Министерство образования Республики Беларусь

Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра Программного обеспечения информационных технологий

Е.В. Мельникова

# Электронный учебно-методический комплекс по дисциплине КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ

Для студентов специальности Т.10 02 00
«Программное обеспечение информационных технологий»

Минск 2009

СОДЕРЖАНИЕ

**Введение**

Архитектура_– это термин, обычно использующийся для описания состава, принципа действия, конфигурации и взаимного соединения основных узлов вычислительной системы на некотором общем уровне, включая описание пользовательских возможностей программирования, системы команд и средств пользовательского интерфейса, организации памяти и системы адресации, операций ввода–вывода и управления и т.д. Таким образом , термин «архитектура» относится как к аппаратным средствам, так и к программному обеспечению, и их комбинациям.

Существуют две основные архитектуры построения вычислительных систем – фон Неймана и Гарвардская.

Американский математик фон Нейман (1903-1957) предложил концепцию вычислительной машины (и в частности, хранимой в памяти программы), которая лежит в основе большинства современных машин. Одним из основных моментов этой концепции является то, что система обладает единой памятью, в которой хранятся и команды программы и данные. Система содержит одну шину данных (ШД) , по которой передаются и команды программы, и данные. Следовательно, в такой системе требуется три цикла для выборки команды и двух сомножителей (т.е .для выполнения операции MAC – базовая операция умножения и добавления результата умножения) На рис. 1 показана традиционная структура вычислительной системы, соответствующая «фон-неймановской» архитектуре.

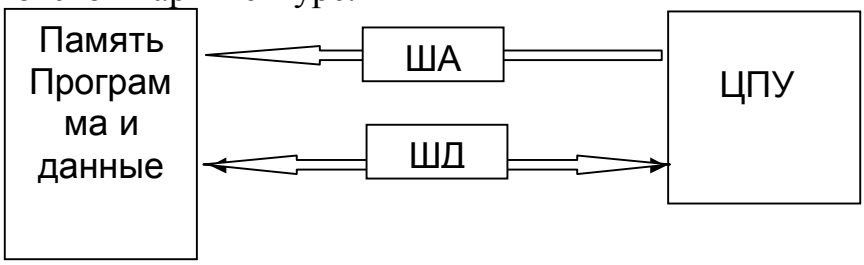


Рис.1

Гарвадская архитектура названа по работе, выполненной в 40-х годах XX века в университете Гарварда под руководством Г.Айкена (1900-1973). В соответствии с этой концепцией для хранения программы (команд) и данных используются различные устройства памяти. Соответственно в системе имеется два комплекта шин для этих устройств: шина адреса памяти программ (ШАПП), шина данных памяти программ для работы с памятью программ (ПП) и шина адреса памяти данных (ШАПД), шина данных памяти данных (ШДПД) для работы с памятью данных (ПД). В системе с гарвардской архитектурой можно одновременно производить операции обращения к различным устройствам памяти, т.е. синхронно выбирать команду из памяти программ ПП по шине ШДПП и сомножитель из памяти данных ПД по шине ШДПД. Соответственно при этом для выполнения операции MAC требуется два цикла работы процессора, Реально за счет различных дополнительных мер почти

всегда время операции MAC сводится к одному циклу. Гарвадская архитектура приведена на рис.2.

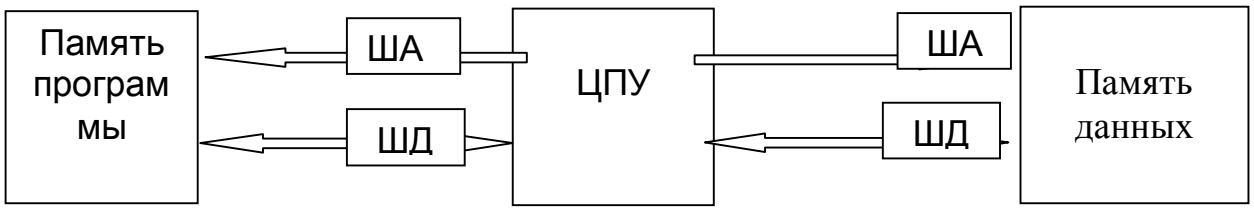


Рис.2

Таким образом, Гарвардская архитектура разделяя пространства памяти данных и программ и предусматривая раздельные шины доступа к каждой из них, обеспечивает доступность и данных, и программ в одном цикле выполнения операций процессором, что увеличивает общую скорость обработки.

В современных процессорах часто применяется модифицированная Гарвардская архитектура, когда для общения с внешней памятью используется один набор шин, в то время как на самом кристалле для увеличения быстродействия они разделены Такой подход минимизирует общую стоимость системы, сохраняя преимущества Гарвардской архитектуры. В дополнение к этому применяются различные схемы кэширования и конвейерной обработки.

Архитектура ЭВМ — это абстрактное представление или определение физической системы (микропрограммы или комплекса аппаратных средств) с точки зрения программиста, разрабатывающего программы на машинно-ориентированном языке, или разработчика компилятора. Она определяет принципы организации вычислительной системы и функции процессора и не отражает такие проблемы, как управление и передача данных внутри процессора, конструктивные особенности элементной базы и специфику технологии ее производства.

Основа любой ЭВМ - центральный процессор (ЦПУ). Он работает под управлением программных средств, преобразуя входную информацию в выходную. Преобразования осуществляются системой команд, последовательность которых реализует программу решения задачи. Команда, попадая в ЦПУ, проходит несколько этапов : выборка команды, выполнение действий над операндами, формирование адресов и статусных флагов. Это обеспечивается в ЦПУ микропрограммным автоматом, формирующим микропрограмму для каждой команды. Для ЦПУ команда представлена на машинном языке в виде последовательности двоичных кодов. Формат команды включает поля: код операции, сведения об операнде источнике и приемнике, Команды и операнды располагаются в памяти машины. Быстрее всего выполняются действия над операндами, заключенными во внутренние регистры ЦПУ. Для расширения возможностей доступа к данным используется система режимов адресации.

ЦПУ как отдельный аппаратный узел объединен с рядом регулярных узлов сопровождения в один кристалл – микропроцессор (МП).

## 1.Архитектура микропроцессоров

ЭВМ получили широкое распространение, начиная с 50-х годов. Прежде это были очень большие и дорогие устройства, используемые лишь в государственных учреждениях и крупных фирмах. Размеры и форма цифровых ЭВМ неузнаваемо изменились в результате разработки новых устройств, называемых микропроцессорами.

Микропроцессор (МП) - это программно-управляемое электронное цифровое устройство, предназначенное для обработки цифровой информации и управления процессом этой обработки, выполненное на одной или нескольких интегральных схемах с высокой степенью интеграции электронных элементов.

В 1970 году Маршиан Эдвард Хофф из фирмы Intel сконструировал интегральную схему, аналогичную по своим функциям центральному процессору большой ЭВМ - первый микропроцессор Intel-4004, который уже в 1971 году был выпущен в продажу.

15 ноября 1971 г. можно считать началом новой эры в электронике. В этот день компания приступила к поставкам первого в мире микропроцессора Intel 4004.

Это был настоящий прорыв, ибо МП Intel-4004 размером менее 3 см был производительнее гигантской машины ENIAC. Правда работал он гораздо медленнее и мог обрабатывать одновременно только 4 бита информации (процессоры больших ЭВМ обрабатывали 16 или 32 бита одновременно), но и стоил первый МП в десятки тысяч раз дешевле.

Кристалл представлял собой 4-разрядный процессор с классической архитектурой ЭВМ гарвардского типа и изготавливался по передовой p-канальной МОП технологии с проектными нормами 10 мкм. Электрическая схема прибора насчитывала 2300 транзисторов. МП работал на тактовой частоте 750 кГц при длительности цикла команд 10,8 мкс. Чип i4004 имел адресный стек (счетчик команд и три регистра стека типа LIFO), блок РОНов (регистры сверхоперативной памяти или регистровый файл - РФ), 4-разрядное параллельное АЛУ, аккумулятор, регистр команд с дешифратором команд и схемой управления, а также схему связи с внешними устройствами. Все эти функциональные узлы объединялись между собой 4-разрядной ШД. Память команд достигала 4 Кбайт (для сравнения: объем ЗУ миниЭВМ в начале 70-х годов редко превышал 16 Кбайт), а РФ ЦП насчитывал 16 4-разрядных регистров, которые можно было использовать и как 8 8-разрядных. Такая организация РОНов сохранена и в последующих МП фирмы Intel. Три регистра стека обеспечивали три уровня вложения подпрограмм. МП i4004 монтировался в пластмассовый или металлокерамический корпус типа DIP (Dual In-line Package) всего с 16 выводами.

В систему его команд входило всего 46 инструкций.

Вместе с тем кристалл располагал весьма ограниченными средствами ввода/вывода, а в системе команд отсутствовали операции логической обработки данных (И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ), в связи с чем их приходилось реализовывать с помощью специальных подпрограмм. Модуль i4004 не имел возможности останова (команды HALT) и обработки прерываний.

Цикл команды процессора состоял из 8 тактов задающего генератора. Была мультиплексированная ША (шина адреса)/ШД (шина данных), адрес 12-разрядный передавался по 4-разряда.

1 апреля 1972 г. фирма Intel начала поставки первого в отрасли 8-разрядного прибора i8008. Кристалл изготавливался по p-канальной МОП-технологии с проектными нормами 10 мкм и содержал 3500 транзисторов. Процессор работал на частоте 500 кГц при длительности машинного цикла 20 мкс (10 периодов задающего генератора).

В отличие от своих предшественников МП имел архитектуру ЭВМ принстонского типа, а в качестве памяти допускал применение комбинации ПЗУ и ОЗУ.

По сравнению с i4004 число РОН уменьшилось с 16 до 8, причем два регистра использовались для хранения адреса при косвенной адресации памяти (ограничение технологии - блок РОН аналогично кристаллам 4004 и 4040 в МП 8008 был реализован в виде динамической памяти). Почти вдвое сократилась длительность машинного цикла (с 8 до 5 состояний). Для синхронизации работы с медленными устройствами был введен сигнал готовности READY.

Система команд насчитывала 65 инструкций. МП мог адресовать память объемом 16 Кбайт. Его производительность по сравнению с четырехразрядными МП возрасла в 2,3 раза. В среднем для сопряжения процессора с памятью и устройствами ввода/вывода требовалось около 20 схем средней степени интеграции.

Возможности p-канальной технологии для создания сложных высокопроизводительных МП были почти исчерпаны, поэтому "направление главного удара" перенесли на n-канальную МОП технологию.

1 апреля 1974 МП Intel 8080 был представлен вниманию всех заинтересованных лиц. Благодаря использованию технологии n-МОП с проектными нормами 6 мкм, на кристалле удалось разместить 6 тыс. транзисторов. Тактовая частота процессора была доведена до 2 Мгц, а длительность цикла команд составила уже 2 мкс. Объем памяти, адресуемой процессором, был увеличен до 64 Кбайт. За счет использования 40-выводного корпуса удалось разделить ША и ШД, общее число микросхем, требовавшихся для построения системы в минимальной конфигурации сократилось до 6 .

В регистровый файл были введены указатель стека, активно используемый при обработке прерываний, а также два программнонедоступных регистра для внутренних пересылок. Блок РОНов был реализован на микросхемах

статической памяти. Исключение аккумулятора из РФ и введение его в состав АЛУ упростило схему управления внутренней шиной.
Новое в архитектуре МП - использование многоуровневой системы прерываний по вектору. Такое техническое решение позволило довести общее число источников прерываний до 256 (до появления БИС контроллеров прерываний схема формирования векторов прерываний требовала применения до 10 дополнительных чипов средней интеграции). В i8080 появился механизм прямого доступа в память (ПДП) (как ранее в универсальных ЭВМ IBM System 360 и др.).
ПДП открыл зеленую улицу для применения в микроЭВМ таких сложных устройств, как накопители на магнитных дисках и лентах дисплеи на ЭЛТ, которые и превратили микроЭВМ в полноценную вычислительную систему.
Традицией компании, начиная с первого кристалла, стал выпуск не отдельного чипа ЦП, а семейства БИС, рассчитанных на совместное использование.

## 1.1.Классификация

По числу больших интегральных схем (БИС) в микропроцессорном комплекте различают микропроцессоры однокристальные, многокристальные и многокристальные секционные.
Процессоры даже самых простых ЭВМ имеют сложную функциональную структуру, содержат большое количество электронных элементов и множество разветвленных связей. Изменять структуру процессора необходимо так, чтобы полная принципиальная схема или ее части имели количество элементов и связей, совместимое с возможностями БИС. При этом микропроцессоры приобретают внутреннюю магистральную архитектуру, т. е. в них к единой внутренней информационной магистрали подключаются все основные функциональные блоки (арифметико-логический, рабочих регистров, стека, прерываний, интерфейса, управления и синхронизации и др.).
Для обоснования классификации микропроцессоров по числу БИС надо распределить все аппаратные блоки процессора между основными тремя функциональными частями: операционной, управляющей и интерфейсной. Сложность операционной и управляющей частей процессора определяется их разрядностью, системой команд и требованиями к системе прерываний; сложность интерфейсной части разрядностью и возможностями подключения других устройств ЭВМ (памяти, внешних устройств, датчиков и исполнительных механизмов и др.). Интерфейс процессора содержит несколько десятков информационных шин данных (ШД), адресов (ША) и управления (ШУ).
Однокристальные микропроцессоры получаются при реализации всех аппаратных средств процессора в виде одной БИС или СБИС (сверхбольшой интегральной схемы). По мере увеличения степени интеграции элементов в кристалле и числа выводов корпуса параметры однокристальных микропроцессоров улучшаются. Однако возможности однокристальных

микропроцессоров ограничены аппаратными ресурсами кристалла и корпуса. Для получения многокристального микропроцессора необходимо провести разбиение его логической структуры на функционально законченные части и реализовать их в виде БИС (СБИС). Функциональная законченность БИС многокристального микропроцессора означает, что его части выполняют заранее определенные функции и могут работать автономно.

На рис. 1.1.1 показано функциональное разбиение структуры процессора при создании трехкристального микропроцессора (пунктирные линии), содержащего БИС операционного (ОП), БИС управляющего (УП) и БИС интерфейсного (ИП) процессоров.

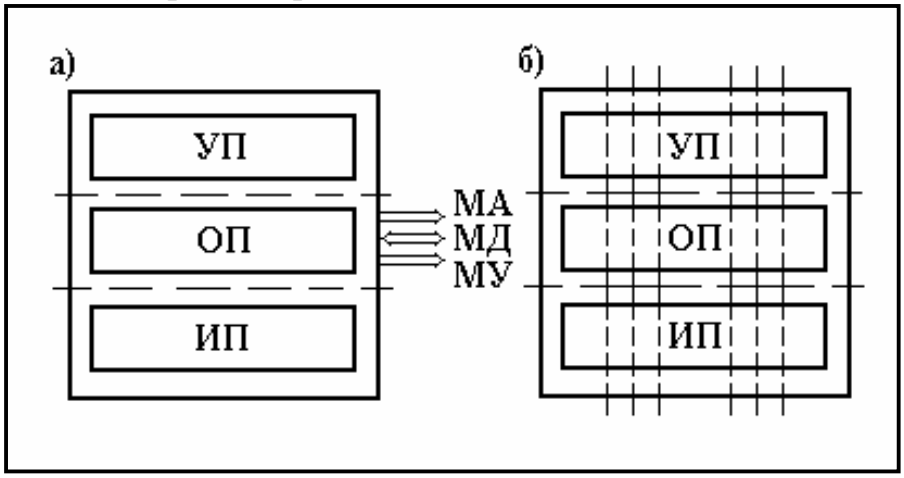


Рис. 1.1.1. Функциональная структура процессора (а)
и ее разбиение для реализации процессора
в виде комплекта секционных БИС.

Операционный процессор служит для обработки данных, управляющий процессор выполняет функции выборки, декодирования и вычисления адресов операндов и также генерирует последовательности микрокоманд. Автономность работы и большое быстродействие БИС УП позволяет выбирать команды из памяти с большей скоростью, чем скорость их исполнения БИС ОП. При этом в УП образуется очередь еще не исполненных команд, а также заранее подготавливаются те данные, которые потребуются ОП в следующих циклах работы. Такая опережающая выборка команд экономит время ОП на ожидание операндов, необходимых для выполнения команд программ. Интерфейсный процессор позволяет подключить память и периферийные средства к микропроцессору; он, по существу, является сложным контроллером для устройств ввода/вывода информации. БИС ИП выполняет также функции канала прямого доступа к памяти.

Выбираемые из памяти команды распознаются и выполняются каждой частью микропроцессора автономно и поэтому может быть обеспечен режим одновременной работы всех БИС МП, т.е. конвейерный поточный режим исполнения последовательности команд программы (выполнение

последовательности с небольшим временным сдвигом). Такой режим работы значительно повышает производительность микропроцессора.
Многокристальные секционные микропроцессоры получаются в том случае, когда в виде БИС реализуются части (секции) логической структуры процессора при функциональном разбиении ее вертикальными плоскостями Для построения многоразрядных микропроцессоров при параллельном включении секций БИС в них добавляются средства "стыковки".
Для создания высокопроизводительных многоразрядных микропроцессоров требуется столь много аппаратных средств, не реализуемых в доступных БИС, что может возникнуть необходимость еще и в функциональном разбиении структуры микропроцессора горизонтальными плоскостями. В результате рассмотренного функционального разделения структуры микропроцессора на функционально и конструктивно законченные части создаются условия реализации каждой из них в виде БИС. Все они образуют комплект секционных БИС МП.
Таким образом, микропроцессорная секция это БИС, предназначенная для обработки нескольких разрядов данных или выполнения определенных управляющих операций. Секционность БИС МП определяет возможность "наращивания" разрядности обрабатываемых данных или усложнения устройств управления микропроцессора при "параллельном" включении большего числа БИС.
Однокристальные и трехкристальные БИС МП, как правило, изготовляют на основе микроэлектронных технологий униполярных полупроводниковых приборов, а многокристальные секционные БИС МП на основе технологии биполярных полупроводниковых приборов. Использование многокристальных микропроцессорных высокоскоростных биполярных БИС, имеющих функциональную законченность при малой физической разрядности обрабатываемых данных и монтируемых в корпус с большим числом выводов, позволяет организовать разветвление связи в процессоре, а также осуществить конвейерные принципы обработки информации для повышения его производительности.
По назначению различают универсальные и специализированные микропроцессоры.
Универсальные микропроцессоры могут быть применены для решения широкого круга разнообразных задач. При этом их эффективная производительность слабо зависит от проблемной специфики решаемых задач. Специализация МП, т.е. его проблемная ориентация на ускоренное выполнение определенных функций позволяет резко увеличить эффективную производительность при решении только определенных задач.
Среди специализированных микропроцессоров можно выделить различные микроконтроллеры, ориентированные на выполнение сложных последовательностей логических операций, математические МП, предназначенные для повышения производительности при выполнении

арифметических операций за счет, например, матричных методов их выполнения, МП для обработки данных в различных областях применений и т. д. С помощью специализированных МП можно эффективно решать новые сложные задачи параллельной обработки данных. Например, конволюция позволяет осуществить более сложную математическую обработку сигналов, чем широко используемые методы корреляции. Последние в основном сводятся к сравнению всего двух серий данных: входных, передаваемых формой сигнала, и фиксированных опорных и к определению их подобия. Конволюция дает возможность в реальном масштабе времени находить соответствие для сигналов изменяющейся формы путем сравнения их с различными эталонными сигналами, что, например, может позволить эффективно выделить полезный сигнал на фоне шума.

Разработанные однокристальные конвольверы используются в устройствах опознавания образов в тех случаях, когда возможности сбора данных превосходят способности системы обрабатывать эти данные.

По виду обрабатываемых входных сигналов различают цифровые и аналоговые микропроцессоры. Сами микропроцессоры цифровые устройства, однако могут иметь встроенные аналого-цифровые и цифро-аналоговые преобразователи. Поэтому входные аналоговые сигналы передаются в МП через преобразователь в цифровой форме, обрабатываются и после обратного преобразования в аналоговую форму поступают на выход. С архитектурной точки зрения такие микропроцессоры представляют собой аналоговые функциональные преобразователи сигналов и называются аналоговыми микропроцессорами. Они выполняют функции любой аналоговой схемы (например, производят генерацию колебаний, модуляцию, смещение, фильтрацию, кодирование и декодирование сигналов в реальном масштабе времени и т.д., заменяя сложные схемы, состоящие из операционных усилителей, катушек индуктивности, конденсаторов и т.д.). При этом применение аналогового микропроцессора значительно повышает точность обработки аналоговых сигналов и их воспроизводимость, а также расширяет функциональные возможности за счет программной "настройки" цифровой части микропроцессора на различные алгоритмы обработки сигналов.

Обычно в составе однокристальных аналоговых МП имеется несколько каналов аналого-цифрового и цифро-аналогового преобразования. В аналоговом микропроцессоре разрядность обрабатываемых данных достигает 24 бит и более, большое значение уделяется увеличению скорости выполнения арифметических операций.

Отличительная черта аналоговых микропроцессоров способность к переработке большого объема числовых данных, т. е. к выполнению операций сложения и умножения с большой скоростью при необходимости даже за счет отказа от операций прерываний и переходов. Аналоговый сигнал, преобразованный в цифровую форму, обрабатывается в реальном масштабе времени и передается на выход обычно в аналоговой форме через цифро-аналоговый

преобразователь. При этом согласно теореме Котельникова частота квантования аналогового сигнала должна вдвое превышать верхнюю частоту сигнала.

Сравнение цифровых микропроцессоров производится сопоставлением времени выполнения ими списков операций. Сравнение же аналоговых микропроцессоров производится по количеству эквивалентных звеньев аналого-цифровых фильтров рекурсивных фильтров второго порядка. Производительность аналогового микропроцессора определяется его способностью быстро выполнять операции умножения: чем быстрее осуществляется умножение, тем больше эквивалентное количество звеньев фильтра в аналоговом преобразователе и тем более сложный алгоритм преобразования цифровых сигналов можно задавать в микропроцессоре.

Одним из направлений дальнейшего совершенствования аналоговых микропроцессоров является повышение их универсальности и гибкости. Поэтому вместе с повышением скорости обработки большого объема цифровых данных будут развиваться средства обеспечения развитых вычислительных процессов обработки цифровой информации за счет реализации аппаратных блоков прерывания программ и программных переходов.

По характеру временной организации работы микропроцессоры делят на синхронные и асинхронные.

Синхронные микропроцессоры - микропроцессоры, в которых начало и конец выполнения операций задаются устройством управления (время выполнения операций в этом случае не зависит от вида выполняемых команд и величин операндов).

Асинхронные микропроцессоры позволяют начало выполнения каждой следующей операции определить по сигналу фактического окончания выполнения предыдущей операции. Для более эффективного использования каждого устройства микропроцессорной системы в состав асинхронно работающих устройств вводят электронные цепи, обеспечивающие автономное функционирование устройств. Закончив работу над какой-либо операцией, устройство вырабатывает сигнал запроса, означающий его готовность к выполнению следующей операции. При этом роль естественного распределителя работ принимает на себя память, которая в соответствии с заранее установленным приоритетом выполняет запросы остальных устройств по обеспечению их командной информацией и данными.

По организации структуры микропроцессорных систем различают микроЭВМ одно- и многомагистральные.

В одномагистральных микроЭВМ все устройства имеют одинаковый интерфейс и подключены к единой информационной магистрали, по которой передаются коды данных, адресов и управляющих сигналов.

В многомагистральных микроЭВМ устройства группами подключаются к своей информационной магистрали. Это позволяет осуществить одновременную передачу информационных сигналов по нескольким (или всем) магистралям.

Такая организация систем усложняет их конструкцию, однако увеличивает производительность.
По количеству выполняемых программ различают одно- и многопрограммные микропроцессоры.
В однопрограммных микропроцессорах выполняется только одна программа. Переход к выполнению другой программы происходит после завершения текущей программы.
В много- или мультипрограммных микропроцессорах одновременно выполняется несколько (обычно несколько десятков) программ. Организация мультипрограммной работы микропроцессорных управляющих систем позволяет осуществить контроль за состоянием и управлением большим числом источников или приемников информации.

## 1.2.Основные характеристики микропроцессора

Микропроцессор характеризуется:
1) тактовой частотой, определяющей максимальное время выполнения переключения элементов в ЭВМ;
2) разрядностью, т.е. максимальным числом одновременно обрабатываемых двоичных разрядов.
Разрядность МП обозначается m/n/k/ и включает:
m - разрядность внутренних регистров, определяет принадлежность к тому или иному классу процессоров;
n - разрядность шины данных, определяет скорость передачи информации;
k - разрядность шины адреса, определяет размер адресного пространства. Например, МП i8088 характеризуется значениями m/n/k=16/8/20;
3) архитектурой. Понятие архитектуры микропроцессора включает в себя систему команд и способы адресации, возможность совмещения выполнения команд во времени, наличие дополнительных устройств в составе микропроцессора, принципы и режимы его работы. Выделяют понятия микроархитектуры и макроархитектуры.
Микроархитектура микропроцессора - это аппаратная организация и логическая структура микропроцессора, регистры, управляющие схемы, арифметико-логические устройства, запоминающие устройства и связывающие их информационные магистрали.
Макроархитектура - это система команд, типы обрабатываемых данных, режимы адресации и принципы работы микропроцессора.
В общем случае под архитектурой ЭВМ понимается абстрактное представление машины в терминах основных функциональных модулей, языка ЭВМ, структуры данных.

## 1.3.Структура типового микропроцессора

Архитектура типичной небольшой вычислительной системы на основе микроЭВМ показана на рис. 2.1 Такая микроЭВМ содержит все 5 основных

блоков цифровой машины: устройство ввода информации, управляющее устройство (УУ), арифметико-логическое устройство (АЛУ) (входящие в состав микропроцессора), запоминающие устройства (ЗУ) и устройство вывода информации.

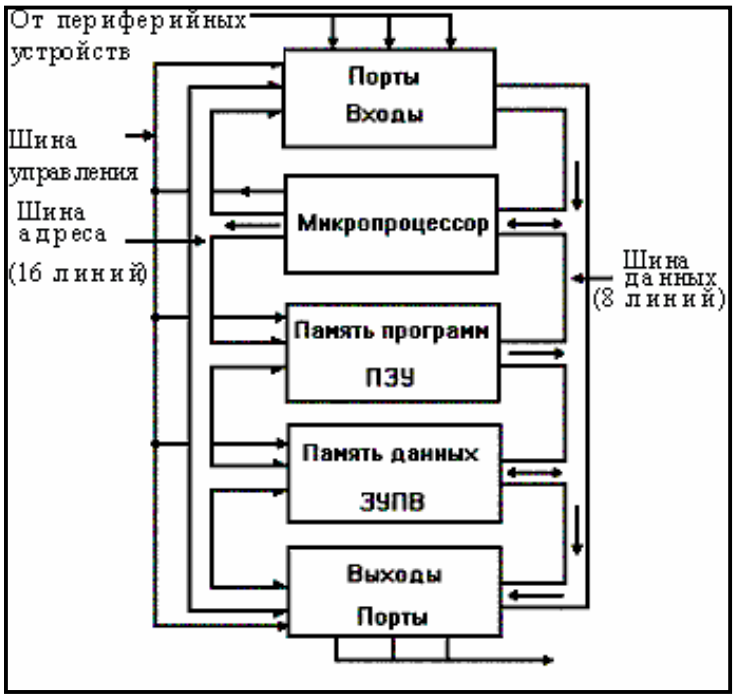


Рис. 1.3.1. Архитектура типового микропроцессора.

Микропроцессор координирует работу всех устройств цифровой системы с помощью шины управления (ШУ). Помимо ШУ имеется 16-разрядная адресная шина (ША), которая служит для выбора определенной ячейки памяти, порта ввода или порта вывода. По 8-разрядной информационной шине или шине данных (ШД) осуществляется двунаправленная пересылка данных к микропроцессору и от микропроцессора. Важно отметить, что МП может посылать информацию в память микроЭВМ или к одному из портов вывода, а также получать информацию из памяти или от одного из портов ввода.

Постоянное запоминающее устройство (ПЗУ) в микроЭВМ содержит некоторую программу (на практике программу инициализации ЭВМ). Программы могут быть загружены в запоминающее устройство с произвольной выборкой (ЗУПВ) и из внешнего запоминающего устройства (ВЗУ). Это программы пользователя.

В качестве примера, иллюстрирующего работу микроЭВМ, рассмотрим процедуру, для реализации которой нужно выполнить следующую последовательность элементарных операций:

1. Нажать клавишу с буквой "А" на клавиатуре.
2. Поместить букву "А" в память микроЭВМ.
3. Вывести букву "А" на экран дисплея.

Это типичная процедура ввода-запоминания-вывода, рассмотрение которой дает возможность пояснить принципы использования некоторых устройств, входящих в микроЭВМ.

На рис. 1.3.2 приведена подробная диаграмма выполнения процедуры ввода-запоминания-вывода. Обратите внимание, что команды уже загружены в первые шесть ячеек памяти. Хранимая программа содержит следующую цепочку команд:

1. Ввести данные из порта ввода 1.
2. Запомнить данные в ячейке памяти 200.
3. Переслать данные в порт вывода 10.

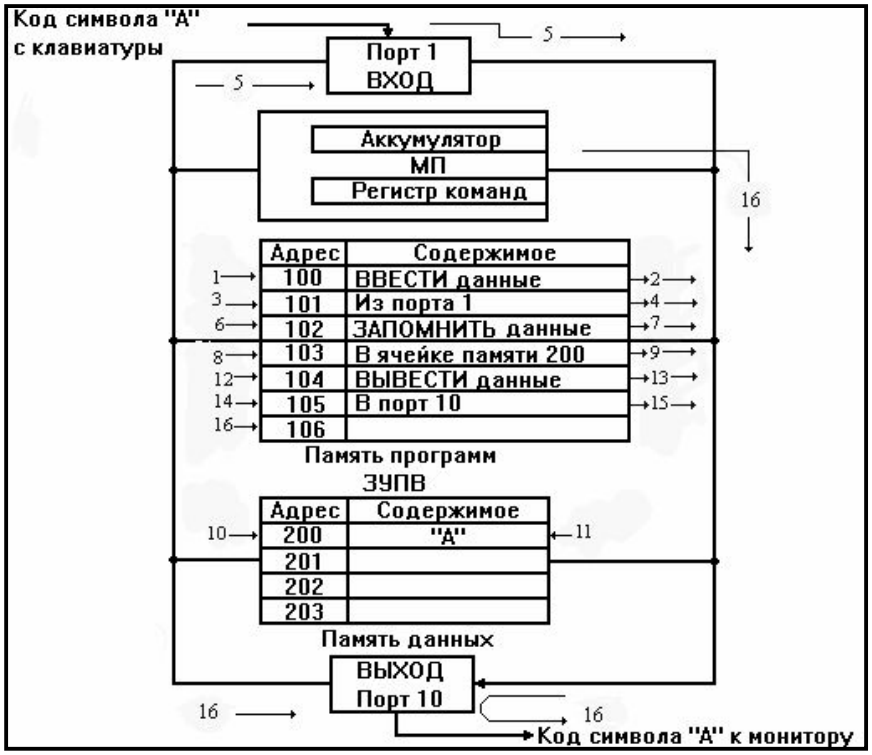


Рис. 1.3.2. Диаграмма выполнения процедуры ввода-запоминания-вывода.

В данной программе всего три команды, хотя на рис. 1.3.2. может показаться, что в памяти программ записано шесть команд. Это связано с тем, что команда обычно разбивается на части. Первая часть команды 1 в приведенной выше программе - команда ввода данных. Во второй части команды 1 указывается, откуда нужно ввести данные (из порта 1). Первая часть команды, предписывающая конкретное действие, называется кодом операции (КОП), а вторая часть - операндом. Код операции и операнд размещаются в отдельных ячейках памяти программ. На рис. 1.3.2. КОП хранится в ячейке 100, а код операнда - в ячейке 101 (порт 1); последний указывает откуда нужно взять информацию.

В МП на рис. 1.3.2. выделены еще два новых блока - регистры: аккумулятор и регистр команд.

Рассмотрим прохождение команд и данных внутри микроЭВМ с помощью занумерованных кружков на диаграмме. Напомним, что микропроцессор - это центральный узел, управляющий перемещением всех данных и выполнением операций.

Итак, при выполнении типичной процедуры ввода-запоминания-вывода в микроЭВМ происходит следующая последовательность действий:

1. МП выдает адрес 100 на шину адреса. По шине управления поступает сигнал, устанавливающий память программ (конкретную микросхему) в режим считывания.
2. ЗУ программ пересылает первую команду ("Ввести данные") по шине данных, и МП получает это закодированное сообщение. Команда помещается в регистр команд. МП декодирует (интерпретирует) полученную команду и определяет, что для команды нужен операнд.
3. МП выдает адрес 101 на ША; ШУ используется для перевода памяти программ в режим считывания.
4. Из памяти программ на ШД пересылается операнд "Из порта 1". Этот операнд находится в программной памяти в ячейке 101. Код операнда (содержащий адрес порта 1) передается по ШД к МП и направляется в регистр команд. МП теперь декодирует полную команду ("Ввести данные из порта 1").
5. МП, используя ША и ШУ, связывающие его с устройством ввода, открывает порт 1. Цифровой код буквы "А" передается в аккумулятор внутри МП и запоминается.Важно отметить, что при обработке каждой программной команды МП действует согласно микропроцедуре выборки-декодирования-исполнения.
6. МП обращается к ячейке 102 по ША. ШУ используется для перевода памяти программ в режим считывания.
7. Код команды "Запомнить данные" подается на ШД и пересылается в МП, где помещается в регистр команд.
8. МП дешифрирует эту команду и определяет, что для нее нужен операнд. МП обращается к ячейке памяти 103 и приводит в активное состояние вход считывания микросхем памяти программ.
9. Из памяти программ на ШД пересылается код сообщения "В ячейке памяти 200". МП воспринимает этот операнд и помещает его в регистр команд. Полная команда "Запомнить данные в ячейке памяти 200" выбрана из памяти программ и декодирована.
10. Теперь начинается процесс выполнения команды. МП пересылает адрес 200 на ША и активизирует вход записи, относящийся к памяти данных.
11. МП направляет хранящуюся в аккумуляторе информацию в память данных. Код буквы "А" передается по ШД и записывается в ячейку 200 этой памяти. Выполнена вторая команда. Процесс запоминания не разрушает содержимого аккумулятора. В нем по-прежнему находится код буквы "А".
12. МП обращается к ячейке памяти 104 для выбора очередной команды и переводит память программ в режим считывания.

13. Код команды вывода данных пересылается по ШД к МП, который помещает ее в регистр команд, дешифрирует и определяет, что нужен операнд.
14. МП выдает адрес 105 на ША и устанавливает память программ в режим считывания.
15. Из памяти программ по ШД к МП поступает код операнда "В порт 10", который далее помещается в регистр команд.
16. МП дешифрирует полную команду "Вывести данные в порт 10". С помощью ША и ШУ, связывающих его с устройством вывода, МП открывает порт 10, пересылает код буквы "А" (все еще находящийся в аккумуляторе) по ШД. Буква "А" выводится через порт 10 на экран дисплея.

В большинстве микропроцессорных систем (МПС) передача информации осуществляется способом, аналогичным рассмотренному выше. Наиболее существенные различия возможны в блоках ввода и вывода информации.

Подчеркнем еще раз, что именно микропроцессор является ядром системы и осуществляет управление всеми операциями. Его работа представляет последовательную реализацию микропроцедур выборки-дешифрации-исполнения. Однако фактическая последовательность операций в МПС определяется командами, записанными в памяти программ.

Таким образом, в МПС микропроцессор выполняет следующие функции:
- выборку команд программы из основной памяти;
- дешифрацию команд;
- выполнение арифметических, логических и других операций, закодированных в командах;
- управление пересылкой информации между регистрами и основной памятью, между устройствами ввода/вывода;
- отработку сигналов от устройств ввода/вывода, в том числе реализацию прерываний с этих устройств;
- управление и координацию работы основных узлов МП.

### 1.4.Устройство управления

Коды операции команд программы, воспринимаемые управляющей частью микропроцессора, расшифрованные и преобразованные в ней, дают информацию о том, какие операции надо выполнить, где в памяти расположены данные, куда надо направить результат и где расположена следующая за выполняемой команда.

Управляющее устройство имеет достаточно средств для того, чтобы после восприятия и интерпретации информации, получаемой в команде, обеспечить переключение (срабатывание) всех требуемых функциональных частей машины, а также для того, чтобы подвести к ним данные и воспринять полученные результаты. Именно срабатывание, т. е. изменение состояния двоичных логических элементов на противоположное, позволяет посредством коммутации вентилей выполнять элементарные логические и арифметические

действия, а также передавать требуемые операнды в функциональные части микроЭВМ.
Устройство управления в строгой последовательности в рамках тактовых и цикловых временных интервалов работы микропроцессора (такт - минимальный рабочий интервал, в течение которого совершается одно элементарное действие; цикл - интервал времени, в течение которого выполняется одна машинная операция) осуществляет: выборку команды; интерпретацию ее с целью анализа формата, служебных признаков и вычисления адреса операнда (операндов); установление номенклатуры и временной последовательности всех функциональных управляющих сигналов; генерацию управляющих импульсов и передачу их на управляющие шины функциональных частей микроЭВМ и вентили между ними; анализ результата операции и изменение своего состояния так, чтобы определить месторасположение (адрес) следующей команды.

### 1.5.Особенности программного и микропрограммного управления

В микропроцессорах используют два метода выработки совокупности функциональных управляющих сигналов: программный и микропрограммный.
Выполнение операций в машине сводится к элементарным преобразованиям информации (передача информации между узлами в блоках, сдвиг информации в узлах, логические поразрядные операции, проверка условий и т.д.) в логических элементах, узлах и блоках под воздействием функциональных управляющих сигналов блоков (устройств) управления. Элементарные преобразования, неразложимые на более простые, выполняются в течение одного такта сигналов синхронизации и называются микрооперациями.
В аппаратных (схемных) устройствах управления каждой операции соответствует свой набор логических схем, вырабатывающих определенные функциональные сигналы для выполнения микроопераций в определенные моменты времени. При этом способе построения устройства управления реализация микроопераций достигается за счет однажды соединенных между собой логических схем, поэтому ЭВМ с аппаратным устройством управления называют ЭВМ с жесткой логикой управления. Это понятие относится к фиксации системы команд в структуре связей ЭВМ и означает практическую невозможность каких-либо изменений в системе команд ЭВМ после ее изготовления.
При микропрограммной реализации устройства управления в состав последнего вводится ЗУ, каждый разряд выходного кода которого определяет появление определенного функционального сигнала управления. Поэтому каждой микрооперации ставится в соответствие свой информационный код - микрокоманда. Набор микрокоманд и последовательность их реализации обеспечивают выполнение любой сложной операции. Набор микроопераций называют микропрограммами. Способ управления операциями путем последовательного считывания и интерпретации микрокоманд из ЗУ (наиболее

часто в виде микропрограммного ЗУ используют быстродействующие программируемые логические матрицы), а также использования кодов микрокоманд для генерации функциональных управляющих сигналов называют микропрограммным, а микроЭВМ с таким способом управления - микропрограммными или с хранимой (гибкой) логикой управления.

К микропрограммам предъявляют требования функциональной полноты и минимальности. Первое требование необходимо для обеспечения возможности разработки микропрограмм любых машинных операций, а второе связано с желанием уменьшить объем используемого оборудования. Учет фактора быстродействия ведет к расширению микропрограмм, поскольку усложнение последних позволяет сократить время выполнения команд программы.

Преобразование информации выполняется в универсальном арифметико-логическом блоке микропроцессора. Он обычно строится на основе комбинационных логических схем.

Для ускорения выполнения определенных операций вводятся дополнительно специальные операционные узлы (например, циклические сдвигатели). Кроме того, в состав микропроцессорного комплекта (МПК) БИС вводятся специализированные оперативные блоки арифметических расширителей.

Операционные возможности микропроцессора можно расширить за счет увеличения числа регистров. Если в регистровом буфере закрепление функций регистров отсутствует, то их можно использовать как для хранения данных, так и для хранения адресов. Подобные регистры микропроцессора называются регистрами общего назначения (РОН). По мере развития технологии реально осуществлено изготовление в микропроцессоре 16, 32 и более регистров.

В целом же, принцип микропрограммного управления (ПМУ) включает следующие позиции:

1) любая операция, реализуемая устройством, является последовательностью элементарных действий - микроопераций;

2) для управления порядком следования микроопераций используются логические условия;

3) процесс выполнения операций в устройстве описывается в форме алгоритма, представляемого в терминах микроопераций и логических условий, называемого микропрограммой;

4) микропрограмма используется как форма представления функции устройства, на основе которой определяются структура и порядок функционирования устройства во времени.

ПМУ обеспечивает гибкость микропроцессорной системы и позволяет осуществлять проблемную ориентацию микро- и миниЭВМ.

## 1.6.Логическая структура микропроцессора

Логические блоки микропроцессора с развитой архитектурой показаны на рис. 2.3.

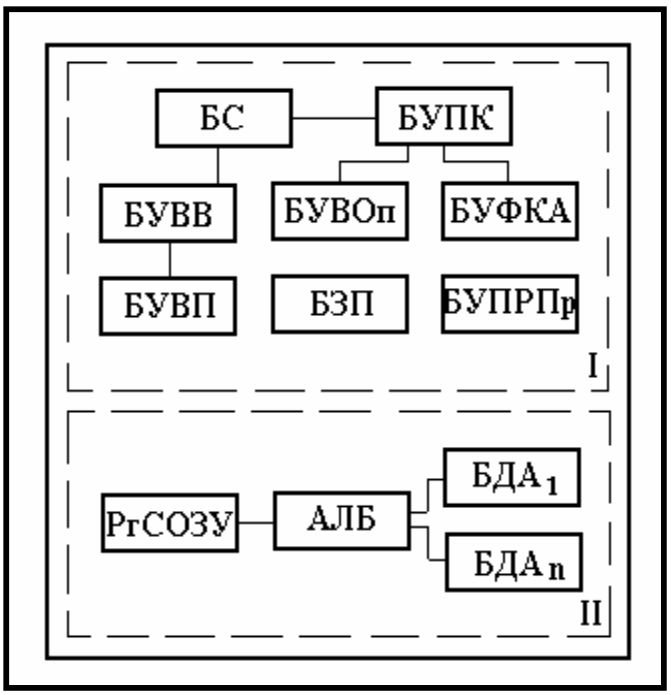


Рис. 2.3. Общая логическая структура микропроцессора:

I - управляющая часть, II - операционная часть; БУПК - блок управления последовательно-стью команд; БУВОп - блок управления выполнением операций; БУФКА - блок управления формированием кодов адресов; БУВП - блок управления виртуальной памятью; БЗП - блок защиты памяти; БУПРПр - блок управления прерыванием работы процессора; БУВВ - блок управления вводом/выводом; РгСОЗУ - регистровое сверхоперативное запоминающее устройство; АЛБ - арифметико-логический блок; БДА - блок дополнительной арифметики; БС - блок синхронизации.

Логическая структура микропроцессора, т. е. конфигурация составляющих микропроцессор логических схем и связей между ними, определяется функциональным назначением. Именно структура задает состав логических блоков микропроцессора и то, как эти блоки должны быть связаны между собой, чтобы полностью отвечать архитектурным требованиям. Срабатывание электронных блоков микропроцессора в определенной последовательности приводит к выполнению заданных архитектурой микропроцессора функций, т. е. к реализации вычислительных алгоритмов. Одни и те же функции можно выполнить в микропроцессорах со структурой, отличающейся набором, количеством и порядком срабатывания логических блоков. Различные структуры микропроцессоров, как правило, обеспечивают их различные возможности, в том числе и различную скорость обработки данных.

При проектировании логической структуры микропроцессоров необходимо рассмотреть:
1) номенклатуру электронных блоков, необходимую и достаточную для реализации архитектурных требований;
2) способы и средства реализации связей между электронными блоками;
3) методы отбора если не оптимальных, то наиболее рациональных вариантов логических структур из возможного числа структур с отличающимся составом блоков и конфигурацией связей между ними.
При проектировании микропроцессора приводятся в соответствие внутренняя сложность кристалла и количество выводов корпуса. Относительный рост числа элементов по мере развития микроэлектронной технологии во много раз превышает относительное увеличение числа выводов корпуса, поэтому проектирование БИС в виде конечного автомата, а не в виде набора схем, реализующих некоторый набор логических переключательных функций и схем памяти, дает возможность получить функционально законченные блоки и устройства ЭВМ.
Использование микропроцессорных комплектов БИС позволяет создать микроЭВМ для широких областей применения вследствие программной адаптации микропроцессора к конкретной области применения: изменяя программу работы микропроцессора, изменяют функции информационно-управляющей системы. Поэтому за счет составления программы работы микропроцессоров в конкретных условиях работы определенной системы можно получить оптимальные характеристики последней.
Если уровень только программной "настройки" микропроцессоров не позволит получить эффективную систему, доступен следующий уровень проектирования - микропрограммный. За счет изменения содержимого ПЗУ или программируемой логической матрицы (ПЛМ) можно "настроиться" на более специфичные черты системы обработки информации. В этом случае частично за счет изменения микропрограмм затрагивается аппаратный уровень системы. Технико-экономические последствия здесь связаны лишь с ограниченным вмешательством в технологию изготовления управляющих блоков микроЭВМ.
Изменение аппаратного уровня информационно-управляющей микропроцессорной системы, включающего в себя функциональные БИС комплекта, одновременно с конкретизацией микропрограммного и программного уровней позволяет наилучшим образом удовлетворить требованиям, предъявляемым к системе.
Решение задач управления в конкретной системе чисто аппаратными средствами (аппаратная логика) дает выигрыш в быстродействии, однако приводит к сложностям при модификации системы. Микропроцессорное решение (программная логика) является более медленным, но более гибким решением, позволяющим развивать и модифицировать систему. Изменение технических требований к информационно-управляющей микропроцессорной системе ведет лишь к необходимости перепрограммирования работы

микропроцессора. Именно это качество обеспечивает высокую логическую гибкость микропроцессоров, определяет возможность их широкого использования, а значит и крупносерийного производства.

### 1.7.Система команд

Проектирование системы команд оказывает влияние на структуру ЭВМ. Оптимальную систему команд иногда определяют как совокупность команд, которая удовлетворяет требованиям проблемно-ориентированных применений таким образом, что избыточность аппаратных и аппаратно-программных средств на реализацию редко используемых команд оказывается минимальной. В различных программах ЭВМ частота появления команд различна; например, по данным фирмы DEC в программах для ЭВМ семейства PDP-11 наиболее часто встречается команда передачи MOV(B), на ее долю приходится приблизительно 32% всех команд в типичных программах. Систему команд следует выбирать таким образом, чтобы затраты на редко используемые команды были минимальными.

При наличии статистических данных можно разработать (выбрать) ЭВМ с эффективной системой команд. Одним из подходов к достижению данной цели является разработка команд длиной в одно слово и кодирование их таким образом, чтобы разряды таких коротких команд использовать оптимально, что позволит сократить время реализации программы и ее длину.

Другим подходом к оптимизации системы команд является использование микроинструкций. В этом случае отдельные биты или группы бит команды используются для кодирования нескольких элементарных операций, которые выполняются в одном командном цикле. Эти элементарные операции не требуют обращения к памяти, а последовательность их реализации определяется аппаратной логикой.

Сокращение времени выполнения программ и емкости памяти достигается за счет увеличения сложности логики управления.

Важной характеристикой команды является ее формат, определяющий структурные элементы команды, каждый из которых интерпретируется определенные образом при ее выполнении. Среди таких элементов (полей) команды выделяют следующие: код операции, определяющий выполняемое действие; адрес ячейки памяти, регистра процессора, внешнего устройства; режим адресации; операнд при использовании непосредственной адресации; код анализируемых признаков для команд условного перехода.

Классификация команд по основным признакам представлена на рис. 1.7.1.Важнейшим структурным элементом формата любой команды является код операции (КОП), определяющей действие, которое должно быть выполнено. Большое число КОП в процессоре очень важно, так как аппаратная реализация команд экономит память и время. Но при выборе ЭВМ необходимо концентрировать внимание на полноте операций с конкретными типами данных, а не только на числе команд, на доступных режимах адресации. Число

бит, отводимое под КОП, является функцией полного набора реализуемых команд.

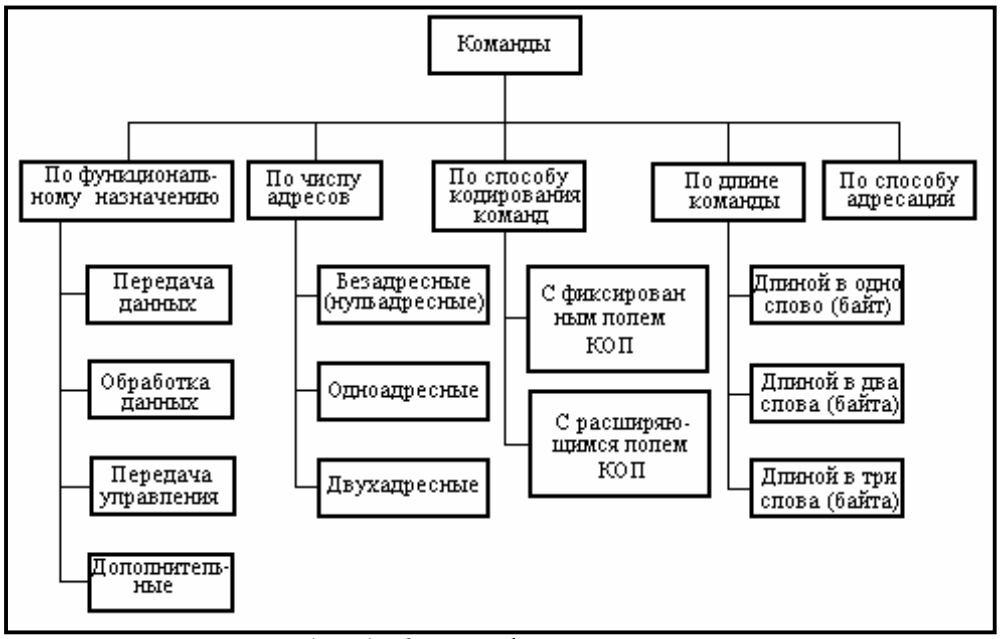


Рис. 1.7.1.Классификация команд.

При использовании фиксированного числа бит под КОП для кодирования всех m команд необходимо в поле КОП выделить двоичных разрядов. Однако, учитывая ограниченную длину слова мини- и микроЭВМ, различное функциональное назначение команд, источники и приемники результатов операций, а также то, что не все команды содержат адресную часть для обращения к памяти и периферийным устройствам, в малых ЭВМ для кодирования команд широко используется принцип кодирования с переменным числом бит под поле КОП для различных групп команд.

В некоторых командах необходим только один операнд и они называются однооперандными (или одноадресными) командами в отличие от двухоперандных (или двухадресных), в которых требуются два операнда. При наличии двух операндов командой обычно изменяется только один из них. Так как информация берется только из одной ячейки, эту ячейку называются источником; ячейка, содержимое которой изменяется, называется приемником.

Ниже приведен формат двухадресной (двухоперандной) команды процессоров СМ.

| а | 15 11 | 10 6 | 5 0 | б | 15 11 | 10 0 |
|---|---|---|---|---|---|---|
| | КОП | Источник | Приемник | | КОП | Приемник |

Формат команд процессоров СМ:

а) двухадресная команда;

б) одноадресная команда.

Примеры кодирования двухадресных команд в процессорах СМ

| КОП | Мнемоника команды | Комментарий |
|---|---|---|
| 0001<br>0010<br>0110<br>1110 | MOV<br>CMP<br>ADD<br>SUB | Передача данных<br>Сравнение<br>Сложение<br>Вычитание |
| 0000<br>1000 | -<br>- | Кодирование группы<br>одноадресных команд |

Четырехбитный КОП (биты 15-12) кодирует ряд двухоперандных операций, приведенных в таблице 1. Биты (11-6) и (5-0) для команд данного типа определяют адреса источника и приемника данных. Как видно из таблицы, комбинации 0000 и 1000 поля КОП определяют группы одноадресных команд (рис 1,б). КОП 1 (биты 15-12), соответствующий кодам 0000 и 1000, определяет группу одноадресных команд, а КОП 2 (биты 11-6) кодирует конкретную операцию команд данной группы. Таким образом, команды, использующие один операнд, кодируются 10-битным КОП (биты 15-6).

Наиболее гибкая команда требует до четырех операндов. Например, команда сложения может указывать адреса слагаемых, адрес результата и адрес следующей команды. Если для задания адреса требуется 16 бит, то четырехоперандная команда займет 8 байт памяти, не учитывая код операции. Следовательно, получится медленнодействующая ЭВМ с огромной памятью. Поэтому в большинстве микроЭВМ любой команде требуется не более двух операндов. Это достигается следующими приемами:

1. Адрес следующей команды указывается только в командах переходов; в остальных случаях очередная команда выбирается из ячеек памяти, следующих за выполненной командой.
2. Использование ячейки, в которой находится один из операндов, для запоминания результата (например, сумма запоминается в ячейки первого операнда).

Локализацию и обращение к операндам обеспечивают режимы адресации. При введении нескольких режимов адресации необходимо отвести в команде биты, указывающие режимы адресации для каждого операнда. Если предусмотрено восемь режимов адресации, то для задания каждого из них нужно три бита.

Почти во всех форматах команд первые биты отводятся для кода операции, но далее форматы команд разных ЭВМ сильно отличаются друг от друга. Остальные биты должны определять операнды или их адреса, и поэтому они используются для комбинации режимов, адресов регистров, адресов памяти, относительных адресов и непосредственных операндов. Обычно длина команды варьируется от 1 до 3 и даже 6 байт.

По форматам команд можно судить о возможностях ЭВМ.

## 1.8.Режимы адресации

Для взаимодействия с различными модулями в ЭВМ должны быть средства идентификации ячеек внешней памяти, ячеек внутренней памяти, регистров МП и регистров устройств ввода/вывода. Поэтому каждой из запоминающих ячеек присваивается адрес, т.е. однозначная комбинация бит. Количество бит определяет число идентифицируемых ячеек. Обычно ЭВМ имеет различные адресные пространства памяти и регистров МП, а иногда - отдельные адресные пространства регистров устройств ввода/вывода и внутренней памяти. Кроме того, память хранит как данные, так и команды. Поэтому для ЭВМ разработано множество способов обращения к памяти, называемых режимами адресации.

Режим адресации памяти - это процедура или схема преобразования адресной информации об операнде в его исполнительный адрес.

Все способы адресации памяти можно разделить на:

1) прямой, когда исполнительный адрес берется непосредственно из команды или вычисляется с использованием значения, указанного в команде, и содержимого какого-либо регистра (прямая адресация, регистровая, базовая, индексная и т.д.);

2) косвенный, который предполагает, что в команде содержится значение косвенного адреса, т.е. адреса ячейки памяти, в которой находится окончательный исполнительный адрес (косвенная адресация).

В каждой микроЭВМ реализованы только некоторые режимы адресации, использование которых, как правило, определяется архитектурой МП.

## 1.9.Типы архитектур

Существует несколько подходов к классификации микропроцессоров по типу архитектуры. Так, выделяют МП с CISC (Complete Instruction Set Computer) архитектурой, характеризуемой полным набором команд, и RISC (Reduce Instruction Set Computer) архитектурой, которая определяет систему с сокращенным набором команд одинакового формата, выполняемых за один такт МП.

Определяя в качестве основной характеристики МП разрядность, выделяют следующие типы МП архитектуры:

-с фиксированной разрядностью и списком команд (однокристальные);
-с наращиваемой разрядностью (секционные) и микропрограммным управлением.

Анализируя адресные пространства программ и данных, определяют МП с архитектурой фон Неймана (память программ и память данных находятся в едином пространстве и нет никаких признаков, указывающих на тип информации в ячейке памяти) и МП с архитектурой Гарвардской лаборатории (память программ и память данных разделены, имеют свои адресные пространства и способы доступа к ним).

Мы рассмотрим более подробно основные типы архитектурных решений, выделяя связь со способами адресации памяти.

1. Регистровая архитектура определяется наличием достаточно большого регистрового файла внутри МП. Команды получают возможность обратиться к операндам, расположенным в одной из двух запоминающих сред: оперативной памяти или регистрах. Размер регистра обычно фиксирован и совпадает с размером слова, физически реализованного в оперативной памяти. К любому регистру можно обратиться непосредственно, поскольку регистры представлены в виде массива запоминающих элементов - регистрового файла. Типичным является выполнение арифметических операций только в регистре, при этом команда содержит два операнда (оба операнда в регистре или один операнд в регистре, а второй в оперативной памяти).

Предельный вариант - архитектура с адресацией посредством аккумуляторов (меньший набор команд).

2. Стековая архитектура дает возможность создать поле памяти с упорядоченной последовательностью записи и выборки информации.В общем случае команды неявно адресуются к элементу стека, расположенному на его вершине, или к двум верхним элементам стека.

3. Архитектура МП, ориентированная на оперативную память (типа "память-память"), обеспечивает высокую скорость работы и большую информационную емкость рабочих регистров и стека при их организации в оперативной памяти. Архитектура этого типа не предполагает явного определения аккумулятора, регистров общего назначения или стека; все операнды команд адресуются к области основной памяти.

С точки зрения важности для пользователя-программиста под архитектурой в общем случае понимают совокупность следующих компонентов и характеристик:

- разрядности адресов и данных;
- состава, имен и назначения программно-доступных регистров;
- форматов и системы команд;
- режимов адресации памяти;
- способов машинного представления данных разного типа;
- структуры адресного пространства;
- способа адресации внешних устройств и средств выполнения операций ввода/вывода;
- классов прерываний, особенностей инициирования и обработки прерываний.

## 1.10.Организация ввода/вывода в микропроцессорной системе.

Вводом/выводом (ВВ) называется передача данных между ядром ЭВМ, включающим в себя микропроцессор и основную память, и внешними устройствами (ВУ). Это единственное средство взаимодействия ЭВМ с "внешним миром", и архитектура ВВ (режимы работы, форматы команд, особенности прерываний, скорость обмена и др.) непосредственно влияет на

эффективность всей системы. За время эволюции ЭВМ подсистема ВВ претерпела наибольшие изменения благодаря расширению сферы применения ЭВМ и появлению новых внешних устройств. Особенно важную роль средства ВВ играют в управляющих ЭВМ. Разработка аппаратных средств и программного обеспечения ВВ является наиболее сложным этапом проектирования новых систем на базе ЭВМ, а возможности ВВ серийных машин представляют собой один из важных параметров, определяющих выбор машины для конкретного применения.

**Программная модель внешнего устройства**

Подключение внешних устройств к системной шине осуществляется посредством электронных схем, называемых контроллерами ВВ (интерфейсами ВВ). Они согласуют уровни электрических сигналов, а также преобразуют машинные данные в формат, необходимый устройству, и наоборот. Обычно контроллеры ВВ конструктивно оформляются вместе с процессором в виде интерфейсных плат.

В процессе ввода/вывода передается информация двух видов: управляющие данные (слова) и собственно данные, или данные-сообщения. Управляющие данные от процессора, называемые также командными словами или приказами, инициируют действия, не связанные непосредственно с передачей данных, например запуск устройства, запрещение прерываний и т.п. Управляющие данные от внешних устройств называются словами состояния; они содержат информацию об определенных признаках, например о готовности устройства к передаче данных, о наличии ошибок при обмене и т.п. Состояние обычно представляется в декодированной форме - один бит для каждого признака.

Регистр, содержащий группу бит, к которой процессор обращается в операциях ВВ, образует порт ВВ. Таким образом, наиболее общая программная модель внешнего устройства, которое может выполнять ввод и вывод, содержит четыре регистра ВВ: регистр выходных данных (выходной порт), регистр входных данных (входной порт), регистр управления и регистр состояния (рис. 1.10.1). Каждый из этих регистров должен иметь однозначный адрес, который идентифицируется дешифратором адреса. В зависимости от особенностей устройства общая модель конкретизируется, например, отдельные регистры состояния и управления объединяются в один регистр, в устройстве ввода (вывода) имеется только регистр входных (выходных) данных, для ввода и вывода используется двунаправленный порт.

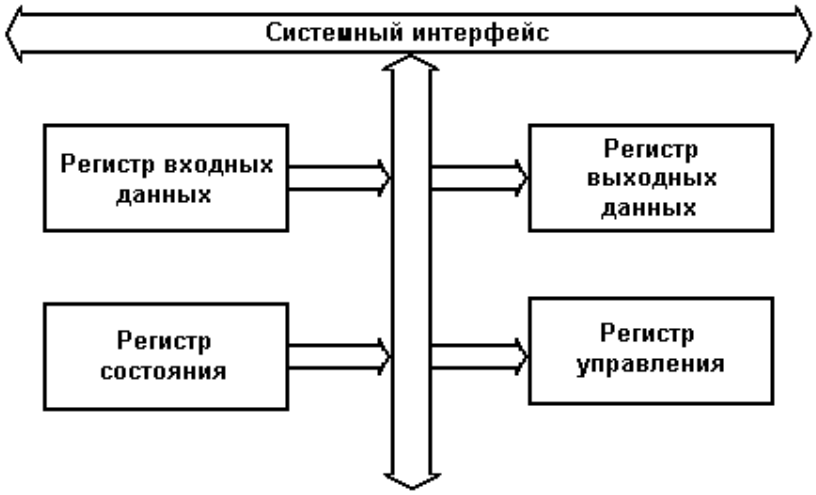


Рис. 1.10.1.Программная модель внешнего устройства

Непосредственные действия, связанные с вводом/выводом, реализуются одним из двух способов, различающихся адресацией регистров ВВ.

Интерфейс с изолированными шинами характеризуется раздельной адресацией памяти и внешних устройств при обмене информацией. Изолированный ВВ предполагает наличие специальных команд ввода/вывода, общий формат которых показан на рис. 1.10.2. При выполнении команды ввода IN содержимое адресуемого входного регистра PORT передается во внутренний регистр REG процессора, а при выполнении команды OUT содержимое регистра REG передается в выходной порт PORT. В процессоре могут быть и другие команды, относящиеся к ВВ и связанные с проверкой и модификацией содержимого регистра управления и состояния.

| КОП(IN) | REG | PORT |
|---|---|---|

| КОП(OUT) | REG | PORT |
|---|---|---|

Рис. 1.10.2. Команды ввода/вывода (общий формат)

Нетрудно заметить, что в этом способе адресное пространство портов ввода и вывода изолировано от адресного пространства памяти, т.е. в ЭВМ один и тот же адрес могут иметь порт ВВ и ячейка памяти. Разделение адресных пространств осуществляется с помощью управляющих сигналов, относящихся к системам ВВ и памяти (MEMRD# - считывание данных из памяти, MEMWR# - запись данных в память, IORD# - чтение порта ВВ, IOWR# - запись в порт ВВ) (# - активный низкий уровень сигналов).

В ЭВМ, рассчитанной на изолированный ВВ, нетрудно перейти к ВВ, отображенному на память. Если, например, адресное пространство памяти составляет 64 Кбайт, а для программного обеспечения достаточно 32 Кбайт, то область адресов от 0 до 32 К-1 используется для памяти, от 32 К до 64 К-1 - для ввода/вывода. При этом признаком, дифференцирующим обращения к памяти и портам ВВ, может быть старший бит адреса.

Таким образом, интерфейс с общими шинами (ввод/вывод с отображением на память) имеет организацию, при которой часть общего адресного пространства отводится для внешних устройств, регистры которых адресуются так же, как и ячейки памяти. В этом случае для адресации портов ВВ используются полные адресные сигналы: READ - чтение, WRITE - запись.
В операционных системах ЭВМ имеется набор подпрограмм (драйверов ВВ), управляющих операциями ВВ стандартных внешних устройств. Благодаря им пользователь может не знать многих особенностей ВУ и интерфейсов ВВ, а применять четкие программные протоколы.

### 1.11. Форматы передачи данных

Рассмотрим некоторые общие вопросы, связанные с обменом данными между ВУ и микроЭВМ. Существуют два способа передачи слов информации по линиям данных: параллельный, когда одновременно пересылаются все биты слова, и последовательный, когда биты слова пересылаются поочередно, начиная, например, с его младшего разряда.
Так как между отдельными проводниками шины для параллельной передачи данных существует электрическая емкость, то при изменении сигнала, передаваемого по одному из проводников, возникает помеха (короткий выброс напряжения) на других проводниках. С увеличением длины шины (увеличением емкости проводников) помехи возрастают и могут восприниматься приемником как сигналы. Поэтому рабочее расстояние для шины параллельной передачи данных ограничивается длиной 1-2 м, и только за счет существенного удорожания шины или снижения скорости передачи длину шины можно увеличить до 10-20 м.
Указанное обстоятельство и желание использовать для дистанционной передачи информации телеграфные и телефонные линии обусловили широкое распространение способа последовательного обмена данными между ВУ и микроЭВМ и между несколькими микроЭВМ. Возможны два режима последовательной передачи данных: синхронный и асинхронный.
При синхронной последовательной передаче каждый передаваемый бит данных сопровождается импульсом синхронизации, информирующим приемник о наличии на линии информационного бита. Следовательно, между передатчиком и приемником должны быть протянуты минимум три провода: два для передачи импульсов синхронизации и бит данных, а также общий заземленный проводник. Если же передатчик (например, микроЭВМ) и приемник (например, дисплей) разнесены на несколько метров, то каждый из сигналов (информационный и синхронизирующий) придется посылать либо по экранированному (телевизионному) кабелю, либо с помощью витой пары проводов, один из которых заземлен или передает сигнал, инверсный основному.
Синхронная последовательная передача начинается с пересылки в приемник одного или двух символов синхронизации (не путать с импульсами

синхронизации). Получив такой символ (символы), приемник начинает прием данных и их преобразование в параллельный формат. Естественно, что при такой организации синхронной последовательной передачи она целесообразна лишь для пересылки массивов слов, а не отдельных символов. Это обстоятельство, а также необходимость использования для обмена сравнительно дорогих (четырехпроводных или кабельных) линий связи помешало широкому распространению синхронной последовательности передачи данных.

Асинхронная последовательная передача данных означает, что у передатчика и приемника нет общего генератора синхроимпульсов и что синхронизирующий сигнал не посылается вместе с данными. Как же в таком случае приемник будет узнавать о моментах начала и завершения передачи бит данных. Опишем простую процедуру, которую можно использовать, если передатчик и приемник асинхронной последовательной передачи данных согласованы по формату и скорости передачи.

Стандартный формат асинхронной последовательной передачи данных, используемый в ЭВМ и ВУ, содержит n пересылаемых бит информации (при пересылке символов n равно 7 или 8 битам) и 3-4 дополнительных бита: стартовый бит, бит контроля четности (или нечетности) и 1 или 2 стоповых бита (рис. 1.11.1,а). Бит четности (или нечетности) может отсутствовать. Когда передатчик бездействует (данные не посылаются на линию), на линии сохраняется уровень сигнала, соответствующий логической 1.


Рис. 1.11.1.Формат асинхронной последовательной передачи данных

Передатчик может начать пересылку символа в любой момент времени посредством генерирования стартового бита, т. е. перевода линии в состояние логического 0 на время, точно равное времени передачи бита. Затем происходит передача битов символа, начиная с младшего значащего бита, за которым следует дополнительный бит контроля по четности или нечетности. Далее с помощью стопового бита линия переводится в состояние логической 1 (рис.

1.11.1,б). При единичном бите контроля стоповый бит не изменяет состояния сигнала на линии. Состояние логической 1 должно поддерживаться в течение промежутка времени, равного 1 или 2 временам передачи бита.
Промежуток времени от начала стартового бита до конца стопового бита (стоповых бит) называется кадром. Сразу после стоповых бит передатчик может посылать новый стартовый бит, если имеется другой символ для передачи; в противном случае уровень логической 1 может сохраняться на протяжение всего времени, пока бездействует передатчик. Новый стартовый бит может быть послан в любой момент времени после окончания стопового бита, например, через промежуток времени, равный 0.43 или 1.5 времени передачи бита.
В линиях последовательной передачи данных передатчик и приемник должны быть согласованы по всем параметрам формата, изображенного на рис. 8, включая номинальное время передачи бита. Для этого в приемнике устанавливается генератор синхроимпульсов, частота которого должна совпадать с частотой аналогичного генератора передатчика. Кроме того, для обеспечения оптимальной защищенности сигнала от искажения, шумов и разброса частоты синхроимпульсов приемник должен считывать принимаемый бит в середине его длительности. Рассмотрим работу приемника с того момента, когда он закончил прием символа данных и перешел в режим обнаружения стартового бита следующего слова.
Если линия перешла в состояние логического нуля и находится в этом состоянии в течение времени, не меньшего половины временного интервала передачи бита, то приемник переводится в режим считывания бит информации. В противном случае приемник остается в режиме обнаружения, так как вероятнее всего это был не стартовый бит, а шумовая помеха. В новом режиме приемник вырабатывает сигналы считывания через интервалы, равные времени передачи бита, т. е. выполняет считывание и сохранение принимаемых бит примерно на середине их передачи. Аналогичным образом будут считаны бит контроля четности и сигнал логической единицы (стоповый бит). Если оказалось, что на месте стопового бита обнаружен сигнал логического нуля, то произошла "Ошибка кадра" и символ принят неправильно. Иначе проверяется, четно ли общее число единиц в информационных битах и бите контроля, и если оно четно, производится запись принятого символа в буфер приемника.
Передний фронт стартового бита сигнализирует о начале поступления передаваемой информации, а момент его появления служит точкой отсчета времени для считывания бит данных. Стоповый бит предоставляет время для записи принятого символа в буфер приемника и обеспечивает возможность выявления ошибки кадра. Наиболее часто ошибки кадра появляются тогда, когда приемник ошибочно синхронизирован с битом 0, который в действительности не является стартовым битом. Если передатчик бездействует (посылает сигнал логической единицы) в течение одного кадра или более, то всегда можно восстановить правильную синхронизацию. Хуже обстоит дело

при рассинхронизации генераторов передатчика и приемника, когда временной интервал между сигналами считывания принимаемых битов будет меньше или больше времени передачи бита.
Например, если при считывании битов посылки, показанной на рис. 1.11.1,б, временной интервал между сигналами считывания станет на 6% меньше, чем время передачи бита, то восьмой и девятый сигналы считывания будут выработаны тогда, когда на линии находится бит контроля четности (рис. 3.4). Следовательно, не будет обнаружен стоповый бит и будет зафиксирована ошибка кадра, несмотря на правильность принятой информации. Однако при 18%-й рассинхронизации генераторов, когда вместо кода (01110001) приемник зафиксирует код (11100001), никаких ошибок не будет обнаружено - четность соблюдена и стоповый (девятый по порядку) бит равен 1 (см. рис. 1.11.2.).

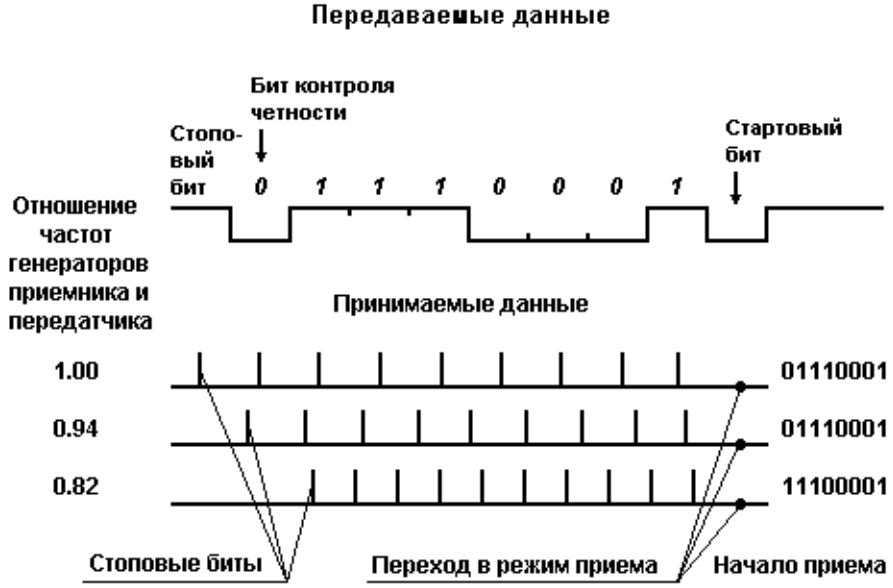


Рис. 1.11.2. Ошибка из-за рассинхронизации генераторов передатчика и приемника

### 1.11.1.Параллельная передача данных

Параллельная передача данных между контроллером и ВУ является по своей организации наиболее простым способом обмена. Для организации параллельной передачи данных помимо шины данных, количество линий в которой равно числу одновременно передаваемых битов данных, используется минимальное количество управляющих сигналов.
В простом контроллере ВУ, обеспечивающем побайтную передачу данных на внешнее устройство (рис. 1.11.1.1.), в шине связи с ВУ используются всего два управляющих сигнала: "Выходные данные готовы" и "Данные приняты".

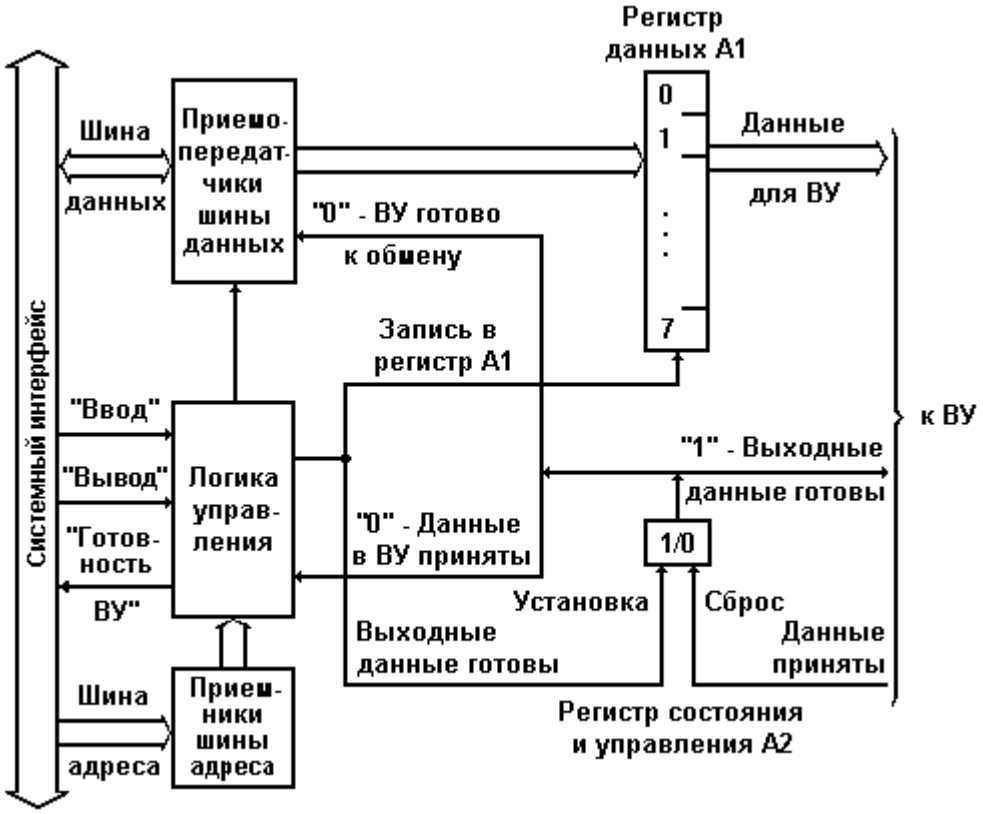


Рис. 1.11.1.1. Простой параллельный контроллер вывода.

Для формирования управляющего сигнала "Выходные данные готовы" и приема из ВУ управляющего сигнала "Данные приняты" в контроллере используется одноразрядный адресуемый регистр состояния и управления A2 (обычно используются раздельные регистр состояния и регистр управления). Одновременно с записью очередного байта данных с шины данных системного интерфейса в адресуемый регистр данных контроллера (порт вывода A1) в регистр состояния и управления записывается логическая единица. Тем самым формируется управляющий сигнал "Выходные данные готовы" в шине связи с ВУ.

ВУ, приняв байт данных, управляющим сигналом "Данные приняты" обнуляет регистр состояния контроллера. При этом формируются управляющий сигнал системного интерфейса "Готовность ВУ" и признак готовности ВУ к обмену, передаваемый в процессор по одной из линий шины данных системного интерфейса посредством стандартной операции ввода при реализации программы асинхронного обмена.

Логика управления контроллера обеспечивает селекцию адресов регистров контроллера, прием управляющих сигналов системного интерфейса и формирование на их основе внутренних управляющих сигналов контроллера, формирование управляющего сигнала системного интерфейса "Готовность ВУ". Для сопряжения регистров контроллера с шинами адреса и данных системного интерфейса в контроллере используются соответственно приемники шины адреса и приемопередатчики шины данных.

Рассмотрим на примере, каким образом контроллер ВУ обеспечивает параллельную передачу данных в ВУ под управлением программы асинхронного обмена. Алгоритм асинхронного обмена в данном случае передачи прост.

1. Процессор микроЭВМ проверяет готовность ВУ к приему данных.
2. Если ВУ готово к приему данных (в данном случае это логический 0 в нулевом разряде регистра А2), то данные передаются с шины данных системного интерфейса в регистр данных А1 контроллера и далее в ВУ. Иначе повторяется п. 1.

Пример 1. Фрагмент программы передачи байта данных в асинхронном режиме с использованием параллельного контроллера ВУ (рис. 1.11.1.1.). Для написания программы асинхронной передачи воспользуемся командами процессора 8086.

| | | |
|---|---|---|
| MOV | DX, A2 | номер порта А2 помещаем в DX |
| m1:IN | AL, DX | чтение байта из порта А2 |
| TEST | AL, 1 | проверка нулевого состояния регистра А2 |
| JNS | ml | переход на метку ml если разряд не нулевой |
| MOV | AL, 64 | выводимый байт данных помещается в AL |
| MOV | DX, A1 | номер порта А1 записываем в DX |
| OUT | DX, AL | содержимое регистра АХ передаем в порт А1 |

Команда во второй строке приводит к следующим действиям. При ее выполнении процессор по шине адреса передает в контроллер адрес А2, сопровождая его сигналом "Ввод" (IORD#; здесь и далее в скобках указаны сигналы на шине ISA). Логика управления контроллера, реагируя на эти сигналы, обеспечивает передачу в процессор содержимого регистра состояния А2 по шине данных системного интерфейса.

Команда в третьей строке приводит к следующим действиям. Процессор проверяет значение соответствующего разряда принятых данных. Нуль в этом разряде указывает на неготовность ВУ к приему данных и, следовательно, на необходимость возврата к проверке содержимого А2, т. е. процессор, выполняя три первые команды, ожидает готовности ВУ к приему данных. Единица в этом разряде подтверждает готовность ВУ и, следовательно, возможность передачи байта данных.

В седьмой строке осуществляется пересылка данных из регистра АХ процессора в регистр данных контроллера А1. Процессор по шине адреса передает в контроллер адрес А1, а по шине данных - байт данных, сопровождая их сигналом "Вывод" (IOWR#). Логика управления контроллера обеспечивает запись данных с шины данных в регистр данных А1 и устанавливает в ноль бит готовности регистра состояния А2, формируя тем самым управляющий сигнал для ВУ "Выходные данные готовы". ВУ принимает байт данных и управляющим сигналом "Данные приняты" устанавливает в единицу регистр

состояния А2. (Далее контроллер ВУ по этому сигналу может сформировать и передать в процессор сигнал "Готовность ВУ", который в данном случае извещает процессор о приеме данных внешним устройством и разрешает процессору снять сигнал "Вывод" и тем самым завершить цикл вывода данных в команде пересылки, однако в IBM-совместимых персональных компьютерах с шиной ISA сигнал "Готовность ВУ" не формируется, а имеется сигнал IO CH RDY#, позволяющий продлить цикл обмена, если устройство недостаточно быстрое. В данном случае нет необходимости в сигнале "Готовность ВУ", т.к. шина ISA является синхронной и, следовательно, все операции выполняются по тактовым импульсам.)

Блок-схема простого контроллера ВУ, обеспечивающего побайтный прием данных из ВУ, приведена на рис. 1.11.1.2. В этом контроллере при взаимодействии с внешним устройством также используются два управляющих сигнала: "Данные от ВУ готовы" и "Данные приняты".

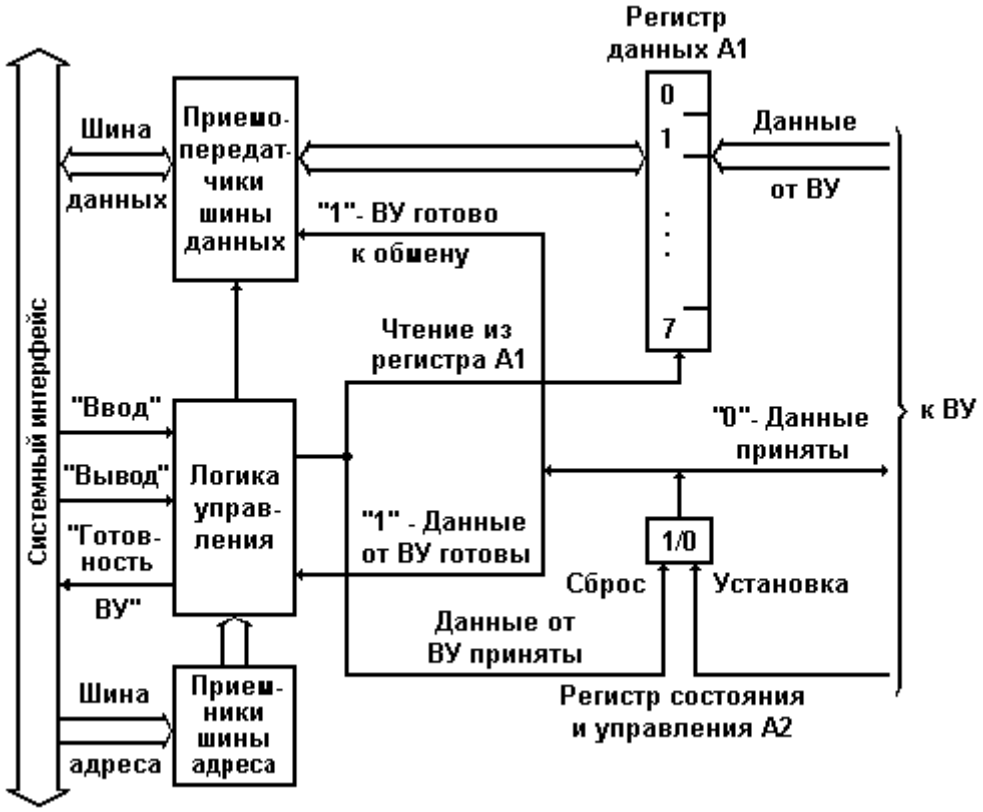


Рис. 1.11.1.2. Простой параллельный контроллер ввода

Для формирования управляющего сигнала "Данные приняты" и приема из ВУ управляющего сигнала " Данные от ВУ готовы" используется одноразрядный адресуемый регистр состояния и управления А2.

Внешнее устройство записывает в регистр данных контроллера А1 очередной байт данных и управляющим сигналом "Данные от ВУ готовы" устанавливает в единицу регистр состояния и управления А2.

При этом формируются: управляющий сигнал системного интерфейса "Готовность ВУ"; признак готовности ВУ к обмену, передаваемый в процессор

по одной из линий шины данных системного интерфейса посредством операции ввода при реализации программы асинхронного обмена.
Тем самым контроллер извещает процессор о готовности данных в регистре A1. Процессор, выполняя программу асинхронного обмена, читает байт данных из регистра данных контроллера и обнуляет регистр состояния и управления A2. При этом формируется управляющий сигнал "Данные приняты" в шине связи с внешним устройством.
Логика управления контроллера и приемопередатчики шин системного интерфейса выполняют те же функции, что и в контроллере вывода (см. рис. 1.11.1.1.),
Рассмотрим работу параллельного интерфейса ввода при реализации программы асинхронного обмена. Алгоритм асинхронного ввода так же прост, как и асинхронного вывода.
1. Процессор проверяет наличие данных в регистре данных контроллера A1.
2. Если данные готовы (логическая 1 в регистре A2), то они передаются из регистра данных A1 на шину данных системного интерфейса и далее в регистр процессора или ячейку памяти микрокомпьютера. Иначе повторяется п. 1.
Пример 2. Фрагмент программы приема байта данных в асинхронном режиме с использованием параллельного интерфейса (контроллер ВУ, рис. 1.11.1.2.):

| | | |
|---|---|---|
| MOV | DX, A2 | номер порта A2 помещаем в DX |
| m1:IN | AL, DX | чтение байта из порта A2 |
| TEST | AL, 1 | проверка нулевого разряда состояния регистра A2 |
| JZ | ml | переход на метку ml если разряд не нулевой |
| MOV | DX, A1 | номер порта A1 записываем в DX |
| IN | AL, DX | содержимое регистра A1 передаем в регистр AL |

В третьей строке выполняется проверка содержимого регистра A2, т.е. признака наличия данных в регистре данных A1. Команда выполняется точно так же, как и в примере 1. Единица в нулевом разряде (содержимое регистра A2) подтверждает, что данные от ВУ записаны в регистр данных контроллера и необходимо переслать их на шину данных. Нуль в знаковом разряде указывает на неготовность данных от ВУ и, следовательно, на необходимость вернуться к проверке готовности.
IN AL, DX - пересылка данных из регистра данных контроллера A1 в регистр процессора AL. Процессор передает в контроллер по шине адреса системного интерфейса адрес A1, сопровождая его сигналом "Ввод". Логика управления контроллера в ответ на сигнал "Ввод" (IORD#) обеспечивает передачу байта данных из регистра данных A1 на шину данных и, в общем случае, но не в IBM-совместимом персональном компьютере с шиной ISA, сопровождает его сигналом "Готовность ВУ", который подтверждает наличие данных от ВУ на шине данных и по которому процессор считывает байт с шины данных и помещает его в указанный регистр. (В IBM-совместимом персональном компьютере с шиной ISA процессор считывает байт с шины данных по

истечении определенного времени после установки сигнала IORD#.) Затем логика управления обнуляет регистр состояния и управления A2, формируя тем самым управляющий сигнал для внешнего устройства "Данные приняты". Таким образом, завершается цикл ввода данных.

Как видно из рассмотренных примеров, для приема или передачи одного байта данных процессору необходимо выполнить всего несколько команд, время выполнения которых и определяет максимально достижимую скорость обмена данными при параллельной передаче. Таким образом, при параллельной передаче обеспечивается довольно высокая скорость обмена данными, которая ограничивается только быстродействием ВУ.

### 1.11.2. Последовательная передача данных

Использование последовательных линий связи для обмена данными с внешними устройствами возлагает на контроллеры ВУ дополнительные по сравнению с контроллерами для параллельного обмена функции. Во-первых, возникает необходимость преобразования формата данных: из параллельного формата, в котором они поступают в контроллер ВУ из системного интерфейса микроЭВМ, в последовательный при передаче в ВУ и из последовательного в параллельный при приеме данных из ВУ. Во-вторых, требуется реализовать соответствующий режиму работы внешнего устройства способ обмена данными: синхронный или асинхронный.

**Синхронный последовательный интерфейс**

Простой контроллер для синхронной передачи данных в ВУ по последовательной линии связи (последовательный интерфейс) представлен на рис. 1.11.2.1.

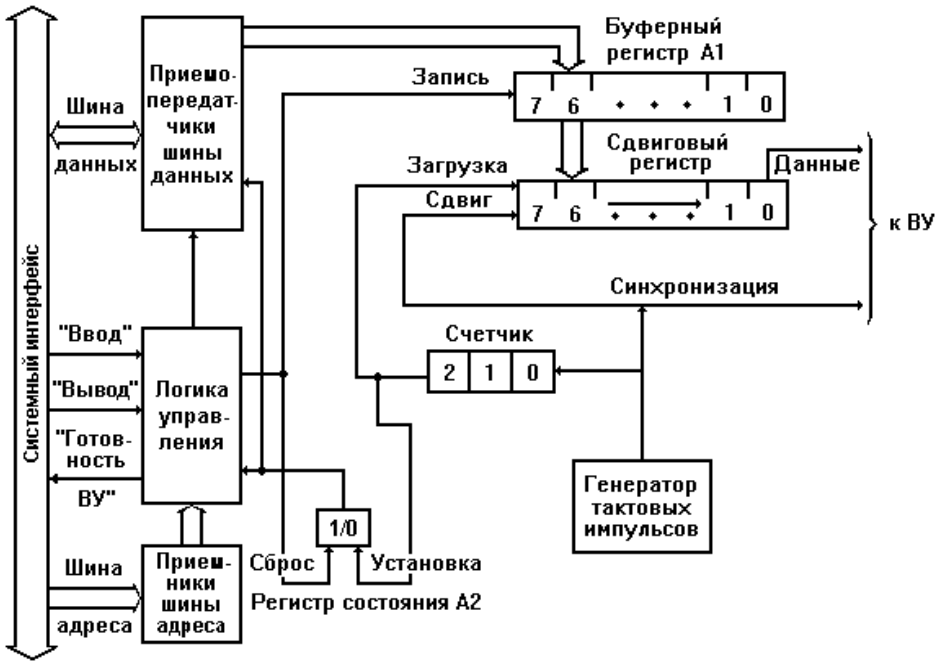


Рис. 1.11.2.1.Контроллер последовательной синхронной передачи

Восьмиразрядный адресуемый буферный регистр контроллера А1 служит для временного хранения байта данных до его загрузки в сдвиговый регистр. Запись байта данных в буферный регистр с шины данных системного интерфейса производится так же, как и в параллельном интерфейсе, только при наличии единицы в одноразрядном адресуемом регистре состояния контроллера А2. Единица в регистре состояния указывает на готовность контроллера принять очередной байт в буферный регистр. Содержимое регистра А2 передается в процессор по одной из линий шины данных системного интерфейса и используется для формирования управляющего сигнала системного интерфейса "Готовность ВУ". При записи очередного байта в буферный регистр А1 обнуляется регистр состояния А2.

Программа записи байта данных в буферный регистр аналогична программе из примера 1. за исключением команды перехода: вместо команды JNZ m1 (переход, если не ноль) необходимо использовать команду JZ m1 (переход, если ноль).

Преобразование данных из параллельного формата, в котором они поступили в буферный регистр контроллера из системного интерфейса, в последовательный и передача их на линию связи производятся в сдвиговом регистре с помощью генератора тактовых импульсов и двоичного трехразрядного счетчика импульсов следующим образом.

Последовательная линия связи контроллера с ВУ подключается к выходу младшего разряда сдвигового регистра. По очередному тактовому импульсу содержимое сдвигового регистра сдвигается на один разряд вправо и в линию связи "Данные" выдается значение очередного разряда. Одновременно со сдвигом в ВУ передается по отдельной линии "Синхронизация" тактовый импульс. Таким образом, каждый передаваемый по линии "Данные" бит информации сопровождается синхронизирующим сигналом по линии "Синхронизация", что обеспечивает его однозначное восприятие на приемном конце последовательной линии связи.

Количество переданных в линию тактовых сигналов, а следовательно, и переданных бит информации подсчитывается счетчиком тактовых импульсов. Как только содержимое счетчика становится равным 7, т. е. в линию переданы 8 бит (1 байт) информации, формируется управляющий сигнал "Загрузка", обеспечивающий запись в сдвиговый регистр очередного байта из буферного регистра. Этим же управляющим сигналом устанавливается в "1" регистр состояния. Очередным тактовым импульсом счетчик будет сброшен в "0", и начнется очередной цикл выдачи восьми битов информации из сдвигового регистра в линию связи.

Синхронная последовательная передача отдельных битов данных на линию связи должна производиться без какого-либо перерыва, и следующий байт данных должен быть загружен в буферный регистр из системного интерфейса за время, не превышающее времени передачи восьми битов в последовательную линию связи.

При записи байта данных в буферный регистр обнуляется регистр состояния контроллера. Нуль в этом регистре указывает, что в линию связи передается байт данных из сдвигового регистра, а следующий передаваемый байт данных загружен в сдвиговый регистр.

Контроллер для последовательного синхронного приема данных из ВУ состоит из тех же компонентов, что и контроллер для синхронной последовательной передачи, за исключением генератора тактовых импульсов.

**Асинхронный последовательный интерфейс**

Организация асинхронного последовательного обмена данными с внешним устройством осложняется тем, что на передающей и приемной стороне последовательной линии связи используются настроенные на одну частоту, но физически разные генераторы тактовых импульсов и, следовательно, общая синхронизация.

Обмен данными с ВУ по последовательным линиям связи широко используется в микроЭВМ, особенно в тех случаях, когда не требуется высокой скорости обмена. Вместе с тем применение в них последовательных линий связи с ВУ обусловлено двумя важными причинами. Во-первых, последовательные линии связи просты по своей организации: два провода при симплексной и полудуплексной передаче и максимум четыре - при дуплексной. Во-вторых, в микроЭВМ используются внешние устройства, обмен с которыми необходимо вести в последовательном коде.

В современных микроЭВМ применяют, как правило, универсальные контроллеры для последовательного ВВ, обеспечивающие как синхронный, так и асинхронный режим обмена данными с ВУ.

## 1.12.Способы обмена информацией в микропроцессорной системе

В ЭВМ применяются три режима ввода/вывода: программно-управляемый ВВ (называемый также программным или нефорсированным ВВ), ВВ по прерываниям (форсированный ВВ) и прямой доступ к памяти. Первый из них характеризуется тем, что инициирование и управление ВВ осуществляется программой, выполняемой процессором, а внешние устройства играют сравнительно пассивную роль и сигнализируют только о своем состоянии, в частности, о готовности к операциям ввода/вывода. Во втором режиме ВВ инициируется не процессором, а внешним устройством, генерирующим специальный сигнал прерывания. Реагируя на этот сигнал готовности устройства к передаче данных, процессор передает управление подпрограмме обслуживания устройства, вызвавшего прерывание. Действия, выполняемые этой подпрограммой, определяются пользователем, а непосредственными операциями ВВ управляет процессор. Наконец, в режиме прямого доступа к памяти, который используется, когда пропускной способности процессора недостаточно, действия процессора приостанавливаются, он отключается от системной шины и не участвует в передачах данных между основной памятью и быстродействующим ВУ. Заметим, что во всех вышеуказанных случаях

основные действия, выполняемые на системной магистрали ЭВМ, подчиняются двум основным принципам.
1. В процессе взаимодействия любых двух устройств ЭВМ одно из них обязательно выполняет активную, управляющую роль и является задатчиком, второе оказывается управляемым, исполнителем. Чаще всего задатчиком является процессор.
2. Другим важным принципом, заложенным в структуру интерфейса, является принцип квитирования (запроса - ответа): каждый управляющий сигнал, посланный задатчиком, подтверждается сигналом исполнителя. При отсутствии ответного сигнала исполнителя в течение заданного интервала времени формируется так называемый тайм-аут, задатчик фиксирует ошибку обмена и прекращает данную операцию.

### 1.12.1.Программно-управляемый ввод/вывод

Данный режим характеризуется тем, что все действия по вводу/выводу реализуются командами прикладной программы. Наиболее простыми эти действия оказываются для "всегда готовых" внешних устройств, например индикатора на светодиодах. При необходимости ВВ в соответствующем месте программы используются команды IN или OUT. Такая передача данных называется синхронным или безусловным ВВ.
Однако для большинства ВУ до выполнения операций ВВ надо убедиться в их готовности к обмену, т.е. ВВ является асинхронным. Общее состояние устройства характеризуется флагом готовности READY, называемым также флагом готовности/занятости (READY/BUSY). Иногда состояния готовности и занятости идентифицируются отдельными флагами READY и BUSY, входящими в слово состояния устройства.
Процессор проверяет флаг готовности с помощью одной или нескольких команд. Если флаг установлен, то инициируются собственно ввод или вывод одного или нескольких слов данных. Когда же флаг сброшен, процессор выполняет цикл из 2-3 команд с повторной проверкой флага READY до тех пор, пока устройство не будет готово к операциям .
Основной недостаток программного ВВ связан с непроизводительными потерями времени процессора в циклах ожидания. К достоинствам следует отнести простоту его реализации, не требующей дополнительных аппаратных средств.

### 1.12.2.Организация прерываний в микроЭВМ

Одной из разновидностей программно-управляемого обмена данными с ВУ в микроЭВМ является обмен с прерыванием программы, отличающийся от асинхронного программно-управляемого обмена тем, что переход к выполнению команд, физически реализующих обмен данными, осуществляется с помощью специальных аппаратных средств. Команды обмена данными в этом случае выделяют в отдельный программный модуль - подпрограмму обработки

прерывания. Задачей аппаратных средств обработки прерывания в процессоре микроЭВМ как раз и является приостановка выполнения одной программы (ее еще называют основной программой) и передача управления подпрограмме обработки прерывания. Действия, выполняемые при этом процессором, как правило, те же, что и при обращении к подпрограмме. Только при обращении к подпрограмме они инициируются командой, а при обработке прерывания - управляющим сигналом от ВУ, который называют "Запрос на прерывание" или "Требование прерывания".

Эта важная особенность обмена с прерыванием программы позволяет организовать обмен данными с ВУ в произвольные моменты времени, не зависящие от программы, выполняемой в микроЭВМ. Таким образом, появляется возможность обмена данными с ВУ в реальном масштабе времени, определяемом внешней по отношению к микроЭВМ средой. Обмен с прерыванием программы существенным образом экономит время процессора, затрачиваемое на обмен. Это происходит за счет того, что исчезает необходимость в организации программных циклов ожидания готовности ВУ на выполнение которых тратится значительное время, особенно при обмене с медленными ВУ.

Прерывание программы по требованию ВУ не должно оказывать на прерванную программу никакого влияния кроме увеличения времени ее выполнения за счет приостановки на время выполнения подпрограммы обработки прерывания. Поскольку для выполнения подпрограммы обработки прерывания используются различные регистры процессора (счетчик команд, регистр состояния и т.д.), то информацию, содержащуюся в них в момент прерывания, необходимо сохранить для последующего возврата в прерванную программу.

Обычно задача сохранения содержимого счетчика команд и регистра состояния процессора возлагается на аппаратные средства обработки прерывания. Сохранение содержимого других регистров процессора, используемых в подпрограмме обработки прерывания, производится непосредственно в подпрограмме. Отсюда следует достаточно очевидный факт: чем больший объем информации о прерванной программе сохраняется программным путем, тем больше время реакции микроЭВМ на сигнал прерывания, и наоборот. Предпочтительными с точки зрения повышения производительности микроЭВМ (сокращения времени выполнения подпрограмм обработки, а, следовательно, и основной программы) являются уменьшение числа команд, обеспечивающих сохранение информации о прерванной программе, и реализация этих функций аппаратными средствами.

Формирование сигналов прерываний - запросов ВУ на обслуживание происходит в контроллерах соответствующих ВУ. В простейших случаях в качестве сигнала прерывания может использоваться сигнал "Готовность ВУ", поступающий из контроллера ВУ в системный интерфейс микроЭВМ. Однако такое простое решение обладает существенным недостатком - процессор не

имеет возможности управлять прерываниями, т. е. разрешать или запрещать их для отдельных ВУ. В результате организация обмена данными в режиме прерывания с несколькими ВУ существенно усложняется.

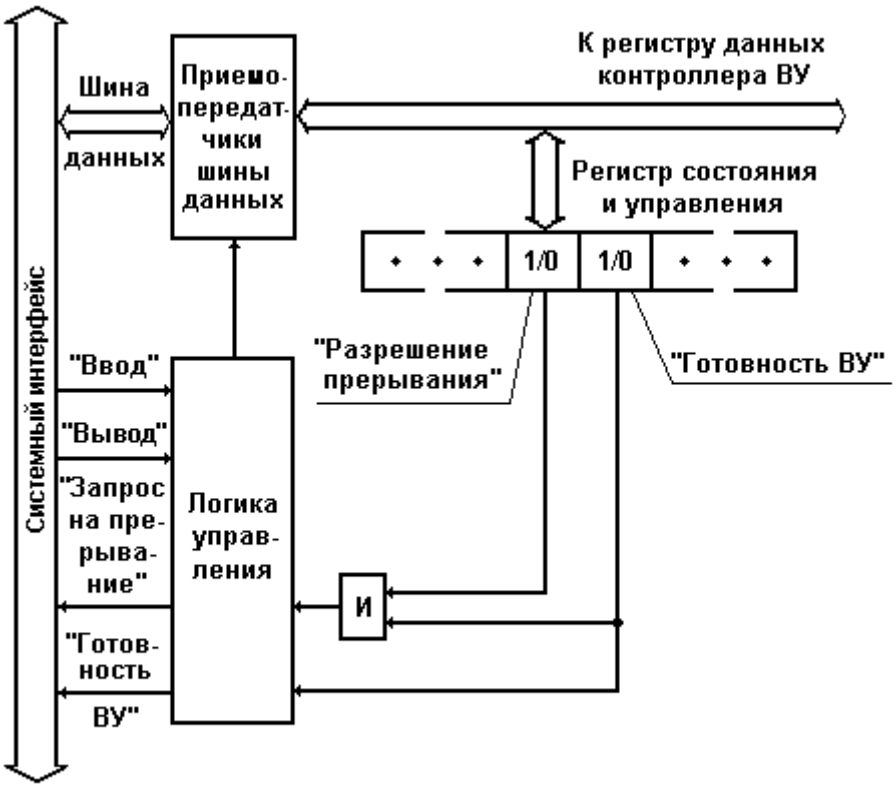


Рис. 1.12.2.1. Фрагмент блок-схемы контроллера ВУ
с разрядом "Разрешение прерывания"
в регистре состояния и управления

Для решения этой проблемы регистр состояния и управления контроллера ВУ дополняют еще одним разрядом - "Разрешение прерывания". Запись 1 или 0 в разряд "Разрешение прерывания" производится программным путем по одной из линий шины данных системного интерфейса. Управляющий сигнал системного интерфейса "Запрос на прерывание" формируется с помощью схемы совпадения только при наличии единиц в разрядах "Готовность ВУ" и "Разрешение прерывания" регистра состояния и управления контроллера.

Аналогичным путем решается проблемам управления прерываниями в микроЭВМ, в целом. Для этого в регистре состояния процессора выделяется разряд, содержимое которого определяет, разрешены или запрещены прерывания от внешних устройств. Значение этого разряда может устанавливаться программным путем.

В микроЭВМ обычно используется одноуровневая система прерываний, т. е. сигналы "Запрос на прерывание" от всех ВУ поступают на один вход процессора. Поэтому возникает проблема идентификации ВУ, запросившего обслуживание, и реализации заданной очередности (приоритета) обслуживания ВУ при одновременном поступлении нескольких сигналов прерывания.

Существуют два основных способа идентификации ВУ, запросивших обслуживания:
-программный опрос регистров состояния (разряд "Готовность ВУ") контроллеров всех ВУ;
-использование векторов прерывания.
Организация прерываний с программным опросом готовности предполагает наличие в памяти микроЭВМ единой подпрограммы обслуживания прерываний от всех внешних устройств. Структура такой подпрограммы приведена на рис. 1.12.2.2

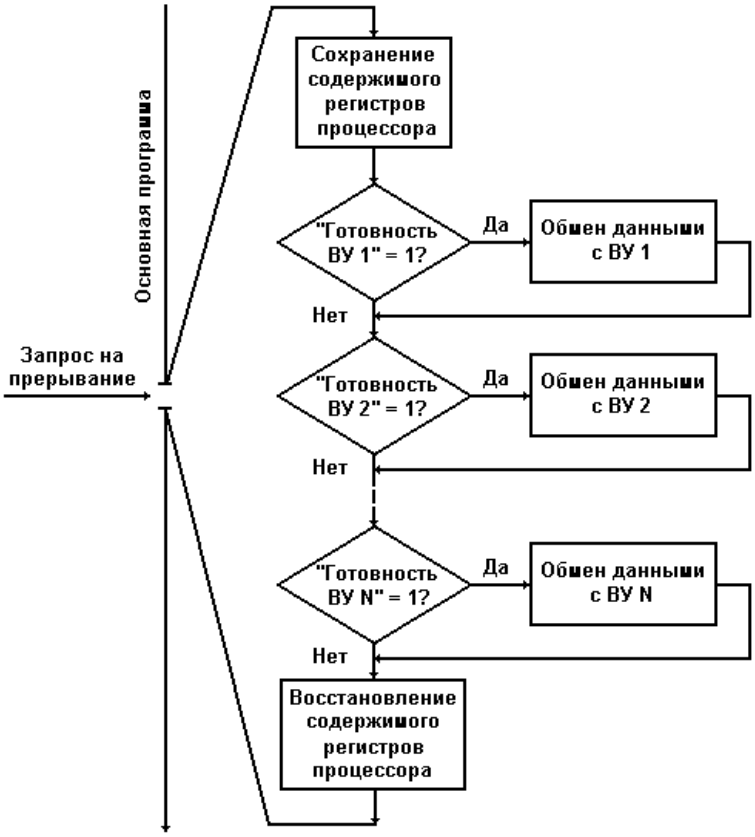


Рис. 1.12.2.2.Структура единой программы
обработки прерываний и
ее связь с основной программой

Обслуживание ВУ с помощью единой подпрограммы обработки прерываний производится следующим образом. В конце последнего машинного цикла выполнения очередной команды основной программы процессор проверяет наличие требования прерывания от ВУ. Если сигнал прерывания есть и в процессоре прерывание разрешено, то процессор переключается на выполнение подпрограммы обработки прерываний.
После сохранения содержимого регистров процессора, используемых в подпрограмме, начинается последовательный опрос регистров состояния контроллеров всех ВУ, работающих в режиме прерывания. Как только подпрограмма обнаружит готовое к обмену ВУ, сразу выполняются действия по его обслуживанию. Завершается подпрограмма обработки прерывания после

опроса готовности всех ВУ и восстановления содержимого регистров процессора.

Приоритет ВУ в микроЭВМ с программным опросом готовности внешнего устройства однозначно определяется порядком их опроса в подпрограмме обработки прерываний. Чем раньше в подпрограмме опрашивается готовность ВУ, тем меньше время реакции на его запрос и выше приоритет. Необходимость проверки готовности всех внешних устройств существенно увеличивает время обслуживания тех ВУ, которые опрашиваются последними. Это является основным недостатком рассматриваемого способа организации прерываний. Поэтому обслуживание прерываний с опросом готовности ВУ используется только в тех случаях, когда отсутствуют жесткие требования на время обработки сигналов прерывания внешних устройств.

Организация системы прерываний в микроЭВМ с использованием векторов прерываний позволяет устранить указанный недостаток. При такой организации системы прерываний ВУ, запросившее обслуживания, само идентифицирует себя с помощью вектора прерывания - адреса ячейки основной памяти микроЭВМ, в которой хранится либо первая команда подпрограммы обслуживания прерывания данного ВУ, либо адрес начала такой подпрограммы. Таким образом, процессор, получив вектор прерывания, сразу переключается на выполнение требуемой подпрограммы обработки прерывания. В микроЭВМ с векторной системой прерывания каждое ВУ должно иметь собственную подпрограмму обработки прерывания.

Различают векторные системы с интерфейсным и внеинтерфейсным вектором. В первом случае вектор прерывания формирует контроллер ВУ, запросившего обслуживания, во втором - контроллер прерываний, общий для всех устройств, работающих в режиме прерываний (IBM-совместимые персональные компьютеры).

Рассмотрим организацию векторной системы с интерфейсным вектором. Вектор прерывания выдается контроллером не одновременно с запросом на прерывание, а только по разрешению процессора, как это реализовано в схеме на рис. 1.12.2.3. Это делается для того, чтобы исключить одновременную выдачу векторов прерывания от нескольких ВУ. В ответ на сигнал контроллера ВУ "Запрос на прерывание" процессор формирует управляющий сигнал "Предоставление прерывания (вх.)", который разрешает контроллеру ВУ, запросившему обслуживание, выдачу вектора прерывания в шину адреса системного интерфейса. Для этого в контроллере используются регистр вектора прерывания и схема совпадения И3. Регистр вектора прерывания обычно реализуется с помощью перемычек или переключателей, что позволяет пользователю устанавливать для конкретных ВУ требуемые значения векторов прерывания.

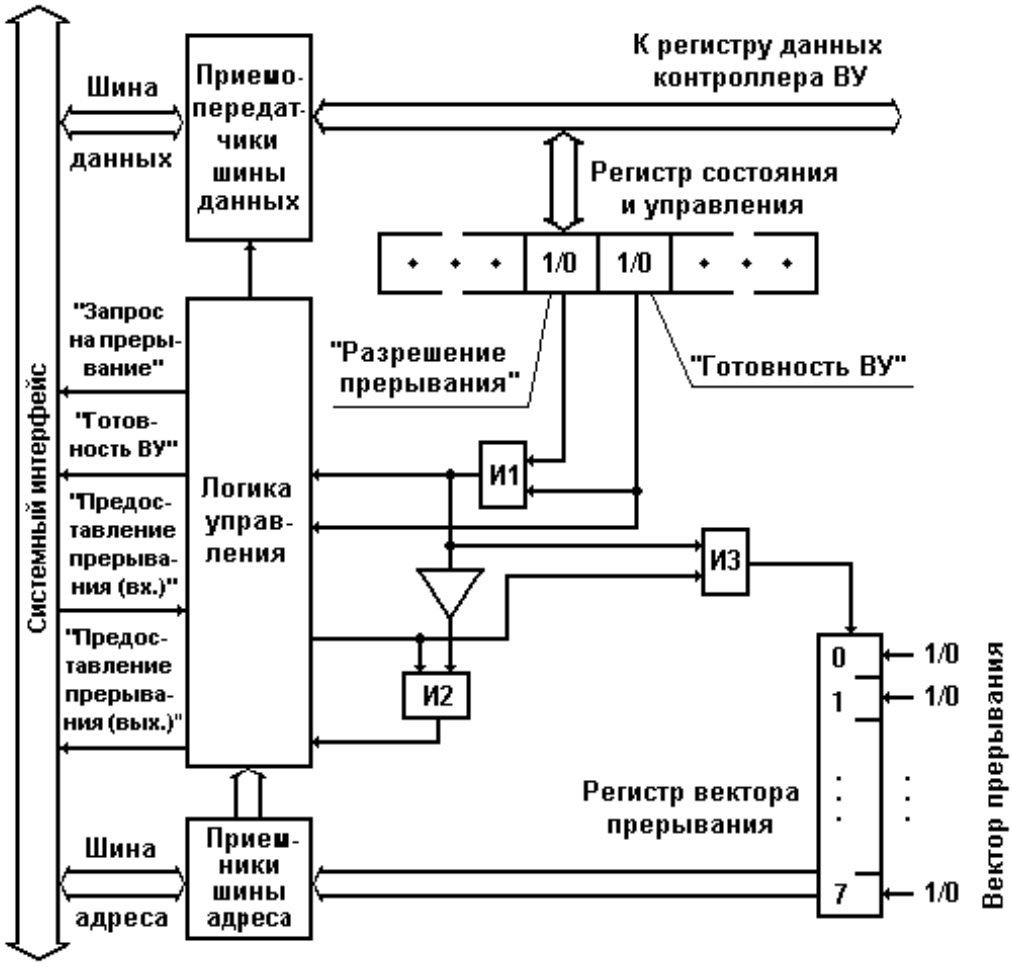


Рис. 1.12.2.3. Формирование векторов прерывания
в контроллере ВУ

Управляющий сигнал "Предоставление прерывания (вых.)" формируется в контроллере ВУ с помощью схемы совпадения И2. Этот сигнал используется для организации последовательного аппаратного опроса готовности ВУ и реализации тем самым требуемых приоритетов ВУ. Процессор при поступлении в него по общей линии системного интерфейса "Запрос на прерывание" сигнала прерывания формирует управляющий сигнал "Предоставление прерывания (вх.)", который поступает сначала в контроллер ВУ с наивысшим приоритетом. Если это устройство не требовало обслуживания, то его контроллер пропускает сигнал "Предоставление прерывания" на следующий контроллер, иначе дальнейшее распространение сигнала прекращается и контроллер выдает вектор прерывания на адресноинформационную шину.

Аппаратный опрос готовности ВУ производится гораздо быстрее, нежели программный. Но если обслуживания запросили одновременно два или более ВУ, обслуживание менее приоритетных ВУ будет отложено на время обслуживания более приоритетных, как и в системе прерывания с программным опросом.

Векторная система с внеинтерфейсным вектором прерывания используется в IBM-совместимых персональных компьютерах. В этих компьютерах контроллеры внешних устройств не имеют регистров для хранения векторов прерывания, а для идентификации устройств, запросивших обслуживания, используется общий для всех ВУ контроллер прерываний .
Упрощенная схема взаимодействия контроллера прерываний с процессором и контроллером шины имеет следующий вид.


Рис. 1.12.2.4.Упрощенная схема

Эта схема функционирует следующим образом. Пусть в некоторый момент времени контроллер клавиатуры с помощью единичного сигнала по линии IRQ 1 известил контроллер прерываний о своей готовности к обмену. В ответ на запрос контроллер прерываний генерирует сигнал INTR (запрос на прерывание) и посылает его на соответствующий вход процессора. Процессор, если маскируемые прерывания разрешены (т.е. установлен флаг прерываний IF в регистре флагов процессора), посылает на контроллер шины сигналы R# - чтение, C# - управление и IO# - ввод/вывод, определяющие тип цикла шины. Контроллер шины, в свою очередь, генерирует два сигнала подтверждения прерывания INTA# и направляет их на контроллер прерываний. По второму импульсу контроллер прерываний выставляет на шину данных восьмибитный номер вектора прерывания, соответствующий данной линии IRQ.
В режиме реального адреса ("реальном" режиме) векторы прерываний хранятся в таблице векторов прерываний, которая находится в первом килобайте оперативной памяти. Под каждый вектор отведено 4 байта (2 байта под адрес сегмента и 2 байта под смещение), т.е. в таблице может содержаться 256 векторов. Адрес вектора в таблице - номер вектора * 4.
Далее процессор считывает номер вектора прерывания. Сохраняет в стеке содержимое регистра флагов, сбрасывает флаг прерываний IF и помещает в стек адрес возврата в прерванную программу (регистры CS и IP). После этого процессор извлекает из таблицы векторов прерываний адрес подпрограммы обработки прерываний для данного устройства и приступает к ее выполнению.
Процедура обработки аппаратного прерывания должна завершаться командой

конца прерывания EOI (End of Interruption), посылаемой контроллеру прерываний. Для этого необходимо записать байт 20h в порт 20h (для первого контроллера) и в порт A0h (для второго).

### 1.12.3.Организация прямого доступа к памяти

Одним из способов обмена данными с ВУ является обмен в режиме прямого доступа к памяти (ПДП). В этом режиме обмен данными между ВУ и основной памятью микроЭВМ происходит без участия процессора. Обменом в режиме ПДП управляет не программа, выполняемая процессором, а электронные схемы, внешние по отношению к процессору. Обычно схемы, управляющие обменом в режиме ПДП, размещаются в специальном контроллере, который называется контроллером прямого доступа к памяти.

Обмен данными в режиме ПДП позволяет использовать в микроЭВМ быстродействующие внешние запоминающие устройства, такие, например, как накопители на жестких магнитных дисках, поскольку ПДП может обеспечить время обмена одним байтом данных между памятью и ВЗУ, равное циклу обращения к памяти.

Для реализации режима прямого доступа к памяти необходимо обеспечить непосредственную связь контроллера ПДП и памяти микроЭВМ. Для этой цели можно было бы использовать специально выделенные шины адреса и данных, связывающие контроллер ПДП с основной памятью. Но такое решение нельзя признать оптимальным, так как это приведет к значительному усложнению микроЭВМ в целом, особенно при подключении нескольких ВЗУ. В целях сокращения количества линий в шинах микроЭВМ контроллер ПДП подключается к памяти посредством шин адреса и данных системного интерфейса. При этом возникает проблема совместного использования шин системного интерфейса процессором и контроллером ПДП. Можно выделить два основных способа ее решения: реализация обмена в режиме ПДП с "захватом цикла" и в режиме ПДП с блокировкой процессора.

Существуют две разновидности прямого доступа к памяти с "захватом цикла". Наиболее простой способ организации ПДП состоит в том, что для обмена используются те машинные циклы процессора, в которых он не обменивается данными с памятью. В такие циклы контроллер ПДП может обмениваться данными с памятью, не мешая работе процессора. Однако возникает необходимость выделения таких циклов, чтобы не произошло временного перекрытия обмена ПДП с операциями обмена, инициируемыми процессором. В некоторых процессорах формируется специальный управляющий сигнал, указывающий циклы, в которых процессор не обращается к системному интерфейсу. При использовании других процессоров для выделения таких циклов необходимо применение в контроллерах ПДП специальных селектирующих схем, что усложняет их конструкцию. Применение рассмотренного способа организации ПДП не снижает производительности микроЭВМ, но при этом обмен в режиме ПДП возможен только в случайные

моменты времени одиночными байтами или словами.
Более распространенным является ПДП с "захватом цикла" и принудительным отключением процессора от шин системного интерфейса. Для реализации такого режима ПДП системный интерфейс микроЭВМ дополняется двумя линиями для передачи управляющих сигналов "Требование прямого доступа к памяти" (ТПДП) и "Предоставление прямого доступа к памяти" (ППДП).
Управляющий сигнал ТПДП формируется контроллером прямого доступа к памяти. Процессор, получив этот сигнал, приостанавливает выполнение очередной команды, не дожидаясь ее завершения, выдает на системный интерфейс управляющий сигнал ППДП и отключается от шин системного интерфейса. С этого момента все шины системного интерфейса управляются контроллером ПДП. Контроллер ПДП, используя шины системного интерфейса, осуществляет обмен одним байтом или словом данных с памятью микроЭВМ и затем, сняв сигнал ТПДП, возвращает управление системным интерфейсом процессору. Как только контроллер ПДП будет готов к обмену следующим байтом, он вновь "захватывает" цикл процессора и т.д. В промежутках между сигналами ТПДП процессор продолжает выполнять команды программы. Тем самым выполнение программы замедляется, но в меньшей степени, чем при обмене в режиме прерываний.
Применение в микроЭВМ обмена данными с ВУ в режиме ПДП всегда требует предварительной подготовки, а именно: для каждого ВУ необходимо выделить область памяти, используемую при обмене, и указать ее размер, т.е. количество записываемых в память или читаемых из памяти байт (слов) информации. Следовательно, контроллер ПДП должен обязательно иметь в своем составе регистр адреса и счетчик байт (слов). Перед началом обмена с ВУ в режиме ПДП процессор должен выполнить программу загрузки. Эта программа обеспечивает запись в указанные регистры контроллера ПДП начального адреса выделенной ВУ памяти и ее размера в байтах или словах в зависимости от того, какими порциями информации ведется обмен. Сказанное не относится к начальной загрузке программ в память в режиме ПДП. В этом случае содержимое регистра адреса и счетчика байт слов устанавливается переключателями или перемычками непосредственно на плате контроллера.
Блок-схема простого контроллера ПДП, обеспечивающего ввод данных в память микроЭВМ по инициативе ВУ в режиме ПДП "Захват цикла", приведена на рис. 1.12.3

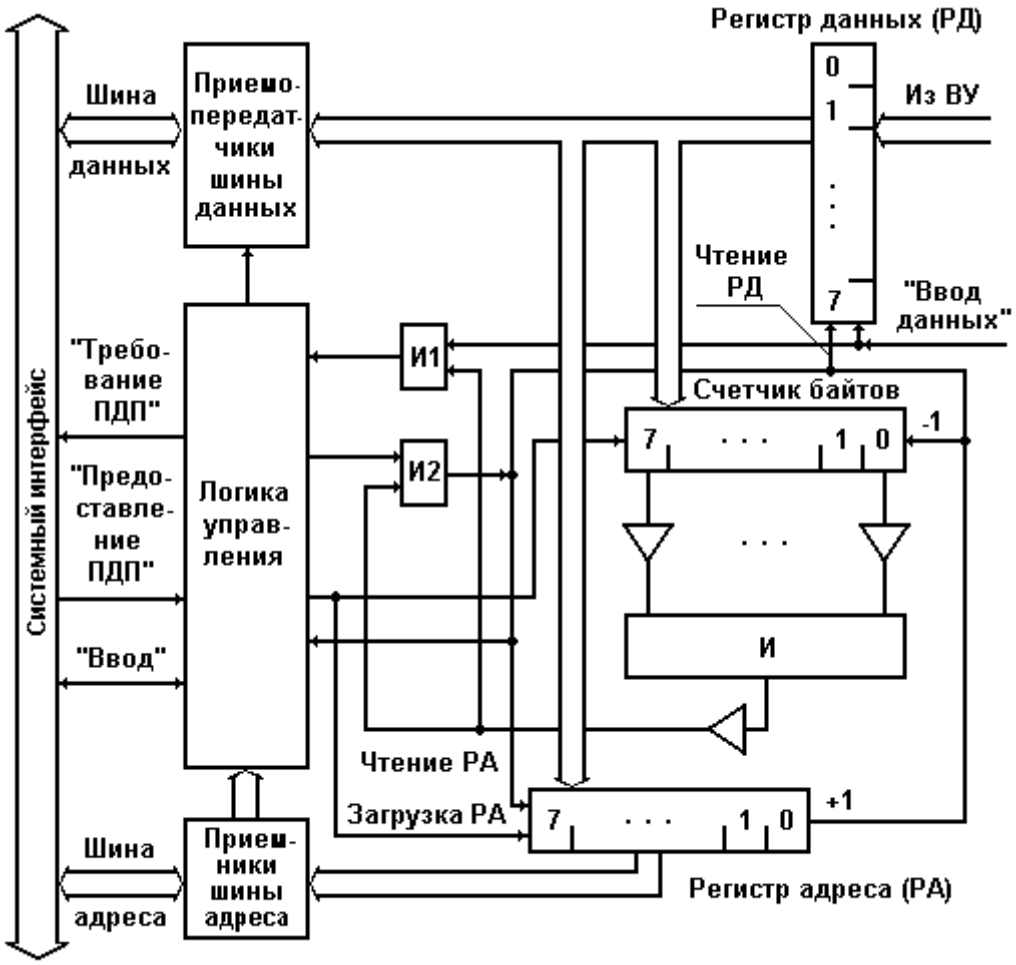


Рис. 1.12.3.Контроллер ПДП для ввода данных
из ВУ в режиме "Захват цикла" и отключением процессора
от шин системного интерфейса

Перед началом очередного сеанса ввода данных из ВУ процессор загружает в регистры его контроллера следующую информацию: в счетчик байт - количество принимаемых байт данных, а в регистр адреса - начальный адрес области памяти для вводимых данных. Тем самым контроллер подготавливается к выполнению операции ввода данных из ВУ в память микроЭВМ в режиме ПДП.

Байты данных из ВУ поступают в регистр данных контроллера в постоянном темпе. При этом каждый байт сопровождается управляющим сигналом из ВУ "Ввод данных", который обеспечивает запись байта данных в регистр данных контроллера. По этому же сигналу и при ненулевом состоянии счетчика байт контроллер формирует сигнал ТПДП. По ответному сигналу процессора ППДП контроллер выставляет на шины адреса и данных системного интерфейса содержимое своих регистров адреса и данных соответственно. Формируя управляющий сигнал "Вывод", контроллер ПДП обеспечивает запись байта данных из своего регистра данных в память микроЭВМ. Сигнал ППДП используется в контроллере и для модификации счетчика байт и регистра адреса. По каждому сигналу ППДП из содержимого счетчика байт вычитается

единица, и как только содержимое счетчика станет равно нулю, контроллер прекратит формирование сигналов "Требование прямого доступа к памяти".
На примере простого контроллера ПДП мы рассмотрели только процесс подготовки контроллера и непосредственно передачу данных в режиме ПДП. На практике любой сеанс обмена данными с ВУ в режиме ПДП всегда инициируется программой, выполняемой процессором, и включает два следующих этапа.
1. На этапе подготовки ВУ к очередному сеансу обмена процессор в режиме программно-управляемого обмена опрашивает состояние ВУ (проверяет его готовность к обмену) и посылает в ВУ команды, обеспечивающие подготовку ВУ к обмену. Такая подготовка может сводиться, например, к перемещению головок на требуемую дорожку в накопителе на жестком диске. Затем выполняется загрузка регистров контроллера ПДП. На этом подготовка к обмену в режиме ПДП завершается и процессор переключается на выполнение другой программы.
2. Обмен данными в режиме ПДП начинается после завершения подготовительных операций в ВУ по инициативе либо ВУ, как это было рассмотрено выше, либо процессора. В этом случае контроллер ПДП необходимо дополнить регистром состояния и управления, содержимое которого будет определять режим работы контроллера ПДП. Один из разрядов этого регистра будет инициировать обмен данными с ВУ. Загрузка информации в регистр состояния и управления контроллера ПДП производится программным путем.
Наиболее распространенным является обмен в режиме прямого доступ к памяти с блокировкой процессора. Он отличается от ПДП с "захватом цикла" тем, что управление системным интерфейсом передается контроллеру ПДП не на время обмена одним байтом, а на время обмена блоком данных. Такой режим ПДП используется в тех случаях, когда время обмена одним байтом с ВУ сопоставимо с циклом системной шины.
В микроЭВМ можно использовать несколько ВУ, работающих в режиме ПДП. Предоставление таким ВУ шин системного интерфейса для обмена данными производится на приоритетной основе. Приоритеты ВУ реализуются так же, как и при обмене данными в режиме прерывания, но вместо управляющих сигналов "Требование прерывания" и "Предоставление прерывания" (рис. 18 Организация прерываний в микроЭВМ) используются сигналы "Требование прямого доступа" и "Предоставление прямого доступа", соответственно.

## 2.Память в микропроцессорной системе

Для хранения информации в микропроцессорных системах используются запоминающие устройства на основе полупроводниковых материалов, а также магнитные и оптические внешние носители. Внутренняя память компьютера представлена в виде отдельных интегральных микросхем (ИМС) собственно памяти и элементов, включенных в состав других ИМС, не выполняющих непосредственно функцию хранения программ и данных - это и внутренняя память центрального процессора, и видеопамять, и контроллеры различных устройств.

Для создания элементов запоминающих устройств, в основном, применяют СБИС со структурой МДП (металл-диэлектрик-полупроводник) на основе кремния (в связи с тем, что в качестве диэлектрика чаще всего используют его оксид Si02, то их обычно называют МОП (металл-оксид-полупроводник) структурами).

Для функционирования компьютерной системы необходимо наличие как оперативного запоминающего устройства (ОЗУ), так и постоянного запоминающего устройства (ПЗУ), обеспечивающего сохранение информации при выключении питания. ОЗУ может быть статическим и динамическим, а ПЗУ однократно или многократно программируемым.

Степень интеграции, быстродействие, электрические параметры ЗУ при записи и хранении информации, помехоустойчивость, долговременная стабильность, стабильность к внешним неблагоприятным факторам при функционировании и т.д. зависят от физических принципов работы приборов, применяемых материалов при производстве ИМС и параметров технологических процессов при их изготовлении.

На развитие микропроцессорной техники решающее значение оказывает технология производства интегральных схем.

Полупроводниковые интегральные микросхемы подразделяются на биполярные ИМС и МОП схемы, причем первые - более быстродействующие, а вторые имеют большую степень интеграции, меньшую потребляемую мощность и меньшую стоимость. Цифровые микросхемы могут по идеологии, конструкторскому решению, технологии относится к разным семействам, но выполнять одинаковую функцию, т.е. быть инвертором, триггером или процессором. Наиболее популярными семействами можно назвать у биполярных ИМС: ТТЛ (транзисторно-транзисторная логика), ТТЛШ (с диодами Шоттки), ЭСЛ (эмиттерно-связанная логика); у МДП: n-МОП и КМОП.

Базовым материалом для изготовления ИМС является кремний. Несмотря на то, что он не обладает высокой подвижностью носителей заряда (mn=1500 см2/Вс), а значит, приборы на его основе теоретически будут уступать по быстродействию приборам на основе арсенида галлия GaAs, однако система Si-SiO2 существенно более технологична. С другой стороны, приборы на

кремниевой основе кремний-оксид кремния) обладают совершенной границей раздела Si-SiO2, химической стойкостью, электрической прочностью и другими уникальными свойствами.
Технологический цикл производства ИМС включает:
-эпитаксиальное наращивание слоя на подготовленную подложку;
-наращивание слоя SiO2 на эпитаксиальный слой;
-нанесение фоторезиста, маскирование и вытравливание окон в слое;
-легирование примесью путем диффузии или имплантацией;
-аналогично повторение операций для подготовки других легированных областей;
-повторение операций для создания окон под контактные площадки;
-металлизацию всей поверхности алюминием или поликремнием;
-повторение операций для создания межсоединений;
-удаление излишков алюминия или поликремния;
- контроль функционирования;
- помещение в корпус;
- выходной контроль.
Наиболее критичным для увеличения степени интеграции является процесс литографии, т.е. процесс переноса геометрического рисунка шаблона на поверхность кремниевой пластины. С помощью этого рисунка формируют такие элементы схемы, как электроды затвора, контактные окна, металлические межкомпонентные соединения и т.п. На первой стадии изготовления ИМС после завершения испытаний схемы или моделирования с помощью ЭВМ формируют геометрический рисунок топологии схемы. С помощью электронно-лучевого устройства или засветки другим способом топологический рисунок схемы последовательно, уровень за уровнем можно переносить непосредственно на поверхность кремниевой пластины, но чаще на фоточувствительные стеклянные пластины, называемые фотошаблонами. Между переносом топологического рисунка с двух шаблонов могут быть проведены операции ионной имплантации, загонки, окисления и металлизации. После экспонирования пластины помещают в раствор, который проявляет изображение в фоточувствительном материале - фоторезисте.
Увеличивая частоту колебаний световой волны, можно уменьшить ширину линии рисунка, т. е. сократить размеры интегральных схем. Но возможности этой технологии ограничены, поскольку рентгеновские лучи трудно сфокусировать. Один из вариантов - использовать сам свет в качестве шаблона (так называемое позиционирование атомов фокусированным лазерным лучом). Этим способом, осветив двумя взаимно перпендикулярными лазерными пучками, можно изготовить решетку на кремниевой пластине из хромированных точек размером 80 нм. Сканируя лазером поверхность для создания произвольного рисунка интегральных наносхем, теоретически можно создавать схемы с шириной линии рисунка в 10 раз меньшей, чем сегодняшние. Второе ограничение при литографии накладывает органическая природа

фоторезиста. Путь ее решения - применение неорганических материалов, например, оксидов ванадия.
Физические процессы, протекающие в изделиях микроэлектроники (и в микросхемах памяти тоже), технология изготовления и конструктивные особенности ИМС высокой степени интеграции могут влиять на архитектуру и методы проектирования ЭВМ и систем. Естественно, уменьшение геометрических размеров транзисторов приводит к увеличению электрических полей, особенно в районе стока. Это может привести к развитию лавинного пробоя и, как следствие, к изменению выходной ВАХ МОП транзистора:
-включению паразитного биполярного транзистора (исток-подложка-сток);
-неравномерному заряжению диэлектрика у стока;
-деградации приповерхностной области полупроводника;
-пробою диэлектрика.
Поэтому необходимо уменьшение напряжения питания СБИС до 3.6, 3.3, 3 В и т.п., при этом известно, что блок питания компьютера обеспечивает обычно напряжения +5В, +12В, -12В.
Однако инжекция и заряжение диэлектрика не всегда процесс отрицательный или паразитный. Уменьшение напряжения записи информационного заряда в репрограммируемых ЗУ ниже 12 В позволяет их программировать внутри микропроцессорной системы, а не специальным устройством (программатором). Тогда для разработчика открываются большие возможности для программирования не только адреса микросхем контроллера или адаптера в пространстве устройств ввода/вывода или номера прерывания, но и творить необходимое устройство самому (если иметь такую ИМС). Однако отметим, что кроме "хозяина" это может сделать и компьютерный вирус, который будет, естественно, разрушать, а не созидать что-либо.

## 2.1.Основные характеристики полупроводниковой памяти

Полупроводниковая память имеет большое число характеристик и параметров, которые необходимо учитывать при проектировании систем:
1. Емкость памяти определяется числом бит хранимой информации. Емкость кристалла обычно выражается также в битах и составляет 1024 бита, 4 Кбит, 16 Кбит, 64 Кбит и т.п. Важной характеристикой кристалла является информационная организация кристалла памяти MxN, где M - число слов, N - разрядность слова. Например, кристалл емкостью 16 Кбит может иметь различную организацию: 16 Кх1, 4 Кх2 Кх8. При одинаковом времени обращения память с большей шириной выборки обладает большей информационной емкостью.
2. Временные характеристики памяти.
Время доступа - временной интервал, определяемый от момента, когда центральный процессор выставил на шину адреса адрес требуемой ячейки памяти и послал по шине управления приказ на чтение или запись данных, до момента осуществления связи адресуемой ячейки с шиной данных.

Время восстановления - это время, необходимое для приведения памяти в исходное состояние после того, как ЦП снял с ША - адрес, с ШУ - сигнал "чтение" или "запись" и с ШД - данные.
3. Удельная стоимость запоминающего устройства определяется отношением его стоимости к информационной емкости, т.е. определяется стоимостью бита хранимой информации.
4. Потребляемая энергия (или рассеиваемая мощность) приводится для двух режимов работы кристалла: режима пассивного хранения информации и активного режима, когда операции записи и считывания выполняются с номинальным быстродействием. Кристаллы динамической МОП-памяти в резервном режиме потребляют примерно в десять раз меньше энергии, чем в активном режиме. Наибольшее потребление энергии, не зависящее от режима работы, характерно для кристаллов биполярной памяти.
5. Плотность упаковки определяется площадью запоминающего элемента и зависит от числа транзисторов в схеме элемента и используемой технологии. Наибольшая плотность упаковки достигнута в кристаллах динамической МОП-памяти.
6. Допустимая температура окружающей среды обычно указывается отдельно для активной работы, для пассивного хранения информации и для нерабочего состояния с отключенным питанием. Указывается тип корпуса, если он стандартный, или чертеж корпуса с указанием всех размеров, маркировкой и нумерацией контактов, если корпус новый. Приводятся также условия эксплуатации: рабочее положение, механические воздействия, допустимая влажность и другие.

## 2.2.Постоянные запоминающие устройства

Программируемые постоянные запоминающие устройства (ППЗУ) делятся на однократно программируемые (например, биполярные ПЗУ с плавкими соединениями) и многократно электрически программируемые МОП ПЗУ. Это полевой транзистор с плавающим затвором и МДОП (металл-диэлектрик-оксид полупроводник) транзистор. Обычно в качестве диэлектрика используют нитрид кремния.
В ПЗУ обеспечена возможность непрерывного считывания без разрушения информации, причем запись и считывание могут быть выполнены в очень короткое время.
Стирание информации (возврат структуры в исходное состояние) может осуществляться:
- ультрафиолетовым излучением с энергией квантов более 5.1 эВ (ширина запрещенной зоны нитрида кремния) через кварцевое окно;
- подачей на структуру импульса напряжения, противоположного по знаку записывающему. Основными факторами, влияющими на запись и хранение заряда, являются электрическое поле, температура и радиация. Количество электрических циклов "запись-стирание" обычно не менее $10^5$.

## 2.3.Оперативные запоминающие устройства

Полупроводниковые ЗУ подразделяются на ЗУ с произвольной выборкой и ЗУ с последовательным доступом.

ЗУПВ подразделяются на:

-статические оперативные запоминающие устройства (СОЗУ);
-динамические оперативные запоминающие устройства (ДОЗУ).

ЗУ с последовательным доступом подразделяются на:

- регистры сдвига;
- приборы с зарядовой связью (ПЗС).

В основе большинства современных ОЗУ лежат комплиментарные МОП ИМС (КМОП), которые отличаются малой потребляемой мощностью.

Как известно, быстродействие МОП транзисторов в первую очередь ограничивается большой входной емкостью затвор-исток (подложка). Уменьшение геометрических размеров приборов (площади затвора и длины канала) при увеличении степени интеграции увеличивает граничную частоту.

Малое потребление энергии позволяет использовать КМОП ИМС с питанием от микробатареи как ПЗУ, где располагается часть операционной системы, которая осуществляет начальную загрузку всей системы (программа Setup).

Чаще всего ОЗУ выполнены в виде ЗУ с произвольной выборкой, которые имеют ряд преимуществ перед ЗУ с последовательным доступом.

## 2.4.Буферная память

В вычислительных системах используются подсистемы с различным быстродействием, и, в частности, с различной скоростью передачи данных. (Например, передача из подсистемы 1 в подсистему 2).Обычно обмен данными между такими подсистемами реализуется с использованием прерываний или канала прямого доступа к памяти. В первую очередь подсистема 1 формирует запрос на обслуживание по мере готовности данных к обмену. Однако обслуживание прерываний связано с непроизводительными потерями времени и при пакетном обмене производительность подсистемы 2 заметно уменьшается. При обмене данными с использованием канала прямого доступа к памяти подсистема 1 передает данные в память подсистемы 2. Данный способ обмена достаточно эффективен с точки зрения быстродействия, но для его реализации необходим довольно сложный контроллер прямого доступа к памяти.

Наиболее эффективно обмен данными между подсистемами с различным быстродействием реализуется при наличии между ними специальной буферной памяти. Данные от подсистемы 1 временно запоминаются в буферной памяти до готовности подсистемы 2 принять их. Емкость буферной памяти должна быть достаточной для хранения тех блоков данных, которые подсистема 1 формирует между считываниями их подсистемой 2. Отличительной особенностью буферной памяти является запись данных с быстродействием и

под управлением подсистемы 1, а считывание - с быстродействием и под управлением подсистемы 2 ("эластичная память"). В общем случае память должна выполнять операции записи и считывания совершенно независимо и даже одновременно, что устраняет необходимость синхронизации подсистем. Буферная память должна сохранять порядок поступления данных от подсистемы 1, т.е. работать по принципу "первое записанное слово считывается первым" (First Input First Output - FIFO). Таким образом, под буферной памятью типа FIFO понимается ЗУПВ, которое автоматически следит за порядком поступления данных и выдает их в том же порядке, допуская выполнение независимых и одновременных операций записи и считывания. На рис. 2.4.1 приведена структурная схема буферной памяти типа FIFO емкостью 64x4.

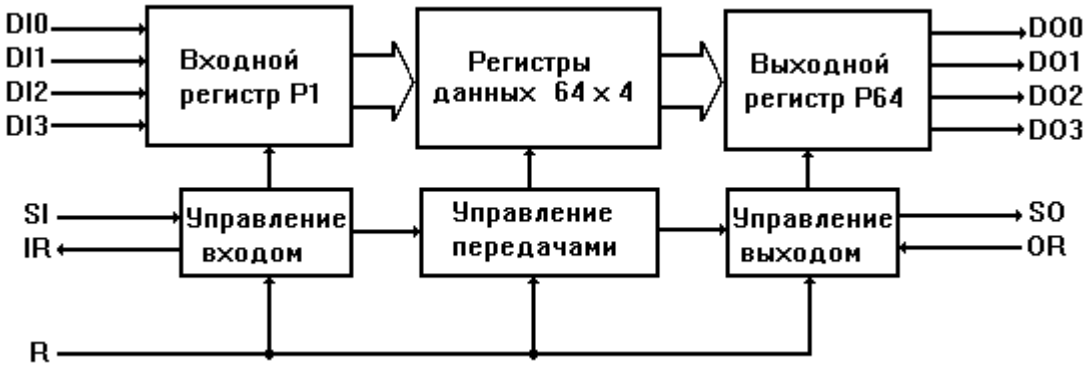


Рис. 2.4.1 Структурная схема буфера 64x4.

На кристалле размещены 64 4-битных регистра с независимыми цепями сдвига, организованных в 4 последовательных 64-битных регистра данных, 64-битный управляющий регистр, а также схема управления. Входные данные поступают на линии DI0-DI3, а вывод данных осуществляется через контакты DO0-DO3. Ввод (запись) данных производится управляющим сигналом SI (shift in), а вывод (считывание) - сигналом вывода SO (shift out). Ввод данных осуществляется только при наличии сигнала готовности ввода IR (input ready), а вывод - при наличии сигнала готовности вывода OR (output ready). Управляющий сигнал R (reset) производит сброс содержимого буфера.

При вводе 4-битного слова под действием сигнала SI оно автоматически передвигается в ближайший к выходу свободный регистр. Состояние регистра данных отображается в соответствующем ему управляющем триггере, совокупность триггеров образует 64-битный управляющий регистр. Если регистр содержит данные, то управляющий триггер находится в состоянии 1, а если регистр не содержит данных, то триггер находится в состоянии 0. Как только управляющий бит соседнего справа регистра изменяется на 0, слово данных автоматически сдвигается к выходу. Перед началом работы в буфер подается сигнал сброса R и все управляющие триггеры переводятся в состояние 0 (все регистры буфера свободны). На выводе IR формируется логическая 1, т.е. буфер готов воспринимать входные данные. При действии сигнала ввода SI входное слово загружается в регистр P1, а управляющий триггер этого регистра

устанавливается в состояние 1: на входе IR формируется логический 0. Связи между регистрами организованы таким образом, что поступившее в P1 слово "спонтанно" копируется во всех регистрах данных FIFO и появляется на выходных линиях DO0-DO3. Теперь все 64 регистра буфера содержат одинаковые слова, управляющий триггер последнего регистра P64 находится в состоянии 1, а остальные управляющие триггеры сброшены при передаче данных в соседние справа регистры. Состояние управляющего триггера P64 выведено на линию готовности выхода OR; OR принимает значение 1, когда в триггер записывается 1. Процесс ввода может продолжаться до полного заполнения буфера; в этом случае все управляющие триггеры находятся в состоянии 1 и на линии IR сохраняется логический 0.

При подаче сигнала SO производится восприятие слова с линий DO0-DO3, управляющий триггер P64 переводится в состояние 1, на линии OR появляется логическая 1, а управляющий триггер P64 сбрасывается в 0. Затем этот процесс повторяется для остальных регистров и нуль в управляющем регистре перемещается ко входу по мере сдвига данных вправо.

В некоторых кристаллах буфера FIFO имеется дополнительная выходная линия флажка заполнения наполовину. На ней формируется сигнал 1, если число слов составляет более половины емкости буфера.

Рассмотренный принцип организации FIFO допускает выполнение записи и считывания данных независимо и одновременно. Скорость ввода определяется временным интервалом, необходимым для передачи данных из P1, а выводить данные можно с такой же скоростью. Единственным ограничением является время распространения данных через FIFO, равное времени передачи входного слова на выход незаполненного буфера FIFO. Оно равняется произведению времени внутреннего сдвига и числа регистра данных. В буферах FIFO, выполненных по МОП-технологии и имеющих емкость 64 слова, время распространения составляет примерно 30 мкс, а в биполярных FIFO такой же емкости - примерно 2 мкс.

Буферы можно наращивать как по числу слов, так и по их длине.

### 2.5.Стековая память

Стековой называют память, доступ к которой организован по принципу: "последним записан - первым считан" (Last Input First Output - LIFO). Использование принципа доступа к памяти на основе механизма LIFO началось с больших ЭВМ. Применение стековой памяти оказалось очень эффективным при построении компилирующих и интерпретирующих программ, при вычислении арифметических выражений с использованием польской инверсной записи. В малых ЭВМ она стала широко использоваться в связи с удобствами реализации процедур вызова подпрограмм и при обработке прерываний.

Принцип работы стековой памяти состоит в следующем . Когда первое слово помещается в стек, оно располагается в первой свободной ячейке памяти. Следующее записываемое слово перемещает предыдущее на одну ячейку вверх

и занимает его место и т.д. Считывание слов из стека осуществляется в обратном порядке, начиная с кода , который был записан последним. Для фиксации переполнения стека желательно формировать признак переполнения.
Перемещение данных при записи и считывании информации в стековой памяти подобно тому, как это имеет место в сдвигающих регистрах. С точки зрения реализации механизма доступа к стековой памяти выделяют аппаратный и аппаратно-программный (внешний) стеки.
Аппаратный стек представляет собой совокупность регистров, связи между которыми организованы таким образом, что при записи и считывании данных содержимое стека автоматически сдвигается. Обычно емкость аппаратного стека ограничена диапазоном от нескольких регистров до нескольких десятков регистров, поэтому в большинстве МП такой стек используется для хранения содержимого программного счетчика и его называют стеком команд. Основное достоинство аппаратного стека - высокое быстродействие, а недостаток - ограниченная емкость.
Наиболее распространенным в настоящее время и, возможно, лучшим вариантом организации стека в ЭВМ является использование области памяти. Для адресации стека используется указатель стека, который предварительно загружается в регистр и определяет адрес последней занятой ячейки. Помимо команд CALL и RET, по которым записывается в стек и восстанавливается содержимое программного счетчика, имеются команды PUSH и POP, которые используются для временного запоминания в стеке содержимого регистров и их восстановления, соответственно. В некоторых МП содержимое основных регистров запоминается в стеке автоматически при прерывании программ. Содержимое регистра указателя стека при записи уменьшается, а при считывании увеличивается на 1 при выполнении команд PUSH и POP, соответственно.

## 3. Интерфейсы

Интерфейсы – промежуточные устройства, которые служат для соединения процессоров, памятей и периферийного оборудования с целью передачи информации между ними. Инженер, создающий систему, изучает генеральную задачу и характеристики отдельных доступных устройств, чтобы выбрать подходящие интерфейсы и режимы их работы для оптимального решения задачи. Ошибки в выборе интерфейса приводят к большим потерям средств и времени.
Интерфейсы прошли длительный путь развития – от примитивных нестандартизованных соединений элементарных схем в первом цифровом компьютере ENIAC в 1944 году до комплекса стандартов на базе современного Scable Coherent Interface (SCI), охватывающего все области информационно-вычислительных технологий наших дней.

## 3.1.Терминология

**Открытые системы** - системы, основанные на открытых спецификациях (стандартах) на интерфейсы, службы и форматы, достаточных для того, чтобы обеспечить:

-возможность переноса прикладных программ, разработанных должным образом, на системы в широком их диапазоне, при минимальных изменениях

-совместную работу с другими прикладными программами на локальных и удаленных системных платформах

-взаимодействие с пользователями в стиле, облегчающем последним переход от системы к системе

**Физический интерфейс** - термин, определяющий совокупность механических и электрических средств, а также физических сред. Такая совокупность служит физической основой для создания логического интерфейса.

**Логический интерфейс** - термин, охватывающий все логические протоколы.

**Логический протокол** - совокупность правил передачи кодированной информации между устройствами, узлами или элементами системы.

**Канал связи** - совокупность передатчика, линии связи и приемника, обеспечивающая передачу информации в одном направлении.

**Линия связи** - техническое устройство или лус в физической среде, используемые для пропускания сигналов.

**Группа линий** - набор линий, служащих для выполнения родственных функций.

**Магистраль** - совокупность групп линий, служащих для передачи данных и управляющих сигналов. Магистраль соединяет все станции в крейте.

**Параллельная магистраль** - магистраль, к которой обслуживаемые устройства присоединены параллельно и в которой передача битов происходит параллельно во времени во всех линиях.

**Последовательная магистраль** - состоит из одной линии с последовательной во времени передачей битов. Обслуживаемые устройства могут быть присоединены к этой магистрали параллельно.

**Запросчик** - устройство, начинающее трансакцию запросом магистрали или посылкой запроса в канал.

**Ответчик** - устройство, исполняющее задание запросчика и отвечающее ему.

**Трансакция** - действие запросчика в виде посылки мообщения через канал связи или магистраль с получением ответа от адресата.

**Крейт** - каркас для установки модулей , неотъемлимой частью которого являются магистраль или каналы, предназначенные для передачи данных и управляющих сигналов, а также проводники питания.

**Модуль** - сменный блок, использующий линии магистрали или канала в соответствии со стандартом и занимающий в крейте одну или более станций.

**Станция** - позиция в крейте для разъема, служащего для соединения модуля с магистралью или каналом.

### 3.2.Система VME

Магистраль в системе VME выполняет 4 функции : передачу данных и служебных команд, арбитрацию и приоритетные прерывания.

В крейте VME устроены гнезда для 21 модуля с разъемами С-96. Крайнее левое гнездо предназначено для системного контроллера . Этот обязательный модуль содержит схему арбитрации, генератор синхроимпульсов, схемы инициализации и обнаружения отказов.

Первоначально слово адреса состояло из 23-х битов, а слово данных всего из 16- ти битов, затем адрес был расширен до 31 разряда и данные до 32-х разрядов.

В логическом протоколе полной магистрали VME предусмотрены варианты адресации. Возможна как прямая адресация к удвоенным байтам данных при двух стробах DS0 и DS1 , так и модификация длины адресного слова : 31, 23 и 15 разрядов. Шесть битов модификатора адресации AM0-AM5 позволяют присвоить каждому из запросчиков персональный код из битов AMx. Этот код использует группа ответчиков, которые получают возможность отвечать по разным адресам в зависимости от значения кода. Эти 64 кода, выражаемые 16-ричными числами от 3F до 00, подразделяются на 3 категории: определяемые стандартом VME, задаваемые пользователем или резервные. Коды выражают комбинации свойств: адресация – короткая (64 Кб), стандартная (16Мб) и расширенная (4Гб); доступ – привилегированный или простой; команды – программа, последовательный доступ, данные, ввод-вывод.

В передачах данных также возможна модификация : одиночный байт можно передать по линиям D00-D07 или D08-D15, слово по линиям D00 – D15 , двойное слово по линиям D00 – D31.

Описанные модификации,заметно затрудняющие программирование, были разработаны с целью максимально эффективного использования устройств памяти .

Протокол VME асинхронный .Каждый последующий шаг выполняется в результате завершения предыдущего шага, при этом могут вводиться задержки.

Протоколом предусмотрен и неразрывный цикл чтение-модификация-запись. Для передачи блока данных служит режим последовательного доступа. Магистраль предоставляется модулю арбитром в зависимости от приоритета, который слагается из 2-х компонентов: фиксированного и переменного. Первый определяется позицией модуля – наивысший приоритет имеет модуль , расположенный рядом с арбитром, наинизший приоритет у модуля, вставленного в крайнее правое гнездо. Кроме того, приоритеты всех модулей изменяются циклически от 1-го уровня до 4-го старшего уровня. Каждый модуль соединен со всеми 4-мя линиями запроса магистрали. Именно этим линиям циклически придаются приоритеты. Модуль может сделать запрос по любой из них, но арбитр примет запрос , получаемый от той линии, приоритет которой в данный момент старше. Если два или больше модулей одновременно запрашивают по одной и той же линии, то магистраль получает тот модуль,

который ближе к арбитру. Арбитр имеет возможность подать всем модулям приказ освобождения магистрали. Этот способ арбитрации имеет существенные недостатки: приоритеты нельзя программировать при инициализации системы, поскольку они фиксированы позицией модуля, требуется много сигналов и много контактов на разъемах, причем сигнал проходит последовательно по длинной цепочке. Все это усложняет программирование и снижает надежность системы.

Модуль, выполняющий свою задачу, может не иметь своего процессора или нужной программы. В таких случаях модуль имеет возможность обратиться за помощью к процессору, расположенному в другом модуле и даже прервать его работу. Одновременно и другие модули также могут запросить прерывание. Для передачи запросов на прерывание служат семь линий IRQ1-IRQ7. Позиционные приоритеты прерывания заданы единственной цепочкой, которая проходит через все гнезда модулей. Модуль, назначенный на обработку запросов, при получении запросов требует доступа к магистрали , а затем выставляет на адресные линии A01- A03 код приоритета запроса, который он назначает на обслуживание. После этого обработчик запросов выставляет сигнал подтверждения прерывания на линию, которая соединена со входом в первый модуль. Модуль, запросивший обслуживание, получив по цепочке сигнал, выставляет на линиях данных свой вектор прерывания , сопровождая его подтверждающим синхросигналом. На магистрали VME может быть от одного до семи обработчиков прерываний. Сложная система приоритетов была в 80-х годах обусловлена необходимостью работать с памятями небольшого объема при слабых процессорах.

**Локальная магистраль VMX**

Обслуживающая магистраль VME , обслуживающая 21 модуль , проходит по всей задней плате на протяжении 430 мм, время распространения сигнала по магистрали достигает величины до 25 нс, кроме того большая емкость линий не позволяет укоротить импульсы и их фронты. Сложная арбитрация также замедляет работу основной магистрали VME, в результате длительность циклов запрос- запись или запрос- чтение достигает 300нс. Для ускорения обменов между процессорными модулями и модулями памяти потребовалась быстрая укороченная магистраль, соединяющая всего лишь 6 соседних модулей или менее. Крейт может содержать несколько локальных магистралей VMX. Буква X здесь указывает на расширение основной системы добавлением вспомогательной магистрали.

Магистраль VMX содержит 32 линии данных, 12 адресных линий и линии управления. Адрес и данные передаются одновременно. Адресные линии мультиплексированы. Стробы данных LDS и UDS определяют передачу 2-х младших или 2-х старших байтов, а при выставлении сигнала LWORD – длинное 32- разрядное слово – одновременно работают два строба. Признаком окончания цикла передачи служит появление либо сигнала подтверждения, либо сигнала ошибки.

Любой из модулей, подключенных к магистрали, может работать в качестве запросчика. Первичный запросчик – главный, он управляет магистралью и контролирует доступ вторичных запросчиков к исполнителям, а также инициирует цикл магистрали. Кроме того, первичный запросчик решает , по какой их магистралей VME или VMX , будет выполняться передача данных, или запросчик удовлетворится памятью собственного модуля. Для системного запроса магистрали служит линия SMRQ, запрос прерываний выполняется по линии IRQ, сигналы SMACKIN и SMACKOUT служат для системных подтверждений передач вторичным запросчикам.
Комбинация 2-х параллельных магистралей в одном крейте существенно повысила производительность системы даже при относительно медленных ТТЛ- сигналах. Однако и такое решение оказалось недостаточным, пришлось создавать еще одну магистраль MVMX32 с мультиплексированием 32-разрядных слов адреса и данных. Комбинация VME и MVMX32 была названа VSB. Эта система позволяет передавать 64- разрядные слова. Поддержка устаревшего базового стандарта VME более совершенными VMX и MVMX32 позволила пользователям не только продлить эксплуатацию большого парка модулей VME , накопившихся в мире, но и фирмам выпускать новые модули.

### 3.3.Система VXI

Доступность микропроцессоров и СБИС памяти при возрастающей их емкости позволила создать новую модификацию VME – систему VXI. Этот модуль представляет собой законченный прибор, вставляемый в крейт VME. Модули VXI поставляются вместе с программой , которая позволяет не только управлять функциями модуля, но и служит для отображения передней панели модуля на экране компьютера. На экране виден целый крейт с индикаторами состояния на модулях и с элементами управления. Указателем мыши можно выбрать на экране нужный управляющий элемент любого модуля и подать команду на изменение параметров системы или даже на изменение задания во включенной реальной системе.
Располагая программами модулей и программой для всей системы, можно ее настраивать и проверять ее работу на виртуальной модели, вовсе не имея реального крейта с модулями. Программирование виртуальных приборных систем значительно облегчает выбор модулей для компоновки реальной системы и отработки ее алгоритма и программ. Если моделирование на виртуальных модулях выполнено корректно, аппаратурная система в идеале правильно работает сразу после сборки.
Виртуальные приборы стали этапом в создании еще более совершенной технологии управления приборными системами “Plug & Play Instrument System Architecture”, которая была подготовлена ведущими фирмами Intel, Microsoft, IBM и другими. Первоначальной целью была разработка таких программ управления параметрами измерительных плат, вставляемых в материнскую

плату компьютера, которые обеспечивали бы полную программную конфигурацию системы.
В 1993 году для широкого использования новой технологии в модульной аппаратуре VXI образовался «VXI Plug & Play System Alliance». Если стандарт на конструктивы и электрические характеристики VXI обеспечивал физическую открытость системы, то технология Plug & Play создала программную открытость и реализовала полную совместимость изделий VXI, выпускаемых разными фирмами.

### 3.4.Система Multibus

Эта система, разработанная фирмой Intel, основана на нескольких магистралях. Как и в системе VME, специализация магистралей в комплексе была вызвана стремлением обеспечить возможно большую информационную мощность при относительно слабых процессорах и маленьких памятях того времени.
В первом фирменном прототипе, названном просто Multibus, магистраль имела 20 адресных линий и 8 линий данных. При разработке стандарта IEEE Std 760-1980 разрядность была увеличена до 24-х и 16-ти линий соответственно. После такого улучшения логический протокол был реализован в 3-х конструктивах. Протокол асинхронный, определяет взаимодействие задатчиков и исполнителей, допускает множество задатчиков, которые получают магистраль, согласно приоритетам. Цикл передачи традиционный – задатчик, получивший магистраль, адресуется к исполнителю, при записи выставляет данные , а при чтении принимает данные. В заключении цикла следует сигнал подтверждения приема. Предусмотрен неразрывный цикл «чтение-модификация-запись», на время которого попытки других запросчиков занять магистраль блокируются специальным сигналом. Длительность цикла около 0.5 мкс.
Одиночные байты передаются по младшим 8-ми линиям. Если нужно передать старший байт, то его переключают на младшие линии данных. Определены 4 типа передач данных; чтение из памяти или из портов ввода-вывода, запись в память или в порты ввода-вывода при единой временной диаграмме.
Арбитрацию на занятие магистрали можно выполнить 3-мя способами. Если задатчики расположены в крейте рядом, то можно реализовать схему позиционных приоритетов, напоминающую цепочку в системе VME. В такой схеме объединяют не более 3-х задатчиков, поскольку велики задержки при передаче сигнала сквозь модуль. При параллельной арбитрации приоритет задатчиков зашифрован. В случае одновременного запроса модулей на магистраль дешифратор приоритетов определяет задатчик с самым высоким приоритетом и предоставляет ему магистраль. В циклической схеме приоритетов при поступлении сразу нескольких запросов магистраль получает запросчик с наивысшим приоритетом. После окончания своей работы этот запросчик получает наинизший приоритет . Поскольку запросчики состоят в замкнутой цепочке , с выбытием первого запросчика приоритеты остальных возрастают. Таким способом реализуется циклическая равнодоступность

задатчиков к магистрали. В системе Multibus1 возможно программное задание приоритетов – в процессе инициализации вводятся шифры приоритетов. В монопольном режиме магистралью владеет один избранный задатчик, поэтому приоритеты игнорируются.
В магистрали имеются 8 линий для передачи запросов модулей на прерывание процессоров. В данный момент принимается запрос лишь по одной из линий. Вариант векторного прерывания более универсален . В этом случае по линиям данных передается вектор (код) прерывания определенного процессора и управление магистралью не передается другому модулю до завершения работы, выполняемой этим прооцессором.
Сбои в передачах и аварии в системе обрабатываются специальными сигналами, чтобы успеть сохранить информацию и , по возможности снова запустить систему. Принимаемые сигналы проверяются на четность.

### 3.5.PCI -локальная магистраль персональных компьютеров

Синхронная магистраль PCI в 32-разрядном варианте содержит 49 информационных линий, при расширении до 64- разряддов добавляются еще 48 линий. В PCI определены следующие типы сигналов на линиях магистрали:
In – только входной сигнал, передаваемый в устройство,
Out – только выходной сигнал, выдаваемый устройством.
T/s - двунаправленный трехуровневый входной/выходной сигнал
S/t/s - активный при низком уровне сигнал, выдаваемый и поддерживаемый в данном интервале времени только одним агентом
O/d - сигнал, который при высоком потенциале коллектора, когда транзистор закрыт, находится в неактивном состоянии, понизить потенциал могут несколько агентов
# - сигнал активен при низком потенциале линии, обозначаемом 0, высокий потенциал линии обозначается 1.
CLK – входной сигнал для всех устройств PCI, задает начало всех трансакций. Другие PCI сигналы стробируются обычно передним фронтом PCI.
RST# - устанавливает регистры системы в исходное состояние , Этот сигнал может быть асинхронным относительно CLK. После выставления RST# только те устройства , которые выполняют инициализацию системы, имеют возможность воспринимать сигналы.
FRAME# - выдает действующий задатчик , чтобы показать начало и длительность трансакции, Этот сигнал снимается в завершение фазы данных.
IRDY# – выставляется задатчиком и показывает его способность завершить текущую фазу данных. IRDY# применяется в связке с TRDY#. В течение записи IRDY# показывает , что на линиях AD присутствуют действительные данные, В течение чтения IRDY# показывает, что задатчик готов принять данные. Фаза данных завершается в любом такте, когда стробируются совместно выставленные сигналы IRDY# и TRDY#

TRDY# - показывает способность исполнителя завершить текущую фазу данных. Во время чтения TRDY# показывает , что на линиях AD присутствуют действительные данные, а в течение записи показывает, что готовность исполнителя принять данные

STOP# - этот сигнал показывает,что исполнитель просит задатчика остановить текущую трансакцию.

LOCK# - указывает на неразделенную операцию, для завершения которой выполняется множественная трансакция. Когда выставлен сигнал LOCK#, неисключаемые трансакции могут пройти в адрес, который в настоящий момент не заблокирован.

IDSEL# - используется при конфигурации системы для выборки устройства с целью выполнения трансакции чтения или записи.

DEVSEL# - при активном выборе показывает , что выбранное устройство декодировало свой адрес. При незапрошенном присутствии DEVSEL# показывает, что на магистрали выбрано, хотя бы одно из устройств.

REQ# - показывает арбитру, что этот агент желает занять магистраль. Этот сигнал передается по индивидуальному проводнику.

GNT# - арбитр показывает задатчику, что ему разрешено занять магистраль.

PERR# - сигнал ошибки четности служит для извещения об ошибке во всех трансакциях, за исключением специальных циклов. Сигнал должен быть выдан агентом, получающим данные, спустя два такта после получения данных в случае обнаружения ошибки четности.

SERR# - сигнал системной ошибки - служит для извещения об ошибке четности при адресации , при ошибке четности данных в специальном цикле или о других системных ошибках, если их результат будет катастрофическим. Сигнал стробируется первым же фронтом импульса CLK. Сигнал снимается возвратом к высокому потенциалу с выдержкой, определяемой компоновщиком системы.

INTA# , INTB# , INTC#, INTD# - сигналы прерываний , активны при низком потенциале и выставляются асинхронно относительно тактовых импульсов CLK.

SDONE# - показывает состояние подслушивания при текущем доступе. Когда сигнал не установлен, подслушивание еще продолжается, Установленный сигнал свидетельствует о завершении подслушивания.

SBO# - показывает успешный отбор модифицированной кэш- строки, Состояние, когда SBO# снят и SDONE# установлен , обозначается CLEAN и показывает отсутствие кэш-конфликтов и возможность нормального завершения обращения к памяти.

Трансакция состоит из фазы адресации, сопровождаемой одной или более фазами данных. Фаза длится один период CLK. Допускается пересылка множества данных при чтении и при записи. Адрес AD при вводе-выводе является адресом байта данных, а при конфигурации системы и при обращении к памяти - адресом 32 разрядного двойного слова. В фазе данных линии

магистрали AD [07-00] содержат наименее значимый байт, в то же время линии AD [31-24] содержат наиболее значимый байт. Данные фиксируются в тех интервалах времени , когда одновременно выставлены сигналы IRDY# и TRDY#.
В фазе адресации код на линиях C/BE[3-0] определяет команду, а в фазе данных определяет, какие байты слова несут информативные данные.
Бит PAR дополняет до нечетности сумму битов на линиях AD и C/BE. Бит выдается задатчиком в следующем периоде после окончания фазы адресации или фазы записи, а исполнитель выдает бит после окончания фазы чтения.
Команды, выдаваемые задатчиком, указывают исполнителю тип начинаемой трансакции . Команды выражаются словом на линиях C/BE в фазе адресации. В представленных ниже кодах команд C -1 указывает на высокий потенциал, 0 – на низкий потенциал линии, В фазе данных BE активным считается низкий потенциал.
0000 Interrupt Ackgjwledge подтверждение прерывания,
это команда чтения, направленного непосредственно контроллеру системных прерываний. В течение фазы адреса биты адреса логически несущественны. Во время фазы данных код BE указывает размер возвращаемого вектора прерывания.
0001 Special Cycle специальный цикл
Команда запускает на PCI широкую передачу сообщений.
0010 I/O Read чтение из зоны ввода/вывода
Команда служит для чтения данных из агента, определенного в зоне памяти для ввода-вывода. Код на линиях AD определяет адрес байта, при этом все биты должны быть декодированы. Код BE показывает объем передачи, при этом код должен соответствовать адресу байта.
0011 I/O Write запись в зону ввода/вывода
Команда служит для записи данных в агент, определенный в зоне памяти для ввода-вывода. Все биты должны быть декодированы. Код BE показывает объем передачи, при этом код должен соответствовать адресу байта.
Reserved
Зарезирвированные коды предназначены для исполнения в будущих расширениях системы. Исполнитель должен не реагировать на эти коды, но если все же они появятся на магистрали, то как правило, задатчик прекратит трансакцию.
0101 Reserved
0110 Memory Read чтение из памяти
Команда служит для чтения данных из агента, определенного в зоне памяти.
Исполнителю допустимо по этой команде выполнять незамедлительное чтение только в случае, если он может гарантировать , что такое чтение не создает побочные эффекты. Более того, исполнитель должен обеспечить связность любых данных, временно сохраняемых в буферах после завершения этой трансакции. Информация в таких буферах должна быть сделана

недействительной до прохождения по пути доступа любых событий синхронизации.

Memory Write запись в рамять

Команда служит для записи данных в агент, определенный в зоне памяти. Когда исполнитель возвращает “готово” он принимает ответственность за связность подчиняемых данных. Это может быть сделано либо полностью синхронизированной подачей команды Memory Write, либо обеспечив предварительное устранение ненужной информации из всех прозрачных для программ буферов до момента, когда синхронизирующие события пройдут по этому пути доступа. При этом предполагается, что задатчик свободен образовать синхронизирующее событие сразу же после выдачи команды Memory Write.

1000 Reserved

Reserved

Configuration Read чтение из зоны конфигурации

Команда служит для чтения из зоны конфигурации каждого агента, Агент выбирается , когда выставлен его сигнал IDSELи код на младших разрядах AD[1-0]=00. В адресной фазе цикла конфигурации код AD[7-2] является адресом одного из 64-х регистров, содержащих двойное 32-разрядное слово в находящихся в зоне конфигурации каждого устройства, а код AD[31-11] не имеет логической значимости для выбранного агента, Код AD[10-8] показывает , какое из устройств многофункционального агента адресуется, а код C/BE выбирает байты внутри каждого двойного 32-разрядного слова.

Configuration Write запись в зону конфигурации

Команда служит для передачи данных в зону конфигурации каждого агента, Оговорки те же , что и в предыдущем определении.

Memory Read Multiplay многократное чтение кэш-строк из памяти

Эта команда подобна команде Memory Read, за исключением того, что она дополнительно указывает, что задатчик может иметь намерение выбрать более одной кэш-строки до момента отсоединения. Контроллеру памяти следует продолжать посылать запросы к памяти, пока выставлен сигнал FRAME#. Эта команда предназначена для последовательных передач больших массивов данных, когда система памяти может получить выигрыш в производительности, выполняя последовательные чтения ранее дополнительной кэш-строки, когда буфер, прозрачный для программ, способен к временному запоминанию.

Dual Address Cycle цикл с 64-разрядным адресом

Команда служит для передачи 64-разрядного адреса устройствам, которые 64-разрядную адресацию Исполнители, работающие только с 32- разрядными адресами, должны рассматривать эту команду как резервную и никоим образом не отвечать на нее в трансакции.

1110 Memory Read Line чтение из множества кэш-строк из памяти

Эта команда подобна команде Memory Read, за исключением того, что дополнительно указывает на намерение запросчика завершить более 2-х 32-разрядных фаз связных данных. Команда предназначена для последовательных передач больших массивов данных, когда система памяти может получить выигрыш в производительности при считывании до границ кэш-строки в ответ на запрос, чем при одиночных циклах памяти. Как и при команде Memory Read, информация в предвыбранных буферах должна быть сделана недействительной до прохождения по пути доступа любых событий синхронизации.

Memory Write and Invalidate запись в память с отменой достоверности

Команда подобна Memory Write , за исключением того, что она дополнительно гарантирует минимальную передачу одной полной кэш-строки, т.е. задатчик намерен записать все байты адресованной кэш-строки за одну трансакцию PCI. Задатчик может допустить выход трансакции за границу кэш-строки только в случае, если он намерен передать также и следующую полную кэш-строку. Для этой команды требуется встроенный в задатчик регистр конфигурации ,указывающий размер кэш-строки. Это допускает оптимизацию работы памяти , устранением сырой кэш-строки в кэше с обратной записью, не прибегая к действительному циклу обратной записи и сокращая , таким способом, время доступа.

Передача неограниченного блока данных (ПБД)является главным способом передач информации в магистрали PCI. ПБД состоит из фазы адресации и одной или более фаз данных. Стандарт поддерживает ПБД как в память , так и при вводе-выводе.

Практически все сигналы (за исключением RST, INTA, INTB,INTC,INTD) фиксируются возрастающим фронтом тактовых импульсовCLK. Передачами данных в PCI управляют 3 сигнала:FRAME, IRDY,TRDY.Если хотя бы один из последних 2-х сигналов не выдан на цикл, то этот цикл является ожиданием. Если не выставлены сигналы FRAME, IRDY, то магистраль свободна. После выставления FRAME, первый период с положительным фронтом CLK является фазой адресации, адрес и команда фиксируются этим фронтом. Фронт следующего импульса CLK начинает первую фазу данных. Данные передаются между запросчиком и исполнителем при каждом положительном фронте, если одновременно выставлены IRDYиTRDY. Циклы ожидания могут устроить и задатчик и исполнитель снятием своего сигнала. Если задатчик выставил IRDY, то он должен не изменять сигналы FRAME или IRDY до текущей фазы данных независимо от состояния TRDY. Если исполнитель выставил TRDY. или STOP, он должен не изменять DEVSEL , TRDY,или STOP до завершения текущей фазы данных.

В случае, если задатчик намеревается завершить еще только одну передачу данных, задатчик снимает FRAME, сохраняя IRDY лишь на следующий цикл. После того, как исполнитель показывает завершение передачи, снимая TRDY, интерфейс возвращается в свободное состояние при снятых FRAME и IRDY.

В магистрали PCI принято распределенное декодирование адреса, при этом исполнитель сам декодирует адрес, выданный задатчиком, наблюдая и сопоставляя его со своим присвоенным адресом. Если значения кодов совпадают, то ответчик подтверждает получение адреса. Такое поведение названо положительным декодированием.
Одно из устройств может быть запрограммировано на исключительное декодирование. Это устройство ожидает в течение определенного времени, достаточного для положительного декодирования всеми другими устройствами. Если никто не принял свой адрес, то упомянутое устройство принимает передачу независимо от значения кода адреса, лишь бы это ни была зарезервированная команда.
Адресация при вводе-выводе обеспечивает выборку отдельных байтов данных. В полном слове адреса AD для указания номера байта служат два младших разряда AD [01-00]. Номера байтов может быть заданы и в фазе данных командами на линиях C/BE. Два способа задания представлены в таблице

| AD01 | AD00 | C/BE3 | C/BE2 | C/BE1 | C/BE0 |
|---|---|---|---|---|---|
| 0 | 0 | X | X | X | 0 |
| 0 | 1 | X | X | 0 | 1 |
| 1 | 0 | X | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |

В таблице C/BE 1- бит не задействован, 0 - бит задействован.
В этой таблице любые другие комбинации значений битов незакончены, иначе говоря , в цикле данных на месте указанных 0 битов C/BE никогда не должно быть 1, и наоборот. Нарушение этого правила приводит к прекращению трансакции исполнителем. При вводе/выводе принято задавать номера байта разрядами AD, при этом не нужно тратить время на лишнюю фазу данных для задания кодом C/BE.
В случае адресации при обращении в зону памяти выбирается 32-разрядное двойное слово данных DWORD, т.е. сразу 4 байта. Для адресации слова служат 30 битов AD 31-02, а биты AD 01-0 использованы для модификации передач блоков данных. При последовательных передачах слов в память или из памяти происходит линейное нарастание адреса по 4( с прибавлением 1 в разряд AD02) при каждой фазе данных, пока не выставлен сигнал FRAME. Передачи слов DWORD в кэш - строках тоже блочные, выполняются они тоже с последовательным возрастанием адреса по 4. Допустимое число слов в кэш-строке определяется значением кода, записанного в регистре размера кэш-строки в зоне конфигурации. Однако использование кэш-строки различное по версиям стандартов PCI 2.0 и PCI 2.1. Это показано в следующей таблице

| AD01 | AD00 | ПБД 2.0 | ПБД 2.1 |
|---|---|---|---|
| 0 | 0 | Линейное нарастание адреса и числа циклов данных | Линейное нарастание адреса и числа циклов данных |
| 0 | 1 | Линейное нарастание числа слов до заполнения кэш-строки | Зарезервировано |
| 1 | 0 | Зарезервировано | Линейное нарастание числа слов в кэш-строке с циклическим возвратом к началу при переполнении |
| 1 | 1 | Зарезервировано | Зарезервировано |

В случае появления зарезервированных кодов AD 01-00 происходит прекращение трансакции после первой фазы данных, представленное резервирование кода 01 в версии 2.1 некорректно: если в дальнейшем этот код будет использован, то аппаратура, исполненная по стандарту 2.1, окажется несовместимой ( или не вполне совместимой) со старой аппаратурой версии 2.0.

В случае адресации в зоне конфигурации в каждом устройстве в этой зоне записаны 64 слова DWORD, содержащие разнообразную информацию о характеристиках устройства, начиная с фирмы-производителя и кончая техническими характеристиками. Команды конфигурации 1010 чтение и 1011 запись выдает процессор - хозяин. Он же выдает сигнал на одну из линий AD31-17, соединенную с контактом IDSEL. Исполнитель подтверждает свою выбранность сигналом DEVSEL.

В случае конфигурации устройств 1-го уровня, присоединенных к магистрали PCI, непосредственно, на линии AD 01-00 выставляется код 00, а на линии AD 07-02 выставляется адрес выбираемого слова DWORD. При иных сочетаниях сигналов исполнитель игнорирует трансакцию.

Мосты обнаруживают конфигурационный запрос к устройству 2-го уровня, расположенному, за мостом, по коду AD 01-00=01

Некоторые сигналы могут быть выданы несколькими агентами. Если после выдачи сигнала одним агентом другой агент намерен сразу же изменить сигнал , то в синхронной магистрали PCI нужно пропустить один цикл, чтобы второй агент успел приготовиться к действию.

Трансакция чтения открывается фазой адресации, в течение которой линии AD 31-00 содержат код адреса, а код на линиях C/BE определяет команду.

В течение фазы данных код C/BE показывает, какой из байтов передается в текущей фазе. Фаза данных может состоять из циклов передачи и ожидания. Сигналы C/BE должно быть действительными с момента первого строба в фазе данных до завершения трансакции.

Перед первой фазой данных при чтении необходим цикл подготовки к перемене сигналов на линияхAD. После стробирования адреса задатчик

прекращает управление линиями AD, чтобы передать управление исполнителю, который сначала должен ввести цикл подготовки с выставлением сигнала DEVSEL. Прием данных запросчиком происходит когда уже установлены оба сигнала IRDY и TRDY (при этом TRDY не может быть установлен , если не установлен DEVSEL) Когда снят хотя бы один из сигналов IRDY или TRDY, наступает цикл ожидания, в котором данные не фиксируются получателем. Только после финального установления IRDY для последнего считывания, задатчик может снять FRAME.

Трансакция записи похожа на трансакцию чтения, за исключением того, что после выставления адреса не требуется цикл подготовки, поскольку коды выставляет сам задатчик. При записи запросчик может изменять номера передаваемых байтов данных изменением кода C/BE в фазах данных. Прекратить трансакцию могут и задатчик и исполнитель, однако задатчик сохраняет за собой управление, приводя все трансакции к упорядоченному систематическому завершению независимо от того, какие причины вызвали прекращение трансакции. Все трансакции завершаются, когда сняты оба сигнала FRAME и IRDY с введением свободного цикла IDLE.

В магистрали PCI реализована центральная арбитрация, в которой каждый задатчик располагает уникальными сигналами запроса REQ и разрешения GNT. Для получения доступа к магистрали служит простое взаимодействие запроса- ответа. Арбитрация «скрытая», это означает, что она происходит во время текущей трансакции, поэтому для арбитрации не требуются дополнительные циклы, за исключением случая, когда магистраль свободна.

Центральный арбитр может быть запрограммирован на выполнение конкретного алгоритма арбитрации, например с циклическими приоритетами, с равнодоступностью и т.п. Алгоритм должно быть определен так, чтобы гарантировать непревышение наибольшей допустимой задержки. Однако, поскольку алгоритм арбитрации принципиально не является частью спецификации магистрали, компоновщики аппаратурных систем могут пойти на модификацию алгоритма, но при этом должны обеспечить выполнение требований в части задержек во вставных платах и контроллерах ввода/вывода. Магистраль допускает возвратные трансакции , запускаемые одним и тем же агентом, и позволяет арбитру гибко оценивать вес запросов. Арбитр может работать по любой схеме, учитывая, что в любом такте может быть выставлен только один сигнал GNT.

Сигналы REQ подведены к СБИС арбитра и сигналы GNT выведены от арбитра индивидуальными проводниками. Это позволяет арбитру в любой момент принять запрос REQ от любого устройства, связанного с магистралью, а также отменить разрешение GNT, данное любому устройству, связанному с магистралью. При инициализации системы арбитр может присвоить один из 2-х уровней приоритетов каждому агенту, при этом мостам обычно присваивают высокий уровень. В пределах уровня магистраль равнодоступна, поскольку, как

правило, арбитр предоставляет магистраль всем запросившим задатчикам поочереди вкруговую.
Для удобства пользователей, стандартом предусмотрена автоконфигурация систем: чтобы скомпоновать и задействовать систему, пользователю достаточно вставить в разъемы магистралей интерфейсные платы, присоединяющие нужные периферийные устройства. В другие разъемы можно добавить платы памяти, наращивая оперативную память до нужного объема. После вставления плат пользователю не требуется вводить или запускать какие-либо программы – достаточно включить питание и конфигурация завершится сама сабой.
Информация о характеристиках подключаемых устройств, необходимая для выполнения алгоритмов конфигурации, содержится в зоне конфигурации основной памяти, а в СБИСах или на платах устройств размещена в регистрах конфигурации. Для процессора-хозяина обеспечен доступ к любым регистрам сразу после подключения устройств- до введения в память системы каких-либо пользовательских программ.
В регистрах каждого устройства определен 8-ми разрядный код номера магистрали, к которой присоединено устройство, 5-тиразрядный номер самого устройства и 3 разряда отведены для 8-ми возможных функций устройства.
После выполнения собственно конфигурации следует инициализация системы. Функции, запомненные в регистрах устройств, позволяют предоставить устройствам потребные системные ресурсы. При инициализации присваиваются приоритеты прерываний и имена прерываемых устройств.
Регистры конфигурации имеют строго определенные адреса в зоне конфигурации изделиях. Вся зона объемом 256 байтов подразделяется на 2 части – 64 байта в позоне регистров с предопределенным назначением и 192 байта в подзоне с кодами, специфичными лишь для данного применения устройства.

## 4.Основные этапы развития параллельной обработки

Идея параллельной обработки возникла одновременно с появлением первых вычислительных машин. В начале 50-х гг. американский математик Дж. фон Нейман предложил архитектуру последовательной ЭВМ, которая приобрела классические формы и применяется практически во всех современных ЭВМ. Однако фон Нейман разработал также принцип **построения процессорной матрицы**, в которой каждый элемент был соединен с четырьмя соседними и имел 29 состояний. Он теоретически показал, что такая матрица может выполнять все операции, поскольку она моделирует поведение машины Тьюринга.
Практическая реализация основных идей параллельной обработки началась только в 60-х гг. нашего столетия. Это связано с появлением транзистора, который благодаря малым размерам и высокой надежности по сравнению с электронными лампами позволил строить машины, состоящие из большого

количества логических элементов, что принципиально необходимо для реализации любой формы параллелизма.

Появление параллельных ЭВМ с различной организацией (конвейерные ЭВМ, процессорные матрицы, многопроцессорные ЭВМ) вызвано различными причинами. Совершенствование этих ЭВМ происходило по внутренним законам развития данного типа машин.

Рассмотрим основные этапы совершенствования параллельных ЭВМ для каждого типа структур.

**Конвейерные ЭВМ.** Основополагающим моментом для развития конвейерных ЭВМ явилось обоснование академиком С.А.Лебедевым в 1956 г. метода, названного "принципом водопровода" (позже он стал называться конвейером). Прежде всего был реализован конвейер команд, на основании которого практически одновременно были построены советская ЭВМ БЭСМ-6 (1957-1966 гг., разработка Института точной механики и вычислительной техники АН СССР) и английская машина ATLAS (1957-1963 гг.). Конвейер команд предполагал наличие многоблочной памяти и секционированного процессора, в котором на разных этапах обработки находилось несколько команд.

Конвейер команд позволил получить в ЭВМ БЭСМ-6 быстродействие в 1 млн. оп/с. В дальнейшем конвейеры команд совершенствовались и стали необходимым элементом всех быстродействующих ЭВМ, в частности, использовались в известных семействах ЭВМ IBM/360 и ЕС ЭВМ.

Следующим заметным шагом в развитии конвейерной обработки, реализованном в ЭВМ CDC-6600 (1964 г.), было введение в состав процессора нескольких функциональных устройств, позволяющих одновременно выполнять несколько арифметико-логических операций: сложение, умножение, логические операции.

В конце 60-х гг. был введен в использование арифметический конвейер, который нашел наиболее полное воплощение в ЭВМ CRAY-1 (1972-1976 гг.). Арифметический конвейер предполагает разбиение цикла выполнения арифметико-логической операции на ряд этапов, для каждого из которых отводится собственное оборудование. Таким образом, на разных этапах обработки нахо-дится несколько чисел, что позволяет производить эффективную обработку вектора чисел.

Сочетание многофункциональности, арифметического конвейера для каждого функционального блока и малой длительности такта синхронизации позволяет получить быстродействие в десятки и сотни миллионов операций в секунду. Такие ЭВМ называются суперЭВМ.

**Процессорные матрицы.** Идея получения сверхвысокого быстродействия в первую очередь связывалась с процессорными матрицами (ПМ). Предполагалось, что, увеличивая в нужной степени число процессорных элементов в матрице, можно получить любое заранее заданное быстродействие. Именно этим объясняется большой объем теоретических разработок по организации процессорных матриц.

В 60-х гг. в Институте математики Сибирского отделения АН СССР под руководством доктора технических наук Э.В.Евреинова сформировалось комплексное научное направление "Однородные системы, структуры и среды", которое получило развитие не только в СССР, но и за рубежом. Книга "Однородные универсальные вычислительные системы высокой производительности", изданная в 1966 г., была одной из первых в мире монографий, посвященных этой теме. В работах по однородным средам и системам исследовались матрицы с различными связями между элементами и конфигурацией, а также с элементами разной сложности и составом функций.

Поскольку в 60-е гг. логические схемы с большим уровнем интеграции отсутствовали, то напрямую реализовать принципы функционирования процессорной матрицы, содержащей множество элементарных процессоров, не представлялось возможным. Поэтому для проверки основных идей строились однородные системы из нескольких больших машин. Так, в 1966 г. была построена система Минск-222, спроектированная совместно с Институтом математики Сибирского отделения АН СССР и минским заводом ЭВМ им. Г.К.Орджоникидзе. Система содержала до 16 соединенных в кольцо ЭВМ Минск-2. Для нее было разработано специальное математическое обеспечение.

Другое направление в развитии однородных сред, основанное на построении процессорных матриц, состоящих из крупных процессорных элементов с достаточно большой локальной памятью, возникло в США и связано с именами Унгера, Холланда, Слотника. Была создана ЭВМ ILLIAC-IV (1966-1975 гг.), которая надолго определила пути развития процессорных матриц. В машине использовались матрицы 8*8 процессоров, каждый с быстродействием около 4 млн. оп/с и памятью 16 кбайт. Для ILLIAC-IV были разработаны кроме Ассемблера еще несколько параллельных языков высокого уровня. ЭВМ ILLIAC-IV позволила отработать вопросы коммутации, связи с базовой ЭВМ, управления вычислительным процессом. Особенно ценным является опыт разработки параллельных алгоритмов вычислений, определивший области эффективного использования подобных машин.

Примерно в то же время в СССР была разработана близкая по принципам построения ЭВМ М-10. Затем процессорные матрицы стали разрабатываться и выпускаться в ряде стран в большом количестве. Широко известной советской ЭВМ этого класса являлась машина ПС-2000, разработанная Институтом проблем управления АН СССР и запущенная в 1982 г. в производство.

Одной из наиболее сложных частей процессорных матриц является система коммутации. Самые простые ее варианты - кольцо и регулярная матрица соединений, где каждый узел связан только с двумя или четырьмя соседними, сильно ограничивают класс эффективно решаемых задач. Поэтому на протяжении всего времени существования параллельной обработки разрабатывались коммутаторы с более широкими возможностями: многомерные кубы, универсальные коммутаторы, коммутационные среды. В

СССР работы, связанные с коммутаторами для ПМ, наибольшее развитие получили в Таганрогском радиотехническом институте.

**Многопроцессорные ЭВМ.** Одной из первых полномасштабных многопроцессорных систем явилась система D825 фирмы "BURROUGHS". Начиная с 1962 г. было выпущено большое число экземпляров и модификаций D825.

Выпуск первых многопроцессорных систем, в частности D825, диктовался необходимостью получения не высокого быстродействия, а высокой живучести ЭВМ, встраиваемых в военные командные системы и системы управления. С этой точки зрения параллельные ЭВМ считались наиболее перспективными.

Система D825 содержала до четырех ПЭ и 16 модулей памяти, соединенных матричным коммутатором, который допускал одновременное соединение любого процессора с любым блоком памяти.

Существует мнение, что система D825 получила широкое распространение потому, что для нее впервые была разработана полноценная операционная система ASOR, обеспечившая синхронизацию процессов и распределение ресурсов.

В дальнейшем в СССР и на Западе были разработаны многопроцессорные системы, в которых все большее внимание уделялось операционным системам, языкам программирования, параллельной вычислительной математике.

Совершенствование микроэлектронной элементной базы, появление в 80-х годах БИС и СБИС позволили перейти к реализации структур с очень большим количеством ПЭ. Появились разработки по систолическим массивам, реализации многопроцессорных систем с программируемой архитектурой, **ЭВМ с управлением от потока данных**.Большая плотность упаковки транзисторов на кристалле позволила разместить в одной микросхеме несколько АЛУ. Это позволило реализовать принцип суперскалярной обработки.

Если в последовательной ЭВМ пользователь в процессе программирования задачи в основном следит за логикой метода вычислений, то в параллельной ЭВМ ему приходится дополнительно заниматься размещением данных и синхронизацией вычислительных процессов. Это означает, что процесс программирования существенно усложняется, особенно если программирование ведется на уровне Ассемблера, чтобы повысить эффективность использования дорогостоящих параллельных ЭВМ. Такое программирование доступно только профессионально подготовленным программистам.

## 5. Принципы конвейерной организации

### 5.1.Простейшая организация конвейера и оценка его производительности

Разработчики архитектуры компьютеров издавна прибегали к методам проектирования, известным под общим названием "совмещение операций", при котором аппаратура компьютера в любой момент времени выполняет одновременно более одной базовой операции. Этот общий метод включает два понятия: параллелизм и конвейеризацию. Хотя у них много общего и их зачастую трудно различать на практике, эти термины отражают два совершенно различных подхода. При параллелизме совмещение операций достигается путем воспроизведения в нескольких копиях аппаратной структуры. Высокая производительность достигается за счет одновременной работы всех элементов структур, осуществляющих решение различных частей задачи.

Конвейеризация (или конвейерная обработка) в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры. Так обработку любой машинной команды можно разделить на несколько этапов (несколько ступеней), организовав передачу данных от одного этапа к следующему. При этом конвейерную обработку можно использовать для совмещения этапов выполнения разных команд. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько команд. Конвейерная обработка такого рода широко применяется во всех современных быстродействующих процессорах.

Для иллюстрации основных принципов построения процессоров мы будем использовать простейшую архитектуру, содержащую 32 целочисленных регистра общего назначения (R0,...,R31), 32 регистра плавающей точки (F0,...,F31) и счетчик команд PC. Будем считать, что набор команд нашего процессора включает типичные арифметические и логические операции, операции с плавающей точкой, операции пересылки данных, операции управления потоком команд и системные операции. В арифметических командах используется трехадресный формат, типичный для RISC-процессоров, а для обращения к памяти используются операции загрузки и записи содержимого регистров в память.

Выполнение типичной команды можно разделить на следующие этапы:

выборка команды - IF (по адресу, заданному счетчиком команд, из памяти извлекается команда);

декодирование команды / выборка операндов из регистров - ID;

выполнение операции / вычисление эффективного адреса памяти - EX;

обращение к памяти - MEM;

запоминание результата - WB.

Работу конвейера можно условно представить в виде временных диаграммы (рисунок 5.1.1), на которых обычно изображаются выполняемые команды, номера тактов и этапы выполнения команд.

| Номер команды | Номер такта | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Команда i | IF | ID | EX | MEM | WB | | | | |
| Команда i+1 | | IF | ID | EX | MEM | WB | | | |
| Команда i+2 | | | IF | ID | EX | MEM | WB | | |
| Команда i+3 | | | | IF | ID | EX | MEM | WB | |
| Команда i+4 | | | | | IF | ID | EX | MEM | WB |

Рис. 5.1.1 Диаграмма работы простейшего конвейера

Конвейеризация увеличивает пропускную способность процессора (количество команд, завершающихся в единицу времени), но она не сокращает время выполнения отдельной команды. В действительности, она даже несколько увеличивает время выполнения каждой команды из-за накладных расходов, связанных с управлением регистровыми станциями. Однако увеличение пропускной способности означает, что программа будет выполняться быстрее по сравнению с простой неконвейерной схемой.

Тот факт, что время выполнения каждой команды в конвейере не уменьшается, накладывает некоторые ограничения на практическую длину конвейера. Кроме ограничений, связанных с задержкой конвейера, имеются также ограничения, возникающие в результате несбалансированности задержки на каждой его ступени и из-за накладных расходов на конвейеризацию. Частота синхронизации не может быть выше, а, следовательно, такт синхронизации не может быть меньше, чем время, необходимое для работы наиболее медленной ступени конвейера. Накладные расходы на организацию конвейера возникают из-за задержки сигналов в конвейерных регистрах (защелках) и из-за перекосов сигналов синхронизации. Конвейерные регистры к длительности такта добавляют время установки и задержку распространения сигналов. В предельном случае длительность такта можно уменьшить до суммы накладных расходов и перекоса сигналов синхронизации, однако при этом в такте не останется времени для выполнения полезной работы по преобразованию информации.

В качестве примера рассмотрим неконвейерную машину с пятью этапами выполнения операций, которые имеют длительность 50, 50, 60, 50 и 50 нс соответственно.Пусть накладные расходы на организацию конвейерной обработки составляют 5 нс. Тогда среднее время выполнения команды в неконвейерной машине будет равно 260 нс. Если же используется конвейерная организация, длительность такта будет равна длительности самого медленного этапа обработки плюс накладные расходы, т.е. 65 нс. Это время соответствует среднему времени выполнения команды в конвейере. Таким образом, ускорение, полученное в результате конвейеризации, будет равно:

| | | | | |
|---|---|---|---|---|
| Среднее время выполнения команды в неконвейерном режиме | | 260 | | |
| Среднее время выполнения команды в конвейерном режиме | = | 65 | = | 4 |

Конвейеризация эффективна только тогда, когда загрузка конвейера близка к полной, а скорость подачи новых команд и операндов соответствует максимальной производительности конвейера. Если произойдет задержка, то параллельно будет выполняться меньше операций и суммарная производительность снизится.

При реализации конвейерной обработки возникают ситуации, которые препятствуют выполнению очередной команды из потока команд в предназначенном для нее такте. Такие ситуации называются конфликтами. Конфликты снижают реальную производительность конвейера, которая могла бы быть достигнута в идеальном случае. Существуют три класса конфликтов:

Структурные конфликты, которые возникают из-за конфликтов по ресурсам, когда аппаратные средства не могут поддерживать все возможные комбинации команд в режиме одновременного выполнения с совмещением.

Конфликты по данным, возникающие в случае, когда выполнение одной команды зависит от результата выполнения предыдущей команды.

Конфликты по управлению, которые возникают при конвейеризации команд переходов и других команд, которые изменяют значение счетчика команд.

Конфликты в конвейере приводят к необходимости приостановки выполнения команд (pipeline stall). Обычно в простейших конвейерах, если приостанавливается какая-либо команда, то все следующие за ней команды также приостанавливаются. Команды, предшествующие приостановленной, могут продолжать выполняться, но во время приостановки не выбирается ни одна новая команда.

### 5.2.Структурные конфликты и способы их минимизации

Совмещенный режим выполнения команд в общем случае требует конвейеризации функциональных устройств и дублирования ресурсов для разрешения всех возможных комбинаций команд в конвейере. Если какая-нибудь комбинация команд не может быть принята из-за конфликта по ресурсам, то говорят, что в машине имеется структурный конфликт. Наиболее типичным примером машин, в которых возможно появление структурных конфликтов, являются машины с не полностью конвейерными функциональными устройствами. Время работы такого устройства может составлять несколько тактов синхронизации конвейера. В этом случае последовательные команды, которые используют данное функциональное устройство, не могут поступать в него в каждом такте. Другая возможность появления структурных конфликтов связана с недостаточным дублированием некоторых ресурсов, что препятствует выполнению произвольной

последовательности команд в конвейере без его приостановки. Например, машина может иметь только один порт записи в регистровый файл, но при определенных обстоятельствах конвейеру может потребоваться выполнить две записи в регистровый файл в одном такте. Это также приведет к структурному конфликту. Когда последовательность команд наталкивается на такой конфликт, конвейер приостанавливает выполнение одной из команд до тех пор, пока не станет доступным требуемое устройство.

Структурные конфликты возникают, например, и в машинах, в которых имеется единственный конвейер памяти для команд и данных (рисунок 5.2.1). В этом случае, когда одна команда содержит обращение к памяти за данными, оно будет конфликтовать с выборкой более поздней команды из памяти. Чтобы разрешить эту ситуацию, можно просто приостановить конвейер на один такт, когда происходит обращение к памяти за данными. Подобная приостановка часто называются "конвейерным пузырем" (pipeline bubble) или просто пузырем, поскольку пузырь проходит по конвейеру, занимая место, но не выполняя никакой полезной работы.

При всех прочих обстоятельствах, машина без структурных конфликтов будет всегда иметь более низкий CPI (среднее число тактов на выдачу команды). Возникает вопрос: почему разработчики допускают наличие структурных конфликтов? Для этого имеются две причины: снижение стоимости и уменьшение задержки устройства. Конвейеризация всех функциональных устройств может оказаться слишком дорогой. Машины, допускающие два обращения к памяти в одном такте, должны иметь удвоенную пропускную способность памяти, например, путем организации раздельных кэшей для команд и данных. Аналогично, полностью конвейерное устройство деления с плавающей точкой требует огромного количества вентилей. Если структурные конфликты не будут возникать слишком часто, то может быть и не стоит платить за то, чтобы их обойти. Как правило, можно разработать неконвейерное, или не полностью конвейерное устройство, имеющее меньшую общую задержку, чем полностью конвейерное. Например, разработчики устройств с плавающей точкой компьютеров CDC7600 и MIPS R2010 предпочли иметь меньшую задержку выполнения операций вместо полной их конвейеризации.

| Номер команды | Номер такта | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Команда загрузки | IF | ID | EX | MEM | WB | | | | | |
| Команда 1 | | IF | ID | EX | MEM | WB | | | | |
| Команда 2 | | | IF | ID | EX | MEM | WB | | | |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Команда 3 | | | | stall | IF | ID | EX | MEM | WB | |
| Команда 4 | | | | | | IF | ID | EX | MEM | WB |
| Команда 5 | | | | | | | IF | ID | EX | MEM |
| Команда 6 | | | | | | | | IF | ID | EX |

Рис. 5.2.1 Диаграмма работы конвейера при структурном конфликте

Конфликты по данным, остановы конвейера и реализация механизма обходов

Одним из факторов, который оказывает существенное влияние на производительность конвейерных систем, являются межкомандные логические зависимости. Такие зависимости в большой степени ограничивают потенциальный параллелизм смежных операций, обеспечиваемый соответствующими аппаратными средствами обработки. Степень влияния этих зависимостей определяется как архитектурой процессора (в основном, структурой управления конвейером команд и параметрами функциональных устройств), так и характеристиками программ.

Конфликты по данным возникают в том случае, когда применение конвейерной обработки может изменить порядок обращений за операндами так, что этот порядок будет отличаться от порядка, который наблюдается при последовательном выполнении команд на неконвейерной машине. Рассмотрим конвейерное выполнение последовательности команд на рисунке 5.2.2

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| ADD | R1,R2,R3 | IF | ID | EX | MEM | WB | | | | |
| | | | | | | | | | | |
| SUB | R4,R1,R5 | | IF | ID | EX | MEM | WB | | | |
| | | | | | | | | | | |
| AND | R6,R1,R7 | | | IF | ID | EX | MEM | WB | | |
| | | | | | | | | | | |
| OR | R8,R1,R9 | | | | IF | ID | EX | MEM | WB | |
| | | | | | | | | | | |
| XOR | R10,R1,R11 | | | | | IF | ID | EX | MEM | WB |
| | | | | | | | | | | |

Рис. 5.2.2, а. Последовательность команд в конвейере и ускоренная пересылка данных

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| ADD | R1,R2,R3 | IF | ID | EX | MEM | WB | | | | |
| | | | R | | | W | | | | |
| SUB | R4,R1,R5 | | IF | ID | EX | MEM | WB | | | |
| | | | | R | | | W | | | |
| AND | R6,R1,R7 | | | IF | ID | EX | MEM | WB | | |
| | | | | | R | | | W | | |
| OR | R8,R1,R9 | | | | IF | ID | EX | MEM | WB | |
| | | | | | | R | | | W | |
| XOR | R10,R1,R11 | | | | | IF | ID | EX | MEM | WB |
| | | | | | | | R | | | W |

Рис. 5.2.2, б. Совмещение чтения и записи регистров в одном такте

В этом примере все команды, следующие за командой ADD, используют результат ее выполнения. Команда ADD записывает результат в регистр R1, а команда SUB читает это значение. Если не предпринять никаких мер для того, чтобы предотвратить этот конфликт, команда SUB прочитает неправильное значение и попытается его использовать. На самом деле значение, используемое командой SUB, является даже неопределенным: хотя логично предположить, что SUB всегда будет использовать значение R1, которое было присвоено какой-либо командой, предшествовавшей ADD, это не всегда так. Если произойдет прерывание между командами ADD и SUB, то команда ADD завершится, и значение R1 в этой точке будет соответствовать результату ADD. Такое непрогнозируемое поведение очевидно неприемлемо.

Проблема, поставленная в этом примере, может быть разрешена с помощью достаточно простой аппаратной техники, которая называется пересылкой или продвижением данных (data forwarding), обходом (data bypassing), иногда закороткой (short-circuiting). Эта аппаратура работает следующим образом. Результат операции АЛУ с его выходного регистра всегда снова подается назад на входы АЛУ. Если аппаратура обнаруживает, что предыдущая операция АЛУ записывает результат в регистр, соответствующий источнику операнда для следующей операции АЛУ, то логические схемы управления выбирают в качестве входа для АЛУ результат, поступающий по цепи "обхода" , а не значение, прочитанное из регистрового файла (рисунок 5.2.3).

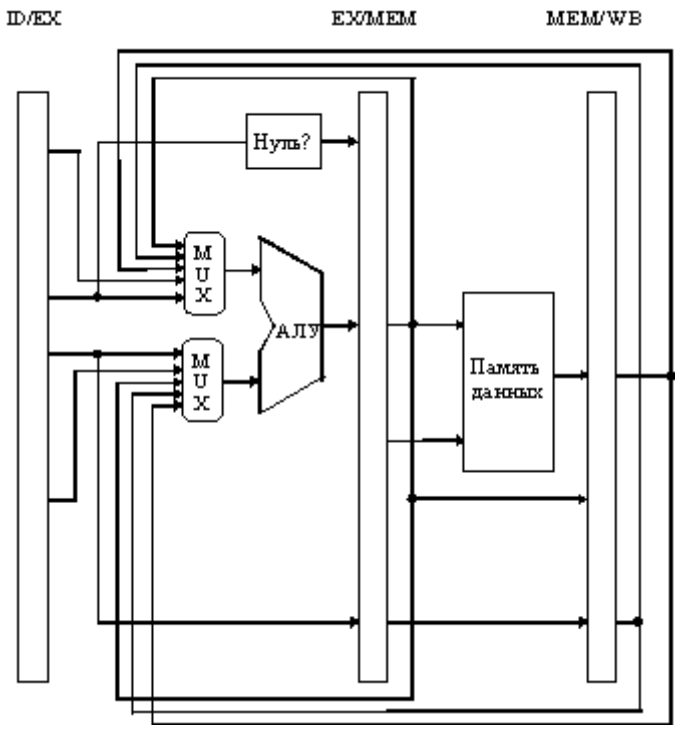


Рис. 5.2.3. АЛУ с цепями обхода
и ускоренной пересылки

Эта техника "обходов" может быть обобщена для того, чтобы включить передачу результата прямо в то функциональное устройство, которое в нем нуждается: результат с выхода одного устройства "пересылается" на вход другого, а не с выхода некоторого устройства только на его вход.

### 5.3. Классификация конфликтов по данным

Конфликт возникает везде, где имеет место зависимость между командами, и они расположены по отношению друг к другу достаточно близко так, что совмещение операций, происходящее при конвейеризации, может привести к изменению порядка обращения к операндам. В нашем примере был проиллюстрирован конфликт, происходящий с регистровыми операндами, но для пары команд возможно появление зависимостей при записи или чтении одной и той же ячейки памяти. Однако, если все обращения к памяти выполняются в строгом порядке, то появление такого типа конфликтов предотвращается.

Известны три возможных конфликта по данным в зависимости от порядка операций чтения и записи. Рассмотрим две команды i и j, при этом i предшествует j. Возможны следующие конфликты:

RAW (чтение после записи) - j пытается прочитать операнд-источник данных прежде, чем i туда запишет. Таким образом, j может некорректно получить старое значение. Это наиболее общий тип конфликтов, способ их преодоления с помощью механизма "обходов" рассмотрен ранее.

WAR (запись после чтения) - j пытается записать результат в приемник прежде, чем он считывается оттуда командой i, так что i может некорректно получить новое значение. Этот тип конфликтов как правило не возникает в системах с централизованным управлением потоком команд, обеспечивающих выполнение команд в порядке их поступления, так как последующая запись всегда выполняется позже, чем предшествующее считывание. Однако конфликты такого рода могут возникать в системах, допускающих выполнение команд не в порядке их расположения в программном коде.
WAW (запись после записи) - j пытается записать операнд прежде, чем будет записан результат команды i, т.е. записи заканчиваются в неверном порядке, оставляя в приемнике значение, записанное командой i, а не j. Этот тип конфликтов присутствует только в конвейерах, которые выполняют запись со многих ступеней (или позволяют команде выполняться даже в случае, когда предыдущая приостановлена).

### 5.3.1.Конфликты по данным, приводящие к приостановке конвейера

К сожалению не все потенциальные конфликты по данным могут обрабатываться с помощью механизма "обходов". Рассмотрим следующую последовательность команд (рисунок 5.3.1.1):

| Команда | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| LW R1,32(R6) | IF | ID | EX | MEM | WB | | | | |
| ADD R4,R1,R7 | | IF | ID | stall | EX | MEM | WB | | |
| SUB R5,R1,R8 | | | IF | stall | ID | EX | MEM | WB | |
| AND R6,R1,R7 | | | | stall | IF | ID | EX | MEM | WB |

Рис. 5.3.1.1 Последовательность команд с приостановкой конвейера

Этот случай отличается от последовательности подряд идущих команд АЛУ. Команда загрузки (LW) регистра R1 из памяти имеет задержку, которая не может быть устранена обычной "пересылкой". Вместо этого нам нужна дополнительная аппаратура, называемая аппаратурой внутренних блокировок конвейера (pipeline interlook), чтобы обеспечить корректное выполнение примера. Вообще такого рода аппаратура обнаруживает конфликты и приостанавливает конвейер до тех пор, пока существует конфликт. В этом случае эта аппаратура приостанавливает конвейер начиная с команды, которая хочет использовать данные в то время, когда предыдущая команда, результат которой является операндом для нашей, вырабатывает этот результат. Эта аппаратура вызывает приостановку конвейера или появление "пузыря" точно также, как и в случае структурных конфликтов.

### 5.3.2 Методика планирования компилятора для устранения конфликтов по данным

Многие типы приостановок конвейера могут происходить достаточно часто. Например, для оператора A = B + C компилятор скорее всего сгенерирует следующую последовательность команд (рисунок 5.3.2.1):

| Команда | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| LW R1,B | IF | ID | EX | MEM | WB | | | | |
| LW R2,C | | IF | ID | EX | MEM | WB | | | |
| ADD R3,R1,R2 | | | IF | ID | stall | EX | MEM | WB | |
| SW A,R3 | | | | IF | stall | ID | EX | MEM | WB |

Рис. 5.3.2.1. Конвейерное выполнение оператора A = B + C

Очевидно, выполнение команды ADD должно быть приостановлено до тех пор, пока не станет доступным поступающий из памяти операнд C. Дополнительной задержки выполнения команды SW не произойдет в случае применения цепей обхода для пересылки результата операции АЛУ непосредственно в регистр данных памяти для последующей записи.

Для данного простого примера компилятор никак не может улучшить ситуацию, однако в ряде более общих случаев он может реорганизовать последовательность команд так, чтобы избежать приостановок конвейера. Эта техника, называемая планированием загрузки конвейера (pipeline scheduling) или планированием потока команд (instruction scheduling), использовалась начиная с 60-х годов и стала особой областью интереса в 80-х годах, когда конвейерные машины стали более распространенными.

Пусть, например, имеется последовательность операторов: a = b + c; d = e - f;

Как сгенерировать код, не вызывающий остановок конвейера? Предполагается, что задержка загрузки из памяти составляет один такт. Ответ очевиден (рисунок 5.3.2.2):

| Неоптимизированная последовательность команд | Оптимизированная последовательность команд |
|---|---|
| LW $R_b$,b | LW $R_b$,b |
| LW $R_c$,c | LW $R_c$,c |
| ADD $R_a$,$R_b$,$R_c$ | LW $R_e$,e |
| SW a,$R_a$ | ADD $R_a$,$R_b$,$R_c$ |
| LW $R_e$,e | LW $R_f$,f |

| | |
|---|---|
| LW $R_f$,f | SW a,$R_a$ |
| SUB $R_d$,$R_e$,$R_f$ | SUB $R_d$,$R_e$,$R_f$ |
| SW d,$R_d$ | SW d,$R_d$ |

Рис. 5.3.2.2. Пример устранения конфликтов компилятором

В результате устранены обе блокировки (командой LW Rc,c команды ADD Ra,Rb,Rc и командой LW Rf,f команды SUB Rd,Re,Rf). Имеется зависимость между операцией АЛУ и операцией записи в память, но структура конвейера допускает пересылку результата с помощью цепей "обхода". Заметим, что использование разных регистров для первого и второго операторов было достаточно важным для реализации такого правильного планирования. В частности, если переменная e была бы загружена в тот же самый регистр, что b или c, такое планирование не было бы корректным. В общем случае планирование конвейера может требовать увеличенного количества регистров. Такое увеличение может оказаться особенно существенным для машин, которые могут выдавать на выполнение несколько команд в одном такте.

Многие современные компиляторы используют технику планирования команд для улучшения производительности конвейера. В простейшем алгоритме компилятор просто планирует распределение команд в одном и том же базовом блоке. Базовый блок представляет собой линейный участок последовательности программного кода, в котором отсутствуют команды перехода, за исключением начала и конца участка (переходы внутрь этого участка тоже должны отсутствовать). Планирование такой последовательности команд осуществляется достаточно просто, поскольку компилятор знает, что каждая команда в блоке будет выполняться, если выполняется первая из них, и можно просто построить граф зависимостей этих команд и упорядочить их так, чтобы минимизировать приостановки конвейера. Для простых конвейеров стратегия планирования на основе базовых блоков вполне удовлетворительна. Однако когда конвейеризация становится более интенсивной и действительные задержки конвейера растут, требуются более сложные алгоритмы планирования.

К счастью, существуют аппаратные методы, позволяющие изменить порядок выполнения команд программы так, чтобы минимизировать приостановки конвейера. Эти методы получили общее название методов динамической оптимизации (в англоязычной литературе в последнее время часто применяются также термины "out-of-order execution" - неупорядоченное выполнение и "out-of-order issue" - неупорядоченная выдача). Основными средствами динамической оптимизации являются:

Размещение схемы обнаружения конфликтов в возможно более низкой точке конвейера команд так, чтобы позволить команде продвигаться по конвейеру до тех пор, пока ей реально не потребуется операнд, являющийся также результатом логически более ранней, но еще не завершившейся команды. Альтернативным подходом является централизованное обнаружение конфликтов на одной из ранних ступеней конвейера.

Буферизация команд, ожидающих разрешения конфликта, и выдача последующих, логически не связанных команд, в "обход" буфера. В этом случае команды могут выдаваться на выполнение не в том порядке, в котором они расположены в программе, однако аппаратура обнаружения и устранения конфликтов между логически связанными командами обеспечивает получение результатов в соответствии с заданной программой.

Соответствующая организация коммутирующих магистралей, обеспечивающая засылку результата операции непосредственно в буфер, хранящий логически зависимую команду, задержанную из-за конфликта, или непосредственно на вход функционального устройства до того, как этот результат будет записан в регистровый файл или в память (short-circuiting, data forwarding, data bypassing - методы, которые были рассмотрены ранее).

Еще одним аппаратным методом минимизации конфликтов по данным является метод переименования регистров (register renaming). Он получил свое название от широко применяющегося в компиляторах метода переименования - метода размещения данных, способствующего сокращению числа зависимостей и тем самым увеличению производительности при отображении необходимых исходной программе объектов (например, переменных) на аппаратные ресурсы (например, ячейки памяти и регистры).

При аппаратной реализации метода переименования регистров выделяются логические регистры, обращение к которым выполняется с помощью соответствующих полей команды, и физические регистры, которые размещаются в аппаратном регистровом файле процессора. Номера логических регистров динамически отображаются на номера физических регистров посредством таблиц отображения, которые обновляются после декодирования каждой команды. Каждый новый результат записывается в новый физический регистр. Однако предыдущее значение каждого логического регистра сохраняется и может быть восстановлено в случае, если выполнение команды должно быть прервано из-за возникновения исключительной ситуации или неправильного предсказания направления условного перехода.

В процессе выполнения программы генерируется множество временных регистровых результатов. Эти временные значения записываются в регистровые файлы вместе с постоянными значениями. Временное значение становится новым постоянным значением, когда завершается выполнение команды (фиксируется ее результат). В свою очередь, завершение выполнения команды происходит, когда все предыдущие команды успешно завершились в заданном программой порядке.

Программист имеет дело только с логическими регистрами. Реализация физических регистров от него скрыта. Как уже отмечалось, номера логических регистров ставятся в соответствие номерам физических регистров. Отображение реализуется с помощью таблиц отображения, которые обновляются после декодирования каждой команды. Каждый новый результат записывается в физический регистр. Однако до тех пор, пока не завершится выполнение соответствующей команды, значение в этом физическом регистре рассматривается как временное.

Метод переименования регистров упрощает контроль зависимостей по данным. В машине, которая может выполнять команды не в порядке их расположения в программе, номера логических регистров могут стать двусмысленными, поскольку один и тот же регистр может быть назначен последовательно для хранения различных значений. Но поскольку номера физических регистров уникально идентифицируют каждый результат, все неоднозначности устраняются.

### 5.3.3.Сокращение потерь на выполнение команд перехода и минимизация конфликтов по управлению

Конфликты по управлению могут вызывать даже большие потери производительности конвейера, чем конфликты по данным. Когда выполняется команда условного перехода, она может либо изменить, либо не изменить значение счетчика команд. Если команда условного перехода заменяет счетчик команд значением адреса, вычисленного в команде, то переход называется выполняемым; в противном случае, он называется невыполняемым.

Простейший метод работы с условными переходами заключается в приостановке конвейера как только обнаружена команда условного перехода до тех пор, пока она не достигнет ступени конвейера, которая вычисляет новое значение счетчика команд (рисунок 5.3.3.1). Такие приостановки конвейера из-за конфликтов по управлению должны реализовываться иначе, чем приостановки из-за конфликтов по данным, поскольку выборка команды, следующей за командой условного перехода, должна быть выполнена как можно быстрее, как только мы узнаем окончательное направление команды условного перехода.

Например, если конвейер будет приостановлен на три такта на каждой команде условного перехода, то это может существенно отразиться на производительности машины. При частоте команд условного перехода в программах, равной 30% и идеальном CPI, равным 1, машина с приостановками условных переходов достигает примерно только половины ускорения, получаемого за счет конвейерной организации. Таким образом, снижение потерь от условных переходов становится критическим вопросом. Число тактов, теряемых при приостановках из-за условных переходов, может быть уменьшено двумя способами:

Обнаружением является ли условный переход выполняемым или невыполняемым на более ранних ступенях конвейера.
Более ранним вычислением значения счетчика команд для выполняемого перехода (т.е. вычислением целевого адреса перехода).

| Номер команды | Номер такта | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Команда перехода | IF | ID | EX | MEM | WB | | | | | |
| Следующая команда | | IF | stall | stall | IF | ID | EX | MEM | WB | |
| Следующая команда + 1 | | | stall | stall | stall | IF | ID | EX | MEM | WB |
| Следующая команда + 2 | | | | stall | stall | stall | IF | ID | EX | MEM |
| Следующая команда + 3 | | | | | stall | stall | stall | IF | ID | EX |
| Следующая команда + 4 | | | | | | stall | stall | stall | IF | ID |
| Следующая команда + 5 | | | | | | | stall | stall | stall | IF |

Рис. 5.3.3.1 Приостановка конвейера при выполнении команды условного перехода

Реализация этих условий требует модернизации исходной схемы конвейера.
В некоторых машинах конфликты из-за условных переходов являются даже еще более дорогостоящими по количеству тактов, чем в нашем примере, поскольку время на оценку условия перехода и вычисление адреса перехода может быть даже большим. Например, машина с раздельными ступенями декодирования и выборки регистров возможно будет иметь задержку условного перехода (длительность конфликта по управлению), которая по крайней мере на один такт длиннее. Многие компьютеры VAX имеют задержки условных переходов в четыре и более тактов, а большие машины с глубокими конвейерами имеют потери по условным переходам, равные шести или семи тактам. В общем случае, чем глубина конвейера больше, тем больше потери на командах условного перехода, исчисляемые в тактах. Конечно эффект снижения относительной производительности при этом зависит от общего CPI машины. Машины с высоким CPI могут иметь условные переходы большей длительности, поскольку процент производительности машины, которая будет потеряна из-за условных переходов, меньше.

### 5.3.4. Снижение потерь на выполнение команд условного перехода

Имеется несколько методов сокращения приостановок конвейера, возникающих из-за задержек выполнения условных переходов. В данном разделе обсуждаются четыре простые схемы, используемые во время компиляции. В этих схемах прогнозирование направления перехода выполняется статически, т.е. прогнозируемое направление перехода фиксируется для каждой команды условного перехода на все время выполнения программы.

**Метод выжидания**

Простейшая схема обработки команд условного перехода заключается в замораживании или подавлении операций в конвейере, путем блокировки выполнения любой команды, следующей за командой условного перехода, до тех пор, пока не станет известным направление перехода. Привлекательность такого решения заключается в его простоте.

**Метод возврата**

Более хорошая и не на много более сложная схема состоит в том, чтобы прогнозировать условный переход как невыполняемый. При этом аппаратура должна просто продолжать выполнение программы, как если бы условный переход вовсе не выполнялся. В этом случае необходимо позаботиться о том, чтобы не изменить состояние машины до тех пор, пока направление перехода не станет окончательно известным. В некоторых машинах эта схема с невыполняемыми по прогнозу условными переходами реализована путем продолжения выборки команд, как если бы условный переход был обычной командой. Поведение конвейера выглядит так, как будто ничего необычного не происходит. Однако, если условный переход на самом деле выполняется, то необходимо просто очистить конвейер от команд, выбранных вслед за командой условного перехода и заново повторить выборку команд (рисунок 5.3.4.1).

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Невыполняемый условный переход | IF | ID | EX | MEM | WB | | | | |
| Команда i + 1 | | IF | ID | EX | MEM | WB | | | |
| Команда i + 2 | | | IF | ID | EX | MEM | WB | | |
| Команда i + 3 | | | | IF | ID | EX | MEM | WB | |
| Команда i + 4 | | | | | IF | ID | EX | MEM | WB |
| Выполняемый условный переход | IF | ID | EX | MEM | WB | | | | |
| Команда i + 1/целевая команда | | IF | IF | ID | EX | MEM | WB | | |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Целевая команда i + 1 | | | stall | IF | ID | EX | MEM | WB | |
| Целевая команда i + 2 | | | | stall | IF | ID | EX | MEM | WB |
| Целевая команда i + 3 | | | | | stall | IF | ID | EX | MEM |

Рис. 5.3.4.1. Диаграмма работы модернизированного конвейера

**Альтернативная схема** прогнозирует переход как выполняемый. Как только команда условного перехода декодирована и вычислен целевой адрес перехода, мы предполагаем, что переход выполняемый, и осуществляем выборку команд и их выполнение, начиная с целевого адреса. Если мы не знаем целевой адрес перехода раньше, чем узнаем окончательное направление перехода, у этого подхода нет никаких преимуществ. Если бы условие перехода зависело от непосредственно предшествующей команды, то произошла бы приостановка конвейера из-за конфликта по данным для регистра, который является условием перехода, и мы бы узнали сначала целевой адрес. В таких случаях прогнозировать переход как выполняемый было бы выгодно. Дополнительно в некоторых машинах (особенно в машинах с устанавливаемыми по умолчанию кодами условий или более мощным (а потому и более медленным) набором условий перехода) целевой адрес перехода известен раньше окончательного направления перехода, и схема прогноза перехода как выполняемого имеет смысл.

**Задержанные переходы**

Четвертая схема, которая используется в некоторых машинах называется "задержанным переходом". В задержанном переходе такт выполнения с задержкой перехода длиною n есть:

команда условного перехода
следующая команда 1
следующая команда 2
.....
следующая команда n
целевой адрес при выполняемом переходе

Команды 1 - n находятся в слотах (временных интервалах) задержанного перехода. Задача программного обеспечения заключается в том, чтобы сделать команды, следующие за командой перехода, действительными и полезными. Аппаратура гарантирует реальное выполнение этих команд перед выполнением собственно перехода. Здесь используются несколько приемов оптимизации.

На рисунке 5.3.4.2, а показаны три случая, при которых может планироваться задержанный переход. В верхней части рисунка для каждого случая показана исходная последовательность команд, а в нижней части - последовательность команд, полученная в результате планирования. В случае (а) слот задержки

заполняется независимой командой, находящейся перед командой условного перехода. Это наилучший выбор. Стратегии (b) и (c) используются, если применение стратегии (a) невозможно.

В последовательностях команд для случаев (b) и (c) использование содержимого регистра R1 в качестве условия перехода препятствует перемещению команды ADD (которая записывает результат в регистр R1) за команду перехода. В случае (b) слот задержки заполняется командой, находящейся по целевому адресу команды перехода. Обычно такую команду приходится копировать, поскольку к ней возможны обращения и из других частей программы. Стратегии (b) отдается предпочтение, когда с высокой вероятностью переход является выполняемым, например, если это переход на начало цикла.

Наконец, слот задержки может заполняться командой, находящейся между командой невыполняемого перехода и командой, находящейся по целевому адресу, как в случае (c). Чтобы подобная оптимизация была законной, необходимо, чтобы можно было все-таки выполнить команду SUB, если переход пойдет не по прогнозируемому направлению. При этом мы предполагаем, что команда SUB выполнит ненужную работу, но вся программа при этом будет выполняться корректно. Это, например, может быть в случае, если регистр R4 используется только для временного хранения промежуточных результатов вычислений, когда переход выполняется не по прогнозируемому направлению.

Рисунок 5.3.4.2, б показывает различные ограничения для всех этих схем планирования условных переходов, а также ситуации, в которых они дают выигрыш. Компилятор должен соблюдать требования при подборе подходящей команды для заполнения слота задержки. Если такой команды не находится, слот задержки должен заполняться пустой операцией.

Планирование задержанных переходов осложняется (1) наличием ограничений на команды, размещение которых планируется в слотах задержки и (2) необходимостью предсказывать во время компиляции, будет ли условный переход выполняемым или нет.

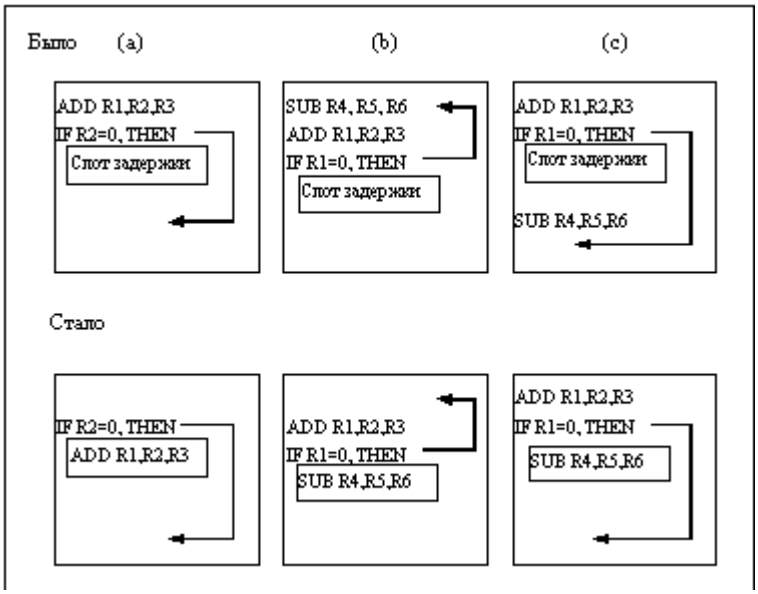


Рис. 5.3.4.2, а. Требования к переставляемым командам при планировании задержанного перехода

Имеются небольшие дополнительные затраты аппаратуры на реализацию задержанных переходов. Из-за задержанного эффекта условных переходов, для корректного восстановления состояния в случае появления прерывания нужны несколько счетчиков команд (один плюс длина задержки).

| Рассматрива-емый случай | Требования | Когда увеличивается производительность |
|---|---|---|
| (a) | Команда условного перехода не должна зависеть от переставляемой команды | Всегда |
| (b) | Выполнение переставляемой команды должно быть корректным, даже если переход не выполняется<br>Может потребоваться копирование команды | Когда переход выполняется.<br>Может увеличивать размер программы в случае копирования команды |
| (c) | Выполнение переставляемой команды должно быть корректным, даже если переход выполняется | Когда переход не выполняется |

Рис. 5.3.4.2, б

### 5.3.5. Статическое прогнозирование условных переходов: использование технологии компиляторов

Имеются два основных метода, которые можно использовать для статического предсказания переходов: метод исследования структуры программы и метод использования информации о профиле выполнения программы, который собран в результате предварительных запусков программы. Использование структуры программы достаточно просто: в качестве исходной точки можно предположить, например, что все идущие назад по программе переходы являются выполняемыми, а идущие вперед по программе - невыполняемыми. Однако эта схема не очень эффективна для большинства программ. Основываясь только на структуре программы просто трудно сделать лучший прогноз.

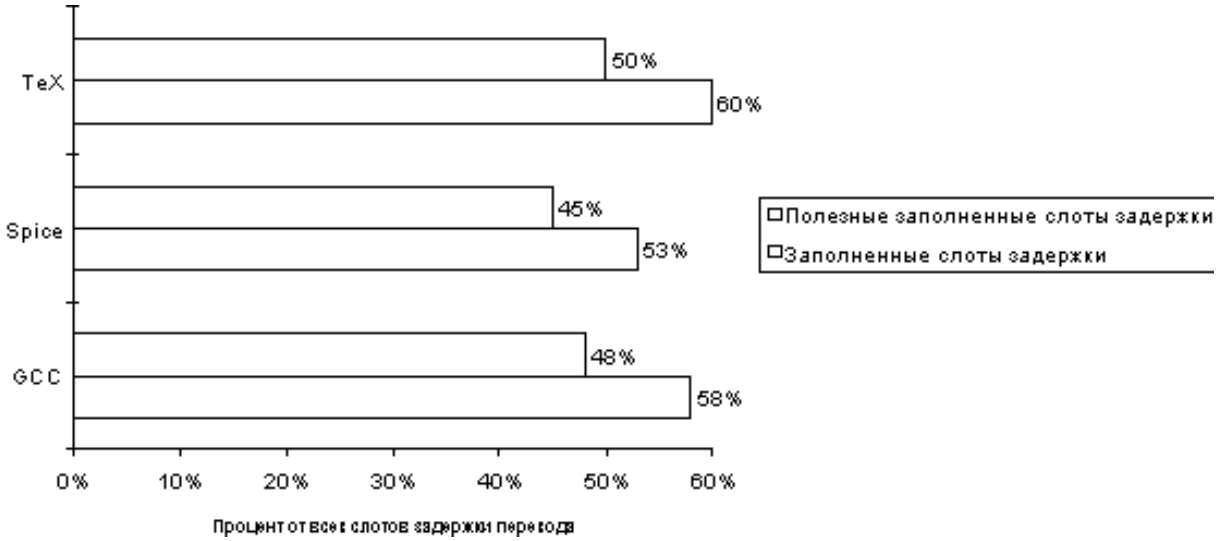


Рис. 5.3.5.1. Частота заполнения одного слота задержки условного перехода

Альтернативная техника для предсказания переходов основана на информации о профиле выполнения программы, собранной во время предыдущих прогонов. Ключевым моментом, который делает этот подход заслуживающим внимания, является то, что поведение переходов при выполнении программы часто повторяется, т.е. каждый отдельный переход в программе часто оказывается смещенным в одну из сторон: он либо выполняемый, либо невыполняемый. Проведенные многими авторами исследования показывают достаточно успешное предсказания переходов с использованием этой стратегии.

### 5.3.6.Проблемы реализации точного прерывания в конвейере

Обработка прерываний в конвейерной машине оказывается более сложной из-за того, что совмещенное выполнение команд затрудняет определение возможности безопасного изменения состояния машины произвольной командой. В конвейерной машине команда выполняется по этапам, и ее

завершение осуществляется через несколько тактов после выдачи для выполнения. Еще в процессе выполнения отдельных этапов команда может изменить состояние машины. Тем временем возникшее прерывание может вынудить машину прервать выполнение еще не завершенных команд.

Как и в неконвейерных машинах двумя основными проблемами при реализации прерываний являются: (1) прерывания возникают в процессе выполнения некоторой команды; (2) необходим механизм возврата из прерывания для продолжения выполнения программы. Например, для нашего простейшего конвейера прерывание по отсутствию страницы виртуальной памяти при выборке данных не может произойти до этапа выборки из памяти (MEM). В момент возникновения этого прерывания в процессе обработки уже будут находиться несколько команд. Поскольку подобное прерывание должно обеспечить возврат для продолжения программы и требует переключения на другой процесс (операционную систему), необходимо надежно очистить конвейер и сохранить состояние машины таким, чтобы повторное выполнение команды после возврата из прерывания осуществлялось при корректном состоянии машины. Обычно это реализуется путем сохранения адреса команды (PC), вызвавшей прерывание. Если выбранная после возврата из прерывания команда не является командой перехода, то сохраняется обычная последовательность выборки и обработки команд в конвейере. Если же это команда перехода, то мы должны оценить условие перехода и в зависимости от выбранного направления начать выборку либо по целевому адресу команды перехода, либо следующей за переходом команды. Когда происходит прерывание, для корректного сохранения состояния машины необходимо выполнить следующие шаги:

В последовательность команд, поступающих на обработку в конвейер, принудительно вставить команду перехода на прерывание.

Пока выполняется команда перехода на прерывание, погасить все требования записи, выставленные командой, вызвавшей прерывание, а также всеми следующими за ней в конвейере командами. Эти действия позволяют предотвратить все изменения состояния машины командами, которые не завершились к моменту начала обработки прерывания.

После передачи управления подпрограмме обработки прерываний операционной системы, она немедленно должна сохранить значение адреса команды (PC), вызвавшей прерывание. Это значение будет использоваться позже для организации возврата из прерывания.

Если используются механизмы задержанных переходов, состояние машины уже невозможно восстановить с помощью одного счетчика команд, поскольку в процессе восстановления команды в конвейере могут оказаться вовсе не последовательными. В частности, если команда, вызвавшая прерывание, находилась в слоте задержки перехода и переход был выполненным, то необходимо заново повторить выполнение команд из слота задержки плюс команду, находящуюся по целевому адресу команды перехода. Сама команда

перехода уже выполнилась и ее повторения не требуется. При этом адреса команд из слота задержки перехода и целевой адрес команды перехода естественно не являются последовательными. Поэтому необходимо сохранять и восстанавливать несколько счетчиков команд, число которых на единицу превышает длину слота задержки. Это выполняется на третьем шаге обработки прерывания.

После обработки прерывания специальные команды осуществляют возврат из прерывания путем перезагрузки счетчиков команд и инициализации потока команд. Если конвейер может быть остановлен так, что команды, непосредственно предшествовавшие вызвавшей прерывание команде, завершаются, а следовавшие за ней могут быть заново запущены для выполнения, то говорят, что конвейер обеспечивает точное прерывание. В идеале команда, вызывающая прерывание, не должна менять состояние машины, и для корректной обработки некоторых типов прерываний требуется, чтобы команда, вызывающая прерывание, не имела никаких побочных эффектов. Для других типов прерываний, например, для прерываний по исключительным ситуациям плавающей точки, вызывающая прерывание команда на некоторых машинах записывает свои результаты еще до того момента, когда прерывание может быть обработано. В этих случаях аппаратура должна быть готовой для восстановления операндов-источников, даже если местоположение результата команды совпадает с местоположением одного из операндов-источников.

Поддержка точных прерываний во многих системах является обязательным требованием, а в некоторых системах была бы весьма желательной, поскольку она упрощает интерфейс операционной системы. Как минимум в машинах со страничной организацией памяти или с реализацией арифметической обработки в соответствии со стандартом IEEE средства обработки прерываний должны обеспечивать точное прерывание либо целиком с помощью аппаратуры, либо с помощью некоторой поддержки со стороны программных средств.

Необходимость реализации в машине точных прерываний иногда оспаривается из-за некоторых проблем, которые осложняют повторный запуск команд. Повторный запуск сложен из-за того, что команды могут изменить состояние машины еще до того, как они гарантировано завершают свое выполнение (иногда гарантированное завершение команды называется фиксацией команды или фиксацией результатов выполнения команды). Поскольку команды в конвейере могут быть взаимозависимыми, блокировка изменения состояния машины может оказаться непрактичной, если конвейер продолжает работать. Таким образом, по мере увеличения степени конвейеризации машины возникает необходимость отката любого изменения состояния, выполненного до фиксации команды. К счастью, в простых конвейерах, подобных рассмотренному, эти проблемы не возникают. На рисунке 5.3.6.1 показаны ступени рассмотренного конвейера и причины прерываний, которые могут возникнуть на соответствующих ступенях при выполнении команд.

| Ступень конвейера | Причина прерывания |
|---|---|
| IF | Ошибка при обращении к странице памяти при выборке команды; невыровненное обращение к памяти; нарушение защиты памяти |
| ID | Неопределенный или запрещенный код операции |
| EX | Арифметическое прерывание |
| MEM | Ошибка при обращении к странице памяти при выборке данных; невыровненное обращение к памяти; нарушение защиты памяти |
| WB | Отсутствует |

Рис. 5.3.6.1 Причины прерываний в простейшем конвейере

### 5.3.7.Обработка многотактных операций и механизмы обходов в длинных конвейерах

В рассмотренном нами конвейере стадия выполнения команды (EX) составляла всего один такт, что вполне приемлемо для целочисленных операций. Однако для большинства операций плавающей точки было бы непрактично требовать, чтобы все они выполнялись за один или даже за два такта. Это привело бы к существенному увеличению такта синхронизации конвейера, либо к сверхмерному увеличению количества оборудования (объема логических схем) для реализации устройств плавающей точки. Проще всего представить, что команды плавающей точки используют тот же самый конвейер, что и целочисленные команды, но с двумя важными изменениями. Во-первых, такт EX может повторяться многократно столько раз, сколько необходимо для выполнения операции. Во-вторых, в процессоре может быть несколько функциональных устройств, реализующих операции плавающей точки. При этом могут возникать приостановки конвейера, если выданная для выполнения команда либо вызывает структурный конфликт по функциональному устройству, которое она использует, либо существует конфликт по данным.

Допустим, что в нашей реализации процессора имеются четыре отдельных функциональных устройства:

Основное целочисленное устройство.

Устройство умножения целочисленных операндов и операндов с плавающей точкой.

Устройство сложения с плавающей точкой.

Устройство деления целочисленных операндов и операндов с плавающей точкой.

Целочисленное устройство обрабатывает все команды загрузки и записи в память при работе с двумя наборами регистров (целочисленных и с плавающей точкой), все целочисленные операции (за исключением команд умножения и деления) и все команды переходов. Поскольку стадия EX является

неконвейерной, никакая команда, использующая функциональное устройство, не может быть выдана для выполнения до тех пор, пока предыдущая команда не покинет ступень EX. Более того, если команда не может поступить на ступень EX, весь конвейер за этой командой будет приостановлен.
В действительности промежуточные результаты возможно не используются циклически ступенью EX, и ступень EX имеет задержки длительностью более одного такта. Мы можем обобщить структуру конвейера плавающей точки, допустив конвейеризацию некоторых ступеней и параллельное выполнение нескольких операций. Чтобы описать работу такого конвейера, мы должны определить задержки функциональных устройств, а также скорость инициаций или скорость повторения операций. Это скорость, с которой новые операции данного типа могут поступать в функциональное устройство. Например, предположим, что имеют место следующие задержки функциональных устройств и скорости повторения операций:

| Функциональное устройство | Задержка | Скорость повторения |
|---|---|---|
| Целочисленное АЛУ | 1 | 1 |
| Сложение с ПТ | 4 | 2 |
| Умножение с ПТ (и целочисленное) | 6 | 3 |
| Деление с ПТ (и целочисленное) | 15 | 15 |

На рисунке 5.3.7.1 представлена структура подобного конвейера. Ее реализация требует введения конвейерной регистровой станции EX1/EX2 и модификации связей между регистрами ID/EX и EX/MEM.

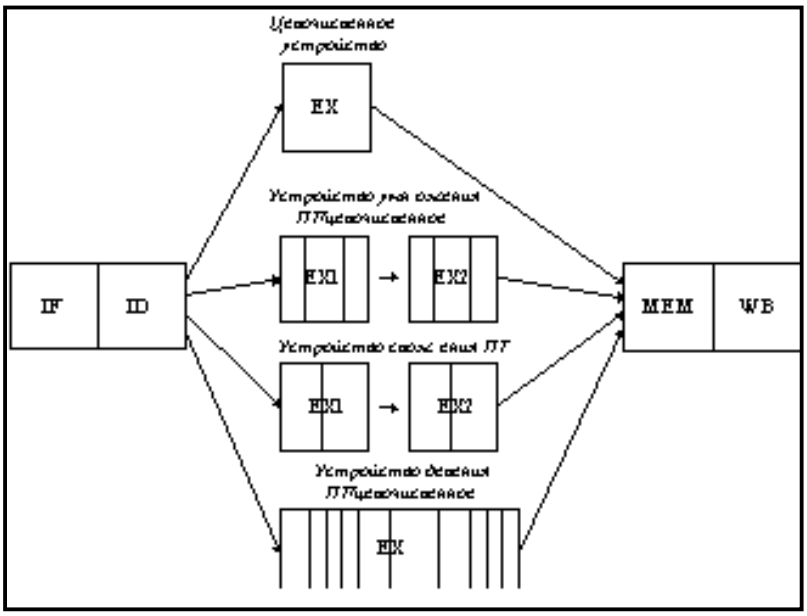


Рис. 5.3.7.1 Конвейер с многоступенчатыми функциональными устройствами

### 5.3.8. Конфликты и ускоренные пересылки в длинных конвейерах

Имеется несколько различных аспектов обнаружения конфликтов и организации ускоренной пересылки данных в конвейерах, подобных представленному на рисунке 5.3.7.1:

Поскольку устройства не являются полностью конвейерными, в данной схеме возможны структурные конфликты. Эти ситуации необходимо обнаруживать и приостанавливать выдачу команд.

Поскольку устройства имеют разные времена выполнения, количество записей в регистровый файл в каждом такте может быть больше 1.

Возможны конфликты типа WAW, поскольку команды больше не поступают на ступень WB в порядке их выдачи для выполнения. Заметим, что конфликты типа WAR невозможны, поскольку чтение регистров всегда осуществляется на ступени ID.

Команды могут завершаться не в том порядке, в котором они были выданы для выполнения, что вызывает проблемы с реализацией прерываний.

Прежде чем представить общее решение для реализации схем обнаружения конфликтов, рассмотрим вторую и третью проблемы.

Если предположить, что файл регистров с ПТ имеет только один порт записи, то последовательность операций с ПТ, а также операция загрузки ПТ совместно с операциями ПТ может вызвать конфликты по порту записи в регистровый файл. Рассмотрим последовательность команд, представленную на рисунке 5.3.8.1. В такте 10 все три команды достигнут ступени WB и должны произвести запись в регистровый файл. При наличии только одного порта записи в регистровый файл машина должна обеспечить последовательное завершение команд. Этот единственный регистровый порт является источником структурных конфликтов. Чтобы решить эту проблему, можно увеличить количество портов в регистровом файле, но такое решение может оказаться неприемлемым, поскольку эти дополнительные порты записи скорее всего будут редко использоваться. Однако в установившемся состоянии максимальное количество необходимых портов записи равно 1. Поэтому в реальных машинах разработчики предпочитают отслеживать обращения к порту записи в регистры и рассматривать одновременное к нему обращение как структурный конфликт.

| Номер команды | Номер такта | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| MULTD F0,F4,F6 | IF | ID | EX11 | EX12 | EX13 | EX21 | EX22 | EX23 | MEM | WB |
| ... | | IF | ID | EX | MEM | WB | | | | |
| ADDD F2,F4,F6 | | | IF | ID | EX1 | EX12 | EX21 | EX22 | MEM | WB |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| ... | | | | IF | ID | EX | MEM | WB | | |
| ... | | | | | IF | ID | EX | MEM | WB | |
| LD F8,0(R2) | | | | | | IF | ID | EX | MEM | WB |

Рис. 5.3.8.1. Пример конфликта по записи в регистровый файл

Имеется два способа для обхода этого конфликта. Первый заключается в отслеживании использования порта записи на ступени ID конвейера и приостановке выдачи команды как при структурном конфликте. Схема обнаружения такого конфликта обычно реализуется с помощью сдвигового регистра. Альтернативная схема предполагает приостановку конфликтующей команды, когда она пытается попасть на ступень MEM конвейера. Преимуществом такой схемы является то, что она не требует обнаружения конфликта до входа на ступень MEM, где это легче сделать. Однако подобная реализация усложняет управление конвейером, поскольку приостановки в этом случае могут возникать в двух разных местах конвейера.

Другой проблемой является возможность конфликтов типа WAW. Можно рассмотреть тот же пример, что и на рисунке 5.3.8.1. Если бы команда LD была выдана на один такт раньше и имела в качестве месторасположения результата регистр F2, то возник бы конфликт типа WAW, поскольку эта команда выполняла бы запись в регистр F2 на один такт раньше команды ADDD. Имеются два способа обработки этого конфликта типа WAW. Первый подход заключается в задержке выдачи команды загрузки до момента передачи команды ADDD на ступень MEM. Второй подход заключается в подавлении результата операции сложения при обнаружении конфликта и изменении управления таким образом, чтобы команда сложения не записывала свой результат. Тогда команда LD может выдаваться для выполнения сразу же. Поскольку такой конфликт является редким, обе схемы будут работать достаточно хорошо. В любом случае конфликт может быть обнаружен на ранней стадии ID, когда команда LD выдается для выполнения. Тогда приостановка команды LD или установка блокировки записи результата командой ADDD реализуются достаточно просто.

Таким образом, для обнаружения возможных конфликтов необходимо рассматривать конфликты между командами ПТ, а также конфликты между командами ПТ и целочисленными командами. За исключением команд загрузки/записи с ПТ и команд пересылки данных между регистрами ПТ и целочисленными регистрами, команды ПТ и целочисленные команды достаточно хорошо разделены, и все целочисленные команды работают с целочисленными регистрами, а команды ПТ - с регистрами ПТ. Таким образом, для обнаружения конфликтов между целочисленными командами и командами

ПТ необходимо рассматривать только команды загрузки/записи с ПТ и команды пересылки регистров ПТ. Это упрощение управления конвейером является дополнительным преимуществом поддержания отдельных регистровых файлов для хранения целочисленных данных и данных с ПТ. (Главное преимущество заключается в удвоении общего количества регистров и увеличении пропускной способности без увеличения числа портов в каждом наборе). Если предположить, что конвейер выполняет обнаружение всех конфликтов на стадии ID, перед выдачей команды для выполнения в функциональные устройства должны быть выполнены три проверки:
Проверка наличия структурных конфликтов. Ожидание освобождения функционального устройства и порта записи в регистры, если он потребуется.
Проверка наличия конфликтов по данным типа RAW. Ожидание до тех пор, пока регистры-источники операндов указаны в качестве регистров результата на конвейерных станциях ID/EX (которая соответствует команде, выданной в предыдущем такте), EX1/EX2 или EX/MEM.
Проверка наличия конфликтов типа WAW. Проверка того, что команды, находящиеся на конвейерных станциях EX1 и EX2, не имеют в качестве месторасположения результата регистр результата выдаваемой для выполнения команды. В противном случае выдача команды, находящейся на ступени ID, приостанавливается.
Хотя логика обнаружения конфликтов для многотактных операций ПТ несколько более сложная, концептуально она не отличается от такой же логики для целочисленного конвейера. То же самое касается логики для ускоренной пересылки данных. Логика ускоренной пересылки данных может быть реализована с помощью проверки того, что указанный на конвейерных станциях EX/MEM и MEM/WB регистр результата является регистром операнда команды ПТ. Если происходит такое совпадение, для пересылки данных разрешается прием по соответствующему входу мультиплексора. Многотактные операции ПТ создают также новые проблемы для механизма прерывания.

### 5.3.9.Поддержка точных прерываний

Другая проблема, связанная с реализацией команд с большим временем выполнения, может быть проиллюстрирована с помощью следующей последовательности команд:

```
DIVF F0,F2,F4
ADDF F10,F10,F8
SUBF F12,F12,F14
```

Эта последовательность команд выглядит очень просто. В ней отсутствуют какие-либо зависимости. Однако она приводит к появлению новых проблем из-за того, что выданная раньше команда может завершиться после команды, выданной для выполнения позже. В данном примере можно ожидать, что команды ADDF и SUBF завершаться раньше, чем завершится команда DIVF.

Этот эффект является типичным для конвейеров команд с большим временем выполнения и называется внеочередным завершением команд (out-of-order completion). Тогда, например, если команда DIVF вызовет арифметическое прерывание после завершения команды ADDF, мы не сможем реализовать точное прерывание на уровне аппаратуры. В действительности, поскольку команда ADDF меняет значение одного из своих операндов, невозможно даже с помощью программных средств восстановить состояние, которое было перед выполнением команды DIVF.

Имеются четыре возможных подхода для работы в условиях внеочередного завершения команд. Первый из них просто игнорирует проблему и предлагает механизмы неточного прерывания. Этот подход использовался в 60-х и 70-х годах и все еще применяется в некоторых суперкомпьютерах, в которых некоторые классы прерываний запрещены или обрабатываются аппаратурой без остановки конвейера. Такой подход трудно использовать в современных машинах при наличии концепции виртуальной памяти и стандарта на операции с плавающей точкой IEEE, которые требуют реализации точного прерывания путем комбинации аппаратных и программных средств. В некоторых машинах эта проблема решается путем введения двух режимов выполнения команд: быстрого, но с возможно не точными прерываниями, и медленного, гарантирующего реализацию точных прерываний.

Второй подход заключается в буферизации результатов операции до момента завершения выполнения всех команд, предшествовавших данной. В некоторых машинах используется этот подход, но он становится все более дорогостоящим, если отличия во времени выполнения разных команд велики, поскольку становится большим количество результатов, которые необходимо буферизовать. Более того, результаты из этой буферизованной очереди необходимо пересылать для обеспечения продолжения выдачи новых команд. Это требует большого количества схем сравнения и многовходовых мультиплексоров. Имеются две вариации этого основного подхода. Первая называется буфером истории (history file), использовавшемся в машине CYBER 180/990. Буфер истории отслеживает первоначальные значения регистров. Если возникает прерывание и состояние машины необходимо откатить назад до точки, предшествовавшей некоторым завершившимся вне очереди командам, то первоначальное значение регистров может быть восстановлено из этого буфера истории. Подобная методика использовалась также при реализации автоинкрементной и автодекрементной адресации в машинах типа VAX. Другой подход называется буфером будущего (future file). Этот буфер хранит новые значения регистров. Когда все предшествующие команды завершены, основной регистровый файл обновляется значениями из этого буфера. При прерывании основной регистровый файл хранит точные значения регистров, что упрощает организацию прерывания. В следующей главе будут рассмотрены некоторые расширения этой идеи.

Третий используемый метод заключается в том, чтобы разрешить в ряде случаев неточные прерывания, но при этом сохранить достаточно информации, чтобы подпрограмма обработки прерывания могла выполнить точную последовательность прерывания. Это предполагает наличие информации о находившихся в конвейере командах и их адресов. Тогда после обработки прерывания, программное обеспечение завершает выполнение всех команд, предшествовавших последней завершившейся команде, а затем последовательность может быть запущена заново. Рассмотрим следующий наихудший случай:
Команда 1 - длинная команда, которая в конце концов вызывает прерывание
Команда 2, ... , Команда n-1 - последовательность команд, выполнение которых не завершилось
Команда n - команда, выполнение которой завершилось
Имея значения адресов всех команд в конвейере и адрес возврата из прерывания, программное обеспечение может определить состояние команды 1 и команды n. Поскольку команда n завершила выполнение, хотелось бы продолжить выполнение с команды n+1. После обработки прерывания программное обеспечение должно смоделировать выполнение команд с 1 по n-1. Тогда можно осуществить возврат из прерывания на команду n+1. Наибольшая неприятность такого подхода связана с усложнением подпрограммы обработки прерывания. Но для простых конвейеров, подобных рассмотренному нами, имеются и упрощения. Если команды с 2 по n все являются целочисленными, то мы просто знаем, что в случае завершения выполнения команды n, все команды с 2 по n-1 также завершили выполнение. Таким образом, необходимо обрабатывать только операцию с плавающей точкой. Чтобы сделать эту схему работающей, количество операций ПТ, выполняющихся с совмещением, может быть ограничено. Например, если допускается совмещение только двух операций, то только прерванная команда должна завершаться программными средствами. Это ограничение может снизить потенциальную пропускную способность, если конвейеры плавающей точки являются достаточно длинными или если имеется значительное количество функциональных устройств. Такой подход использовался в архитектуре SPARC, позволяющей совмещать выполнение целочисленных операций с операциями плавающей точки.
Четвертый метод представляет собой гибридную схему, которая позволяет продолжать выдачу команд только если известно, что все команды, предшествовавшие выдаваемой, будут завершены без прерывания. Это гарантирует, что в случае возникновения прерывания ни одна следующая за ней команда не будет завершена, а все предшествующие будут завершены. Иногда это означает необходимость приостановки машины для поддержки точных прерываний. Чтобы эта схема работала, необходимо, чтобы функциональные устройства плавающей точки определяли возможность появления прерывания на самой ранней стадии выполнения команд так, чтобы предотвратить

завершение выполнения следующих команд. Такая схема используется, например, в микропроцессорах R2000/R3000 и R4000 компании MIPS.

## 5.4.Конвейерная и суперскалярная обработка

### 5.4.1.Параллелизм на уровне выполнения команд, планирование загрузки конвейера и методика разворачивания циклов

В предыдущем разделе мы рассмотрели средства конвейеризации, которые обеспечивают совмещенный режим выполнения команд, когда они являются независимыми друг от друга. Это потенциальное совмещение выполнения команд называется параллелизмом на уровне команд. В данном разделе мы рассмотрим ряд методов развития идей конвейеризации, основанных на увеличении степени параллелизма, используемой при выполнении команд. Мы начнем с рассмотрения методов, позволяющих снизить влияние конфликтов по данным и по управлению, а затем вернемся к теме расширения возможностей процессора по использованию параллелизма, заложенного в программах.

Для начала запишем выражение, определяющее среднее количество тактов для выполнения команды в конвейере:

CPI конвейера = CPI идеального конвейера +
+ Приостановки из-за структурных конфликтов +
+ Приостановки из-за конфликтов типа RAW +
+ Приостановки из-за конфликтов типа WAR +
+ Приостановки из-за конфликтов типа WAW +
+ Приостановки из-за конфликтов по управлению

CPI идеального конвейера есть не что иное, как максимальная пропускная способность, достижимая при реализации. Уменьшая каждое из слагаемых в правой части выражения, мы минимизируем общий CPI конвейера и таким образом увеличиваем пропускную способность команд.

Прежде, чем начать рассмотрение этих методов, необходимо определить концепции, на которых эти методы построены.

### 5.4.2.Параллелизм уровня команд: зависимости и конфликты по данным

Параллелизм, заложенный в последовательности команд, называется параллелизмом уровня команд или ILP. Степень параллелизма, доступная внутри базового блока (линейной последовательности команд, переходы из вне которой разрешены только на ее вход, а переходы внутри которой разрешены только на ее выход) достаточно мала. Например, средняя частота переходов в целочисленных программах составляет около 16%. Это означает, что в среднем между двумя переходами выполняются примерно пять команд. Поскольку эти пять команд возможно взаимозависимые, то степень перекрытия, которую мы можем использовать внутри базового блока, возможно будет меньше чем пять. Чтобы получить существенное улучшение производительности, мы должны использовать параллелизм уровня команд одновременно для нескольких базовых блоков.

| Метод | Снижает |
|---|---|
| Разворачивание циклов | Приостановки по управлению |
| Базовое планирование конвейера | Приостановки RAW |
| Динамической планирование с централизованной схемой управления | Приостановки RAW |
| Динамическое планирование с переименованием регистров | Приостановки WAR и WAW |
| Динамическое прогнозирование переходов | Приостановки по управлению |
| Выдача нескольких команд в одном такте | Идеальный CPI |
| Анализ зависимостей компилятором | Идеальный CPI и приостановки по данным |
| Программная конвейеризация и планирование трасс | Идеальный CPI и приостановки по данным |
| Выполнение по предположению | Все приостановки по данным и управлению |
| Динамическое устранение неоднозначности памяти | Приостановки RAW, связанные с памятью |

Рис. 5.4.2.1.

Самый простой и общий способ увеличения степени параллелизма, доступного на уровне команд, является использование параллелизма между итерациями цикла. Этот тип параллелизма часто называется параллелизмом уровня итеративного цикла. Ниже приведен простой пример цикла, выполняющего сложение двух 1000-элементных векторов, который является полностью параллельным:

```
for (i = 1; i <= 1000; i = i + 1)
x[i] = x[i] + y[i];
```

Каждая итерация цикла может перекрываться с любой другой итерацией, хотя внутри каждой итерации цикла практическая возможность перекрытия небольшая.

Имеется несколько методов для превращения такого параллелизма уровня цикла в параллелизм уровня команд. Эти методы основаны главным образом на разворачивании цикла либо статически, используя компилятор, либо динамически с помощью аппаратуры.

Важным альтернативным методом использования параллелизма уровня команд является использование векторных команд. По существу векторная команда оперирует с последовательностью элементов данных. Например, приведенная выше последовательность на типичной векторной машине может быть выполнена с помощью четырех команд: двух команд загрузки векторов x и y из памяти, одной команды сложения двух векторов и одной команды записи

вектора-результата. Конечно, эти команды могут быть конвейеризованными и иметь относительно большие задержки выполнения, но эти задержки могут перекрываться. Векторные команды и векторные машины заслуживают отдельного рассмотрения, которое выходит за рамки данного курса. Хотя разработка идей векторной обработки предшествовала появлению большинства методов использования параллелизма, машины, использующие параллелизм уровня команд постепенно заменяют машины, базирующиеся на векторной обработке.

### 5.4.2.1.Основы планирования загрузки конвейера и разворачивание циклов

Для поддержания максимальной загрузки конвейера должен использоваться параллелизм уровня команд, основанный на выявлении последовательностей несвязанных команд, которые могут выполняться в конвейере с совмещением. Чтобы избежать приостановки конвейера зависимая команда должна быть отделена от исходной команды на расстояние в тактах, равное задержке конвейера для этой исходной команды. Способность компилятора выполнять подобное планирование зависит как от степени параллелизма уровня команд, доступного в программе, так и от задержки функциональных устройств в конвейере. Будем предполагать задержки, показанные на рисунке 5.4.2.1.1, если только явно не установлены другие задержки. Мы предполагаем, что условные переходы имеют задержку в один такт, так что команда следующая за командой перехода не может быть определена в течение одного такта после команды условного перехода. Мы предполагаем, что функциональные устройства полностью конвейеризованы или дублированы (столько раз, какова глубина конвейера), так что операция любого типа может выдаваться для выполнения в каждом такте и структурные конфликты отсутствуют.

| Команда, вырабатывающая результат | Команда, использующая результат | Задержка в тактах |
|---|---|---|
| Операция АЛУ с ПТ | Другая операция АЛУ с ПТ | 3 |
| Операция АЛУ с ПТ | Запись двойного слова | 2 |
| Загрузка двойного слова | Другая операция АЛУ с ПТ | 1 |
| Загрузка двойного слова | Запись двойного слова | 0 |

Рис. 5.4.2.1.1

Компилятор может увеличить степень параллелизма уровня команд путем разворачивания циклов. Для иллюстрации этих методов используем простой цикл, который добавляет скалярную величину к вектору в памяти; это параллельный цикл, поскольку зависимость между итерациями цикла отсутствует. Предполлжим, что первоначально в регистре R1 находится адрес последнего элемента вектора (например, элемент с наибольшим адресом), а в регистре F2 - скалярная величина, которая должна добавляться к каждому

элементу вектора. Программа для машины, не рассчитанная на использование конвейера, будет выглядеть примерно так:

```
Loop:  LD    F0,0(R1)       ;F0=элемент вектора
       ADDD F4,F0,F2        ;добавляет скаляр из F2
       SD 0(R1),F4          ;запись результата
       SUBI R1,R1,#8        ;пересчитать указатель
                            ;8 байт (в двойном слове)
       BNEZ R1, Loop        ;переход R1!=нулю
```

Для упрощения предположим, что массив начинается с ячейки 0. Если бы он находился в любом другом месте, цикл потребовал бы наличия одной дополнительной целочисленной команды для выполнения сравнения с регистром R1.

Рассмотрим работу этого цикла при выполнении на простом конвейере с задержками, показанными на рисунке 5.4.2.1.1.

Если не делать никакого планирования, работа цикла будет выглядеть следующим образом:

| | Такт выдачи |
|---|---|
| Loop: LD F0,0(R1) | 1 |
| Приостановка | 2 |
| ADDD F4,F0,F2 | 3 |
| Приостановка | 4 |
| Приостановка | 5 |
| SD 0(R1),F4 | 6 |
| SUBI R1,R1,#8 | 7 |
| BNEZ R1,Loop | 8 |
| Приостановка | 9 |

Для его выполнения потребуется 9 тактов на итерацию: одна приостановка для команды LD, две для команды ADDD, и одна для задержанного перехода. Мы можем спланировать цикл так, чтобы получить

| | Такт выдачи |
|---|---|
| Loop: LD F0,0(R1) | 1 |
| Приостановка | 2 |
| ADDD F4,F0,F2 | 3 |
| SUBI R1,R1,#8 | 4 |
| BNEZ R1,Loop;задержанный переход | 5 |
| SD 8(R1),F4 | 6 |

Время выполнения уменьшилось с 9 до 6 тактов.

Заметим, что для планирования задержанного перехода компилятор должен определить, что он может поменять местами команды SUB1 и SD путем изменения адреса в команде записи SD: Адрес был равен 0(R1), а теперь равен 8(R1). Это не тривиальная задача, поскольку большинство компиляторов будут

видеть, что команда SD зависит от SUB1, и откажутся от такой перестановки мест. Более изощренный компилятор смог бы рассчитать отношения и выполнить перестановку. Цепочка зависимостей от команды LD к команде ADDD и далее к команде SD определяет количество тактов, необходимое для данного цикла.

В вышеприведенном примере мы завершаем одну итерацию цикла и выполняем запись одного элемента вектора каждые 6 тактов, но действительная работа по обработке элемента вектора отнимает только 3 из этих 6 тактов (загрузка, сложение и запись). Оставшиеся 3 такта составляют накладные расходы на выполнение цикла (команды SUB1, BNEZ и приостановка). Чтобы устранить эти три такта нам нужно иметь больше операций в цикле относительно числа команд, связанных с накладными расходами. Одним из наиболее простых методов увеличения числа команд по отношению к команде условного перехода и команд, связанных с накладными расходами, является разворачивание цикла. Такое разворачивание выполняется путем многократной репликации (повторения) тела цикла и коррекции соответствующего кода конца цикла.

Разворачивание циклов может также использоваться для улучшения планирования. В этом случае, мы можем устранить приостановку, связанную с задержкой команды загрузки путем создания дополнительных независимых команд в теле цикла. Затем компилятор может планировать эти команды для помещения в слот задержки команды загрузки. Если при разворачивании цикла мы просто реплицируем команды, то результирующие зависимости по именам могут помешать нам эффективно спланировать цикл. Таким образом, для разных итераций хотелось бы использовать различные регистры, что увеличивает требуемое число регистров.

Представим теперь этот цикл развернутым так, что имеется четыре копии тела цикла, предполагая, что R1 первоначально кратен 4. Устраним при этом любые очевидные излишние вычисления и не будем пользоваться повторно никакими регистрами.

Ниже приведен результат, полученный путем слияния команд SUB1 и выбрасывания ненужных операций BNEZ, которые дублируются при разворачивании цикла.

```
Loop:  LD    F0,0(R1)
       ADDD F4,F0,F2
       SD 0(R1),F4            ;выбрасывается SUB1 и BNEZ
       LD F6,-8(R1)
       ADDD F8,F6,F2
       SD -8(R1),F8           ;выбрасывается SUB1 и BNEZ
       LD F10,-16(R1)
       ADDD F12,F10,F2
       SD -16(R1),F12         ;выбрасывается SUB1 и BNEZ
       LD F14,-24(R1)
```

```
        ADDD F16,F14,F2
        SD -24(R1),F16
        SUB1 R1,R1,#32
        BNEZ R1, Loop
```

Мы ликвидировали три условных перехода и три операции декрементирования R1. Адреса команд загрузки и записи были скорректированы так, чтобы позволить слить команды SUB1 в одну команду по регистру R1. При отсутствии планирования за каждой командой здесь следует зависимая команда и это будет приводить к приостановкам конвейера. Этот цикл будет выполняться за 27 тактов (на каждую команду LD потребуется 2 такта, на каждую команду ADDD - 3, на условный переход - 2 и на все другие команды 1 такт) или по 6.8 такта на каждый из четырех элементов. Хотя эта развернутая версия в такой редакции медленнее, чем оптимизированная версия исходного цикла, после оптимизации самого развернутого цикла ситуация изменится. Обычно разворачивание циклов выполняется на более ранних стадиях процесса компиляции, так что избыточные вычисления могут быть выявлены и устранены оптимизатором.

В реальных программах мы обычно не знаем верхней границы цикла. Предположим, что она равна n и мы хотели бы развернуть цикл так, чтобы иметь k копий тела цикла. Вместо единственного развернутого цикла мы генерируем пару циклов. Первый из них выполняется (n mod k) раз и имеет тело первоначального цикла. Развернутая версия цикла окружается внешним циклом, который выполняется (n div k) раз.

В вышеприведенном примере разворачивание цикла увеличивает производительность этого цикла путем устранения команд, связанных с накладными расходами цикла, хотя оно заметно увеличивает размер программного кода. Насколько увеличится производительность, если цикл будет оптимизироваться?

Ниже представлен развернутый цикл из предыдущего примера после оптимизации.

```
Loop:   LD    F0,0(R1)
        LD F6,-8(R1)
        LD F10,-16(R1)
        LD F14,-24(R1)
        ADDD F4,F0,F2
        ADDD F8,F6,F2
        ADDD F12,F10,F2
        ADDD F16,F14,F2
        SD 0(R1),F4
        SD -8(R1),F8
        SD -16(R1),F12
        SUB1 R1,R1,#32
        BNEZ R1, Loop
```

SD 8(R1),F16 ; 8 - 32 = -24

Время выполнения развернутого цикла снизилось до 14 тактов или до 3.5 тактов на элемент, по сравнению с 6.8 тактов на элемент до оптимизации, и по сравнению с 6 тактами при оптимизации без разворачивания цикла.

Выигрыш от оптимизации развернутого цикла даже больше, чем от оптимизации первоначального цикла. Это произошло потому, что разворачивание цикла выявило больше вычислений, которые могут быть оптимизированы для минимизации приостановок конвейера; приведенный выше программный код выполняется без приостановок. При подобной оптимизации цикла необходимо осознавать, что команды загрузки и записи являются независимыми и могут чередоваться. Анализ зависимостей по данным позволяет нам определить, являются ли команды загрузки и записи независимыми.

Разворачивание циклов представляет собой простой, но полезный метод увеличения размера линейного кодового фрагмента, который может эффективно оптимизироваться. Это преобразование полезно на множестве машин от простых конвейеров, подобных рассмотренному ранее, до суперскалярных конвейеров, которые обеспечивают выдачу для выполнения более одной команды в такте. В следующем разделе рассмотрены методы, которые используются аппаратными средствами для динамического планирования загрузки конвейера и сокращения приостановок из-за конфликтов типа RAW, аналогичные рассмотренным выше методам компиляции.

### 5.4.2.2.Устранение зависимостей по данным и механизмы динамического планирования

Главным ограничением методов конвейерной обработки, которые мы рассматривали ранее, является выдача для выполнения команд строго в порядке, предписанном программой: если выполнение какой-либо команды в конвейере приостанавливалось, следующие за ней команды также приостанавливались. Таким образом, при наличии зависимости между двумя близко расположенными в конвейере командами возникала приостановка обработки многих команд. Но если имеется несколько функциональных устройств, многие из них могут оказаться незагруженными. Если команда j зависит от длинной команды i, выполняющейся в конвейере, то все команды, следующие за командой j должны приостановиться до тех пор, пока команда i не завершится и не начнет выполняться команда j. Например, рассмотрим следующую последовательность команд:

DIVD F0,F2,F4
ADDD F10,F0,F8
SUBD F8,F8,F14

Команда SUBD не может выполняться из-за того, что зависимость между командами DIVD и ADDD привела к приостановке конвейера. Однако команда

SUBD не имеет никаких зависимостей от команд в конвейере. Это ограничение производительности, которое может быть устранено снятием требования о выполнении команд в строгом порядке.
В рассмотренном нами конвейере структурные конфликты и конфликты по данным проверялись во время стадии декодирования команды (ID). Если команда могла нормально выполняться, она выдавалась с этой ступени конвейера в следующие. Чтобы позволить начать выполнение команды SUBD из предыдущего примера, необходимо разделить процесс выдачи на две части: проверку наличия структурных конфликтов и ожидание отсутствия конфликта по данным. Когда мы выдаем команду для выполнения, мы можем осуществлять проверку наличия структурных конфликтов; таким образом, мы все еще используем упорядоченную выдачу команд. Однако мы хотим начать выполнение команды как только станут доступными ее операнды. Таким образом, конвейер будет осуществлять неупорядоченное выполнение команд, которое означает и неупорядоченное завершение команд.
Неупорядоченное завершение команд создает основные трудности при обработке исключительных ситуаций. В рассматриваемых в данном разделе машинах с динамическим планированием потока команд прерывания будут неточными, поскольку команды могут завершиться до того, как выполнение более ранней выданной команды вызовет исключительную ситуацию. Таким образом, очень трудно повторить запуск после прерывания
Чтобы реализовать неупорядоченное выполнение команд, мы расщепляем ступень ID на две ступени:
Выдача - декодирование команд, проверка структурных конфликтов.
Чтение операндов - ожидание отсутствия конфликтов по данным и последующее чтение операндов.
Затем, как и в рассмотренном нами конвейере, следует ступень EX. Поскольку выполнение команд ПТ может потребовать нескольких тактов в зависимости от типа операции, мы должны знать, когда команда начинает выполняться и когда заканчивается. Это позволяет нескольким командам выполняться в один и тот же момент времени. В дополнение к этим изменениям структуры конвейера мы изменим и структуру функциональных устройств, варьируя количество устройств, задержку операций и степень конвейеризации функциональных устройств так, чтобы лучше использовать эти методы конвейеризации.

### 5.4.2.3.Динамическая оптимизация с централизованной схемой обнаружения конфликтов

В конвейере с динамическим планированием выполнения команд все команды проходят через ступень выдачи строго в порядке, предписанном программой (упорядоченная выдача). Однако они могут приостанавливаться и обходить друг друга на второй ступени (ступени чтения операндов) и тем самым поступать на ступени выполнения неупорядочено. Централизованная схема обнаружения конфликтов представляет собой метод, допускающий

неупорядоченное выполнение команд при наличии достаточных ресурсов и отсутствии зависимостей по данным. Впервые подобная схема была применена в компьютере CDC 6600.

Прежде чем начать обсуждение возможности применения подобных схем, важно заметить, что конфликты типа WAR, отсутствующие в простых конвейерах, могут появиться при неупорядоченном выполнении команд. В ранее приведенном примере регистром результата для команды SUBD является регистр R8, который одновременно является источником операнда для команды ADDD. Поэтому здесь между командами ADDD и SUBD имеет место антизависимость: если конвейер выполнит команду SUBD раньше команды ADDD, он нарушит эту антизависимость. Этот конфликт WAR можно обойти, если выполнить два правила: (1) читать регистры только во время стадии чтения операндов и (2) поставить в очередь операцию ADDD вместе с копией ее операндов. Чтобы избежать нарушений зависимостей по выходу конфликты типа WAW (например, это могло произойти, если бы регистром результата команды SUBD была бы регистр F10) все еще должны обнаруживаться. Конфликты типа WAW могут быть устранены с помощью приостановки выдачи команды, регистр результата которой совпадает с уже используемым в конвейере.

Задачей централизованной схемы обнаружения конфликтов является поддержание выполнения команд со скоростью одна команда за такт (при отсутствии структурных конфликтов) посредством как можно более раннего начала выполнения команд. Таким образом, когда команда в начале очереди приостанавливается, другие команды могут выдаваться и выполняться, если они не зависят от уже выполняющейся или приостановленной команды. Централизованная схема несет полную ответственность за выдачу и выполнение команд, включая обнаружение конфликтов. Подобное неупорядоченное выполнение команд требует одновременного нахождения нескольких команд на стадии выполнения. Этого можно достигнуть двумя способами: реализацией в процессоре либо множества неконвейерных функциональных устройств, либо путем конвейеризации всех функциональных устройств. Обе эти возможности по сути эквивалентны с точки зрения организации управления.Поэтому предположим, что в машине имеется несколько неконвейерных функциональных устройств.

В нашем случае централизованная схема обнаружения конфликтов имеет смысл только для устройства плавающей точки. Предположим, что имеются два умножителя, один сумматор, одно устройство деления и одно целочисленное устройство для всех операций обращения к памяти, переходов и целочисленных операций. Каждая команда проходит через централизованную схему обнаружения конфликтов, которая определяет зависимости по данным; этот шаг соответствует стадии выдачи команд и заменяет часть стадии ID в нашем конвейере. Эти зависимости определяют затем моменты времени, когда команда может читать свои операнды и начинать выполнение операции. Если

централизованная схема решает, что команда не может немедленно выполняться, она следит за всеми изменениями в аппаратуре и решает, когда команда сможет выполняться. Эта же централизованная схема определяет также когда команда может записать результат в свой регистр результата. Таким образом, все схемы обнаружения и разрешения конфликтов здесь выполняются устройством центрального управления.

Каждая команда проходит четыре стадии своего выполнения. (Поскольку в данный момент мы интересуемся операциями плавающей точки, мы не рассматриваем стадию обращения к памяти). Рассмотрим эти стадии сначала неформально, а затем детально рассмотрим как централизованная схема поддерживает необходимую информацию, которая определяет обработку при переходе с одной стадии на другую. Следующие четыре стадии заменяют стадии ID, EX и WB в стандартном конвейере:

Выдача. Если функциональное устройство, необходимое для выполнения команды, свободно и никакая другая выполняющаяся команда не использует тот же самый регистр результата, централизованная схема выдает команду в функциональное устройство и обновляет свою внутреннюю структуру данных. Поскольку никакое другое работающее функциональное устройство не может записать результат в регистр результата нашей команды, мы гарантируем, что конфликты типа WAW не могут появляться. Если существует структурный конфликт или конфликт типа WAW, выдача команды блокируется и никакие следующие команды не будут выдаваться на выполнение до тех пор, пока эти конфликты существуют. Эта стадия заменяет часть стадии ID в нашем конвейере.

Чтение операндов. Централизованная схема следит за возможностью выборки источников операндов для соответствующей команды. Операнд-источник доступен, если отсутствует выполняющаяся команда, которая записывает результат в этот регистр или если в данный момент времени в регистр, содержащий операнд, выполняется запись из работающего функционального устройства. Если операнды-источники доступны, централизованная схема сообщает функциональному устройству о необходимости чтения операндов из регистров и начале выполнения операции. Централизованная схема разрешает конфликты RAW на этой стадии динамически и команды могут посылаться для выполнения не в порядке, предписанном программой. Эта стадия, совместно со стадией выдачи, завершает работу стадии ID простого конвейера.

Выполнение. Функциональное устройство начинает выполнение операции после получения операндов. Когда результат готов оно уведомляет централизованную схему управления о том, что оно завершило выполнение операции. Эта стадия заменяет стадию EX и занимает несколько тактов в рассмотренном ранее конвейере.

Запись результата. Когда централизованная схема управления узнает о том, что функциональное устройство завершило выполнение операции, она проверяет существование конфликта типа WAR. Конфликт типа WAR существует, если

имеется последовательность команд, аналогичная представленной в нашем примере с командами ADDF и SUBF. В том примере мы имели следующую последовательность команд:
DIVF F0,F2,F4
ADDF F10,F0,F8
SUBF F8,F8,F14
Команда ADDF имеет операнд-источник F8, который является тем же самым регистром, что и регистр результата команды SUBF. Но в действительности команда ADDF зависит от предыдущей команды. Централизованная схема управления будет блокировать выдачу команды SUBF до тех пор, пока команда ADDF не прочитает свои операнды. Тогда в общем случае завершающейся команде не разрешается записывать свои результаты если:
имеется команда, которая не прочитала свои операнды,
один из операндов является регистром результата завершающейся команды.
Если этот конфликт типа WAR не существует, централизованная схема управления сообщает функциональному устройству о необходимости записи результата в регистр назначения. Эта стадия заменяет стадию WB в простом конвейере.
Основываясь на своей собственной структуре данных, централизованная схема управления управляет продвижением команды с одной ступени на другую, взаимодействуя с функциональными устройствами. Но имеется небольшое усложнение: в регистровом файле есть только ограниченное число магистралей для операндов-источников и магистралей для записи результата. Централизованная схема управления должна гарантировать, что количество функциональных устройств, которым разрешено продолжать работу на ступенях 2 и 4 не превышает числа доступных шин.
Централизованная схема управления не обрабатывает несколько ситуаций. Например, когда команда записывает свой результат, зависимая команда в конвейере должна дожидаться разрешения обращения к регистровому файлу, поскольку все результаты всегда записываются в регистровый файл и никогда не используется методика "ускоренной пересылки". Это увеличивает задержку и ограничивает возможность инициирования нескольких команд, ожидающих результата.

### 5.4.2.4. Другой подход к динамическому планированию - алгоритм Томасуло

Другой подход к параллельному выполнению команд при наличии конфликтов был использован в устройстве плавающей точки в машине IBM 360/91. Эта схема приписывается Р. Томасуло и названа его именем. Разработка IBM 360/91 была завершена спустя три года после выпуска CDC 6600, прежде чем кэш-память появилась в коммерческих машинах. Задачей IBM было достижение высокой производительности на операциях с плавающей точкой, используя набор команд и компиляторы, разработанные для всего семейства 360, а не

только для приложений с интенсивным использованием плавающей точки. Архитектура 360 предусматривала только четыре регистра плавающей точки двойной точности, что ограничивало эффективность планирования кода компилятором. Этот факт был другой мотивацией подхода Томасуло. Наконец, машина IBM 360/91 имела большое время обращения к памяти и большие задержки выполнения операций плавающей точки, преодолеть которые и был призван разработанный Томасуло алгоритм. Алгоритм Томасуло может также поддерживать совмещенное выполнение нескольких итераций цикла.

Поясним этот алгоритма на примере устройства ПТ. Основное различие между нашим конвейером ПТ и конвейером машины IBM/360 заключается в наличии в последней машине команд типа регистр-память. Поскольку алгоритм Томасуло использует функциональное устройство загрузки, не требуется значительных изменений, чтобы добавить режимы адресации регистр-память; основное добавление - другая шина. IBM 360/91 имела также конвейерные функциональные устройства, а не несколько функциональных устройств. Единственное отличие между ними заключается в том, что конвейерное функциональное устройство может начинать выполнение только одной операции в каждом такте. IBM 360/91 могла выполнять одновременно три операции сложения ПТ и две операции умножения ПТ. Кроме того, в процессе выполнения могли находиться до 6 операций загрузки ПТ, или обращений к памяти, и до трех операций записи ПТ. Для реализации этих функций использовались буфера данных загрузки и буфера данных записи. Схема Томасуло имеет много общего со схемой централизованного управления CDC 6600, однако имеются и существенные отличия. Во-первых, обнаружение конфликтов и управление выполнением являются распределенными - станции резервирования (reservation stations) в каждом функциональном устройстве определяют, когда команда может начать выполняться в данном функциональном устройстве. В CDC 6600 эта функция централизована. Во-вторых, результаты операций посылаются прямо в функциональные устройства, а не проходят через регистры. В IBM 360/91 имеется общая шина результатов операций (которая называется общей шиной данных (common data bus - CDB)), которая позволяет производить одновременную загрузку всех устройств, ожидающих операнда. CDC 6600 записывает результаты в регистры, за которые ожидающие функциональные устройства могут соперничать. Кроме того, CDC 6600 имеет несколько шин завершения операций (две в устройстве ПТ), а IBM 360/91 - только одну.

В устройствах ПТ на базе алгоритма Томасуло станции резервирования хранят команды, которые выданы и ожидают выполнения в соответствующем функциональном устройстве, а также информацию, требующуюся для управления командой, когда ее выполнение началось в функциональном устройстве. Буфера загрузки и записи хранят данные поступающие из памяти и записываемые в память. Регистры ПТ соединены с функциональными устройствами парой шин и одной шиной с буферами записи. Все результаты из

функциональных устройств и из памяти посылаются на общую шину данных, которая связана со входами всех устройств за исключением буфера загрузки. Все буфера и станции резервирования имеют поля тегов, используемых для управления конфликтами.

В отличие от централизованной схемы управления, имеется всего три стадии выполнения команды:

Выдача - Берет команду из очереди команд ПТ. Если операция является операцией ПТ, выдает ее при наличии свободной станции резервирования и посылает операнды на станцию резервирования, если они находятся в регистрах. Если операция является операцией загрузки или записи, она может выдаваться при наличии свободного буфера. При отсутствии свободной станции резервирования или свободного буфера возникает структурный конфликт и команда приостанавливается до тех пор, пока не освободится станция резервирования или буфер.

Выполнение - Если один или более операндов команды не доступны по каким либо причинам, контролируется состояние CDB и ожидается завершение вычисления значений нужного регистра. На этой стадии выполняется контроль конфликтов типа RAW. Когда оба операнда доступны, выполняется операция.

Запись результата - Когда становится доступным результат, он записывается на CDB и оттуда в регистры и любое функциональное устройство, ожидающее этот результат.

Хотя эти шаги в основном похожи на аналогичные шаги в централизованной схеме управления, имеются три важных отличия. Во-первых, отсутствует контроль конфликтов типа WAW и WAR - они устраняются как побочный эффект алгоритма. Во-вторых, для трансляции результатов используется CDB, а не схема ожидания готовности регистров. В-третьих, устройства загрузки и записи рассматриваются как основные функциональные устройства.

Структуры данных, используемые для обнаружения и устранения конфликтов, связаны со станциями резервирования, регистровым файлом и буферами загрузки и записи. Хотя с разными объектами связана разная информация, все устройства, за исключением буферов загрузки, содержат в каждой строке поле тега. Это поле тега представляет собой четырехбитовое значение, которое обозначает одну из пяти станций резервирования или один из шести буферов загрузки. Поле тега используется для описания того, какое функциональное устройства будет поставлять результат, нужный в качестве источника операнда. Неиспользуемые значения, такие как ноль, показывают что операнд уже доступен. Важно помнить, что теги в схеме Томасуло ссылаются на буфера или устройства, которые будут поставлять результат; когда команда выдается в станцию резервирования номера регистров исключаются из рассмотрения.

Большие преимущества схемы Томасуло заключаются в распределении логики обнаружения конфликтов, и в устранении приостановок, связанных с конфликтами типа WAW и WAR. Первое преимущество возникает из-за наличия распределенных станций резервирования и использования CDB. Если

несколько команд ожидают один и тот же результат и каждая команда уже имеет свой другой операнд, то команды могут выдаваться одновременно посредством трансляции по CDB. В централизованной схеме управления ожидающие команды должны читать свои операнды из регистров когда станут доступными регистровые шины. Конфликты типа WAW и WAR устраняются путем переименования регистров используя станции резервирования.

Эта динамическая схема может достигать очень высокой производительности при условии того, что стоимость переходов может поддерживаться небольшой. Главный недостаток этого подхода заключается в сложности схемы Томасуло, которая требует для своей реализации очень большого объема аппаратуры. Особенно это касается большого числа устройств ассоциативной памяти, которая должна работать с высокой скоростью, а также сложной логики управления. Наконец, увеличение производительности ограничивается наличием одной шины завершения (CDB). Хотя дополнительные шины CDB могут быть добавлены, каждая CDB должна взаимодействовать со всей аппаратурой конвейера, включая станции резервирования. В частности, аппаратуру ассоциативного сравнения необходимо дублировать на каждой станции для каждой CDB.

В схеме Томасуло комбинируются две различных методики: методика переименования регистров и буферизация операндов-источников из регистрового файла. Буферизация источников операндов разрешает конфликты типа WAR, которые возникают когда операнды доступны в регистрах. Возможно также устранять конфликты типа WAR посредством переименования регистра вместе с буферизацией результата до тех пор, пока остаются обращения к старой версии регистра.

Схема Томасуло является привлекательной, если разработчик вынужден делать конвейерную архитектуру, для которой трудно выполнить планирование кода или реализовать большое хранилище регистров. С другой стороны, преимущество подхода Томасуло возможно ощущается меньше, чем увеличение стоимости реализации, по сравнению с методами планирования загрузки конвейера средствами компилятора в машинах, ориентированных на выдачу для выполнения только одной команды в такте. Однако по мере того, как машины становятся все более агрессивными в своих возможностях выдачи команд и разработчики сталкиваются с вопросами производительности кода, который трудно планировать (большинство кодов для нечисловых расчетов), методика типа переименования регистров и динамического планирования будет становиться все более важной. Эти методы являются одним из важных компонентов большинства схем для реализации аппаратного выполнения по предположению.

Ключевыми компонентами увеличения параллелизма уровня команд в алгоритме Томасуло являются динамическое планирование, переименование регистров и динамическое устранение неоднозначности обращений к памяти.

### 5.4.2.5.Аппаратное прогнозирование направления переходов и снижение потерь на организацию переходов

Динамической аппаратной технике планирования загрузки конвейера при наличии зависимостей по данным соответствует и динамическая техника для эффективной обработки переходов. Эта техника используется для двух целей: для прогнозирования того, будет ли переход выполняемым, и для возможно более раннего нахождения целевой команды перехода. Эта техника называется аппаратным прогнозированием переходов.

Простейшей схемой динамического прогнозирования направления условных переходов является буфер прогнозирования условных переходов (branch-prediction buffer) или таблица "истории" условных переходов (branch history table). Буфер прогнозирования условных переходов представляет собой небольшую память, адресуемую с помощью младших разрядов адреса команды перехода. Каждая ячейка этой памяти содержит один бит, который говорит о том, был ли предыдущий переход выполняемым или нет. Это простейший вид такого рода буфера. В нем отсутствуют теги, и он оказывается полезным только для сокращения задержки перехода в случае, если эта задержка больше, чем время, необходимое для вычисления значения целевого адреса перехода. В действительности мы не знаем, является ли прогноз корректным (этот бит в соответствующую ячейку буфера могла установить совсем другая команда перехода, которая имела то же самое значение младших разрядов адреса). Но это не имеет значения. Прогноз - это только предположение, которое рассматривается как корректное, и выборка команд начинается по прогнозируемому направлению. Если же предположение окажется неверным, бит прогноза инвертируется. Конечно такой буфер можно рассматривать как кэш-память, каждое обращение к которой является попаданием, и производительность буфера зависит от того, насколько часто прогноз применялся и насколько он оказался точным.

Однако простая однобитовая схема прогноза имеет недостаточную производительность. Рассмотрим, например, команду условного перехода в цикле, которая являлась выполняемым переходом последовательно девять раз подряд, а затем однажды невыполняемым. Направление перехода будет неправильно предсказываться при первой и при последней итерации цикла. Неправильный прогноз последней итерации цикла неизбежен, поскольку бит прогноза будет говорить, что переход "выполняемый" (переход был девять раз подряд выполняемым). Неправильный прогноз на первой итерации происходит из-за того, что бит прогноза инвертируется при предыдущем выполнении последней итерации цикла, поскольку в этой итерации переход был невыполняемым. Таким образом, точность прогноза для перехода, который выполнялся в 90% случаев, составила только 80% (2 некорректных прогноза и 8 корректных). В общем случае, для команд условного перехода, используемых для организации циклов, переход является выполняемым много раз подряд, а затем один раз оказывается невыполняемым. Поэтому однобитовая схема

прогнозирования будет неправильно предсказывать направление перехода дважды (при первой и при последней итерации).
Для исправления этого положения часто используется схема двухбитового прогноза. В двухбитовой схеме прогноз должен быть сделан неверно дважды, прежде чем он изменится на противоположное значение.
Двухбитовая схема прогнозирования в действительности является частным случаем более общей схемы, которая в каждой строке буфера прогнозирования имеет n-битовый счетчик. Этот счетчик может принимать значения от 0 до 2n - 1. Тогда схема прогноза будет следующей:
Если значение счетчика больше или равно 2n-1 (точка на середине интервала), то переход прогнозируется как выполняемый. Если направление перехода предсказано правильно, к значению счетчика добавляется единица (если только оно не достигло максимальной величины); если прогноз был неверным, из значения счетчика вычитается единица.
Если значение счетчика меньше, чем 2n-1, то переход прогнозируется как невыполняемый. Если направление перехода предсказано правильно, из значения счетчика вычитается единица (если только не достигнуто значение 0); если прогноз был неверным, к значению счетчика добавляется единица.
Исследования n-битовых схем прогнозирования показали, что двухбитовая схема работает почти также хорошо, и поэтому в большинстве систем применяются двухбитовые схемы прогноза, а не n-битовые.
Буфер прогнозирования переходов может быть реализован в виде небольшой специальной кэш-памяти, доступ к которой осуществляется с помощью адреса команды во время стадии выборки команды в конвейере (IF), или как пара битов, связанных с каждым блоком кэш-памяти команд и выбираемых с каждой командой. Если команда декодируется как команда перехода, и если переход спрогнозирован как выполняемый, выборка команд начинается с целевого адреса как только станет известным новое значение счетчика команд. В противном случае продолжается последовательная выборка и выполнение команд. Если прогноз оказался неверным, значение битов прогноза меняется.
Как уже упоминалось, точность двухбитовой схемы прогнозирования зависит от того, насколько часто прогноз каждого перехода является правильным и насколько часто строка в буфере прогнозирования соответствует выполняемой команде перехода. Если строка не соответствует данной команде перехода, прогноз в любом случае делается, поскольку все равно никакая другая информация не доступна. Даже если эта строка соответствует совсем другой команде перехода, прогноз может быть удачным.

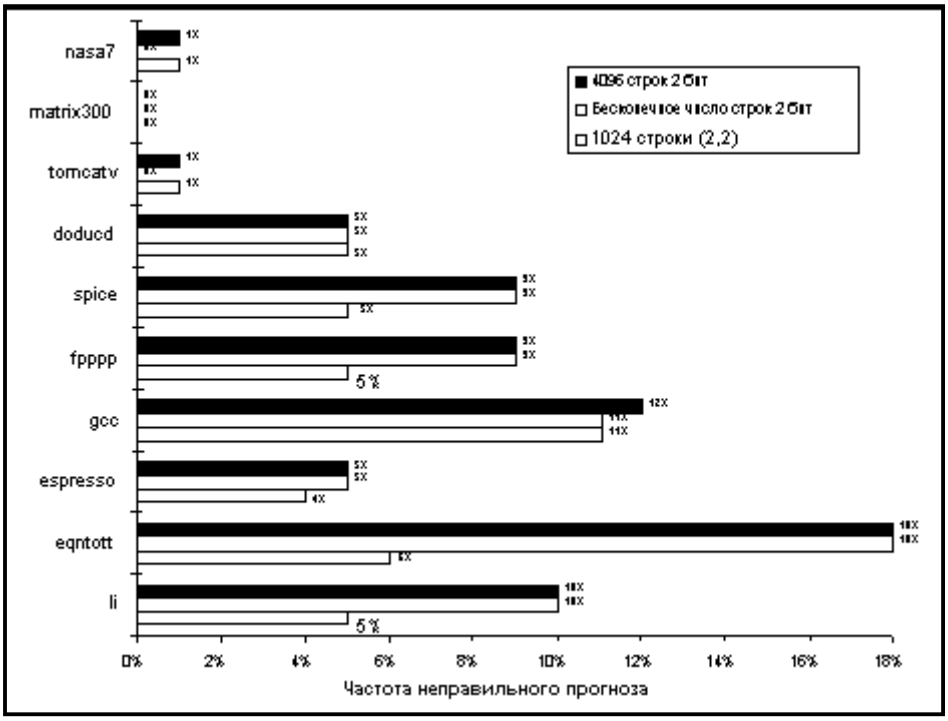


Рис. 5.4.2.5.Сравнение качества 2-битового прогноза

Какую точность можно ожидать от буфера прогнозирования переходов на реальных приложениях при использовании 2 бит на каждую строку буфера? Для набора оценочных тестов SPEC-89 буфер прогнозирования переходов с 4096 строками дает точность прогноза от 99% до 82%, т.е. процент неудачных прогнозов составляет от 1% до 18% (рисунок 5.5.3.2). Следует отметить, что буфер емкостью 4К строк считается очень большим. Буферы меньшего объема дадут худшие результаты.

Однако одного знания точности прогноза не достаточно для того, чтобы определить воздействие переходов на производительность машины, даже если известны время выполнения перехода и потери при неудачном прогнозе. Необходимо учитывать частоту переходов в программе, поскольку важность правильного прогноза больше в программах с большей частотой переходов. Например, целочисленные программы li, eqntott, expresso и gcc имеют большую частоту переходов, чем значительно более простые для прогнозирования программы плавающей точки nasa7, matrix300 и tomcatv.

Поскольку главной задачей является использование максимально доступной степени параллелизма программы, точность прогноза направления переходов становится очень важной. Как видно из рисунка 5.4.2.5., точность схемы прогнозирования для целочисленных программ, которые обычно имеют более высокую частоту переходов, меньше, чем для научных программ с плавающей точкой, в которых интенсивно используются циклы. Можно решать эту проблему двумя способами: увеличением размера буфера и увеличением точности схемы, которая используется для выполнения каждого отдельного

прогноза. Коэффициент попаданий буфера не является лимитирующим фактором, увеличение числа бит в схеме прогноза также имеет малый эффект.
Рассмотренные двухбитовые схемы прогнозирования используют информацию о недавнем поведении команды условного перехода для прогноза будущего поведения этой команды. Вероятно можно улучшить точность прогноза, если учитывать не только поведение того перехода, который мы пытаемся предсказать, но рассматривать также и недавнее поведение других команд перехода.
Схемы прогнозирования, которые для предсказания направления перехода используют поведение других команд перехода, называются коррелированными или двухуровневыми схемами прогнозирования. Схема прогнозирования называется прогнозом (1,1), если она использует поведение одного последнего перехода для выбора из пары однобитовых схем прогнозирования на каждый переход. В общем случае схема прогнозирования (m,n) использует поведение последних m переходов для выбора из $2^m$ схем прогнозирования, каждая из которых представляет собой n-битовую схему прогнозирования для каждого отдельного перехода. Привлекательность такого типа коррелируемых схем прогнозирования переходов заключается в том, что они могут давать больший процент успешного прогнозирования, чем обычная двухбитовая схема, и требуют очень небольшого объема дополнительной аппаратуры. Простота аппаратной схемы определяется тем, что глобальная история последних m переходов может быть записана в m-битовом сдвиговом регистре, каждый разряд которого запоминает, был ли переход выполняемым или нет. Тогда буфер прогнозирования переходов может индексироваться конкатенацией (объединением) младших разрядов адреса перехода с m-битовой глобальной историей. Например, на рисунке 5.4.2.5.2. показана схема прогнозирования (2,2) и организация выборки битов прогноза. прогнозирования (2,2) и организация выборки битов прогноза.

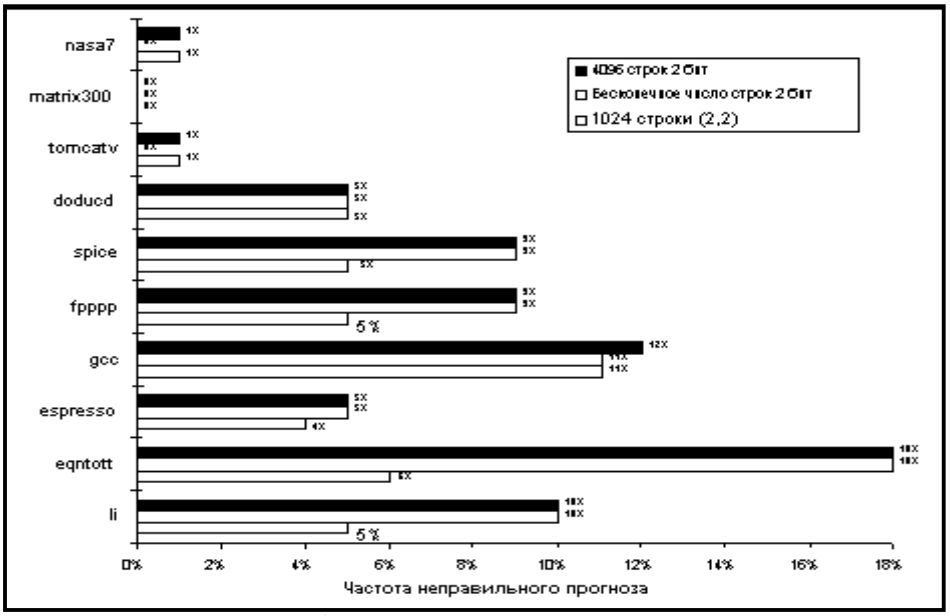

Рис. 5.4.2.5.2 Буфер прогнозирования переходов (2,2)

В этой реализации имеется тонкий эффект: поскольку буфер прогнозирования не является кэш-памятью, счетчики, индексируемые единственным значением глобальной схемы прогнозирования, могут в действительности в некоторый момент времени соответствовать разным командам перехода; то есть, прогноз может не соответствовать текущему переходу. На рисунке 5.4.2.5.2 с целью упрощения понимания буфер изображен как двумерный объект. В действительности он может быть реализован просто как линейный массив двухбитовой памяти; индексация выполняется путем конкатенации битов глобальной истории и соответствующим числом бит, требуемых от адреса перехода. Имеется широкий спектр корреляционных схем прогнозирования, среди которых схемы (0,2) и (2,2) являются наиболее интересными.

### 5.4.2.6.Дальнейшее уменьшение приостановок по управлению: буфера целевых адресов переходов

Допустим на стадии выборки команд находится команда перехода (на следующей стадии будет осуществляться ее дешифрация). Тогда чтобы сократить потери, необходимо знать, по какому адресу выбирать следующую команду. Это означает, что как-то надо выяснить, что еще недешифрированная команда в самом деле является командой перехода, и чему равно следующее значение счетчика адресов команд. Если все это известно, то потери на команду перехода могут быть сведены к нулю. Специальный аппаратный кэш прогнозирования переходов, который хранит прогнозируемый адрес следующей команды, называется буфером целевых адресов переходов (branch-target buffer).

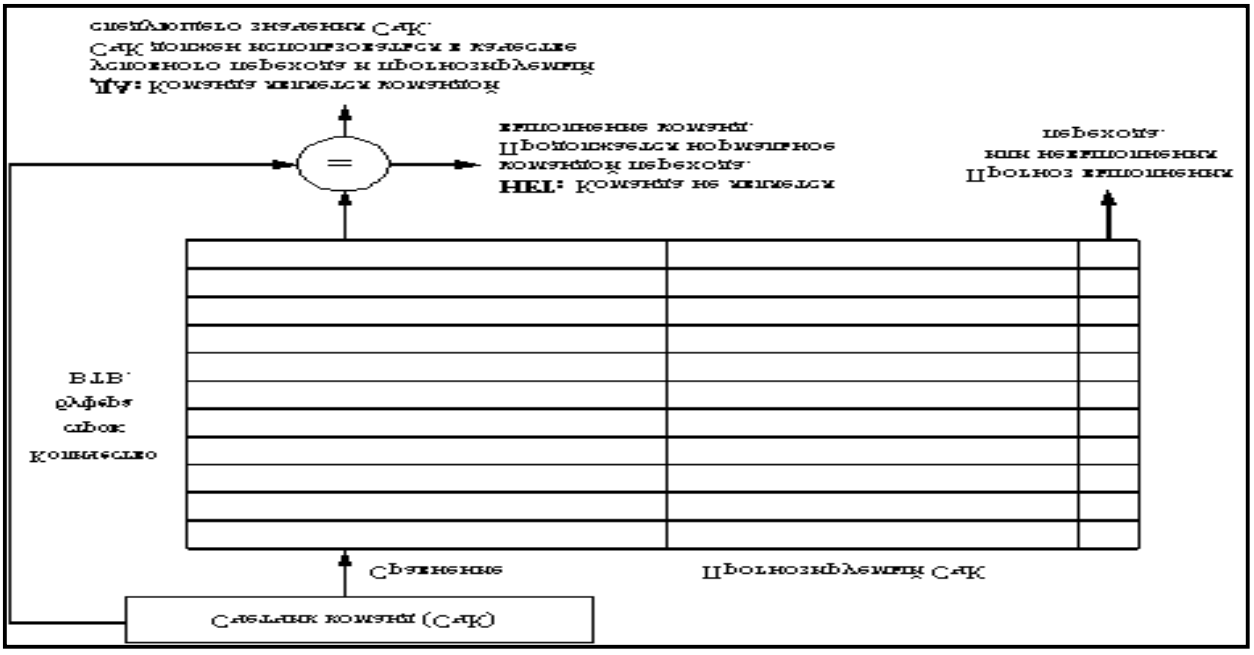

Рис. 5.4.2.6.1. Буфер целевых адресов переходов

Каждая строка этого буфера включает программный адрес команды перехода, прогнозируемый адрес следующей команды и предысторию команды перехода (рисунок 5.4.2.6.1.)**.** Биты предыстории представляют собой информацию о выполнении или невыполнении условий перехода данной команды в прошлом. Обращение к буферу целевых адресов перехода (сравнение с полями программных адресов команд перехода) производится с помощью текущего значения счетчика команд на этапе выборки очередной команды. Если обнаружено совпадение (попадание в терминах кэш-памяти), то по предыстории команды прогнозируется выполнение или невыполнение условий команды перехода, и немедленно производится выборка и дешифрация команд из прогнозируемой ветви программы. Считается, что предыстория перехода, содержащая информацию о двух предшествующих случаях выполнения этой команды, позволяет прогнозировать развитие событий с вполне достаточной вероятностью.

Существуют и некоторые вариации этого метода. Основной их смысл заключается в том, чтобы хранить в процессоре одну или несколько команд из прогнозируемой ветви перехода. Этот метод может применяться как в совокупности с буфером целевых адресов перехода, так и без него, и имеет два преимущества. Во-первых, он позволяет выполнять обращения к буферу целевых адресов перехода в течение более длительного времени, а не только в течение времени последовательной выборки команд. Это позволяет реализовать буфер большего объема. Во-вторых, буферизация самих целевых команд позволяет использовать дополнительный метод оптимизации, который называется свертыванием переходов (branch folding). Свертывание переходов может использоваться для реализации нулевого времени выполнения самих команд безусловного перехода, а в некоторых случаях и нулевого времени выполнения условных переходов. Рассмотрим буфер целевых адресов перехода, который буферизует команды из прогнозируемой ветви. Пусть к нему выполняется обращение по адресу команды безусловного перехода. Единственной задачей этой команды безусловного перехода является замена текущего значения счетчика команд. В этом случае, когда буфер адресов регистрирует попадание и показывает, что переход безусловный, конвейер просто может заменить команду, которая выбирается из кэш-памяти (это и есть сама команда безусловного перехода), на команду из буфера. В некоторых случаях таким образом удается убрать потери для команд условного перехода, если код условия установлен заранее.

Еще одним методом уменьшения потерь на переходы является метод прогнозирования косвенных переходов, а именно переходов, адрес назначения которых меняется в процессе выполнения программы (в run-time). Компиляторы языков высокого уровня будут генерировать такие переходы для реализации косвенного вызова процедур, операторов select или case и вычисляемых операторов goto в Фортране. Однако подавляющее большинство косвенных переходов возникает в процессе выполнения программы при

организации возврата из процедур. Например, для тестовых пакетов SPEC возвраты из процедур в среднем составляют 85% общего числа косвенных переходов.

Хотя возвраты из процедур могут прогнозироваться с помощью буфера целевых адресов переходов, точность такого метода прогнозирования может оказаться низкой, если процедура вызывается из нескольких мест программы или вызовы процедуры из одного места программы не локализуются по времени. Чтобы преодолеть эту проблему, была предложена концепция небольшого буфера адресов возврата, работающего как стек. Эта структура кэширует последние адреса возврата: во время вызова процедуры адрес возврата вталкивается в стек, а во время возврата он оттуда извлекается. Если этот кэш достаточно большой (например, настолько большой, чтобы обеспечить максимальную глубину вложенности вызовов), он будет прекрасно прогнозировать возвраты. На рисунке 5.4.2.6.2. показано исполнение такого буфера возвратов, содержащего от 1 до 16 строк (элементов) для нескольких тестов SPEC.

Точность прогноза в данном случае есть доля адресов возврата, предсказанных правильно. Поскольку глубина вызовов процедур обычно не большая, за некоторыми исключениями даже небольшой буфер работает достаточно хорошо. В среднем возвраты составляют 81% общего числа косвенных переходов для этих шести тестов.

Схемы прогнозирования условных переходов ограничены как точностью прогноза, так и потерями в случае неправильного прогноза. Как мы видели, типичные схемы прогнозирования достигают точности прогноза в диапазоне от 80 до 95% в зависимости от типа программы и размера буфера. Кроме увеличения точности схемы прогнозирования, можно пытаться уменьшить потери при неверном прогнозе. Обычно это делается путем выборки команд по обоим ветвям (по предсказанному и по непредсказанному направлению). Это требует, чтобы система памяти была двухпортовой, включала кэш-память с расслоением, или осуществляла выборку по одному из направлений, а затем по другому (как это делается в IBM POWER-2). Хотя подобная организация увеличивает стоимость системы, возможно это единственный способ снижения потерь на условные переходы ниже определенного уровня. Другое альтернативное решение, которое используется в некоторых машинах, заключается в кэшировании адресов или команд из нескольких направлений (ветвей) в целевом буфере.

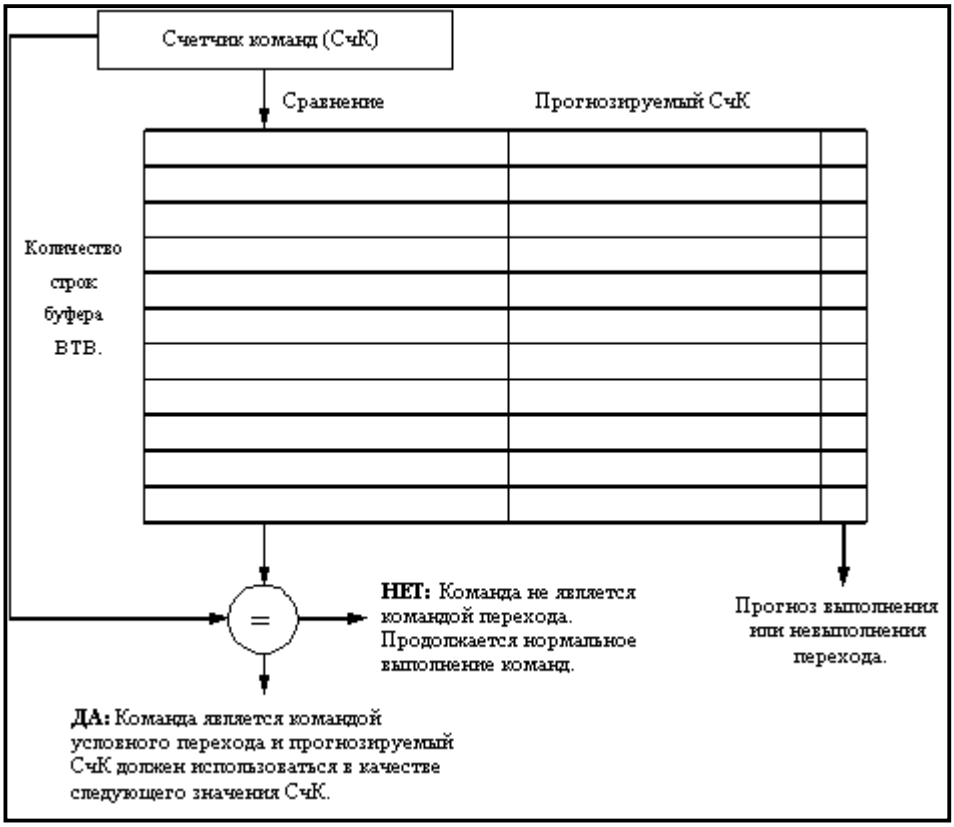


Рис. 5.4.2.6.2. Точность прогноза для адресов возврата

Одновременная выдача нескольких команд для выполнения и динамическое планирование

Методы минимизации приостановок работы конвейера из-за наличия в программах логических зависимостей по данным и по управлению, рассмотренные в предыдущих разделах, были нацелены на достижение идеального CPI (среднего количества тактов на выполнение команды в конвейере), равного 1. Чтобы еще больше повысить производительность процессора необходимо сделать CPI меньшим, чем 1. Однако этого нельзя добиться, если в одном такте выдается на выполнение только одна команда. Следовательно необходима параллельная выдача нескольких команд в каждом такте. Существуют два типа подобного рода машин: суперскалярные машины и VLIW-машины (машины с очень длинным командным словом). Суперскалярные машины могут выдавать на выполнение в каждом такте переменное число команд, и работа их конвейеров может планироваться как статически с помощью компилятора, так и с помощью аппаратных средств динамической оптимизации. В отличие от суперскалярных машин, VLIW-машины выдают на выполнение фиксированное количество команд, которые сформатированы либо как одна большая команда, либо как пакет команд фиксированного формата. Планирование работы VLIW-машины всегда осуществляется компилятором.

Суперскалярные машины используют параллелизм на уровне команд путем посылки нескольких команд из обычного потока команд в несколько

функциональных устройств. Дополнительно, чтобы снять ограничения последовательного выполнения команд, эти машины используют механизмы внеочередной выдачи и внеочередного завершения команд, прогнозирование переходов, кэши целевых адресов переходов и условное (по предположению) выполнение команд. Возросшая сложность, реализуемая этими механизмами, создает также проблемы реализации точного прерывания.

В типичной суперскалярной машине аппаратура может осуществлять выдачу от одной до восьми команд в одном такте. Обычно эти команды должны быть независимыми и удовлетворять некоторым ограничениям, например таким, что в каждом такте не может выдаваться более одной команды обращения к памяти. Если какая-либо команда в потоке команд является логически зависимой или не удовлетворяет критериям выдачи, на выполнение будут выданы только команды, предшествующие данной. Поэтому скорость выдачи команд в суперскалярных машинах является переменной. Это отличает их от VLIW-машин, в которых полную ответственность за формирование пакета команд, которые могут выдаваться одновременно, несет компилятор, а аппаратура в динамике не принимает никаких решений относительно выдачи нескольких команд.

Предположим, что машина может выдавать на выполнение две команды в одном такте. Одной из таких команд может быть команда загрузки регистров из памяти, записи регистров в память, команда переходов, операции целочисленного АЛУ, а другой может быть любая операция плавающей точки. Параллельная выдача целочисленной операции и операции с плавающей точкой намного проще, чем выдача двух произвольных команд. В реальных системах (например, в микропроцессорах PA7100, hyperSPARC, Pentium и др.) применяется именно такой подход. В более мощных микропроцессорах (например, MIPS R10000, UltraSPARC, PowerPC 620 и др.) реализована выдача до четырех команд в одном такте.

Выдача двух команд в каждом такте требует одновременной выборки и декодирования по крайней мере 64 бит. Чтобы упростить декодирование можно потребовать, чтобы команды располагались в памяти парами и были выровнены по 64-битовым границам. В противном случае необходимо анализировать команды в процессе выборки и, возможно, менять их местами в момент пересылки в целочисленное устройство и в устройство ПТ. При этом возникают дополнительные требования к схемам обнаружения конфликтов. В любом случае вторая команда может выдаваться, только если может быть выдана на выполнение первая команда. Аппаратура принимает такие решения в динамике, обеспечивая выдачу только первой команды, если условия для одновременной выдачи двух команд не соблюдаются. На рисунке 5.4.2.6.3. представлена диаграмма работы подобного конвейера в идеальном случае, когда в каждом такте на выполнение выдается пара команд.

Такой конвейер позволяет существенно увеличить скорость выдачи команд. Однако чтобы он смог так работать, необходимо иметь либо полностью

конвейеризованные устройства плавающей точки, либо соответствующее число независимых функциональных устройств. В противном случае устройство плавающей точки станет узким горлом и эффект, достигнутый за счет выдачи в каждом такте пары команд, сведется к минимуму.

| **Тип команды** | **Ступень конвейера** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Целочисленная команда | IF | ID | EX | MEM | WB | | | |
| Команда ПТ | IF | ID | EX | MEM | WB | | | |
| Целочисленная команда | | IF | ID | EX | MEM | WB | | |
| КомандаПТ | | IF | ID | EX | MEM | WB | | |
| Целочисленная команда | | | IF | ID | EX | MEM | WB | |
| КомандаПТ | | | IF | ID | EX | MEM | WB | |
| Целочисленная команда | | | | IF | ID | EX | MEM | WB |
| КомандаПТ | | | | IF | ID | EX | MEM | WB |

Рис. 5.4.2.6.3. Работа суперскалярного конвейера

При параллельной выдаче двух операций (одной целочисленной команды и одной команды ПТ) потребность в дополнительной аппаратуре, помимо обычной логики обнаружения конфликтов, минимальна: целочисленные операции и операции ПТ используют разные наборы регистров и разные функциональные устройства. Более того, усиление ограничений на выдачу команд, которые можно рассматривать как специфические структурные конфликты (поскольку выдаваться на выполнение могут только определенные пары команд), обнаружение которых требует только анализа кодов операций. Единственная сложность возникает, только если команды представляют собой команды загрузки, записи и пересылки чисел с плавающей точкой. Эти команды создают конфликты по портам регистров ПТ, а также могут приводить к новым конфликтам типа RAW, когда операция ПТ, которая могла бы быть выдана в том же такте, является зависимой от первой команды в паре.

Проблема регистровых портов может быть решена, например, путем реализации отдельной выдачи команд загрузки, записи и пересылки с ПТ. В случае составления ими пары с обычной операцией ПТ ситуацию можно рассматривать как структурный конфликт. Такую схему легко реализовать, но она будет иметь существенное воздействие на общую производительность. Конфликт подобного типа может быть устранен посредством реализации в регистровом файле двух дополнительных портов (для выборки и записи).

Если пара команд состоит из одной команды загрузки с ПТ и одной операции с ПТ, которая от нее зависит, необходимо обнаруживать подобный конфликт и блокировать выдачу операции с ПТ. За исключением этого случая, все другие конфликты естественно могут возникать, как и в обычной машине, обеспечивающей выдачу одной команды в каждом такте. Для предотвращения ненужных приостановок могут, правда потребоваться дополнительные цепи обхода.

Другой проблемой, которая может ограничить эффективность суперскалярной обработки, является задержка загрузки данных из памяти. В нашем примере простого конвейера команды загрузки имели задержку в один такт, что не позволяло следующей команде воспользоваться результатом команды загрузки без приостановки. В суперскалярном конвейере результат команды загрузки не может быть использован в том же самом и в следующем такте. Это означает, что следующие три команды не могут использовать результат команды загрузки без приостановки. Задержка перехода также становится длиною в три команды, поскольку команда перехода должна быть первой в паре команд. Чтобы эффективно использовать параллелизм, доступный на суперскалярной машине, нужны более сложные методы планирования потока команд, используемые компилятором или аппаратными средствами, а также более сложные схемы декодирования команд.

Рассмотрим, например, что дает разворачивание циклов и планирование потока команд для суперскалярного конвейера. Ниже представлен цикл, который мы уже разворачивали и планировали его выполнение на простом конвейере.

```
Loop: LD   F0,0(R1)      ;F0=элемент вектора
      ADDD F4,F0,F2      ;добавление скалярной величины из F2
      SD 0(R1),F4        ;запись результата
      SUBI R1,R1,#8      ;декрементирование указателя
                         ;8 байт на двойное слово
      BNEZ R1,Loop       ;переход R1!=нулю
```

Чтобы спланировать этот цикл для работы без задержек, необходимо его развернуть и сделать пять копий тела цикла. После такого разворачивания цикл будет содержать по пять команд LD, ADDD, и SD, а также одну команду SUBI и один условный переход BNEZ. Развернутая и оптимизированная программа этого цикла приведена в таблице 5.1.

Этот развернутый суперскалярный цикл теперь работает со скоростью 12 тактов на итерацию, или 2.4 такта на один элемент (по сравнению с 3.5 тактами для оптимизированного развернутого цикла на обычном конвейере. В этом примере производительность суперскалярного конвейера ограничена существующим соотношением целочисленных операций и операций ПТ, но команд ПТ не достаточно для поддержания полной загрузки конвейера ПТ. Первоначальный оптимизированный неразвернутый цикл выполнялся со скоростью 6 тактов на итерацию, вычисляющую один элемент. Мы получили таким образом ускорение в 2.5 раза, больше половины которого произошло за

счет разворачивания цикла. Чистое ускорение за счет суперскалярной обработки дало улучшение примерно в 1.5 раза.

Таблица 5.1.

| **Целочисленная команда** | **Команда ПТ** | **Номер такта** |
|---|---|---|
| Loop: LD F0,0(R1) | | 1 |
| LD F8,-8(R1) | ADDD | 2 |
| LD F10,-16(R1) | F4,F0,F2 | 3 |
| LD F14,-24(R1) | ADDD | 4 |
| LD F18,-32(R1) | F8,F6,F2 | 5 |
| SD 0(R1),F4 | ADDD | 6 |
| SD -8(R1),F8 | F12,F10,F2 | 7 |
| SD -16(R1),F12 | ADDD | 8 |
| SD -24(R1),F16 | F16,F14,F2 | 9 |
| SUBI R1,R1,#40 | ADDD | 10 |
| BNEZ R1,Loop | F20,F18,F2 | 11 |
| SD -32(R1),F20 | | 12 |

В лучшем случае такой суперскалярный конвейер позволит выбирать две команды и выдавать их на выполнение, если первая из них является целочисленной, а вторая - с плавающей точкой. Если это условие не соблюдается, что легко проверить, то команды выдаются последовательно. Это показывает два главных преимущества суперскалярной машины по сравнению с WLIW-машиной. Во-первых, малое воздействие на плотность кода, поскольку машина сама определяет, может ли быть выдана следующая команда, и нам не надо следить за тем, чтобы команды соответствовали возможностям выдачи. Во-вторых, на таких машинах могут работать неоптимизированные программы, или программы, откомпилированные в расчете на более старую реализацию. Конечно такие программы не могут работать очень хорошо. Один из способов улучшить ситуацию заключается в использовании аппаратных средств динамической оптимизации.

В общем случае в суперскалярной системе команды могут выполняться параллельно и возможно не в порядке, предписанном программой. Если не предпринимать никаких мер, такое неупорядоченное выполнение команд и наличие множества функциональных устройств с разными временами выполнения операций могут приводить к дополнительным трудностям. Например, при выполнении некоторой длинной команды с плавающей точкой (команды деления или вычисления квадратного корня) может возникнуть исключительная ситуация уже после того, как завершилось выполнение более быстрой операции, выданной после этой длинной команды. Для того, чтобы поддерживать модель точных прерываний, аппаратура должна гарантировать

корректное состояние процессора при прерывании для организации последующего возврата.
Обычно в машинах с неупорядоченным выполнением команд предусматриваются дополнительные буферные схемы, гарантирующие завершение выполнения команд в строгом порядке, предписанном программой. Такие схемы представляют собой некоторый буфер "истории", т.е. аппаратную очередь, в которую при выдаче попадают команды и текущие значения регистров результата этих команд в заданном программой порядке.
В момент выдачи команды на выполнение она помещается в конец этой очереди, организованной в виде буфера FIFO (первый вошел - первый вышел). Единственный способ для команды достичь головы этой очереди - завершение выполнения всех предшествующих ей операций. При неупорядоченном выполнении некоторая команда может завершить свое выполнение, но все еще будет находиться в середине очереди. Команда покидает очередь, когда она достигает головы очереди и ее выполнение завершается в соответствующем функциональном устройстве. Если команда находится в голове очереди, но ее выполнение в функциональном устройстве не закончено, она очередь не покидает. Такой механизм может поддерживать модель точных прерываний, поскольку вся необходимая информация хранится в буфере и позволяет скорректировать состояние процессора в любой момент времени.
Этот же буфер "истории" позволяет реализовать и условное (speculative) выполнение команд (выполнение по предположению), следующих за командами условного перехода. Это особенно важно для повышения производительности суперскалярных архитектур. Статистика показывает, что на каждые шесть обычных команд в программах приходится в среднем одна команда перехода. Если задерживать выполнение следующих за командой перехода команд, потери на конвейеризацию могут оказаться просто неприемлемыми. Например, при выдаче четырех команд в одном такте в среднем в каждом втором такте выполняется команда перехода. Механизм условного выполнения команд, следующих за командой перехода, позволяет решить эту проблему. Это условное выполнение обычно связано с последовательным выполнением команд из заранее предсказанной ветви команды перехода. Устройство управления выдает команду условного перехода, прогнозирует направление перехода и продолжает выдавать команды из этой предсказанной ветви программы.
Если прогноз оказался верным, выдача команд так и будет продолжаться без приостановок. Однако если прогноз был ошибочным, устройство управления приостанавливает выполнение условно выданных команд и, если необходимо, использует информацию из буфера истории для ликвидации всех последствий выполнения условно выданных команд. Затем начинается выборка команд из правильной ветви программы. Таким образом, аппаратура, подобная буферу, истории позволяет не только решить проблемы с реализацией точного

прерывания, но и обеспечивает увеличение производительности суперскалярных архитектур.

### 5.4.3 Архитектура машин с длинным командным словом

Архитектура машин с очень длинным командным словом (VLIW - Very Long Instruction Word) позволяет сократить объем оборудования, требуемого для реализации параллельной выдачи нескольких команд, и потенциально чем большее количество команд выдается параллельно, тем больше эта экономия. Например, суперскалярная машина, обеспечивающая параллельную выдачу двух команд, требует параллельного анализа двух кодов операций, шести полей номеров регистров, а также того, чтобы динамически анализировалась возможность выдачи одной или двух команд и выполнялось распределение этих команд по функциональным устройствам. Хотя требования по объему аппаратуры для параллельной выдачи двух команд остаются достаточно умеренными, и можно даже увеличить степень распараллеливания до четырех (что применяется в современных микропроцессорах), дальнейшее увеличение количества выдаваемых параллельно для выполнения команд приводит к нарастанию сложности реализации из-за необходимости определения порядка следования команд и существующих между ними зависимостей.

Архитектура VLIW базируется на множестве независимых функциональных устройств. Вместо того, чтобы пытаться параллельно выдавать в эти устройства независимые команды, в таких машинах несколько операций упаковываются в одну очень длинную команду. При этом ответственность за выбор параллельно выдаваемых для выполнения операций полностью ложится на компилятор, а аппаратные средства, необходимые для реализации суперскалярной обработки, просто отсутствуют.

WLIW-команда может включать, например, две целочисленные операции, две операции с плавающей точкой, две операции обращения к памяти и операцию перехода. Такая команда будет иметь набор полей для каждого функционального устройства, возможно от 16 до 24 бит на устройство, что приводит к команде длиною от 112 до 168 бит.

Рассмотрим работу цикла инкрементирования элементов вектора на подобного рода машине в предположении, что одновременно могут выдаваться две операции обращения к памяти, две операции с плавающей точкой и одна целочисленная операция либо одна команда перехода. На рисунке 5.5.5.1. показан код для реализации этого цикла. Цикл был развернут семь раз, что позволило устранить все возможные приостановки конвейера. Один проход по циклу осуществляется за 9 тактов и вырабатывает 7 результатов. Таким образом, на вычисление каждого результата расходуется 1.28 такта (в нашем примере для суперскалярной машины на вычисление каждого результата расходовалось 2.4 такта).

Для машин с VLIW-архитектурой был разработан новый метод планирования выдачи команд, названный "трассировочным планированием". При

использовании этого метода из последовательности исходной программы генерируются длинные команды путем просмотра программы за пределами базовых блоков. Как уже отмечалось, базовый блок - это линейный участок программы без ветвлений.

| Обращение к памяти 1 | Обращение к памяти 2 | Операция ПТ 1 | Операция ПТ 2 | Целочисленная операция/переход |
|---|---|---|---|---|
| LD F0,0(R1)<br>LD F10,-16(R1)<br>LD F18,-32(R1)<br>LD F26,-48(R1)<br>SD 0(R1),F4<br>SD -16(R1),F12<br>SD -32(R1),F20<br>SD 0(R1),F28 | LD F6,-8(R1)<br>LD F14,-24(R1)<br>LD F22,-40(R1)<br>SD -8(R1),F8<br>SD -24(R1),F16<br>SD -40(R1),F24 | ADDD F4,F0,F2<br>ADDD F12,F10,F2<br>ADDD F20,F18,F2<br>ADDD F28,F26,F2 | ADDD F8,F6,F2<br>ADDD F16,F14,F2<br>ADDD F24,F22,F2 | SUBI R1,R1,#48<br>BNEZ R1,Loop |

Рис. 5.4.3.1.

С точки зрения архитектурных идей машину с очень длинным командным словом можно рассматривать как расширение RISC-архитектуры. Как и в RISC-архитектуре аппаратные ресурсы VLIW-машины предоставлены компилятору, и ресурсы планируются статически. В машинах с очень длинным командным словом к этим ресурсам относятся конвейерные функциональные устройства, шины и банки памяти. Для поддержки высокой пропускной способности между функциональными устройствами и регистрами необходимо использовать несколько наборов регистров. Аппаратное разрешение конфликтов исключается и предпочтение отдается простой логике управления. В отличие от традиционных машин регистры и шины не резервируются, а их использование полностью определяется во время компиляции.

В машинах типа VLIW, кроме того, этот принцип замены управления во время выполнения программы планированием во время компиляции распространен на системы памяти. Для поддержания занятости конвейерных функциональных устройств должна быть обеспечена высокая пропускная способность памяти. Одним из современных подходов к увеличению пропускной способности памяти является использование расслоения памяти. Однако в системе с расслоенной памятью возникает конфликт банка, если банк занят предыдущим обращением. В обычных машинах состояние занятости банков памяти

отслеживается аппаратно и проверяется, когда выдается команда, выполнение которой связано с обращением к памяти. В машине типа VLIW эта функция передана программным средствам. Возможные конфликты банков определяет специальный модуль компилятора - модуль предотвращения конфликтов.

Обнаружение конфликтов не является задачей оптимизации, это скорее функция контроля корректности выполнения операций. Компилятор должен быть способен определять, что конфликты невозможны или, в противном случае, допускать, что может возникнуть наихудшая ситуация. В определенных ситуациях, например, в том случае, когда производится обращение к массиву, а индекс вычисляется во время выполнения программы, простого решения здесь нет. Если компилятор не может определить, что конфликт не произойдет, операции не могут планироваться для параллельного выполнения, а это ведет к снижению производительности.

Компилятор с трассировочным планированием определяет участок программы без обратных дуг (переходов назад), которая становится кандидатом для составления расписания. Обратные дуги обычно имеются в программах с циклами. Для увеличения размера тела цикла широко используется методика раскрутки циклов, что приводит к образованию больших фрагментов программы, не содержащих обратных дуг. Если дана программа, содержащая только переходы вперед, компилятор делает эвристическое предсказание выбора условных ветвей. Путь, имеющий наибольшую вероятность выполнения (его называют трассой), используется для оптимизации, проводимой с учетом зависимостей по данным между командами и ограничений аппаратных ресурсов. Во время планирования генерируется длинное командное слово. Все операции длинного командного слова выдаются одновременно и выполняются параллельно.

После обработки первой трассы планируется следующий путь, имеющий наибольшую вероятность выполнения (предыдущая трасса больше не рассматривается). Процесс упаковки команд последовательной программы в длинные командные слова продолжается до тех пор, пока не будет оптимизирована вся программа.

Ключевым условием достижения эффективной работы VLIW-машины является корректное предсказание выбора условных ветвей. Отмечено, например, что прогноз условных ветвей для научных программ часто оказывается точным. Возвраты назад имеются во всех итерациях цикла, за исключением последней. Таким образом, "прогноз", который уже дается самими переходами назад, будет корректен в большинстве случаев. Другие условные ветви, например ветвь обработки переполнения и проверки граничных условий (выход за границы массива), также надежно предсказуемы.

### 5.4.4. Аппаратные средства поддержки большой степени распараллеливания

Методы, подобные разворачиванию циклов и планированию трасс, могут использоваться для увеличения степени доступного параллелизма, когда поведение условных переходов достаточно предсказуемо во время компиляции. Если же поведение переходов не известно, одной техники компиляторов может оказаться не достаточно для выявления большей степени параллелизма уровня команд. В этом разделе представлены два метода, которые могут помочь преодолеть подобные ограничения. Первый метод заключается в расширении набора команд условными или предикатными командами. Такие команды могут использоваться для ликвидации условных переходов и помогают компилятору перемещать команды через точки условных переходов. Условные команды увеличивают степень параллелизма уровня команд, но имеют существенные ограничения. Для использования большей степени параллелизма разработчики исследовали идею, которая называется "выполнением по предположению" (speculation), и позволяет выполнить команду еще до того, как процессор узнает, что она должна выполняться (т.е. этот метод позволяет избежать приостановок конвейера, связанных с зависимостями по управлению).

Концепция, лежащая в основе условных команд, достаточно проста: команда обращается к некоторому условию, оценка которого является частью выполнения команды. Если условие истинно, то команда выполняется нормально; если условие ложно, то выполнение команды осуществляется, как если бы это была пустая команда. Многие новейшие архитектуры включают в себя ту или иную форму условных команд. Наиболее общим примером такой команды является команда условной пересылки, которая выполняет пересылку значения одного регистра в другой, если условие истинно. Такая команда может использоваться для полного устранения условных переходов в простых последовательностях программного кода.

Например, рассмотрим следующий оператор:

```
if (A=0) {S=T;};
```

Предполагая, что регистры R1, R2 и R3 хранят значения A, S и T соответственно, представим код этого оператора с командой условного перехода и с командой условной пересылки.

Код с использованием команды условного перехода будет иметь следующий вид:

```
BEQZ  R1,L
MOV R2,R3
L:
```

Используя команду условной пересылки, которая выполняет пересылку только если ее третий операнд равен нулю, мы можем реализовать этот оператор с помощью одной команды:

```
  CMOVZ  R2,R3,R1
```

Условная команда позволяет преобразовать зависимость по управлению, присутствующую в коде с командой условного перехода, в зависимость по данным. (Это преобразование используется также в векторных машинах, в которых оно называется if-преобразованием (if-convertion)). Для конвейерной машины такое преобразование позволяет перенести точку, в которой должна разрешаться зависимость, от начала конвейера, где она разрешается для условных переходов, в конец конвейера, где происходит запись в регистр.
Одним из примеров использования команд условной пересылки является реализация функции вычисления абсолютного значения: A = abs (B), которая реализуется оператором
if (B<0) {A=-B} else {A=B}.
Этот оператор if может быть реализован парой команд условных пересылок или командой безусловной пересылки (A=B), за которой следует команда условной пересылки (A=-B).
Условные команды могут использоваться также для улучшения планирования в суперскалярных или VLIW-процессорах. Ниже приведен пример кодовой последовательности для суперскалярной машины с одновременной выдачей для выполнения не более двух команд. При этом в каждом такте может выдаваться комбинация одной команды обращения к памяти и одной команды АЛУ или только одна команда условного перехода:
LW R1,40(R2)  ADD R3,R4,R5
ADD R6,R3,R7
BEQZ R10,L
LW R8,20(R10)
LW R9,0(R8)
Эта последовательность теряет слот операции обращения к памяти во втором такте и приостанавливается из-за зависимости по данным, если переход невыполняемый, поскольку вторая команда LW после перехода зависит от предыдущей команды загрузки. Если доступна условная версия команды LW, то команда LW, немедленно следующая за переходом (LW R8,20(R10)), может быть перенесена во второй слот выдачи. Это улучшает время выполнения на несколько тактов, поскольку устраняет один слот выдачи команды и сокращает приостановку конвейера для последней команды последовательности.
Для успешного использования условных команд в примерах, подобных этому, семантика команды должна определять команду таким образом, чтобы не было никакого побочного эффекта, если условие не выполняется. Это означает, что если условие не выполняется, команда не должна записывать результат по месту назначения, а также не должна вызывать исключительную ситуацию. Как показывает вышеприведенный пример, способность не вызывать исключительную ситуацию достаточно важна: если регистр R10 содержит нуль, команда LW R8,20(R10), выполненная безусловно, возможно вызовет исключительную ситуацию по защите памяти, а эта исключительная ситуация не должна возникать. Именно эта вероятность возникновения исключительной

ситуации не дает возможность компилятору просто перенести команду загрузки R8 через команду условного перехода. Конечно, если условие удовлетворено, команда LW все еще может вызвать исключительную ситуацию (например, ошибку страницы), и аппаратура должна воспринять эту исключительную ситуацию, поскольку она знает, что управляющее условие истинно.

Условные команды определенно полезны для реализации коротких альтернативных потоков управления. Тем не менее полезность условных команд существенно ограничивается несколькими факторами:

Аннулируемые условные команды (т.е. команды, условие которых является ложным) все же отнимают определенное время выполнения. Поэтому перенос команды через команду условного перехода и превращение ее в условную будет замедлять программу всякий раз, когда перенесенная команда не будет нормально выполняться. Важное исключение из этого правила возникает, когда такты, используемые перенесенной невыполняемой командой, были бы в любом случае холостыми (как в вышеприведенном примере с суперскалярной обработкой). Перенос команды через команду условного перехода существенно базируется на предположении о направлении перехода. Условные команды упрощают реализацию такого переноса, но не устраняют время выполнения, которое будет затрачено при неправильном предположении.

Условные команды наиболее полезны, когда условие может быть вычислено заранее. Если условие и условный переход не могут быть отделены друг от друга (из-за зависимости по данным при определении условия), то условная команда не поможет, хотя все еще может оказаться полезной, поскольку она задерживает момент времени, когда условие должно стать известным, почти до конца конвейера.

Использование условных команд ограничено, когда в поток управления вовлечено больше одной простой альтернативной последовательности команд. Например, при переносе команды через пару команд условного перехода необходимо, чтобы она оставалась зависимой от обоих условий, что требует либо спецификации в команде сразу двух условий (маловероятная возможность), либо вставки дополнительных команд для вычисления конъюнкции условий.

Условные команды могут давать некоторые потери скорости по сравнению с безусловными командами. Это может проявиться либо в большем количестве тактов, необходимых для выполнения таких команд, либо в уменьшении общей частоты синхронизации машины. Если условные команды являются более дорогими с точки зрения скорости выполнения, то их следует использовать осмысленно.

По этим причинам во многих современных архитектурах используется небольшое число условных команд (наиболее популярными являются команды условных пересылок), хотя некоторые из них включают условные версии большинства команд.

Поддерживаемое аппаратурой выполнение по предположению позволяет выполнить команду до момента определения направления условного перехода, от которого данная команда зависит. Это снижает потери, которые возникают при наличии в программе зависимостей по управлению.

В альтернативном варианте, при выполнении по предположению, что переход не будет выполняться, мы можем попытаться совместить выполнение последовательных итераций цикла. Действительно, это в точности то, что делает компилятор с планированием трасс. Когда направление переходов может прогнозироваться во время компиляции, и компилятор может найти команды, которые он может безопасно перенести на место перед точкой перехода, решение, базирующееся на технологии компилятора, идеально. Эти два условия являются ключевыми ограничениями для выявления параллелизма уровня команд статически с помощью компилятора. Выполнение по предположению, подобно условным командам, позволяет преодолеть два сложных момента, которые могут возникнуть при более раннем выполнении команд: возможность появления исключительной ситуации и ненужное изменение состояния машины, вызванное выполнением команды. Кроме того, механизмы выполнения по предположению позволяют выполнять команду даже до момента оценки условия командой условного перехода, что невозможно при условных командах. Конечно, аппаратная поддержка выполнения по предположению достаточно сложна и требует значительных аппаратных ресурсов.

Один из подходов, который был хорошо исследован во множестве исследовательских проектов заключается в объединении аппаратных средств динамического планирования и выполнения по предположению. Механизмы, допускающие выполнение по предположению, позволяют команды, зависящие от команд, выполняющихся по предположению. Аппаратура, реализующая алгоритм Томасуло, может быть расширена для обеспечения поддержки выполнения по предположению. С этой целью необходимо отделить средства пересылки результатов команд, которые требуются для выполнения по предположению некоторой команды, от механизма действительного завершения команды. Имея такое разделение функций, мы можем допустить выполнение команды и пересылать ее результаты другим командам, не позволяя ей однако делать никакие обновления состояния машины, которые не могут быть ликвидированы, до тех пор, пока мы не узнаем, что команда должна безусловно выполниться. Использование цепей ускоренной пересылки также подобно выполнению по предположению чтения регистра, поскольку мы не знаем, обеспечивает ли команда, формирующая значение регистра-источника, корректный результат до тех пор, пока ее выполнение не станет безусловным. Если команда, выполняемая по предположению, становится безусловной, ей разрешается обновить регистровый файл или память. Этот дополнительный этап выполнения команд обычно называется стадией фиксации результатов команды (instruction commit).

Главная идея, лежащая в основе реализации выполнения по предположению, заключается в разрешении неупорядоченного выполнения команд, но в строгом соблюдении порядка фиксации результатов и предотвращением любого безвозвратного действия (например, обновления состояния или приема исключительной ситуации) до тех пор, пока результат команды не фиксируется. В простом конвейере с выдачей одиночных команд мы могли бы гарантировать, что команда фиксируется в порядке, предписанном программой, и только после проверки отсутствия исключительной ситуации, вырабатываемой этой командой, просто посредством переноса этапа записи результата в конец конвейера. Когда мы добавляем механизм выполнения по предположению, мы должны отделить процесс фиксации команды, поскольку он может произойти намного позже, чем в простом конвейере. Добавление к последовательности выполнения команды этой фазы фиксации требует некоторых изменений в последовательности действий, а также в дополнительного набора аппаратных буферов, которые хранят результаты команд, которые завершили выполнение, но результаты которых еще не зафиксированы. Этот аппаратный буфер, который можно назвать буфером переупорядочивания, используется также для передачи результатов между командами, которые могут выполняться по предположению.

Буфер переупорядочивания предоставляет дополнительные виртуальные регистры точно так же, как станции резервирования в алгоритме Томасуло расширяют набор регистров. Буфер переупорядочивания хранит результат некоторой операции в промежутке времени от момента завершения операции, связанной с этой командой, до момента фиксации результатов команды. Поэтому буфер переупорядочивания является источником операндов для команд, точно также как станции резервирования обеспечивают промежуточное хранение и передачу операндов в алгоритме Томасуло. Основная разница заключается в том, что когда в алгоритме Томасуло команда записывает свой результат, любая последующая выдаваемая команда будет выбирать этот результат из регистрового файла. При выполнении по предположению регистровый файл не обновляется до тех пор, пока команда не фиксируется (и мы знаем определенно, что команда должна выполняться); таким образом, буфер переупорядочивания поставляет операнды в интервале между завершением выполнения и фиксацией результатов команды. Буфер переупорядочивания не похож на буфер записи в алгоритме Томасуло, и в нашем примере функции буфера записи интегрированы с буфером переупорядочивания только с целью упрощения. Поскольку буфер переупорядочивания отвечает за хранение результатов до момента их записи в регистры, он также выполняет функции буфера загрузки.

Каждая строка в буфере переупорядочивания содержит три поля: поле типа команды, поле места назначения (результата) и поле значения. Поле типа команды определяет, является ли команда условным переходом (для которого отсутствует место назначения результата), командой записи (которая в качестве

места назначения результата использует адрес памяти) или регистровой операцией (команда АЛУ или команда загрузки, в которых местом назначения результата является регистр). Поле назначения обеспечивает хранение номера регистра (для команд загрузки и АЛУ) или адрес памяти (для команд записи), в который должен быть записан результат команды. Поле значения используется для хранения результата операции до момента фиксации результата команды. Буфер переупорядочивания полностью заменяет буфера загрузки и записи. Хотя функция переименования станций резервирования заменена буфером переупорядочивания, нам все еще необходимо некоторое место для буферизации операций (и операндов) между моментом их выдачи и началом выполнения. Эту функцию выполняют регистровые станции резервирования. Поскольку каждая команда имеет позицию в буфере переупорядочивания до тех пор, пока она не будет зафиксирована (и результаты не будут отправлены в регистровый файл), результат тегируется посредством номера строки буфера переупорядочивания, а не номером станции резервирования. Это требует, чтобы номер строки буфера переупорядочивания, присвоенный команде, отслеживался станцией резервирования.

Ниже перечислены четыре этапа выполнение команды:

**Выдача.** Получает команду из очереди команд плавающей точки. Выдает команду для выполнения, если имеется свободная станция резервирования и свободная строка в буфере переупорядочивания; передает на станцию резервирования операнды, если они находятся в регистрах или в буфере переупорядочивания; и обновляет поля управления для индикации того, что буфера используются. Номер отведенной под результат строки буфера переупорядочивания также записывается в станцию резервирования, так что этот номер может использоваться для тегирования (пометки) результата, когда он помещается на CDB. Если все станции резервирования заполнены, или полон буфер переупорядочивания, выдача команды приостанавливается до тех пор, пока в обоих буферах не появится доступной строки.

**Выполнение.** Если один или несколько операндов еще не готовы (отсутствуют), осуществляется просмотр CDB (Common Data Bus) и происходит ожидание вычисления значения требуемого регистра. На этом шаге выполняется проверка наличия конфликтов типа RAW. Когда оба операнда оказываются на станции резервирования, происходит вычисление результата операции.

**Запись результата.** Когда результат вычислен и становится доступным, выполняется его запись на CDB (с посылкой тега буфера переупорядочивания, который был присвоен команде на этапе выдачи для выполнения) и из CDB в буфер переупорядочивания, а также в каждую станцию резервирования, ожидающую этот результат. (Можно было бы также читать результат из буфера переупорядочивания, а не из CDB, точно также, как централизованная схема управления (scoreboard) читает результаты из регистров, а не с шины завершения). Станция резервирования помечается как свободная.

**Фиксация.** Когда команда достигает головы буфера переупорядочивания и ее результат присутствует в буфере, соответствующий регистр обновляется значением результата (или выполняется запись в память, если операция - запись в память), и команда изымается из буфера переупорядочивания.

Когда команда фиксируется, соответствующая строка буфера переупорядочивания очищается, а место назначения результата (регистр или ячейка памяти) обновляется. Чтобы не менять номера строк буфера переупорядочивания после фиксации результата команды, буфер переупорядочивания реализуется в виде циклической очереди, так что позиции в буфере переупорядочивания меняются, только когда команда фиксируется. Если буфер переупорядочивания полностью заполнен, выдача команд останавливается до тех пор, пока не освободится очередная строка буфера.

Поскольку никакая запись в регистры или ячейки памяти не происходит до тех пор, пока команда не фиксируется, машина может просто ликвидировать все свои выполненные по предположению действия, если обнаруживается, что направление условного перехода было спрогнозировано не верно.

Исключительные ситуации в подобной машине не воспринимаются до тех пор, пока соответствующая команда не готова к фиксации. Если выполняемая по предположению команда вызывает исключительную ситуацию, эта исключительная ситуация записывается в буфер упорядочивания. Если обнаруживается неправильный прогноз направления условного перехода и выясняется, что команда не должна была выполняться, исключительная ситуация гасится вместе с командой, когда обнуляется буфер переупорядочивания. Если же команда достигает вершины буфера переупорядочивания, то мы знаем, что она более не является выполняемой по предположению (она уже стала безусловной), и исключительная ситуация должна действительно восприниматься.

Эту методику выполнения по предположению легко распространить и на целочисленные регистры и функциональные устройства. Действительно, выполнение по предположению может быть более полезно в целочисленных программах, поскольку именно такие программы имеют менее предсказуемое поведение переходов. Кроме того, эти методы могут быть расширены так, чтобы обеспечить работу в машинах с выдачей на выполнение и фиксацией результатов нескольких команд в каждом такте. Выполнение по предположению возможно является наиболее интересным методом именно для таких машин, поскольку менее амбициозные машины могут довольствоваться параллелизмом уровня команд внутри базовых блоков при соответствующей поддержке со стороны компилятора, использующего технологию разворачивания циклов.