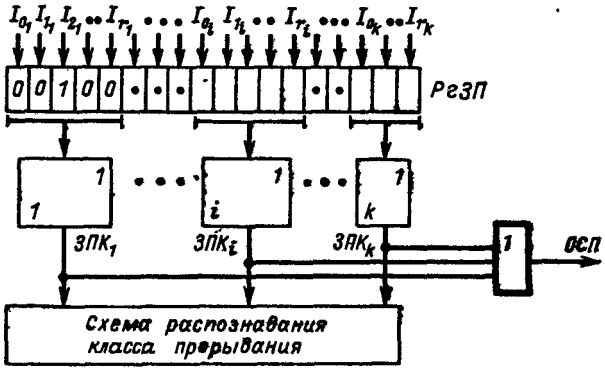
Глубина прерывания — максимальное число программ, которые могут прерывать друг друга. Если после перехода к прерывающей программе и вплоть до ее окончания прием других запросов запрещается, то говорят, что система имеет глубину прерывания, равную 1. Глубина равна *n*, если допускается последовательное прерывание до *n* программ. Глубина прерывания обычно совпадает с числом уровней приоритета в системе прерывании. На Рис. 9.25, *а—в* показано прерывание в системах с различной глубиной прерывания (предполагается, что приоритет каждого следующего запроса выше предыдущего). Системы с большим значением глубины прерывания обеспечивают более быструю реакцию на срочные запросы.

Если запрос окажется необслуженным к моменту прихода нового запроса от того же источника, то возникнет так называемое *насыщение системы прерывания.* В этом случае предыдущий запрос прерывания от данного источника будет машиной утрачен, что недопустимо. Быстродействие ЭВМ, характеристики системы прерывания, число источников прерывания и частота возникновения запросов должны быть согласованы таким образом, чтобы насыщение было невозможным.

Количество уровней прерывания. В ЭВМ число различных запросов прерывания может

достигать нескольких десятков или сотен. В таких случаях часто запросы разделяют на отдельные классы или уровни.

Совокупность запросов, инициирующих одну и ту же прерывающую программу, образует *класс или уровень прерывания* (рис 926).



Запросы всех источников прерывания поступают на регистр запросов прерывания *РгЗП,* устанавливая соответствующие его разряды (флажки) в состояние 1, указывающее на наличие запроса прерывания определенного источника. Запросы классов прерывания $ЗПК_1$ — $ЗПК_k$ формируются элементами ИЛИ, объединяющими разряды *РгЗП,* относящиеся к соответствующим классам (уровням). Еще одна схема ИЛИ формирует общий сигнал прерывания *ОСП,* поступающий в устройство управления процессора. Значение сигнала *ОСП* определяется выражением

$$ОСП = \bigvee_{i=1}^{k} ЗПК_i = \bigvee_{i=1}^{k}\left(\bigvee_{j=0}^{r} I_j \right)$$

Информация о действительной причине прерывания, породившей запрос данного класса, содержится в коде прерывания, который отражает состояние разрядов *РгЗП,* относящихся к данному классу прерывания. После принятия запроса прерывания на исполнение и передачи управления прерывающей программе соответствующий триггер *РгЗП* сбрасывается. Объединение запросов в классы прерывания позволяет уменьшить объем аппаратуры, но связано с замедлением работы системы прерывания.

Организация перехода к. прерывающей программе. Приоритетное обслуживание запросов прерывания. Назовем *вектором прерывания* вектор начального состояния прерывающей программы. Вектор прерывания содержит всю необходимую информацию для перехода к прерывающей программе, в том числе ее начальный адрес. Каждому запросу (уровню) прерывания, а в ряде случаев, например в малых и микроЭВМ и микропроцессорах, каждому периферийному устройству соответствует свой вектор прерывания, способный инициировать выполнение соответствующей прерывающей программы. Векторы прерывания обычно находятся в специально выделенных фиксированных ячейках памяти.

Главное место в процедуре перехода к прерывающей программе занимают передача из соответствующего регистра (регистров) процессора в память (в частности, в стек) на сохранение текущего вектора состояния прерываемой программы (чтобы можно было вернуться к ее исполнению) и загрузка в регистр (регистры) процессора вектора прерывания прерывающей программы, к которой при этом переходит управление процессором.

Процедура организации перехода к прерывающей программе включает в себя выделение из выставленных запросов такого, который имеет наибольший приоритет.

Различают *абсолютный* и, *относительный приоритеты.* Запрос, имеющий абсолютный приоритет, прерывает выполняемую программу и инициирует выполнение соответствующей прерывающей программы. Запрос с относительным приоритетом является первым кандидатом на обслуживание после завершения выполнения текущей программы.

Если наиболее приоритетный из выставленных запросов прерывания не превосходит по уровню приоритета выполняемую процессором программу, то запрос прерывания игнорируется или его обслуживание откладывается до завершения выполнения текущей программы.

Простейший способ установления приоритетных соотношений между запросами (уровнями) прерывания состоит в том, что приоритет определяется порядком присоединения линий сигналов запросов ко входам системы прерывания. При появлении нескольких запросов прерывания первым воспринимается запрос, поступивший на вход с меньшим номером. В этом случае приоритет является жестко фиксированным. Изменить приоритетные соотношения можно лишь пересоединением линий сигналов запросов на входах системы прерывания.

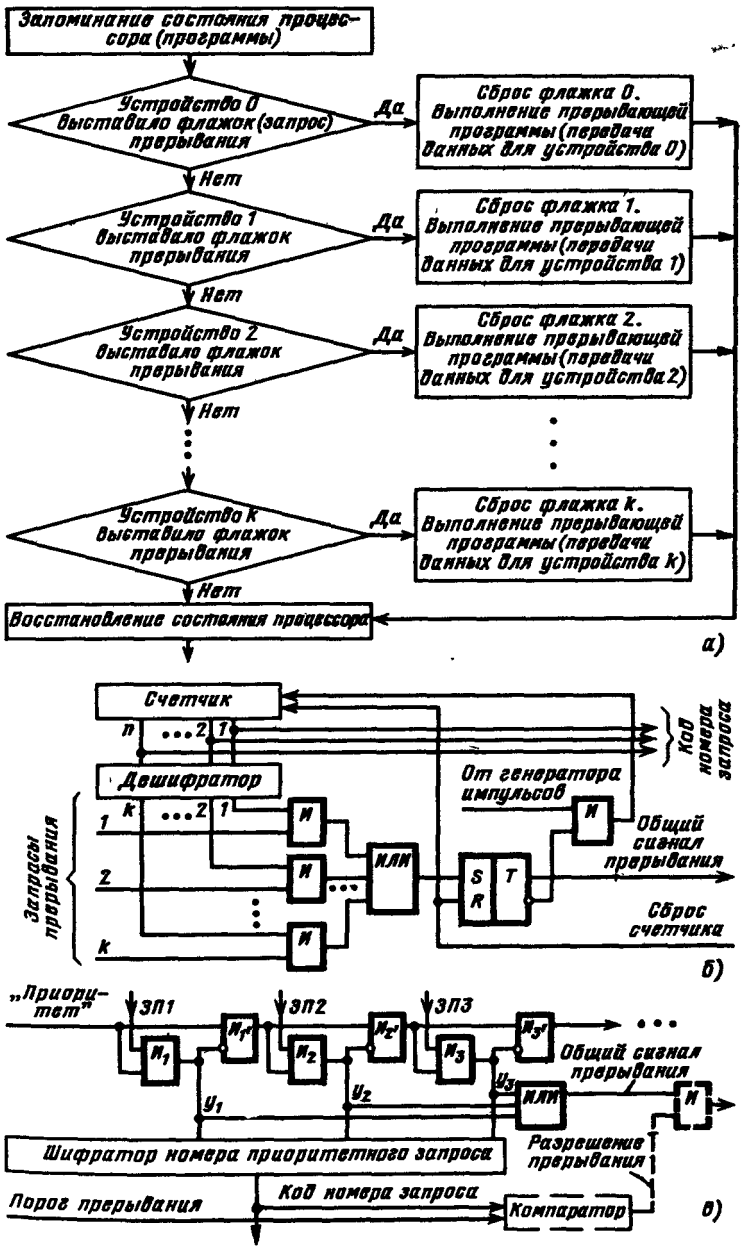


Рис 9.27. Способы опроса источников сигналов прерывания:

а -программный опрос, б -циклический (многотактный) опрос, в -цепочечный однотактный опрос («дейзи-цепочка»)

Процедура прерывания с опросом источников (флажков) прерывания. При указанном способе задания приоритета между запросами каждому источнику запросов соответствует разряд *(флажок)* в регистре запросов прерывания *(регистре флажков).*

При наличии запроса или нескольких запросов прерывания формируется общий сигнал прерывания (ОСП) (как это, например, показано на рис. 9.26), инициирующий выполняемую специальной программой или аппаратурой процедуру опроса регистра прерывания (флажков) или просто линий сигналов прерывания для установления источника, выставившего запрос прерывания наибольшего приоритета. По существу, эта процедура состоит в определении местоположения крайней слева единицы (крайнего флажка) в регистре запросов прерывания.

На рис. 9.27 приведены различные способы реализации процедуры опроса источников сигналов прерывания. На рис 9.27, *а* показан процесс прерывания с программным опросом флажков прерывания (или, другими словами, опросом периферийных устройств, затребовавших передачу данных). Программный опрос источников прерываний занимает сравнительно много времени. Для уменьшения этого времени процедуру опроса реализуют аппаратурным путем.

Схема циклического опроса запросов (источников) прерываний (рис. 9.27,6). Опрос *k* линий запросов прерывания (или разрядов регистра запросов прерывания) производится последовательно (циклически) с помощью *n*-разрядного счетчика ($2^n \geq k$), на который с некоторой частотой поступают импульсы от генератора. Поиск приоритетного запроса прерывания начинается со сброса счетчика и одновременно триггера *Т* в нулевое состояние, при этом импульсы генератора начинают поступать на вход счетчика. При помощи дешифратора и элементов *И в* каждом такте поиска проверяется наличие запроса прерывания, номер которого совпадает с кодом счетчика. Если на данном входе нет запроса прерывания, то после

прибавления 1 к счетчику проверяется следующий по порядку вход. Если имеется запрос, триггер *Т* перебрасывается в 1, при этом в процессор посылается общий сигнал прерывания *ОСП* и прекращается поступление импульсов на вход счетчика, т. е. завершается цикл просмотра входов системы прерывания. Содержимое счетчика — код номера старшего по приоритету выставленного запроса — используется для формирования начального адреса прерывающей программы. После передачи управления прерывающей программе счетчик (и триггер *Т)* сбрасывается в 0, и процедура опроса запросов возобновляется, начиная с первого входа.

Циклический (последовательный) опрос входов системы прерывания в аппаратурном отношении сравнительно прост, однако время реакции и при этом методе все-таки велико, особенно при большом числе источников запросов. Поэтому во многих случаях, например в ряде микропроцессоров, предназначенных для использования при работе в реальном времени, применяют схемы, позволяющие определять номер выставленного запроса или уровня прерывания старшего приоритета за один такт.

Цепочечная однотактная схема определения приоритетного запроса («дейзи-цепочка») представлена на рис. 9.27, *в*. Как и в предыдущих случаях, приоритет запросов прерывания возрастает с уменьшением их номера.

Процедура определения приоритетного запроса инициируется сигналом *Приоритет,* поступающим на цепочку последовательно включенных схем *И.* При отсутствии запросов этот сигнал пройдет через цепочку и сигнал общего запроса прерывания не сформируется. Если среди выставленных запросов прерывания наибольший приоритет имеет *i*-й запрос, то распространение сигнала *Приоритет* правее схемы *И с* номером *i* блокируется. На *i*-м выходе цепочечной схемы будет сигнал $y_i = 1$, на всех других 0. В процессор поступит общий сигнал прерывания, при этом шифратор по сигналу $y_i = 1$ сформирует код номера *i*-го запроса, принятого к обслуживанию. По сигналу процессора *Подтверждение прерывания* (на рис. 9.27 не показан) этот код передается в процессор и используется для формирования начального адреса прерывающей программы.

Схемы, представленные на рис. 9.27, *б* и *в,* производят поиск крайней левой единицы в наборе сигналов прерывания и формируют код номера *i* запроса, удовлетворяющего условию

$$ЗП_i\left(\bigwedge_{j=1}^{i-1}\overline{ЗП_j}\right)=1$$

Векторное прерывание

Представленные на рис. 9.27 способы определения запроса с наибольшим приоритетом включают в себя так или иначе выполняемую процедуру опроса источников прерывания (входов системы прерывания). Эта процедура, даже если она выполняется аппаратурными средствами, требует сравнительно больших временных затрат.

Более гибким и динамичным является *векторное прерывание,* при котором исключается опрос источников прерывания (флажков регистра прерывания).

Прерывание называется векторным, если источник прерывания, выставляя запрос прерывания, посылает в процессор (выставляет на шины интерфейса) код адреса в памяти своего вектора прерывания.

Отметим, что если прерывание на основе опроса источников прерываний всегда сопровождается переходом по одному и тому же адресу и инициирует одну и ту же прерывающую подпрограмму, которая после идентификации источника запроса и формирования адреса начала соответствующей запросу прерывающей программы передает ей управление, то при векторном прерывании каждому запросу прерывания, или, другими словами, устройству — источнику прерывания, соответствует переход к начальному адресу соответствующей прерывающей программы, задаваемому вектором прерывания.

Программно-управляемый приоритет прерывающих программ

Относительная степень важности программ, их частота повторения, относительная степень срочности в ходе вычислительного процесса могут меняться, требуя установления новых прио-

ритетных отношений. Поэтому во многих случаях приоритет между прерывающими программами не может быть зафиксирован раз и навсегда. Необходимо иметь возможность изменять по мере надобности приоритетные соотношения программным путем, другими словами, приоритет между прерывающими программами должен быть динамичным, т. е. программно-управляемым.

В ЭВМ широко применяются два способа реализации программно-управляемого приоритета прерывающих программ, в которых используются соответственно *порог прерывания* и *маски прерывания*.

Порог прерывания. Этот способ позволяет в ходе вычислительного процесса программным путем изменять уровень приоритета процессора (а следовательно, и обрабатываемой в данный момент на процессоре программы) относительно приоритетов запросов источников прерывания (в основном периферийных устройств), другими словами, задавать порог прерывания, т. е. минимальный уровень приоритета запросов, которым разрешается прерывать программу, идущую на процессоре.

Порог прерывания задается командой программы, устанавливающей в регистре порога прерывания *код порога прерывания.* Специальная схема выделяет наиболее приоритетный запрос прерывания, сравнивает его приоритет с порогом прерывания и, если он оказывается выше порога, вырабатывает общий сигнал прерывания, и начинается процедура прерывания (рис 9.27,в).

В современных ЭВМ общего назначения наибольшее распространение получило программное управление приоритетом на основе маски прерывания (рис. 9.28).

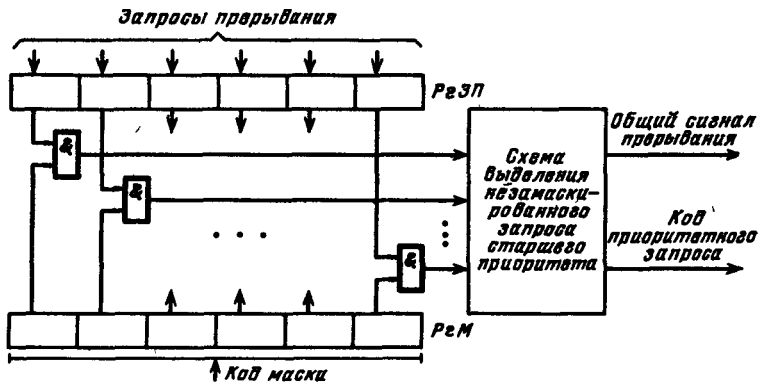


Рис 9.28. Программно управляемый приоритет на основе маски прерывания

Маска прерывания представляет обой двоичный код, разряды оторого поставлены в соответствие апросам или классам прерывания. Маска загружается командой рограммы в регистр маски. остояние 1 в данном разряде егистра маски разрешает, а состояние запрещает *(маскирует)* прерывание екущей программы от оответствующего запроса. Таким образом, программа, изменяя маску в регистре маски, может устанавливать произвольные приоритетные соотношения между программами без перекоммутации линий, по которым поступают запросы прерывания. Каждая прерывающая программа может установить свою маску. При формировании маски 1 устанавливаются в разряды, соответствующие запросам (прерывающим программам) с более высоким, чем у данной программы, приоритетом.

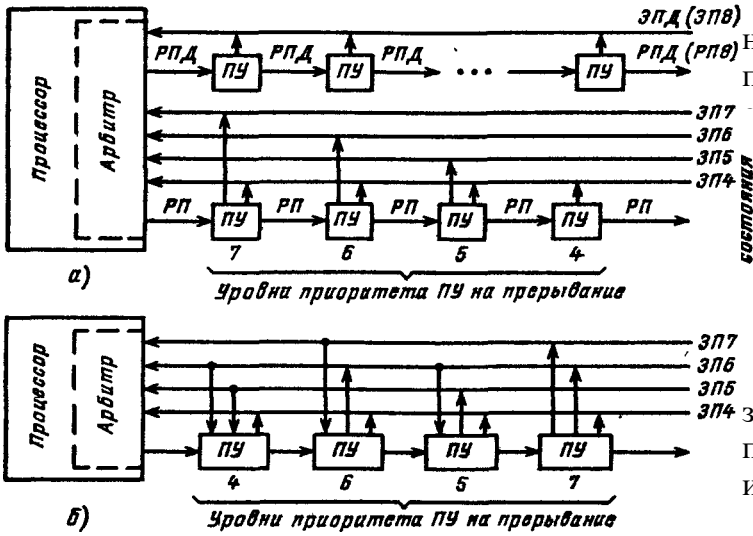


Рис. 9.30. Схема цепей запросов и разрешений прерываний в малых и микроЭВМ с интерфейсом «Q-шина»: а — с позиционно-зависимым приоритетом; б - с позиционно-независимым приоритетом; *ЗПi РПi* - соответственно запрос и разрешение

Схемы И выделяют поступившие незамаскированные запросы прерывания, из которых специальная хема, аналогичная цепочечной схеме а' рис. 9.28, *в,* выделяет наиболее риоритетный и формирует код его омера *i*, удовлетворяющего условию

$$\left(\bigwedge_{j=1}^{i-1}\overline{ЗП_j РгМ_j}\right) ЗП_i РгМ_i = 1$$

С замаскированным запросом в зависимости от причины прерывания поступают двояким образом: или он игнорируется, или запоминается, с тем

чтобы осуществить затребованные действия, когда запрет будет снят. Например, если прерывание вызвано окончанием операции в периферийном устройстве, то его следует, как правило, запомнить, так как иначе ЭВМ останется неосведомленной о том, что периферийное устройство освободилось. Прерывание, вызванное переполнением разрядной сетки при арифметической операции, следует при его маскировании игнорировать, так как запоминание этого запроса может привести к тому, что он окажет действие на часть программы или другую программу, к которым это переполнение не относится.

9.17. Особенности систем прерывания малых ЭВМ

Во многих малых ЭВМ, микропроцессорах и построенных на микропроцессорах микроЭВМ реализованы многоуровневые векторные системы прерывания с порогом прерывания и с использованием стековой памяти в процедурах перехода к прерывающей программе и возврата к прерванной программе.

Векторная система прерывания в малых машинах СМ ЭВМ

Рассмотрим особенности системы прерывания малых и микроЭВМ, в которых используется интерфейс «Q-шина».

Запросы прерываний. Запросы внешних прерываний генерируются периферийными устройствами, подсоединенными к интерфейсу «Q-шина». На рис. 9.30 представлены варианты схем присоединения периферийных устройств и процессора (схемы арбитража) к линиям сигналов запросов и разрешения прерывания и прямого доступа к памяти. Имеется четыре уровня приоритета запросов прерывания — с четвертого по седьмой (в порядке возрастания). Еще более высокий (восьмой) уровень приоритета имеют запросы прямого доступа к памяти. Каждый уровень обслуживает своя линия запросов прерывания $ЗП_i$,, к которой параллельно (по схеме ИЛИ) подсоединяются ПУ соответствующего уровня приоритета. Имеется одна линия для выдаваемого арбитром сигнала разрешения прерывания РП, проходящая последовательно через все ПУ с приоритетом от четвертого до седьмого. Кроме того, имеется отдельная линия для сигнала разрешения прямого доступа к памяти РПД, также проходящая последовательно через все ПУ, подключенные к линии запросов прямого доступа ЗПД.

При наличии одной линии разрешения прерывания для выделения устройства, которому разрешается прерывание, используется цепочечный метод, при этом возможны два варианта схем прерывания с позиционно зависимым (рис 9 30, *а)* и с позиционно независимым приоритетом (рис 9.30,6). В позиционно-зависимой схеме устройства подсоединяются к процессору, точнее, к линии РП в порядке убывания приоритета. Если это неудобно, может применяться позиционно-независимая схема, в которой благодаря дополнительным связям при появлении *ЗП* на линиях более высокого приоритета выставившие *ЗП* устройства меньшего приоритета игнорируют сигнал разрешения прерывания и пропускают его на соседние устройства. Во второй схеме позиционность сохраняется только в отношении устройств, имеющих одинаковый приоритет. Из них преимущественное право на прерывание имеет устройство, расположенное электрически ближе к процессору.

Схема *Арбитр* из выставленных запросов выделяет запрос старшего уровня приоритета и сравнивает его уровень с приоритетом процессора, т.е. с программно- устанавливаемым в регистре слова состояния процессора *порогом прерывания* (может принимать значения 4—7). Если уровень наиболее приоритетного из выставленных запросов прерывания превышает порог прерывания, арбитр (процессор) после завершения выполнения текущей команды выдает сигнал разрешения прерывания на линию *РП.* Этот сигнал поступает в первое по пути его прохождения выставившее запрос (и не заблокированное в схеме рис 9 30, б) устройство, которое прекращает дальнейшее распространение сигнала *РП.*

Устройство, пославшее $ЗП_i$ и получившее разрешение на прерывание, передает в процессор адрес соответствующего вектора прерывания. Процессор, получив адрес вектора прерывания, помещает в стек, т. е. в ячейки памяти, адресуемые указателем стека, два слова вектора состояния: сначала текущее слово состояния процессора (второе слово вектора состояния), затем первое слово — содержимое счетчика команд (продвинутый адрес прерванной программы). Перед каждой передачей в стек значение указателя стека уменьшается на два.

Далее в счетчик команд из ячейки, хранящей первое слово вектора прерывания, передается начальный адрес прерывающей программы, а из следующей ячейки второе слово вектора прерывания заносится в регистр слова состояния процессора. В новом слове состояния процессора порог прерывания должен быть не меньше уровня приоритета принятого к обслуживанию запроса, чтобы повторный запрос от этого источника прерывания не мог прервать выполняемую прерывающую программу. Управление переходит к программе обработки прерывания, заданной вектором прерывания. Если эта программа использует общие регистры, то она начинается с передачи их содержимого в стековую память с помощью команд передачи с автодекрементной прямой адресации по регистру указателя стека.

Возврат к прерванной программе осуществляет заключительная часть прерывающей программы, в которой команды передачи данных с автоинкрементной прямой адресацией по указателю стека производят передачу из стека сохраненных в нем состояний общих регистров в соответствующие регистры. Последней командой прерывающей программы — командой «Возврат из прерываний»—.первое слово вектора состояния прерванной программы загружается из стека в счетчик команд, а второе слово — в регистр слова состояния процессора. Передача каждого слова сопровождается увеличением УС на два. После этого восстанавливается выполнение прерванной программы.

Имеются особенности в процедуре выполнения запросов прерываний *ЗП8* (запросов прямого доступа к памяти). Их приоритет всегда выше приоритета процессора. Поэтому в ответ на запрос *ЗПД (ЗП8)* сигнал разрешения *РПД* посылается немедленно, даже если не завершено выполнение текущей команды, и производится обмен данными между периферийным устройством и *ОП* без участия процессора.

9.19. Процедура выполнения команд. Рабочий цикл процессора

Функционирование процессора в основном состоит из повторяющихся *рабочих циклов,* каждый из которых соответствует выполнению одной команды программы. Завершив рабочий цикл для текущей команды, процессор переходит к выполнению рабочего цикла для следующей команды программы.

На рис. 9.34 представлена схема рабочего цикла процессора. Эта схема имеет достаточно общий характер.

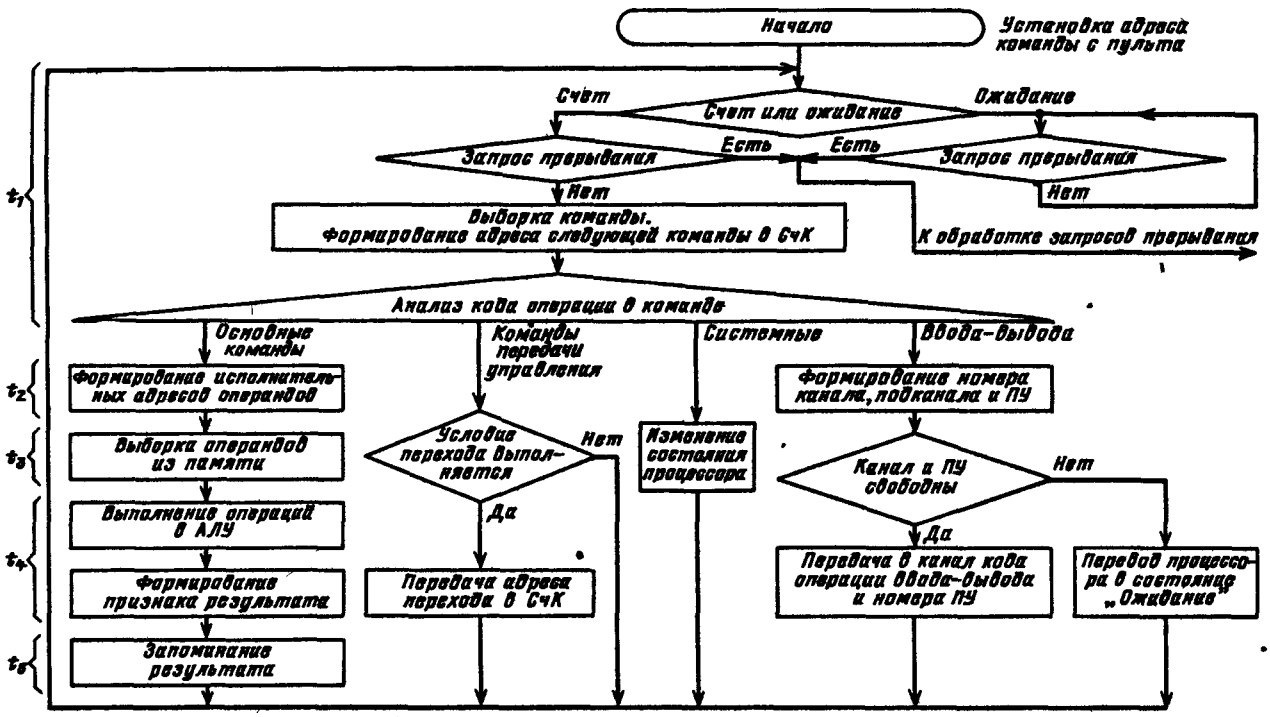


Рис. 9.34. Рабочий цикл процессора

На схеме показаны варианты рабочего цикла для четырех групп команд: 1) основных

(осуществляющих арифметические, логические и пересылочные операции), 2) передачи управления, 3) ввода-вывода и 4) системных (устанавливающих состояние процессора, маску прерывания, слово состояния программы и др.).

Рабочий цикл начинается с распознавания состояния процессора. Устанавливается, какое из альтернативных состояний — *Счет* или *Ожидание* — имеет место. Далее проверяется наличие незамаскированных прерываний.

В состоянии *Ожидание* никакие программы не выполняются. Процессор ждет прихода запроса прерывания, после чего управление переходит к соответствующей прерывающей программе, переводящей процессор в состояние *Счет.*

В состоянии *Счет* при наличии незамаскированных прерываний происходят выход из нормального рабочего цикла н переход к процедуре обработки запросов прерывания.

При отсутствии в состоянии *Счет* запросов прерывания последовательно выполняются этапы рабочего цикла: выборка очередной команды и определение по коду операции команды ее группы, подготовка операндов (формирование исполнительных адресов и выборка операндов из памяти), обработка операндов в АЛУ и запоминание результата.

На этапе выборки очередной команды образуется согласно естественному порядку адрес следующей за ней команды (продвинутый адрес), при этом содержимое счетчика команд (соответствующего поля ССП) увеличивается на число, равное числу байт в очередной команде. В некоторых ЭВМ формирование адреса следующей команды составляет отдельный этап, завершающий рабочий цикл.

В процессе выполнения заданной командой операции формируется *признак результата операции,* используемый командами условного перехода при организации ветвлений в программах.

Указанная выше последовательность этапов составляет основной вариант рабочего цикла, реализуемый при выполнении основных команд.

При выполнении команд передачи управления проверяется заданное командой (например, ее полем маски) условие. Если условие не выполняется, то следующую команду указывает продвинутый адрес, ранее установленный в *СчК* (регистре ССП). Если условие выполняется или имеется один из вариантов команды безусловного перехода, то адрес, задаваемый командой передачи управления, передается в *СчК.*

Команды ввода-вывода инициируют в канале операцию обмена информацией между ядром ЭВМ (основной памятью) и периферийным устройством. Сама эта операция выполняется каналом под управлением его собственной. программы. Поэтому на долю процессора остается только процедура опроса состояний канала и периферийного устройства — свободны ли они для операции ввода-вывода. Если свободны, процессор выдает в канал информацию, необходимую для начала операции ввода-вывода. В противном случае процессор переключается в состоянии *Ожидание* и ждет сигнала прерывания от этого канала.

Системные команды осуществляют переключение состояния процессора (программы) путем загрузки нового ССП или его части. В частности, эти команды изменяют маски прерывания, устанавливают ключи памяти и ключи защиты в ССП, реализуют операции прямого управления.

9.20. Принцип совмещения операций академика С. А. Лебедева. Конвейер операций

Вернемся к схеме рабочего цикла (рис. 9.34) и рассмотрим совокупность этапов цикла для основных команд (основной вариант цикла). Если эти этапы выполняются последовательно во времени, то, суммируя обозначенные на рисунке продолжительности отдельных этапов, получаем время цикла

$$t_{\text{посл}} = t_1 + t_2 + t_3 + t_4 + t_5$$

и производительность процессора, операций (команд)/с,

$$P_{посл} = 1 / t_{посл} = 1 / (t_1 + t_2 + t_3 + t_4 + t_5)$$

Во многих случаях последовательная процедура выполнения этапов цикла не обеспечивает требуемую производительность процессора.

Академик С. А. Лебедев в 1956 г. предложил повышать производительность, используя *принцип совмещения во времени отдельных операций (этапов) рабочего цикла,* и реализовал этот принцип в ЭВМ М-20 в форме параллельного выполнения во времени операции в АЛУ и выборки из памяти следующей команды.

Пусть рабочий цикл процессора состоит из k этапов, причем 1-й этап имеет продолжительность t_i, тогда при последовательном выполнении этапов продолжительность процедуры

$$t_{посл} = \sum_{i=1}^{k} t_i \qquad (*)$$

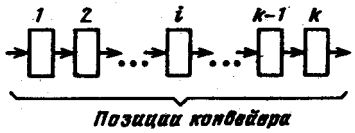


Рис. 9.35. Конвейер операций

и общая производительность процессора, операций/с,

$$P_{посл} = 1 / \sum_{i=1}^{k} t_i \, .$$

Скорость работы машины может быть увеличена, если для выполнения каждого этапа иметь отдельный аппаратурный блок и соединить эти блоки в обрабатывающую линию — конвейер операций (в данном случае конвейер команд) так, чтобы результат выполнения в данном блоке некоторого этапа передавался для реализации очередного этапа на следующий блок, и т. д. (рис. 9.35).

Синхронный конвейер операций. Если конвейер работает в принудительном темпе и для выполнения любого этапа выделено одно и то же время t_T, (такт конвейера), то такой конвейер называется синхронным.

Разбиение процедуры на этапы и выбор длительности такта производятся согласно условиям

$$t_T = \max\{t_i\}, \qquad i = 1, \ldots, k; \qquad (**)$$

$$t_i + t_{i+1} > t_{T,} \qquad i = 1, \ldots, k. \qquad (***)$$

причем в силу цикличности рабочего процесса в последнем неравенстве принимаем $t_{k+1} = t_1$.

Если для каких-либо смежных этапов второе условие не выполняется, то их следует объединить в один этап либо наиболее длинный этап разбить на несколько этапов. В последнем случае заново выбирается t_T и вновь проверяется условие (**).

На рис. 9.36 показана временная диаграмма выполнения команд на 5-позиционном синхронном конвейере. Одинаковыми символами помечены разные этапы рабочего цикла одной и той же команды.

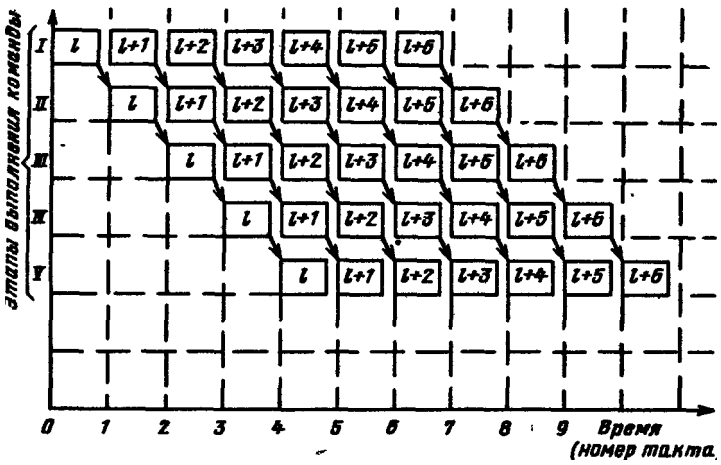


Рис. 9.36. Синхронный конвейер команд

После того как все позиции конвейера окажутся заполненными, параллельно во времени обрабатывается столько команд, сколько в конвейере обрабатывающих блоков (позиций).

Конвейер характеризуется коэффициентом совмещения операций, равным числу одновременно выполняемых этапов обработки информации.

Номинальная производительность синхронного конвейера при его полной загрузке

$$P_{конв}^{ном} = 1 / t_T$$

Найдем соотношение производительностей процессора при конвейерной обработке и при последовательном выполнении этапов рабочего цикла.

Из (*) и (**) имеем $$kt_T \geq \sum_{i=1}^{k} t_i = t_{посл} \qquad (****)$$

а из (*) и (***) получаем $$k t_T < \sum_{i=1}^{k} (t_i + t_{i+1}) = 2 t_{посл} \qquad (*****)$$

Из (****) и (*****) получаем $$k/2 < P_{конв}^{ном} / P_{посл} \leq k \qquad (******)$$

В действительности рост реальной производительности процессора окажется ниже из-за простоев (задержек) конвейера. В процедурах выполнения некоторых команд (например, команд пересылки данных) отдельные этапы общего рабочего цикла отсутствуют, и, следовательно, простаивают отдельные блоки конвейера. Для команды условного перехода по результату предыдущей операции выборка следующей команды должна быть задержана (конвейер простаивает несколько тактов), пока не будет сформирован признак результата (формируется на более позднем этапе) предыдущей операции.

Если p_m — вероятность выборки команды, вызывающей задержку конвейера на m тактов (m = 1, 2, ..., k), то действительная производительность конвейера

$$P_{конв}^{д} = P_{конв}^{ном} / \left(1 + \sum_{m=1}^{k} p_m m \right)$$

Асинхронный конвейер команд. При большой зависимости продолжительности выполнения процедур отдельных этапов от типа команды и вида операндов целесообразно применение асинхронного конвейера, в котором отсутствует единый такт работы его блоков, а информация с одного блока конвейера передается на следующий, когда данный блок закончит свою процедуру, а следующий полностью освободится от обработки предыдущей команды.

Управление передачей информации между соседними блоками в асинхронном конвейере осуществляется с помощью двух триггеров — готовности блока (сигнализирует о завершении операции в блоке) и освобождения последующего блока.

В качестве примера применения асинхронного конвейера команд может служить процессор ЭВМ ЕС-1050, в котором реализован конвейер, выполняющий одновременно три команды. Рабочий цикл выполнения команды разбит на три этапа: I — выборка очередной команды, II — формирование исполнительных адресов и выборка операндов, III — операция в АЛУ, формирование признака результата и запись результата в память.

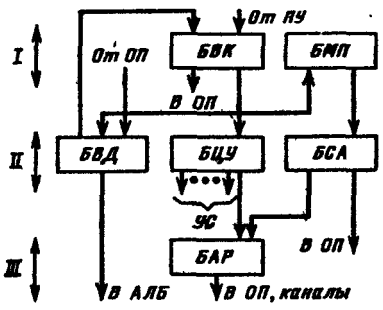


Рис. 9.37. Структура управляющего устройства процессора: *БВК.* — блок выборки команд; *БМП* — блок местной памяти; *БВД* — блок выборки данных;

Для каждого из указанных этапов выполнения команды имеется соответствующая аппаратура. Например, кроме сумматора АЛУ есть отдельный сумматор для формирования исполнительного адреса на этапе II. На рис. 9.37 представлена структура управляющего устройства с «жесткой» логикой процессора ЭВМ ЕС-1050, на которой показаны блоки, управляющие процедурами отдельных этапов выполнения команды.

На рис. 9.38 показана временная диаграмма совмещения выполнения трех команд в ЭВМ ЕС-1050 Временная диаграмма построена для случая, когда выбираемый за одно обращение к памяти «участок программы» содержит четыре команды формата «регистр-регистр».

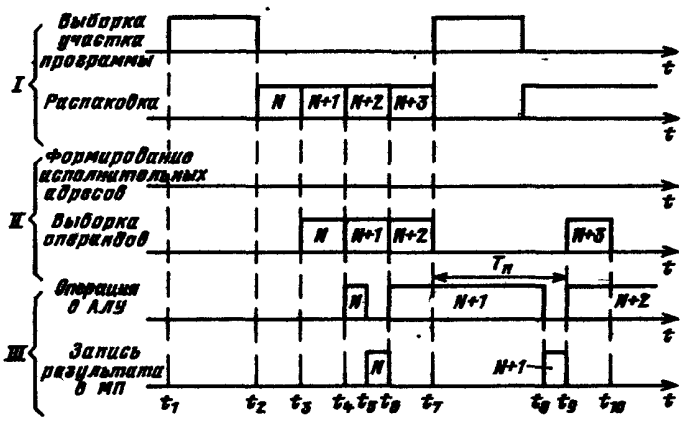


Рис. 9.38. Временная диаграмма совмещения выполнения трех команд (формата “регистр-регистр”) *в* асинхронном конвейере команд.

Этап I содержит две процедуры, выборку из ОП участка программы (8 байт) и распаковку участка — выделение из него очередной команды и размещение ее в регистре команды.

Этап II в общем случае включает в себя формирование исполнительных адресов (при выполнении команд формата “регистр-регистр” отсутствует) и выборку операндов.

Этап III состоит также из двух процедур: выполнения операций в АЛУ и записи результата в память.

Из диаграммы видно, что, начиная с момента времени t_4 выполняются одновременно три этапа цикла соответственно для трех команд. В приведенном примере с момента t_7 из-за большой длительности в команде N+1 операции в АЛУ приостанавливается работа блоков аппаратуры, соответствующих этапам I и II.

Арифметический конвейер. Выше был рассмотрен конвейер команд. Однако в целях повышения производительности машины принцип конвейерной обработки широко используется и в самих выполняющих содержательную обработку информации устройствах (АЛУ), которые строятся в виде *арифметического конвейера,* причем таких арифметических конвейерных линий может быть в процессоре несколько, в том числе и специализированных для определенных операций с данными Подобные операционные (арифметические) устройства часто называют *магистральными.*

Пусть операционное устройство должно вычислять некоторую функцию Ф от входных данных (выполнять некоторую операцию над входными данными). Можно функцию Ф представить в виде последовательности более простых подфункций

$$\varphi_1 \rightarrow \varphi_2 \rightarrow \varphi_3 \rightarrow \ldots \rightarrow \varphi_k$$

причем такой, что результаты преобразования, выполняемые подфункцией φ_i, используются в качестве входных данных при вычислении подфункции φ_{i+1}, и если при этом для каждой подфункции иметь реализующую ее схемный блок, то получим арифметический конвейер, который может быть выполнен как синхронный или как асинхронный.

Приведенные выше элементы теории синхронного конвейера команд остаются в силе и для синхронного арифметического конвейера. Если t_T — такт конвейера, то после полной загрузки он станет выдавать значения функции Ф через интервалы времени t_T. Увеличение производительности процессора за счет использования арифметического конвейера можно оценить по (******).

Если арифметический конвейер используется для выполнения разных операций, то осложняется определение состава рабочих позиций (блоков) конвейера и может потребоваться настройка (диспетчеризация) с соответствующей коммутацией блоков конвейера на операцию, задаваемую текущей командой.

Рассмотрим в качестве примера использование арифметического конвейера для сложения двух векторов **X+Y=Z,** компонентами» которых являются числа, представленные в форме с плавающей точкой и в нормализованном виде.

Выделим в операции сложения чисел с плавающей точкой четыре этапа: 1) сравнение и определение разности порядков, 2) выравнивание порядков - сдвиг мантиссы числа с меньшим

порядком на число разрядов, равное разности порядков; 3) сложение мантисс; 4) нормализация результата.

В арифметическом конвейере эти этапы выполняются отдельными блоками, образующими конвейер, по которому перемещаются операнды или промежуточные результаты операции По мере их перемещения в конвейер вводятся новые компоненты векторов.

Пусть времена, необходимые для выполнения этапов сложения чисел с плавающей точкой, есть t_1, t_2, t_3, t_4.

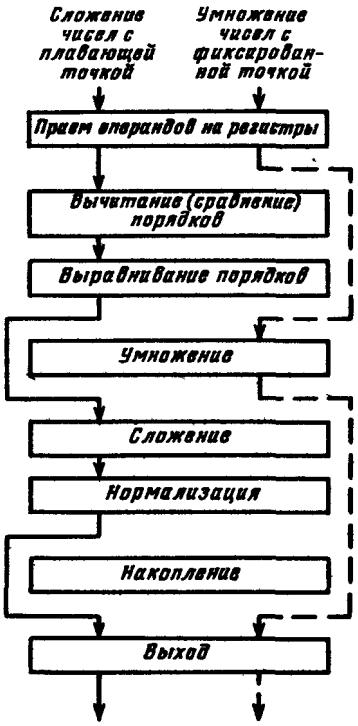


Рис.9.39 Пример настройки арифметического конвейера на выполнение различных операций

Следовательно, если не организовать конвейер и выполнять все этапы операции последовательно, то для получения компонента $z_i = x_i + y_i$ потребуется время $T = t_1 + t_2 + t_3 + t_4$.

В синхронном конвейере, как указывалось выше, продолжительность каждого этапа устанавливается по самому длинному из них, пусть в данном случае это t_3. Тогда, если конвейер заполнен, результаты сложения элементов векторов будут выдаваться через каждые промежутки времени t_3, т. е. значительно быстрее, чем в случае отсутствия конвейерной обработки.

На рис. 9.39 в качестве примера представлена структура конвейерного (магистрального) АЛУ, соответствующего АЛУ известной в свое время ЭВМ ASC фирмы Texas Instruments, и показаны варианты коммутации блоков конвейера для выполнения разных операций, в данном случае сложения чисел с плавающей точкой и умножения чисел с фиксированной точкой.

Особенно эффективно использование операционных (арифметических) конвейеров в специализированных вычислительных устройствах с ограниченным набором алгоритмов обработки входных потоков данных, так как в этом случае возможно разбиение АЛУ на большое число простейших быстродействующих конвейерных блоков при небольших схемных и временных потерях на их коммутацию.

В ряде микропроцессоров одновременно присутствуют конвейер команд и арифметический конвейер, при этом часто в процессоре (микропроцессоре) выделяют *I*-часть — аппаратуру, относящуюся к обработке собственно команд и *E*-часть — аппаратуру, связанную с операциями над данными[1].

I — от Instruction (инструкция, команда) и *E* — от Execution (выполнение).

Контрольные вопросы

Что относится к элементам архитектуры ЭВМ.

Что определяет остроту проблемы при выборе структуры и формата команд современных ЭВМ. Каковы пути решения этой проблемы.

Что такое самоопределяемые данные? Почему при использовании тегов сокращается количество различных команд в системе команд машины.

Почему в малоразрядных ЭВМ и микропроцессорах широко используется косвенная адресация? Приведите пример совместного использования регистровой и косвенной адресации.
|

Поясните, почему стековая память позволяет использовать безадресные команды?
1

Каковы назначение и особенности реализации команды безусловного перехода с возвратом?

Как с помощью индексации организуется обработка упорядоченных массивов данных?

Каковы назначение и процедуры автоинкрементной и автодекрементной адресаций?

Что общего между вектором состояния программы (процессора) и вектором прерывания?

Каковы назначение и процедура прерывания программ ЭВМ?

Что такое векторное прерывание? Опишите процедуру векторного прерывания с использованием стековой памяти.

В чем различие синхронного и асинхронного конвейеров?

Каким образом особенности RISC-архитектуры способствуют повышению ее быстродействия? Какова при этом роль «перекрывающихся регистровых окон»?

Иерархическая структура памяти

Идеальная память должна обеспечивать процессор командами и данными так, чтобы не вызывать простоев процессора. При этом память должна иметь большую емкость. В современных условиях уменьшение времени доступа достигается введением многоуровневой иерархии памяти. Время доступа зависит от объема и типа используемой памяти.

Типовая современная иерархия памяти имеет следующую структуру:

- регистры 64 - 256 слов с временем доступа 1 такт процессора;
- кэш 1 уровня - 8к слов с временем доступа 2 такта;
- кэш 2 уровня - 256к слов с временем доступа 3-5 тактов;
- основная память - до 4 Гигаслов с временем доступа 12-55 тактов.

Используя помимо основной памяти небольшую и более быструю буферную память, можно значительно сократить количество обращений к основной памяти, за счет аккумуляции текущего фрагмента программного кода в буферной памяти. Создание иерархической многоуровневой памяти, пересылающей блоки программ и данных между уровнями памяти за время, пока предшествующие блоки обрабатываются процессором, позволяет существенно сократить простои процессора в ожидании данных. При этом эффект уменьшения времени доступа в память будет тем больше, чем больше время обработки данных в буферной памяти по сравнению с временем пересылки между буферной и основной памятью. Это достигается при локальности обрабатываемых данных, когда процессор многократно использует одни и те же данные для выработки некоторого результата. Например, такая ситуация имеет место при решении систем уравнений в научных и инженерных расчетах, когда короткие участки программного кода с большим количеством вложенных и зацепленных друг с другом циклов обрабатывают поочередно, переходя от точки к точке, небольшие порции данных, многократно используя одни и те же данные и внутренние результаты.

В связи с тем, что локально обрабатываемые данные могут возникать в динамике вычислений и не обязательно сконцентрированы в одной области при статическом размещении в основной памяти, буферную память организуют как ассоциативную, в которой данные содержатся в совокупности с их адресом в основной памяти. Такая буферная память получила название кэш-памяти. Кэш-память позволяет гибко согласовывать структуры данных, требуемые в динамике вычислений, со статическими структурами данных основной памяти.

Кэш имеет совокупность строк (cache-lines), каждая из которых состоит из фиксированного количества адресуемых единиц памяти (байтов, слов) с последовательными адресами. Типичный размер строки: 16, 64, 128, 256 байтов.

Наиболее часто используются три способа организации кэш-памяти, отличающиеся объемом аппаратуры, требуемой для их реализации. Это так называемые кэш-память с прямым отображением (direct-mapped cache), частично ассоциативная кэш-память (set-associative cache) и ассоциативная кэш-память (fully associative cache).

При использовании кэш-памяти с прямым отображением адрес представляется как набор трех компонент, составляющих группы старших, средних и младших разрядов адреса,

соответственно тега, номера строки, смещения. Например, при 16-разрядном адресе старшие 5 разрядов могут представлять тег, следующие 7 разрядов - номер строки и последние 4 разряда - смещение в строке. В этом случае строка состоит из 16 адресуемых единиц памяти, всего строк в кэше 128. Кэш-память с прямым отображением представляет собой набор строк, каждая из которых содержит компоненту тег и элементы памяти строки, адрес которых идентифицируется смещением относительно начала строки.

При этом устанавливается однозначное соответствие между адресом элемента памяти и возможным расположением этого элемента памяти в кэше, а именно: элемент памяти всегда располагается в строке, задаваемой компонентой "номер строки" адреса, и находится на позиции строки, задаваемой компонентой "смещение" адреса.

Наличие элемента данных по запрашиваемому адресу в кэше определяется значением тега. Если тег строки кэш-памяти равен компоненте "тег" адреса, то элемент данных содержится в кэш-памяти.

Иначе необходима подкачка в кэш-память строки, с заданным в адресе тегом.

Так как для определения наличия нужной строки данных в кэш-памяти требуется только одно сравнение тегов заданной строки и адреса, а само замещение строк выполняется по фиксированному местоположению, то объем оборудования, необходимый для реализации этого типа кэш-памяти, достаточно мал.

Недостатки этой организации - очевидны. Если программа использует поочередно элементы памяти из одной строки, но с различными значениями тегов, то это вызывает при каждом обращении замену строки с обращением к данным основной памяти.

Ассоциативная кэш-память использует двухкомпонентное представление адреса: группа старших разрядов трактуется как тег, а группа младших разрядов - как смещение в строке.

Нахождение строки в кэше определяется совпадением тега-строки со значением тега адреса. Количество строк в кэше может быть произвольным (естественное ограничение - количество возможных значений тегов). Поэтому при определении нахождения требуемой строки в кэш-памяти необходимо сравнение тега адреса с тегами всех строк кэша. Если выполнять это последовательно, строка за строкой, то время выполнения сравнений будет непозволительно большим. Поэтому сравнение выполняется параллельно во всех строках с использованием принципов построения ассоциативной памяти, что и дало название этому способу организации кэш-памяти.

При отсутствии необходимой строки в кэш-памяти одна из его строк должна быть заменена на требуемую. Используются разнообразные алгоритмы определения заменяемой строки, например циклический, замена наиболее редко используемой строки, замена строки, к которой дольше всего не было обращений, и другие.

Частично-ассоциативная кэш-память комбинирует оба вышеописанных подхода: кэш-память состоит из набора ассоциативных блоков кэш-памяти. Средняя компонента адреса задает в отличие от прямо адресуемой кэш-памяти не номер строки, а номер одного из ассоциативных блоков. При поиске данных ассоциативное сравнение тегов выполняется только для набора блоков (возможна организация кэша, когда таких наборов несколько), номер которого совпадает со средней компонентой адреса. По количеству n строк в наборе кэш-память называется n-входовой.

Соответствие между данными в оперативной памяти и кэш-памяти обеспечивается внесением изменений в те области оперативной памяти, для которых данные в кэш-памяти подверглись модификации. Соответствие данных обеспечивается параллельно с основными вычислениями. Существует несколько способов его реализации (и, соответственно, несколько режимов работы кэш-памяти).

Один способ предполагает внесение изменений в оперативную память сразу после изменения данных в кэше. При этом процессор простаивает в ожидании завершения записи в основную память. В основной памяти поддерживается правильная копия данных кэша, и при замене строк не требуется никаких дополнительных действий. Кэш-память, работающая в та-

ком режиме, называется памятью со сквозной записью (write- through).

Другой способ предполагает отображение изменений в основной памяти только в момент вытеснения строки данных из кэша. Если данные по адресу памяти, в который необходимо произвести запись, находятся в кэш-памяти, то идет запись только в кэш-память. При отсутствии данных в кэш-памяти производится запись в основную память. Такой режим работы кэша получил название обратной записи (write-back).

Существуют также промежуточные варианты (buffed write though), при которых запросы на изменение в основной памяти буферизуются и не задерживают процессор на время операции записи в память. Эта запись выполняется по мере возможности доступа контроллера кэш-памяти к основной памяти.

Кэш-память с обратной записью (write-back) создает меньшую нагрузку на шину процессора и обеспечивает большую производительность, однако контроллер для write-back кэша значительно сложнее.

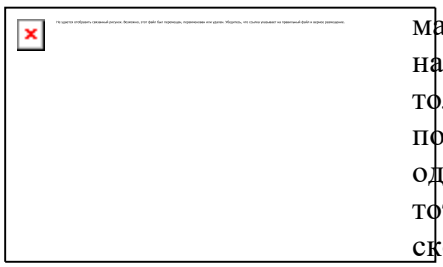
Контроллер кэша отслеживает адреса памяти, выдаваемые процессором, и если адрес соответствует данным, содержащимся в одной из строк кэша, то отмечается "попадание в кэш", и данные из кэша направляются в процессор. Если данных в кэше не оказывается, то фиксируется "промах", и инициируются действия по доставке в кэш из памяти требуемой строки. В ряде процессоров, выполняющих одновременно совокупность команд, допускается несколько промахов, прежде чем будет запущен механизм замены строк.

Рассуждения о том, какой способ организации кэш-памяти более предпочтителен, должны учитывать особенности генерации программ компилятором, а также использование программистом при подготовке программы сведений о работе компилятора и контроллера кэш-памяти. То есть более простой способ организации кэш-памяти, поддерживаемый компилятором, при исполнении программ, написанных в соответствии с некоторыми правилами, обусловленными особенностями компиляции и организации кэш-памяти, может дать лучший результат, чем сложный способ организации кэш-памяти.

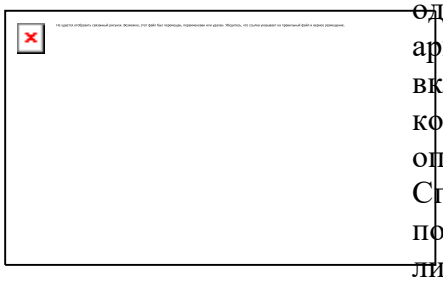
Так как области памяти программ и данных различны и к ним происходит одновременный доступ, то для повышения параллелизма при работе с памятью делают отдельные кэши команд и данных.

Классификация вычислительных систем

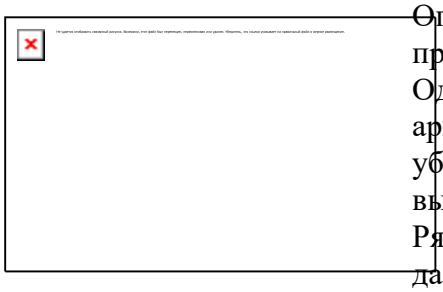
По-видимому, самой ранней и наиболее известной является классификация архитектур вычислительных систем, предложенная в 1966 году М.Флинном [1,2]. Классификация базируется на понятии *потока*, под которым понимается последовательность элементов, команд или данных, обрабатываемая процессором. На основе числа потоков команд и потоков данных Флинн выделяет четыре класса архитектур: SISD,MISD,SIMD,MIMD.



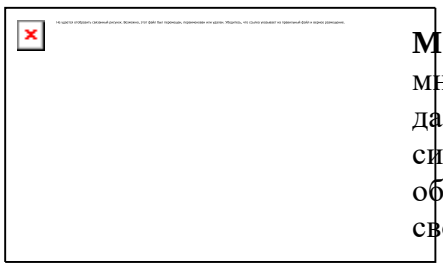
SISD (single instruction stream / single data stream) - одиночный поток команд и одиночный поток данных. К этому классу относятся, прежде всего, классические последовательные машины, или иначе, машины фон-неймановского типа, например, PDP-11 или VAX 11/780. В таких машинах есть только один поток команд, все команды обрабатываются последовательно друг за другом и каждая команда инициирует одну операцию с одним потоком данных. Не имеет значения тот факт, что для увеличения скорости обработки команд и скорости выполнения арифметических операций может применяться конвейерная обработка - как машина CDC 6600 со скалярными функциональными устройствами, так и CDC 7600 с конвейерными попадают в этот класс.



SIMD (single instruction stream / multiple data stream) - одиночный поток команд и множественный поток данных. В архитектурах подобного рода сохраняется один поток команд, включающий, в отличие от предыдущего класса, векторные команды. Это позволяет выполнять одну арифметическую операцию сразу над многими данными - элементами вектора. Способ выполнения векторных операций не оговаривается, поэтому обработка элементов вектора может производится либо процессорной матрицей, как в ILLIAC IV, либо с помощью конвейера, как, например, в машине CRAY-1.



MISD (multiple instruction stream / single data stream) - множественный поток команд и одиночный поток данных. Определение подразумевает наличие в архитектуре многих процессоров, обрабатывающих один и тот же поток данных. Однако ни Флинн, ни другие специалисты в области архитектуры компьютеров до сих пор не смогли представить убедительный пример реально существующей вычислительной системы, построенной на данном принципе. Ряд исследователей [3,4,5] относят конвейерные машины к данному классу, однако это не нашло окончательного признания в научном сообществе. Будем считать, что пока данный класс пуст.



MIMD (multiple instruction stream / multiple data stream) - множественный поток команд и множественный поток данных. Этот класс предполагает, что в вычислительной системе есть несколько устройств обработки команд, объединенных в единый комплекс и работающих каждое со своим потоком команд и данных.

Итак, что же собой представляет каждый класс? В SISD, как уже говорилось, входят однопроцессорные последовательные компьютеры типа VAX 11/780. Однако, многими критиками подмечено, что в этот класс можно включить и векторно-конвейерные машины, если рассматривать вектор как одно неделимое данное для соответствующей команды. В таком случае в этот класс попадут и такие системы, как CRAY-1, CYBER 205, машины семейства FACOM VP и многие другие.

Бесспорными представителями класса SIMD считаются матрицы процессоров: ILLIAC IV, ICL DAP, Goodyear Aerospace MPP, Connection Machine 1 и т.п. В таких системах единое управляющее устройство контролирует множество процессорных элементов. Каждый процессорный элемент получает от устройства управления в каждый фиксированный момент времени одинаковую команду и выполняет ее над своими локальными данными. Для классических процессорных матриц никаких вопросов не возникает, однако в этот же класс можно включить и векторно-конвейерные машины, например, CRAY-1. В этом случае каждый элемент вектора надо рассматривать как отдельный элемент потока данных.

Класс MIMD чрезвычайно широк, поскольку включает в себя всевозможные мультипроцессорные системы: Cm*, C.mmp, CRAY Y-MP, Denelcor HEP,BBN Butterfly, Intel Paragon, CRAY T3D и многие другие. Интересно то, что если конвейерную обработку рассматривать как выполнение множества команд (операций ступеней конвейера) не над одиночным векторным потоком данных, а над множественным скалярным потоком, то все рассмотренные выше векторно-конвейерные компьютеры можно расположить и в данном классе.

Предложенная схема классификации вплоть до настоящего времени является самой применяемой при начальной характеристике того или иного компьютера. Если говорится, что компьютер принадлежит классу SIMD или MIMD, то сразу становится понятным базовый принцип его работы, и в некоторых случаях этого бывает достаточно. Однако видны и явные недостатки. В частности, некоторые заслуживающие внимания архитектуры, например dataflow и векторно--конвейерные машины, четко не вписываются в данную классификацию. Другой недостаток - это чрезмерная заполненность класса MIMD. Необходимо средство, более избирательно систематизирующее архитектуры, которые по Флинну попадают в один класс, но совершенно различны по числу процессоров, природе и топологии связи между ними, по способу организации памяти и, конечно же, по технологии программирования.

Наличие пустого класса (MISD) не стоит считать недостатком схемы. Такие классы, по мнению некоторых исследователей в области классификации архитектур [6,7], могут стать чрезвычайно полезными для разработки принципиально новых концепций в теории и практике построения вычислительных систем.

Основные классы современных параллельных компьютеров
MPP, SMP, NUMA, PVP, кластеры.

Введение. Основным параметром классификации параллельных компьютеров является наличие общей (SMP) или распределенной памяти (MPP). Нечто среднее между SMP и MPP представляют собой NUMA-архитектуры, где память физически распределена, но логически общедоступна. Кластерные системы являются более дешевым вариантом MPP. При поддержке команд обработки векторных данных говорят о векторно-конвейерных процессорах, которые, в свою очередь могут объединяться в PVP-системы с использованием общей или распределенной памяти. Все большую популярность приобретают идеи комбинирования различных архитектур в одной системе и построения неоднородных систем.

При организациях распределенных вычислений в глобальных сетях (Интернет) говорят о мета-компьютерах, которые, строго говоря, не представляют из себя параллельных архитектур.

Подробно рассмотрим особенности всех перечисленных архитектур, а также в описаниях конкретных компьютеров - представителей этих классов. Для каждого класса приводится следующая информация:

- краткое описание особенностей архитектуры,
- примеры конкретных компьютеров,
- перспективы масштабируемости,
- типичные особенности построения операционных систем,
- наиболее характерная модель программирования (хотя возможны и другие).

Рассмотрим наиболее типичные классы архитектур современных параллельных компьютеров и супер-ЭВМ.

Массивно-параллельные системы (MPP)

Архитектура	Система состоит из однородных *вычислительных узлов*, включающих: • один или несколько центральных процессоров (обычно RISC), • локальную память (прямой доступ к памяти других узлов невозможен), • коммуникационный процессор или сетевой адаптер • иногда - жесткие диски (как в SP) и/или другие устройства В/В К системе могут быть добавлены специальные узлы ввода-вывода и управляющие узлы. Узлы связаны через некоторую коммуникационную среду (высокоскоростная сеть, коммутатор и т.п.)
Примеры	IBM RS/6000 SP2, Intel PARAGON/ASCI Red, SGI/CRAY T3E, Hitachi SR8000, транспьютерные системы Parsytec.
Масштабируемость	Общее число процессоров в реальных системах достигает нескольких тысяч (ASCI Red, Blue Mountain).
Операционная система	Существуют два основных варианта: 6. Полноценная ОС работает только на управляющей машине (front-end), на каждом узле работает сильно урезанный вариант ОС, обеспечивающие только работу расположенной в нем ветви параллельного приложения. Пример: Cray T3E. 7. На каждом узле работает полноценная UNIX-подобная ОС (вариант, близкий к кластерному подходу). Пример: IBM RS/6000 SP + ОС AIX, устанавливаемая отдельно на каждом узле.
Модель программирования	Программирование в рамках модели **передачи сообщений** (MPI, PVM, BSPlib)

Симметричные мультипроцессорные системы (SMP)

Архитектура	Система состоит из нескольких однородных процессоров и массива общей памяти (обычно из нескольких независимых блоков). Все процессоры имеют доступ к любой точке памяти с одинаковой скоростью. Процессоры подключены к памяти либо с помощью общей шины (базовые 2-4 процессорные SMP-сервера), либо с помощью crossbar-коммутатора (HP 9000). Аппаратно поддерживается когерентность кэшей.
Примеры	HP 9000 V-class, N-class; SMP-сервера и рабочие станции на базе процессоров Intel (IBM, HP, Compaq, Dell, ALR, Unisys, DG, Fujitsu и др.).
Масштабируемость	Наличие общей памяти сильно упрощает взаимодействие процессоров между собой, однако накладывает сильные ограничения на их число - не более 32 в реальных системах. Для построения масштабируемых систем на базе SMP используются кластерные или NUMA-архитектуры.
Операционная система	Вся система работает под управлением единой ОС (обычно UNIX-подобной, но для Intel-платформ поддерживается Windows NT). ОС автоматически (в процессе работы) распределяет процессы/нити по процессорам (scheduling), но иногда возможна и явная привязка.
Модель программирования	Программирование в модели общей памяти. (POSIX threads, OpenMP). Для SMP-систем существуют сравнительно эффективные средства автоматического распараллеливания.

Системы с неоднородным доступом к памяти (NUMA)

Архитектура	Система состоит из однородных базовых модулей (плат), состоящих из небольшого числа процессоров и блока памяти. Модули объединены с помощью высокоскоростного коммутатора. Поддерживается единое адресное пространство, аппаратно поддерживается доступ к удаленной памяти, т.е. к памяти других модулей. При этом доступ к локальной памяти в несколько раз быстрее, чем к удаленной. В случае, если аппаратно поддерживается когерентность кэшей во всей системе (обычно это так), говорят об архитектуре **cc-NUMA** (cache-coherent NUMA)
Примеры	HP HP 9000 V-class в SCA-конфигурациях, SGI Origin2000, Sun HPC 10000, IBM/Sequent NUMA-Q 2000, SNI RM600.
Масштабируемость	Масштабируемость NUMA-систем ограничивается объемом адресного пространства, возможностями аппаратуры поддежки когерентности кэшей и возможностями операционной системы по управлению большим числом процессоров. На настоящий момент, максимальное число процессоров в NUMA-системах составляет 256 (Origin2000).
Операционная система	Обычно вся система работает под управлением единой ОС, как в SMP. Но возможны также варианты динамического "подразделения" системы, когда отдельные "разделы" системы работают под управлением разных ОС (например, Windows NT и UNIX в NUMA-Q 2000).
Модель программирования	Аналогично SMP.

Параллельные векторные системы (PVP)

Архитектура	Основным признаком PVP-систем является наличие специальных векторно-конвейерных процессоров, в которых предусмотрены команды однотипной обработки векторов независимых данных, эффективно выполняющиеся на конвейерных функциональных устройствах. Как правило, несколько таких процессоров (1-16) работают одновременно над общей памятью (аналогично SMP) в рамках многопроцессорных конфигураций. Несколько таких узлов могут быть объединены с помощью коммутатора (аналогично MPP).
Примеры	NEC SX-4/SX-5, линия векторно-конвейерных компьютеров CRAY: от CRAY-1, CRAY J90/T90, CRAY SV1, серия Fujitsu VPP.
Модель программирования	Эффективное программирование подразумевает векторизацию циклов (для достижения разумной производительности одного процессора) и их распараллеливание (для одновременной загрузки нескольких процессоров одним приложением).

Кластерные системы

Архитектура	Набор рабочих станций (или даже ПК) общего назначения, используется в качестве дешевого варианта массивно-параллельного компьютера. Для связи узлов используется одна из стандартных сетевых технологий (Fast/Gigabit

	Ethernet, Myrinet) на базе шинной архитектуры или коммутатора. При объединении в кластер компьютеров разной мощности или разной архитектуры, говорят о **гетерогенных** (неоднородных) кластерах. Узлы кластера могут одновременно использоваться в качестве пользовательских рабочих станций. В случае, когда это не нужно, узлы могут быть существенно облегчены и/или установлены в стойку.
Примеры	NT-кластер в NCSA, Beowulf-кластеры.
Операционная система	Используются стандартные для рабочих станций ОС, чаще всего, свободно распространяемые - Linux/FreeBSD, вместе со специальными средствами поддержки параллельного программирования и распределения нагрузки.
Модель программирования	Программирование, как правило, в рамках модели передачи сообщений (чаще всего - MPI). Дешевизна подобных систем оборачивается большими накладными расходами на взаимодействие параллельных процессов между собой, что сильно сужает потенциальный класс решаемых задач.

Рассмотрим устаревшие и проектируемые архитектуры.

Denelcor HEP (Heterogeneous Element Processor)

Данный компьютер считается первой коммерчески доступной вычислительной системой с множественным потоком команд. В своей полной конфигурации Denelcor HEP содержит 16 процессорных модулей (Process Execution Module - PEM), через многокаскадный переключатель связанных со 128 модулями памяти данных (Data Memory Module - DMM). Все процессорные модули могут работать независимо друг от друга со своими потоками команд. В свою очередь каждый процессорный модуль может поддерживать до 50 потоков команд пользователей. На уровне процессорного модуля множественность потоков команд обеспечивается одним восьмиуровневым конвейерным устройством для обработки команд. На каждой ступени конвейера должны находиться команды из разных потоков. Следовательно, скорость вычислений увеличивается с увеличением количества потоков команд, пока конвейер не будет заполнен. После заполнения конвейера эта величина остается постоянной.

C.mpp

Содержит до 16 машин типа DEC PDP-11, связанных с 16 модулями памяти через перекрестный переключатель размерности 16x16.

PASM (Partitioned SIMD/MIMD computer)

Содержит до $N=2^n$ процессорных элементов, каждый из которых содержит свое устройство обработки данных и модуль памяти из двух блоков. Все процессорные элементы между собой связываются через многокаскадный переключатель. Отличительной особенностью этой архитектуры является возможность динамически менять свою конфигурацию в зависимости от прикладных задач. Система может быть сконфигурирована либо как SIMD, либо как MIMD компьютер. Кроме локальной памяти, каждый процессорный элемент имеет доступ к общей памяти.

PRINGLE

Это вычислительная система типа MIMD с распределенной памятью, состоящая из 64 процессорных элементов (ПЭ). Каждый ПЭ содержит 8-разряд\-ный микропроцессор Intel 8031 с 32--разрядным сопроцессором Intel 8231 и локальную память объемом 2 Кбайта. В качестве контроллера используется 16-разрядный микропроцессор Intel 8086. Связь процессорных элементов осуществляется через общую шину.

ICAP (loosely Coupled Array Processors

Данная система состоит из нескольких машин FPS 164, которые контролируются одной управляющей машиной. В демонстрационных образцах было использовано семь FPS 164, каждая из которых имела по 4 Мбайта основной памяти. Управляющей машиной служила IBM 4381.

Cm*

Основной компонентой этой системы является "вычислительный модуль", состоящий из микропроцессора DEC LSI-11 с 64Мбайтами dynamic MOS memory. Данный модуль может работать как отдельный компьютер. В то же время до 14 таких модулей могут быть подключены к шине (intracluster bus), формируя таким образом сильносвязанную систему (кластер - tightly-coupled cluster). Внутри этого кластера передача данных происходит путем прямого доступа к памяти. Построенные таким образом кластеры можно связать в более сложную систему через две соединяющие кластеры шины (intercluster buses). При этом получается слабо связанная сеть (loosely-coupled network), в которой обмен данными происходит путем коммутации пакетов (packet switching techniques).

CEDAR

В состав системы входит шестнадцать кластеров по восемь процессорных элементов (ПЭ) в каждом. Кластеры связаны через расширенную сеть типа Omega (extended Omega global switching network) с 256 модулями глобальной памяти. Каждый модуль памяти имеет объем от 4 до 16 мегаслов. Процессорные элементы, составляющие кластер, имеют по 16 мегаслов локальной памяти. Все процессорные элементы конвейерного типа и связаны между собой через локальную коммутационную сеть (local switching network).

STARAN

В ее состав входят четыре матричных модуля, управляемых последовательной машиной PDP-11. Каждый модуль содержит 256 ПЭ и общую память емкостью от 64 Кбит до 64 Мбит. Связь между ПЭ и памятью осуществляется через гибкую коммутационную сеть.

PEPE (Parallel Element Processor Ensemble)

Это система из 288 ПЭ с низкой степенью связности. Каждый процессорный элемент содержит по три процессора (каждый процессор предназначался для выполнения определенной функции, связанной с задачей радиолокации), управляемых в синхронном режиме тремя устройствами управления (по одному на каждый тип процессора в ПЭ). Эти три устройства управления подключались к трем стандартным каналам ввода-вывода машины CDC 7600, которая была главной для всей системы. Связь между ПЭ осуществлялась через блоки памяти устройств управления.

PRIME

Система состоит из пяти процессоров. Каждый процессор через матричный коммутатор имеет доступ к блокам памяти (количество блоков варьируется от одного и более). Через сеть внешнего доступа процессоры соединяются с памятью на внешних носителях и устройствами ввода--вывода. В каждый момент времени некоторый процессор с памятью работает как управляющий процессор (монитор), регулируя активность остальных рабочих процессоров.