А.М. Сергиєнко

Архитектура компьютеров

Киев-2015

УДК 004.383 ББК 32.973-018 Рецензенти: А. М. Романкевіч, доктор технічних наук, професор (кафедра спеціализованих обчислювальних систем Національного технічного університету України "КПИ"); В. П. Сімоненко, доктор технічних наук, професор (кафедра обчислювальної техніки Національного технічного університету України "КПИ")

Сергієнко А. М. Архітектура комп'ютерів: Конспект лекцій. – К.: НТУУ«КПІ», 2015. – 198 с. Розглянуті теоретичні відомості про архітектури сучасних комп'ютерів. Для студентів, аспірантів, викладачів вузів в галузі електроніки, вимірювальної і обчислювальної техніки, зв'язку та захисту інформації. УДК 004.383 ББК 32.973-018

© А. М. Сергієнко, 2015

СОДЕРЖАНИЕ

Предисловие 3 1. Архитектура процессорного ядра компьютеров 4
1.1 Введение. Основные определения и принципы построения компьютеров 4
1.2 Элементная база для построения компьютеров 19
1.3 Архитектура компьютеров фон Неймана 26
1.4 Система команд компьютеров и ее выбор 28
1.5 Форматы данных и операции с ними 30
1.6 Способы адресации 38
1.7 RISC- и CISC - процессоры 44
1.8 Система прерываний 50
2. Архитектура процессорного среды 60
2.1 Основы организации интерфейсов 60
2.2 Организация памяти компьютеров 70
2.3. Внешние запоминающие устройства 79
2.4. Архитектура консоли компьютера 86
2.5. Структура однопроцессорного компьютера 92
3. Параллелизм в однопроцессорных компьютерах 94
3.1. Конвейерная и векторная обработка информации 96
3.2. Векторно - конвейерные компьютеры 101
3.3. Суперскалярные процессоры 104
3.4. Повышено быстродействие суперскалярных процессоров 107
3.5. Многопоточная работа процессора 114
3.6. Параллелизм доступа памяти 116

Предисловие

Архитектура компьютеров - это основные сведения о компьютере, которые необходимы как для его проектирования, так и для программирования. Создание новой программы - это творческий процесс реализации заданного алгоритма на компьютере с определенной архитектурой. При создании программы программист должен четко осознавать, какие процессы и в каком порядке выполняются в компьютере при выполнении его программы. Иначе вычислительный процесс, который задумал реализовать программист в своей программе, может отличаться от процесса выполнения программы в определенном компьютере и, в результате, цель составления этой программы будет недостижима.

С каждым годом усложняется устройство компьютеров, увеличивается степень присущего им параллелизма и повышается уровень защиты информации. При этом понимание и восприятие архитектуры для многих программистов остается неизменным на протяжении десятилетий. С одной стороны, это означает, что без знаний о новых архитектурах программисты не могут достичь наиболее эффективного выполнения своих программ в современных компьютерах, а с другой - особенности организации параллельных вычислительных процессов в этих компьютерах могут приводить к неэффективному и даже неправильному выполнению программ, составленных для устаревших архитектур.

Также без знания архитектуры компьютеров невозможны разработка, настройка и анализ многих сложных и ответственных применений, которые имеют особые требования по быстродействию, надежности, защиты информации, энергосбережению.

В дисциплине "Архитектура компьютеров" студенты изучают теоретические сведения о архитектурные элементы современных компьютеров и связи между ними, а также получают навыки по их применению при выборе и программировании микропроцессоров. Поэтому эта дисциплина занимает центральное место в подготовке специалистов по специальностям "Информационные управляющие системы и технологии", "Компьютерные системы и сети".

1. Архитектура процессорного ядра компьютера

1.1. Введение. Основные определения и принципы построения компьютеров

Чтобы понять, что такое архитектура и зачем ее изучать и исследовать, следует вспомнить историю развития компьютеров.

История компьютеров и их архитектур

Наиболее весомым толчком для появления электронных вычислительных них машин была потребность в совершенствовании оружия во время Второй мировой войне. Вопрос расчета траектории полета снарядов и бомб стало важным для составления таблиц стрельбы и бомбометания. Желательно было вести такие расчеты в реальном времени. Для ускорения решения этой задачи в 1943 по поручению МО США в Пенсильванском университете началась разработка первого электронного компьютера Eniac. Он приступил к работе в 1946 и работал до 1955 г. Компьютер Eniac имел 18000 электронных лам и выполнял умножения десятичных 10-разрядных чисел с 2,8 мс. Он состоял из нескольких интеграторов- аккумуляторов, регистров, таблиц констант и устройства ввода-вывода. Выполняемые алгоритмы - это итерационные алгоритмы решения дифференциальных уравнений. Программирование такой машины состояло в соединении ее блоков в сеть с помощью кабелей передачи данных в соответствии с графом алгоритма выполнения цикла вычислений.

Сначала работа по выполнению такого соединения осуществлялась вручную с помощью коммутационного поля, по конструкции напоминавшего коммутатор на телефонной станции, который задавал конфигурацию связей между отдельными регистрами и сумматорами. Изменение программы требовало несколько дней. Для ускорение этого процесса внедрили коммутационные панели, а изменение программы (конфигурации) выполнялось ручной заменой "запрограммированной "коммутационной панели.

В 1943 в Англии были разработаны и изготовлены под руководством Н.Винера десять экземпляров компьютеров Colossus, которые использовались для расшифровки немецких радиограмм. По своему принципу работы это были знаковые корреляторы, каждый из которых состоял из 1500 ламп. При этом их

структура также отражала структуру алгоритма, но алгоритм вычислений был неизменным.

Еще раньше - с 1939 по 1942 - Д. Атанасоф в Государственном колледже штата Айова разрабатывал электронный компьютер ABC, который выполнял единственный алгоритм - решение системы линейных уравнений. Это была параллельная структура из тридцати последовательных ламповых арифметико-логических устройств (АЛУ) с параллельной памятью на конденсаторном барабане. Изменение алгоритма также не предполагалось.

Итак, первые электронные компьютеры были способны выполнять только один или несколько похожих алгоритмов, которые были закодированы в их устройствах управления или структуре межсоединений. По идеологии, это было заимствование принципа работы механических или электромеханических вычислителей, которые были распространены до середины двадцатого столетия.

В 1936 А.Тьюрингом была предложена модель вычислителя, в которой произвольный алгоритм мог быть закодирован как таблица в блоке управления и как исходное состояние запоминающей ленты. Так было положено начало идее компьютера, который может выполнять гипотетически любой алгоритм.

Считается, что идея реализации универсального программируемого компьютера впервые возникла в США в электротехнической школе Мура Пенсильванского университета. Там коллектив ученых во главе с фон-Нейманом в 1946 составил отчет, в котором предлагалось в памяти компьютеров хранить как данные, так и программу, а также выполнять обработку данных, используя их двоичное представление и аппарат булевой алгебры. Итак, память компьютера стала считаться основой его функциональности и универсальности.

В те времена архитектура компьютеров существенно зависела от конструкции памяти. Триггерная память на лампах была быстродействующей, но слишком дорогой и ненадежной. Была распространена память на ртутных линиях задержки. Такая память имела последовательный, не произвольный доступ данных и поэтому определяла последовательную, поразрядную обработку этих данных. Осциллографические трубки Вильямса позволяли получить произвольный доступ к данным, записываемым в виде пикселов на экране трубки, но имели небольшой объем памяти.

Т.Килберн и Ф.Вильямс были первыми, кто в 1948 году просчитали первую программу на компьютере Mark1 в Манчестерском университете. Этот компьютер имел память на трубках Вильямса, с емкостью по 1024 бит данных каждая.

Профессор Кембриджа М.Вилкс перенял идею группы Неймана и запустил свою первую программу в 1949 г. на компьютере Edsac. Годом позже заработала

машина Edvac в школе Мура, для разработки которой, собственно, был составлен отчет группы Неймана. Эти машины были построены на линиях задержки и работали синхронно-последовательно, то есть слова в них обрабатывались бит за битом.

В тот же период в Киеве С. А. Лебедев разрабатывал машину МЭСМ, в которой было заложено много новых идей. Это параллельная обработка данных (такая же впервые была применена для компьютере Wirlwind в 1951г. в США) и принцип микропрограммного управления (который был предложен М.Вилксом в 1951 году, а впервые реализован в 1957 году). МЭСМ впервые заработала в 1950 и была принята в эксплуатацию в конце 1951г. Ее успешное использование для сложных вычислений в оборонных проектах способствовало появлению и развитию компьютерной отрасли в СССР.

Большой толчок для коммерческого распространения Компьютеров дала машина Univac-1, которая выпускалась с 1951г. филиалом Юнивак фирмы Ремингтон. Она была на 5000 лампах и сотне линий задержки для 1000 двенадцатиразрядный десятичных чисел. Ввод-вывод выполнялся с помощью переключателей, электрической пишущей машинки и устройства с магнитной лентой. В 1953 эта фирма одной из первых внедрила оперативное запоминающее устройство (ОЗУ) на магнитных кольцах, что повысило производительность компьютеров в 50 раз.

Повышенная емкость ОЗУ и произвольный доступ к памяти позволили внедрять программирования на языке высокого уровня, таком как Fortran. Еще в 1952 фирма Ремингтон первой предложила использовать интерпретатор для вычисления программ в алгебраической записи. А женщина-ученый Грейс Хоппер для машины Univac разработала первый компилятор и также принимала участие в разработке языка Cobol. Так в 50-х годах появились языки программирования, а некоторые из них, такие как Алгол, Fortran, Cobol стали стандартными в 60-х годах.

Несмотря на высокую стоимость компьютеров первого поколения (около миллиона долларов США), их производство становилось коммерчески выгодным. Большинство внедрений компьютеры нашли в банковских и государственных учреждениях, где они использовались для обработки статистических данных и денежных расчетов. В этих областях компьютер мог заменять сотни служащих, многократно ускорять решение задач и поэтому его применение давало прибыль. Поэтому основной задачей компьютеров первого поколения была накопление и обработка больших массивов структурированных данных, которые хранились на магнитных лентах. Возникло большое число компьютерных фирм, которые соревновались по скорости и цене компьютеров.

Так, фирма DEC в 1959 году выпустила восемнадцатиразрядную модель компьютера PDP-1 с ОЗУ объемом до 32 Кслов по цене всего лишь 120 тыс. долл.

Миниатюризация ОЗУ и внедрение в нем полупроводников (компьютеры второго поколения) способствовало еще бо'льшему повышению его емкости и быстродействия. Достижение емкости ОЗУ нескольких десятков тысяч слов позволило в 1960 году внедрить концепцию операционной системы. Операционная система начала предоставлять пользователям новые возможности, такие как работа с виртуальной памятью и разделением времени. Толчок этому дало исследование в начале 60-х годов в Дартмундском колледже, проводившемся фирмой Дженерал Электрик. Системы с разделением времени дали возможность нескольким пользователям распоряжаться одним компьютером одновременно. Благодаря этому, стало возможным существенно уменьшить арендную плату за использование компьютера.

Программное обеспечение в те годы было дорогим и необходимым дополнением к компьютерам. И это было причиной для исчезновения с рынка программно несовместимых компьютеров. Например, в 1960 году высоким техническим достижением фирмы IBM была машина Stretch, в которой были внедрены конвейерная обработка команд и предварительный анализ очереди команд. Но эта и другие модели компьютеров потеряли коммерческий успех из-за программной несовместимости и трудностей работы с ними.

Поэтому фирма IBM в 1961 году взяла курс на создание единого модельного ряда компьютеров. Компьютеры должны быть на самом деле универсальными - использоваться как для коммерческих, так и научных работ, соединяться с разнообразными периферийными устройствами с помощью стандартных интерфейсов. В разработке новых компьютеров руководящую роль играли Джим Амдал и Герит Блау, которые ввели название архитектуры компьютеров. Основная идея понятия архитектуры заключалась в том, чтобы все члены ряда компьютеров характеризовались теми же архитектурными признаками: одинаковый набор команд, такие же способы адресации и.т.д. Благодаря этому, все они могли выполнять одни и те же программы с одинаковыми результатами для одих исходных данных. Для реализации одной системы сложных команд во всех моделях компьютеров было внедрено микропрограммирование.

Одновременно проектировались шесть моделей компьютеров с архитектурой IBM-360, которые отличались между собой быстродействием и ценой. В поставленных компьютерах могли использоваться до 47 различных периферийных устройств в любой комплектации. В модели 195 этой системы в 1969 г. впервые была внедрена кэш-память. В 1970 г. архитектуру IBM -360 заменила

архитектура IBM-370. В старших моделях IBM-370 явно прослеживаются признаки современных суперскалярных микропроцессоров.

Такой компьютер занимал несколько шкафов и находился в отдельной комнате. В основной раме (main frame) главного шкафа компьютера размещался центральный процессор. Поэтому большой компьютер назывался (и до сих пор называется) майнфреймом. В нашей стране аналоги компьютеров с такой архитектурой известны как ЕС1010-ЕС1065. Следующие поколения больших компьютеров фирмы IBM, выполненные на микропроцессорах, наследовали систему команд IBM-360/370, чтобы удовлетворить требования переносимости матобеспечения.

История компьютеров показывает, что успешными архитектурами становятся не те архитектуры, в которых внедрены передовые научные идеи, а те, которые поддержаны удачным торговым менеджментом и рыночной конъюнктурой. Например, фирма Барроуз в своих компьютерах внедрила много технических новинок, но это не способствовало ее успеху. Так, в ее компьютере B5000 впервые использованы операционная система с мультипрограммированием и вычислительные средства с виртуальной памятью, которые предшествовали их широкому внедрению на десятилетие. Но его быстродействие в 1963 году не удовлетворило пользователей. В наиболее совершенном компьютере этой фирмы B6500 образца 1966 года операционная система обеспечивала мультипрограммирование, параллельную обработку данных, работу в реальном масштабе времени с разделением времени и даже выполнение программ с различными системами команд.

Другой пример - это фирма Контрол Дейта Корпорейшн (CDC). Сеймур Крей, ее основатель, в 1963 году разработал модель CDC-6600, состоящую из нескольких АЛУ и периферийных процессоров. В ней нашли внедрение многие признаки современных RISC-процессоров, такие как: - простота команд; - команды работы с регистрами отделены от команд пересылок; - регистровые команды имеют три поля адресов; - формат команд простой и одинаковый; - выполнение команд – конвейризованнное; - несколько аппаратных ресурсов, которые перераспределяются между командами из потока команд, как в суперскалярных процессорах; - переименование регистров.

В результате, быстродействие этого компьютера втрое превышало быстродействие компьютера Stretch, но он не получил популярность. Идеи, проверенные в нем, были внедрены Креем в первом суперкомпьютере Cray1, который вступил в строй в 1976 году.

В 1965 фирма DEC появлением своего 12-разрядного компьютера PDP-8 дала толчок для развития рынка мини-компьютеров. Это был первый компьютер по цене менее 20 тыс. долл. Наиболее значимым стало представление этой

фирмой архитектуры PDP-11 в 1970 году. Архитектура мини-компьютера отличается тем, что все его процессоры и периферийные устройства соединены двунаправленной асинхронной общей шиной. Это позволило формировать конфигурацию компьютера по модульному принципу.

Несколькими годами позже эта архитектура была повторена в СССР в компьютерах СМ-4, М-4030, Саратов, а впоследствии - также в микрокомпьютерах Электроника-60 и ДВК. Архитектура PDP-11 была совершенной, благодаря чему она приобрела большую популярность среди программистов. Эта архитектура после модернизации нашла внедрение в миникомпьютерах и рабочих станциях с архитектурой VAX. На архитектуре VAX история фирмы DEC оборвалась, так как она не выдержала конкуренции с процессорами Intel и архитектурой RISC.

В 1968 году в историю миникомпьютеров вошла также фирма Дейта Дженерал с компьютером Nova по цене всего 8 тыс.долл. Эта модель получила наибольшее распространение в мире среди мини-компьютеров. Ее архитектура отличалась тем, что часть матобеспечения сохранялась в постоянном запоминающем устройстве (ПЗУ). Такая архитектурная особенность затем закрепилась во всех микрокомпьютерах.

В конце 60-х годов в Киеве в Институте Кибернетики был разработан и внедрен в производство мини-компьютер МИР-1. Он отличался тем, что его интерпретатор с языка высокого уровня был выполнен в виде микропрограммы, зашитой (в прямом смысле проводами как нитками) в ПЗУ. Этот компьютер пользовался уважением у инженеров, исследователей благодаря дружественному интерфейсу с оператором-программистом. Его последователь - компьютер МИР-2 - имел консоль с дисплеем и световым пером и использовался как персональный компьютер инженера.

Быстрый рост рынка калькуляторов стал непосредственной причиной производства первого микропроцессора. Считается, что появление в 1971 году кристалла I4004, разработанного инженерами фирмы Intel, стало началом эры микропроцессоров. Главный разработчик этого микропроцессора - М. Е. Гофф создавал его под впечатлением от архитектуры PDP-8. Появление микропроцепрессоров стало возможным, очевидным и обязательным, когда количество транзисторов на кристалле стало большим нескольких тысяч.

Архитектурные возможности микропроцессоров постоянно расширялись с ростом количества транзисторов на кристалле. Еще в середине 1960-х годов Гордон Мур, один из основателей фирмы Intel, анализируя микросхемы различных успешных фирм-производителей, заметил, что примерно каждые два года количество транзисторов в них удваивается. Этот закон, названный именем Мура, взяли на вооружение экономисты фирмы Intel и они требовали

от проектировщиков новых микросхем его выполнения, так как он был залогом успешности бизнеса в области микроэлектроники. В результате, этот закон неукоснительно выполнялся в течение десятилетий и выполняется до сих пор.

Поэтому архитектурные возможности микропроцессоров постоянно расширялись с ростом количества транзисторов на кристалле. Так, микропроцессор I8080 имеет 4,5 тыс. транзисторов и обеспечивает выполнение только одной программы; микропроцессор I8086 с 29 тыс. транзисторов уже выполняет несколько программ под управлением операционной системы; 275 тыс. транзисторов микропроцессора I80386 обеспечивают уже работу в многозадачном режиме с виртуальной памятью.

По закону Мура развивались также микросхемы интегральной памяти. Появление дешевых микросхем динамической памяти в начале 1980-х лет позволило конструировать малогабаритные ОЗУ объемом до 1 Мбайт и тем самым обеспечить массовый выпуск персональных компьютеров. После достижения емкости ОЗУ величины 4–10 Мбайт стало возможным реализовать в компьютере операционную систему с графическим интерфейсом, такую как Windows-3. Когда эта емкость достигла примерно 100 Мбайт, появилась возможность обрабатывать в компьютерах мультимедийные данные.

Новые архитектурные признаки в микропроцессорах повторяли те же признаки в больших компьютерах образцов 1960-х - 1980-х годов, пока количество транзисторов на кристалле не стало превышать суммарное число транзисторов в микросхемах майнфреймов. Этот предел, равный примерно 1 млн. транзисторов, был пересечен в начале 90-х годов прошлого столетия.

Сейчас в распоряжении разработчиков новых микропроцессоров есть несколько миллиардов транзисторов. Все архитектурные идеи, внедренные в старых компьютерах, уже исчерпаны. Кроме того, приближается предел миниатюризациии с применением кремниевой технологии. Нужны новые архитектурные идеи, чтобы постоянный рост производительности компьютеров, наблюдаемый до сих пор, не прерывался.

На основе обзора истории компьютеров и их архитектур можно сформулировать следующие выводы.

- Появление компьютеров с одной и той же архитектурой было обусловлено высокой трудоемкостью программирования и необходимостью переносимости матобеспечения между компьютерами с различным устройством.
- Современные архитектуры, в значительной степени - это множество переосмысленных архитектурных признаков, изобретенных раньше.
- Большинство архитектурных признаков было выработано в период, когда аппаратные ресурсы были существенно ограничены.

- На архитектуру компьютеров влияло их назначение: для военных и научных расчетов, для экономических расчетов, для управления.
- На развитие компьютеров и, соответственно, их архитектуру существенно влияли коммерческая выгода и милитаристская потребность.
- Миниатюризация и новые изобретения позволяли увеличивать объем памяти. При достижении определенного предела объема памяти в корне изменялись как архитектура, так и методы программирования и устройство операционной системы.
- На успешность распространения архитектуры и компьютеров, в основном, влияют не столько их потенциальная эффективность, быстродействие, габариты и др., как коммерческий менеджмент, конкуренция и рыночная конъюнктура.
- Высокая цена аппаратных и временных ресурсов, малый объем ОЗУ компьютеров в начале 60-х годов заставили внедрить системы с разделением времени, многозадачный режим, защиту памяти, виртуальную память, кэш-память, удаленный доступ к процессору.

Основные определения относительно архитектур

Чтобы понять строение существующих компьютеров или успешно выполнить разработку новых компьютеров, свободно общаться со специалистами и соответствующей технической литературой, необходимо знать ряд определений, принципов и аксиом компьютерной науки. Некоторые из них перечислены ниже.

Состояние вычислителя - это состояние St_i всех его ячеек памяти в дискретный момент времени его работы. Например, состояние микропроцессора определяется содержанием его регистров команды, данных, счетчика команд, триггеров флажков, буферов интерфейсов, а также всех ячеек его оперативной памяти. В ячейках памяти могут храниться различные конструкные объекты, которыми могут быть отдельные биты, символы, слова, строки, числа, а также более сложные объекты, такие как записи, списки, файлы, массивы и тому подобное.

Вычислительный процесс - это последовательный ряд состояний St_i некоторого вычислителя, начиная с начального состояния St_0 и кончая результирующим состоянием St_E. Причем результирующее состояние может быть недостижимым, например, в процессоре для цифровой обработки сигналов или управления. В начальном состоянии определенные ячейки памяти хранят исходные данные, а в конечном состоянии в ячейках находятся результаты вычислений. Во время выполнения вычислительного процесса конструктивные объекты, хранящиеся в памяти, преобразуются и пересылаются с помощью

определенных операций, команд, присущих данному вычислителю.

Алгоритм - это задание вычислительного процесса вычисления определенной функции *F* на модели вычислителя, которая описана с помощью строгих математических понятий.

Это определение было впервые сформулировано Е.Л.Постом и А.Тьюрингом. При этом за модель вычислителя принималась машина Тьюринга. Эта машина представляется бесконечной лентой и устройством управления. На ленте головка записи-чтения может записывать и считывать биты данных. Устройство управления, согласно алгоритму, управляет передвижением и записью-чтением головки. Состояние этой модели определяется состоянием управляющего автомата, положением головки и состоянием ленты и называется конфигурацией машины Тьюринга.

Понятие алгоритма существует как интуитивное понятие, а не как строго ограниченное определение. Это понятие предполагает, во-первых, то, что должен существовать определенный субъект или процессор, который способен читать, распознавать конструктивные объекты и правильно выполнять операции над ними согласно алгоритму.

Во-вторых, несомненно, алгоритм создается с целью эффективно вычислить некоторую полезную функцию *F* ради получения правильного результата для определенного рода исходных данных.

В-третьих, для простоты восприятия алгоритма несущественные детали выполнения вычислительного процесса (которые, как правило, не отражаются модели вычислителя) не принимаются во внимание.

Поэтому вышеприведенное определение является лишь разъяснением алгоритма, чтобы представить дальше понятие архитектуры. Так, алгоритм может существовать отдельно от вычислительной модели - как список правил функционирования модели или быть частью модели, как устройство управления в машине Тьюринга, или быть самой моделью, например, моделью конечного автомата или графовой моделью. Например, если исполнителем алгоритма является человек или процессор последовательного действия, то традиционно принято называть алгоритмом список действий, инструкций, которые следует выполнять последовательно.

Архитектура - это модель реального вычислителя с уровнем детализации, достаточным для его разработки или программирования, которая способна выполнять соответствующую множество алгоритмов.

Программа - это алгоритм, который задан на архитектуре вычислителя по помощью алгоритмического языка или машинных кодов в зависимости от уровня детализации архитектуры. Итак, программа, написанная для определенной архитектуры, должна выполняться для одинаковых исходных

данных с одинаковыми промежуточными и конечными результатами на различных компьютерах, которые имеют эту архитектуру.

Архитектура с точки зрения разработчика - это модель компьютера с уровнем детализации, который является достаточным для его разработки и изготовления. Архитектура, как правило, включает в себя сведения о системе команд, адресное пространство, управление адресами, систему прерываний, защиту памяти, интерфейсы, периферийные устройства, то есть все сведения по детального технического задания на разработку компьютера. Описание такой архитектуры не включает сведения об элементной базе, ее быстродействии, габаритах, энергопотреблении, параметры надежности и т.д., так как они непосредственно не влияют на ход вычислительного процесса.

Архитектура с точки зрения программиста - это модель компьютера с уровнем детализации, который является достаточным для успешного программирования определенной вычислительной задачи на том или ином алгоритмическом языке.

Так, программист на языке Pascal принимает во внимание модель компьютера с памятью, которая адресуется 32-разрядной шиной, с АЛУ, которое обрабатывает целые числа, или числа с плавающей запятой, с памятью на жестком магнитном диске произвольного объема, с клавиатурой и дисплеем, доступ к которым обеспечивается процедурами, которые он может найти в библиотеке процедур компилятора Pascal.

Каждая строка алгоритма, описанного на языке ассемблера, представляет определенную машинную команду, выполняет элементарные действия с данными или управляет выбором следующей команды. Поэтому программист на языке ассемблера должен знать архитектуру компьютера в совершенстве с точностью до регистра и его бита, кода прерывания, адреса периферийного устройства, и тому подобное.

Определенная часть архитектуры для языка высокого уровня задается как ***интерфейс*** между этим языком и системным матобеспечением, которое непосредственно обеспечивает выполнение скомпилированных программ. Таким матобеспечением выступает операционная система или модель виртуальной машины.

Архитектурная платформа - это распространенная архитектура компьютеров, которая гарантированно неизменна в течение следующих нескольких лет. Архитектурную платформу можно использовать в новых вычислительных средствах и компьютерах, гарантируя совместимость матобеспечения, возможность использования готовых элементов структуры компьютеров, подключение существующих периферийных устройств и устройств, изготавливаемых другими фирмами. Самой известной архитектур-

ной платформой является I80x86. Ее синоним – 32-bit Intel Architecture (IA-32).

Архитектурная парадигма - это набор общеупотребительных принципов и подходов к проектированию архитектур компьютеров. Например, компьютер Eniac, выполненный по парадигме потокового спецпроцессора с единичным отображением графа алгоритма в структуру с конфигурируемыми связями. То есть, вершине графа ставится в соответствие операционный блок - сумматор или блок умножения, а его дуге - линия связи между операционными блоками. Большинство вышеупомянутых компьютеров построены по парадигмы машины фон Неймана, которая будет рассмотрена ниже.

Принципы построения компьютеров

Чтобы объяснить выбор архитектуры компьютеров и конкретной системы команд, вспомним принципы их построения, которые были изобретены и отработаны за годы развития компьютеров.

Принцип иерархии означает, что сложная вычислительная функция $F = (F_1, F_2, ..., F_i, ...)$, которая представляется композицией функций F_i, реализуется вычислителем $S = (S_1, S_2, ..., S_i, ...)$, причем каждая из функций F_i выполняется определенным блоком S_i. В свою очередь, функции F_i выполняются системой модулей на более низком уровне иерархии.

Принцип иерархии позволяет человеку понять, спроектировать и реализовать сложные технические системы, которыми являются компьютеры. При этом обзор, синтез (проектирование) или анализ системы идет ***снизу вверх***, то есть начиная от элементов и кончая сложными блоками, или ***сверху вниз***, когда сначала система рассматривается на высоком уровне и постепенно обзор спускается до уровня элементов.

Согласно этому принципу, компьютер строят как набор блоков, соединенных согласно структуре, которые в свою очередь, реализованы из модулей, таких как арифметическое устройство, регистр, счетчик, и тому подобное. Те, в свою очередь, выполняются на основе логических элементов с определенного элементного базиса. Точно так же массив данных иерархически состоит из отдельных слов, которые, соответственно, являются наборами битов. Сложная программа состоит из нескольких программных модулей, в состав которых входят подпрограммы, которые написаны как последовательность операторов, причем в откомпилированной программе каждый оператор выполняется как одна или несколько машинных команд.

Концепция операционного и управляющего автоматов утверждает, что вычислитель S эффективно осознавать, разрабатывать и реализовать как композицию операционного и управляющего автоматов. Первый из них выполняет определенный набор операций над данными, а второй - управление выполне-

нием этих операций по заданному алгоритму. Например, блок обработки данных, который рассматриваться ниже, состоит из арифметико-логического устройства (АЛУ), регистров данных, представляющих операционный автомат, и управляющего автомата, который обеспечивает подачу управляющих сигналов на регистры и АЛУ согласно алгоритмам выполнения команд процессора.

Согласно ***принципу программного управления***, операции по вычислению функции *F* кодируются командами, которые дешифруются управляющим автоматом и выполняются операционным автоматом. Процесс вычисления функции *F* описывается алгоритмом, который представляется как список выполняемых команд, в которых закодирован также порядок их выполнения, то есть в виде программы. Для изменения порядка выполнения действий используются управляющие команды.

Принцип микропрограммного управления. Он является производным от предыдущего принципа. Согласно ему, выполнение сложной операции, например, команды процессора, разлагается на микрооперации, выполняемые последовательно-параллельно отдельными блоками и узлами операционного автомата. Микрооперации, выполняемые параллельно, кодируются одной ***микрокомандой***, в которой также закодировано управления порядком выполнения таких микрокоманд. Совокупность микрокоманд составляет прошивку постоянной памяти микропрограмм.

Принцип хранения программы и данных в одной памяти (принцип фон Неймана) - принцип командного управления, согласно которому команды хранятся в памяти и обрабатываются так же, как и данные; следующая команда начинает выполнение только после окончания выполнения предыдущей команды; последовательность выполнения команд задается естественным порядком их нахождения в памяти за исключением случаев передачи управления по командам перехода. Так как память, как правило, имеет линейную адресацию, последовательные команды выбираются из нее по счетчику команд.

Благодаря этому принципу, можно сравнительно просто задавать вычислительный процесс в компьютерах, а также выполнять в них компиляцию программ. В наше время задание вычислительного процесса можно выполнить многими другими методами, благодаря автоматизации программирования, проектированию архитектур, работающих по другим принципам.

В компьютере, работающем по этому принципу, команды и данные разных типов различаются неявно. Считается, что арифметическая команда обращается к числовому данному определенного типа, а команда перехода - к конкретной команде программы. Но возможна ситуация, когда на месте данного сохраняется данное другого типа или команда. Эта ситуация является причиной большинства случаев неправильной работы компьютеров. Также это про-

тиворечит требованиям языков программирования, согласно которым операции обращаются к конструктивным объектам (числам, строкам, массивам) явным образом и не допускается несовпадение типов операции и объектов.

Согласно этому принципу, невозможно выполнять одновременно несколько команд в процессоре. Поэтому из-за строгого соблюдения этого принципа невозможен рост быстродействия компьютеров за счет распараллеливания операций, тормозится развитие вычислительной техники. В современных архитектурах этот принцип нарушается на аппаратном уровне для увеличения быстродействия. Из-за этого может снижаться надежность матобеспечения, созданного в соответствии с принципом фон Неймана.

Принцип синхронного управления означает, что все основные элементы памяти компьютера меняют свое состояние St_i одновременно под управлением одного источника синхронизации, а ход времени отсчитывается дискретно по номеру такта. При этом период интервала синхронизации определяется максимальной задержкой информационных сигналов при выполнении любой из микроопераций.

Каждый вычислитель, который проектировался в течение последних трех десятилетий, можно рассматривать как множество логических схем и множество регистров, которые тактируются от общего синхросигнала. Процесс обработки информации заключается в том, что сигналы, выходящие из регистров в начале каждого такта, распространяются через логические схемы с присущей им задержкой, преобразуются в них в соответствии с логическими функциями и запоминаются в регистрах в конце такта. Такой порядок обработки информации на физическом уровне понятен разработчикам и поддерживается всеми соврнменными системами автоматизированного проектирования (**САПР**) дискретных схем.

На более высоком уровне - уровне параллельных процессов - по этому принципу параллельные вычислительные процессы одновременно завершаются или обмениваются результатами под управлением источника управляющих команд - диспетчера операционной системы. Принцип синхронного управления на уровне процедур проявляется, например, при планировании последовательного выполнения программных потоков по квантами времени или при программировании многопроцессорных систем.

Используется также ***принцип асинхронного управления***, когда элемент памяти вычислителя меняет свое состояние только по специальному сигналу готовности данных. Этот сигнал вырабатывается логической схемой, которая обрабатывает данные. Поэтому состояния вычислителя меняются быстрее, и вычислитель может иметь более высокое быстродействие. Но проектировать такие вычислители значительно труднее, их скоростные характеристики

теряются при их использовании, когда необходимо выполнять соединение с устройствами, управляемыми синхронно или при выполнении алгоритмов с обратными связями. Этот принцип часто используется в телекоммуникационных системах, например, в протоколах компьютерных сетей.

Согласно ***принципу сбалансированности универсальности и специализации*** оптимальная архитектура компьютеров находится на границе множеств универсальных и специализированных архитектур. Из практики проектирования средств вычислительной техники известно, что универсальный процессор, способный выполнять большое множество алгоритмов, имеет время выполнения алгоритмов, энергозатраты, габариты или цену, которые на два – три порядка больше, чем у специализированного вычислителя, настроенного на выполнение одного алгоритма. Универсальные компьютеры, которые предназначены для выполнения определенного множества сложных алгоритмов за малый промежуток времени, то есть суперкомпьютеры, имеют наибольшие показатели цены, габаритов, энергопотребления.

Кроме того, множество алгоритмов компьютера с новой архитектурой определяется количеством программ, которые можно создать для него. Эта количество ограничено так же, как и ограничены время разработки и коллектив разработчиков этих программ.

Поэтому проектировщики новых вычислительных средств всегда ищут компромисс между универсальностью вычислителя и затратами на его производство и использование, привлекая матобеспечение, наработанное для предыдущих архитектур. Например, мобильные средства связи выполняют все большею множество алгоритмов, но оно ограничено, по крайней мере, возможностями этих средств экономно расходовать энергию при их выполнении, зарядом их аккумулятора и весом. Это также соответствует следующему принципу.

Согласно ***принципу эффективной многофункциональности***, увеличение функциональности компьютеров не должно приводить к пропорциональному увеличению их стоимости и сложности. В этом плане архитектура фон Неймана является совершенной, так как обеспечивает выполнение произвольного алгоритма при неизменных аппаратных затратах.

Принцип параллельной обработки информации. Для достижения высокой производительности и (или) надежности компьютеров независимые этапы вычисления функции F и управления разделяются между несколькими операционными и управляющими автоматами (или процессорными элементами), связанными некоторой системой коммутации.

Наиболее понятны и поэтому более распространены последовательные алгоритмы, которые описываются, например, последовательными программами. Операторы такого алгоритма выполняются последовательно на одном

ресурсе, например, в АЛУ неймановского процессора. Но ускорить выполнение алгоритма, используя ту же элементную базу, можно только задействовав принцип параллельной обработки информации.

Согласно ***принципу конвейерной обработки информации***, вычисления функции F разделяется на последовательные этапы F_1, F_2, ..., F_K, каждый из которых вычисляется на отдельном вычислительном ресурсе S_i конвейерного вычислителя, или просто конвейера. Данное X_j обрабатывается последовательно, проходя через ресурсы S_i, которые называются ступенями конвейера. Благодаря тому, что поток данных X_j через конвейер является непрерывным, увеличивается его производительность до K раз.

Принцип изменяемости заключается в возможности изменения множества элементов компьютера и связей между ними. Этот принцип проявляется в свойствах живучести и наращиваемости компьютеров, возможности адаптации архитектуры к специфике выполняемой задачи и обеспечивает реализацию принципа многофункциональности.

Принцип однородности заключается в том, что структура компьютера или его частей состоит из повторяющихся однотипных элементов и связей. Это уменьшает затраты на проектирование, изготовление и эксплуатацию компьютера. Память компьютера, состоящая из тысяч и миллионов одинаковых ячеек, строится по этому принципу. Все мультипроцессорные компьютеры состоят из нескольких одинаковых процессорных элементов. Причем благодаря принципу однородности, планирование вычислений в мультипроцесорном компьютере значительно проще, чем у компьютеров с, так называемой, гетерогенной структорой, состоящих из неоднородных процессоров.

Принцип унификации является универсальным принципом создания технических систем. Он заключается в том, что объект, который должен взаимодействовать с другими объектами, создается согласно стандартным требованиями. Благодаря этому, этот объект будет соединяться без проблем с другими унифицированными объектами, даже тогда когда они были изготовлены в разных местах и в произвольное время. Сама идея компьютерной архитектуры является результатом действия этого принципа, благодаря чему любая программа выполняется правильно на произвольном компьютере той же архитектуры.

Основным фактором унификации в вычислительной технике является интерфейс. ***Интерфейс***, как стандартное средство соединения, используется как в компьютерных шинах, так и программах. Особую роль играет интерфейс между человеком и аппаратно-программным комплексом, который помогает эффективно наладить взаимодействие между различными операторами и компьютером, используя одно и то же матобеспечение.

1.2 Элементная база для построения компьютеров

Как было показано выше, архитектура компьютеров развивалась взаимосвязано с развитием их элементной базы. Революционным толчком к информатизации и компьютеризации общества были появление и широкое распространение технологии разработки и производства интегральных схем (ИС) большой степени интеграции. Хотя архитектура компьютера, в принципе, не имеет сведений о примененной элементной базе, она в значительной степени определяется характеристиками ИС, которые в нем используются. Поэтому далее кратко рассмотрим современную элементную базу компьютеров.

Элементная база интегральных схем

ИС являются как важнейшей продукцией промышленности электроники, так и основой всех изделий компьютерной техники. ***Транзистор*** – это элементарный компонент ИС. Простейший элемент ИС – ***инвертор*** – состоит из двух комплементарных транзисторов со структурой метал-оксид-полупроводник (**КМОП**), как показано на функциональной схеме на рис. 1.1.

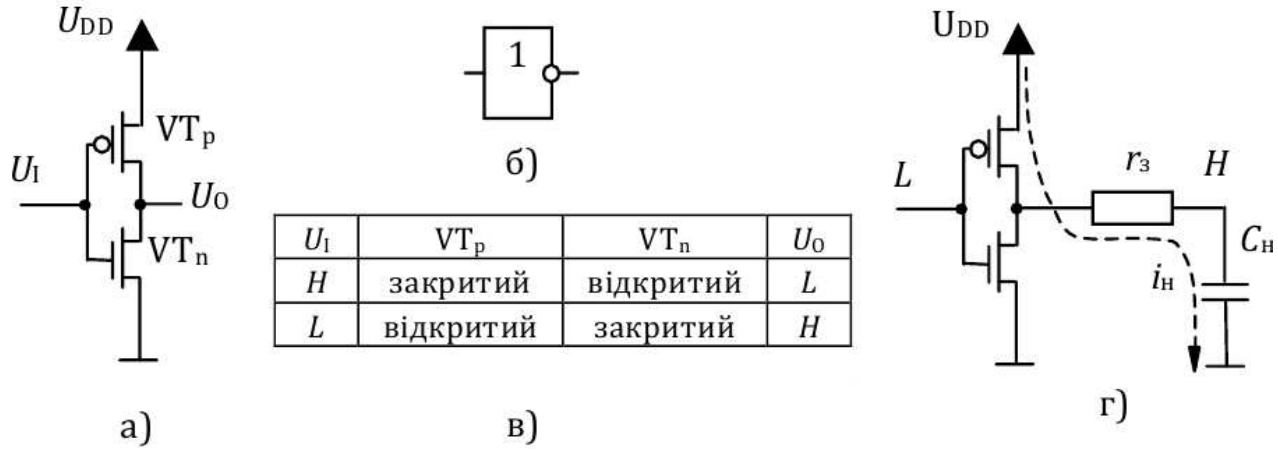


Рис.1.1. Функциональная схема инвертора а), его условное графическое обозначение б), таблица режимов работы инвертора в), работа инвертора с нагрузкой

p-канальный транзистор VT_p в этом инверторе всегда находится в противоположном, то есть комплементарном состоянии относительно n-канального транзистора VT_n, как показано в таблице истинности на рис.1.1.в. В этой таблицы символ H (high) означает высокий уровень сигнала, то есть логическую 1, а L (low) - низкий или логический 0. КМОП-инвертор имеет три составляющие энергопотребления: статическое, динамическое потребление и потреблени в нагрузке.

В статическом режиме или верхний, или нижний транзистор инвертора закрыт. Поэтому КМОП-инвертор в этом режиме практически ничего не потребляет. В момент переключения инвертора оба транзистора на очень короткое время могут стать открытыми и тогда возникает небольшой импульс динамического тока потребления.

Но больше всего энергопотребления инвертора связано с током нагрузки, который заряжает или разряжает емкость нагрузки $C_н$ и течет через сопротивление соединения $r_з$ и сопротивление открытого транзистора. При этом энергия $E_п$, которая считается потребляемой инвертором при его переключении, равна энергии, рассеиваемой на сопротивлении $r_з$ и открытом транзисторе и составляет энергию заряда емкости $C_п$, то есть ее можно оценить как

$$E_п = \frac{C_н U^2_{DD}}{2}, \qquad (1.1)$$

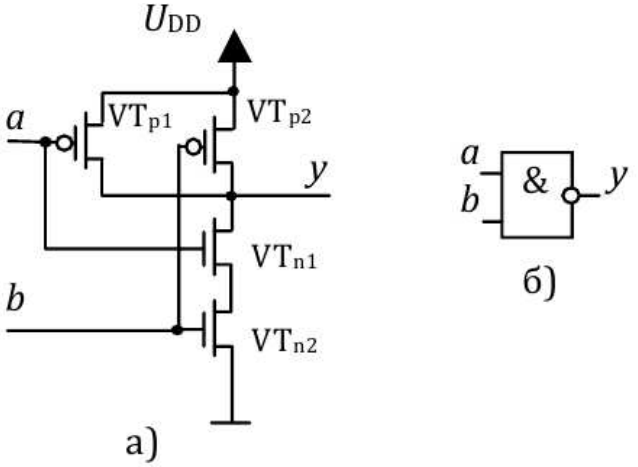
где U_{DD} - напряжение питания, которое примерно равно перепаду уровней H и L, $C_п$ - емкость нагрузки, равная суммарной емкости затворов КМОП-транзисторов всех схем, которые подключены к выходу инвертора, а также емкости линий связи.

Так как количество переключений в секунду инверторов и подобных им схем пропорционально тактовой частоте f_C ИС, то энергопотребление компьюров также пропорциональное этой частоте. Также, согласно (1.1), оно пропорционально емкости, которая зависит не только от сложности схемы, но и от технологии ее изготовления. Значение $C_н$ обратно пропорционально проектной норме ИС, равной минимальной ширине конструктивного элемента ИС, например, ширине линии связи или затвора транзистора. Кроме того, уменьшение C_H означает пропорциональное уменьшение продолжительности заряда этой емкости, а следовательно, увеличение быстродействия инвертора и ИС в целом. Поэтому с каждым новым поколением технологии ИС существенно уменьшается энергопотребление производимых микросхем и увеличивается их быстродействие.

Согласно формуле (1.1), эффективно минимизировать енергопотребление, уменьшая напряжение питания U_{DD}. Но при его уменьшении ниже предела около 0,9 В один из КМОП-транзисторов инвертора становится недостаточно закрытым и через комплементарную пару транзисторов протекает существенный постоянный ток.

Один двохвходовой логический элемент (ЛЭ), названный ***логическим вентилем***, может быть сформирован из четырех транзисторов, как на рис.1.2. Если на обоих входах *a* и *b* установлен высокий уровень, то транзисторы VT_{n1} и VT_{n2} открываются, так что выход *y* устанавливается в низкий уровень, в то время, когда транзисторы VT_{p1} и VT_{p2} являются закрытыми. При другой

комбинации входных сигналов, по крайней мере, один из транзисторов VT_{p1}, VT_{p2} откроется, а один из VT_{n1}, VT_{n2} - закроется и на выходе будет высокий уровень сигнала.



Входи		Вихід
a	*b*	*y*
L	*L*	*H*
L	*H*	*H*
H	*L*	*H*
H	*H*	*L*

в)

Рис.1.2. Функциональная схема элемента И-не а), его условное графическое изображение б) и таблица истинности в)

Основным элементом памяти компьютеров является триггер, который сохраняет один бит информации. Различают такие триггеры, как защелка и синхронный триггер. ***Защелка*** (latch) – это триггер, который имеет два основных режима работы: режим прозрачности, в котором выход триггера повторяет информацию на своем входе, и режим хранения в своей внутренней бистабильной схеме, которая имеет состояние Q = 0 или Q = 1. Режим защелки меняется по уровню входного информационного или синхронизирующего сигнала. Так как такой триггер имеет режим прозрачности, в котором выходной сигнал изменяется безотносительно синхросигнала, его часто называют ***асинхронным триггером***.

Синхросигнал - это управляющий сигнал, который периодически переходит из состояния 0 в 1, а затем обратно в 0. Синхросигнал обычно обозначается как C или CLK или CLOCK. Использование синхросигнала для управления работой триггера дает разработчику возможность задавать заранее определеные моменты времени, в которые данные могут быть запомнены в триггерах устройства.

Синхронизированная D-защелка в ИС создается по схеме как на рис.1.3. В ней инверторы D_1 и D_2 через обратную связь формируют бистабильную схему. Если синхросигнал C = 1 открывает транзистор VT_{n3}, то входной сигнал D откроет или транзистор VT_{n1} или VT_{n2} так, что бистабильная схема перейдет в состояние, соответствующее сигналу D. При этом триггер переходит в режим прозрачности - произвольные изменения сигнала D передаются на выход.

После закрытия транзистора VT_{n3} при C = 0 триггер переходит в режим хранения, а состояние, запомненное в последний момент, когда C = 1, выдается на выход неизменным.

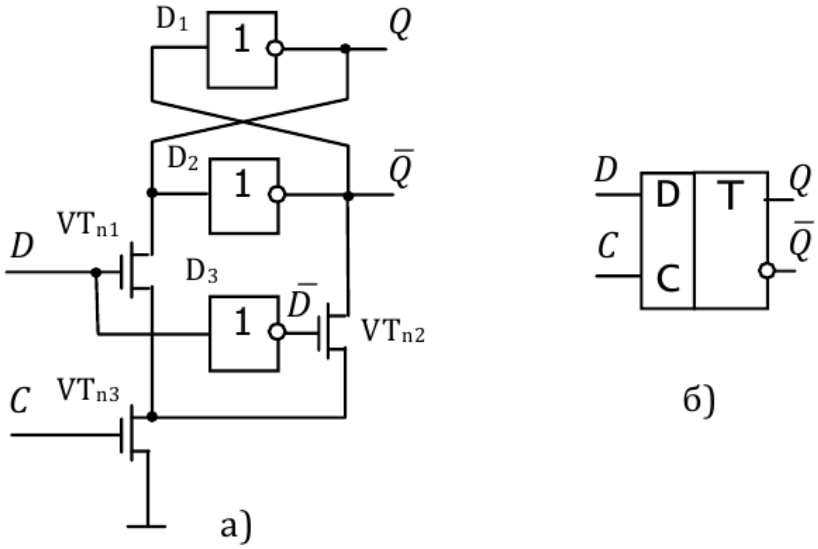


Рис.1.3. Функциональная схема элемента D-защелки а) и его условное графическое изображение б)

Схемы, где используются защелки, могут иметь непредсказуемое поведение. Так, сигнал D может распространяться через цепочку последовательно соединенных защелок, когда они находятся в режиме прозрачности, или в схеме, в которой защелки стоят в обратной связи, возможно возбуждение.

Поэтому для предотвращения таких явлений во всех современных компьютерах, как правило, используют только синхронные триггеры. В простейшей схеме ***синхронного триггера*** две синхронизированные защелки соединены последовательно, как показано на рис.1.4.

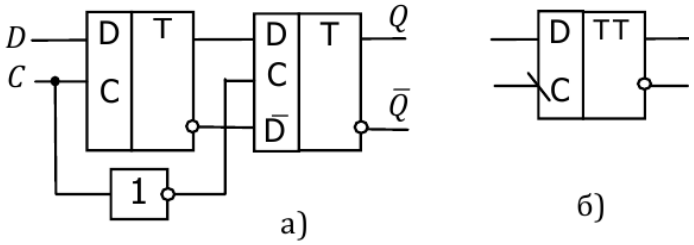


Рис.1.4. Функциональная схема элемента синхронного D-триггера а), и его условное графическое изображение б)

Первая защелка играет роль ведущей ступени и отвечает за прием входящего данного. Вторая защелка - это ведомая ступень и предназначена для хранения данного, которое получено от ведущей ступени. Для того, чтобы не произошло распространение данного сразу через две ступени в режиме

прозрачности, первая и вторая защелки синхронизируются противоположными фазами синхросигнала C. Следовательно, данное с ведущей защелки переписывается в ведомую защелку (и становится видимым на выходе Q) в момент, когда синхросигнал переходит из C = 1 в C = 0. Из-за такого свойства эта схема считается чувствительной к спаду синхросигнала.

Любую схему компьютера можно представить как множество комбинационных схем и множество регистров, набранных из синхронных триггеров. Эти множества связаны между собой соответствующей системой связей, как например, на рис.1.5.

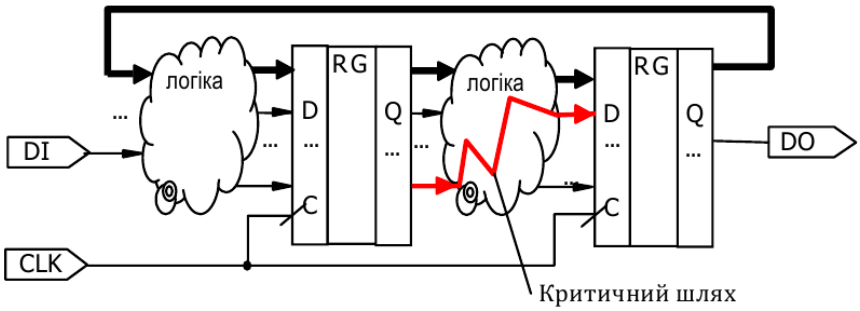


Рис.1.5. Функциональная схема произвольного компьютера

При однотактной синхронизации все триггеры компьютера выполнены как синхронные и тактируются фронтом одного синхросигнала. В современных микросхемах сеть передачи синхросигнала от своего источника ко всем триггерам выполняется с особой тщательностью. Благодаря этому, она обеспечивает минимальный дисбаланс задержек прихода фронта синхросигнала до каждого триггера. Поэтому минимальный период синхросерии T_{Cmin} микросхемы оценивается как максимальная задержка между выходом одного произвольного триггера и входом другого, то есть сигнал между этими триггерами проходит через критический путь в комбинационной схеме (рис.1.5). Соответственно, максимальная тактовая частота процессора, в определенной мере определяет его быстродействие и определяется как $f_{Cmax} = 1 / T_{Cmin}$.

С постоянным уменьшением проектных норм технология производства ИС перешла в такой период, когда задержки в вентилях стали меньше задержки соединений между ними. Кроме того, истончение проводников межсоединений и соответствующий рост их сопротивления, а также повышение частоты f_{Cmax} до нескольких гигагерц приводит к существенному угасанию сигналов в межсоединениях, а также к уменьшению скорости их распространения (рис. 1.1, г). Это привело к тому, что с одной стороны, для поддержания требуемой формы логических сигналов необходимо вставлять дополнительные буферы и триггеры по ходу их распространения, с другой стороны, рост тактовой частоты f_{Cmax} приостановился на предельных значениях 5-7 ГГц.

Классификация интегральных схем

Сложность цифровых ИС обычно выражают в числе эквивалентных двухвходовых логических вентилей. По этому числу все ИС делятся на ИС малой степени интеграции, большой степени интеграции (БИС, large scale integration – LSI) и сверхбольшой степени интеграции (**СБИС**, very large scale integration – **VLSI**). СБИС состоят более чем из сотни тысяч вентилей.

Множество схем СБИС состоит из микропроцессоров, микроконтроллеров, ИС памяти, заказных СБИС (application specific integral circuits – **ASIC**), и многофункциональных СБИС (application specific standard products – **ASSP**). Современная схема СБИС может содержать до десятков миллионов вентилей.

Кроме того, микросхемы памяти имеют объем до миллиардов бит, причем на запоминание каждого бита необходимо от одного до десяти транзисторов. ***Микропроцессор*** – это главный операционный блок компьютера. Его функциональность не определена не только во время его разработки, производства, но и в течение его использования. Это зависит от пользовательских программ и операционной системы (см. табл.1.1).

Микроконтроллер, как программно управляемый процессор, имеет такие же свойства, как и микропроцессор. Но его функциональность обычно неизменна и определяет функциональность изделия, в котором он встроен. Это означает, что микроконтроллер выполняет единую программу, которая, как правило, остается неизменной в течение всего времени использования изделия.

Сигнальные микропроцессоры (digital signal processor – DSP) формируют отдельное подмножество микроконтроллеров.

Микросхема ***заказной*** **СБИС** имеет такое название из-за ее функциональности, которая закладывается в нее во время проектирования и производства. Поэтому заказная СБИС осуществляет одну, но сложную функцию. Примеры заказной СБИС - схема модема, контроллер дисковода, ИС чипсета компьютера, ИС шифрования. Так как постоянно увеличивается стоимость проектирования СБИС, их разработка и производство могут приносить прибыль, когда серия производства достигает миллионов микросхем.

Микросхемы ***многофункциональных*** **СБИС** – ASSP формируют широкий набор различных устройств. Функциональность такой ИС меньше функциональности микроконтроллера, но ее достаточно, чтобы ее настроить на множество различных приложений. С точки зрения проектировщика, многофункциональная СБИС - это заказная СБИС, которая имеет возможность адаптации к области применения. Примерами многофункциональных СБИС является ИС специализированной памяти, микроконтроллер со специфическим набором периферийных устройств, например, MP3-плеер, дешифратор изображений стандарта MPEG.

Таблица 1.1. Функциональность микросхем

Вид микросхемы	Этап разработки	Этап изготовление	Этап внедрения и использование
ASIC	известна	известна	известна
ASSP	неизвестна	известна	известна
Микроконтроллер	неизвестна	неизвестна	известна
Микропроцессор	неизвестна	неизвестна	неизвестна

Сложные ПЛМ (complex programmable logic device - **CPLD**) и программируемые логические интегральные схемы (**ПЛИС**, field programmable gate array - **FPGA**) формируют особые подмножества многофункциональных СБИС. ПЛИС представляет собой массив 2-6-входных логических элементов, триггеров и металлических полосок, которые соединены между собой большим множеством программируемых мостиков. Эти мостики сформированы полевыми транзисторами, которые управляются специальными программирующими триггерами. Маршруты соединение элементов ПЛИС программируются изменением электрического поля в затворах полевых транзисторов мостиков, что породило название FPGA. Перед использованием ПЛИС последовательный программируемый поток, называемый конфигурацией, автоматически загружается в ПЛИС с выхода внешнего ПЗУ. Этот процесс называется ***конфигурированием* ПЛИС**.

Современные ПЛИС содержат блоки ОЗУ, блоки умножения, быстрые интерфейсы, ядра процессоров и другие специализированные модули. Логический объем современных ПЛИС достигает двух-четырех десятков миллионов эквивалентных вентилей. Иногда ПЛИС является частью другой многофункциональной СБИС.

Проектирование систем на сложных ПЛМ и ПЛИС экономически эффективно и выполняется быстро. Поэтому сейчас ПЛИС очень часто рассматривается как альтернатива заказным СБИС, особенно, когда запланированная серия устройств не превышает сотни тысяч единиц. В результате, ежегодно число новых проектов на ПЛИС и сложных ПЛМ растет, а число проектов заказных СБИС уменьшается. ИС памяти делятся на ИС оперативной памяти или памяти с произвольным доступом, или ИС оперативного запоминающего устройства (**ОЗУ**, random access Memory – **RAM**) и ИС постоянного запоминающего устройства (**ПЗУ**, read-only memory – **ROM**).

Система, которая включает в себя процессор, специализированные процессоры, ОЗУ, ПЗУ, периферийные устройства и т.п., которые реализованы в одной ИС, получила название ***система на кристалле*** (СНК, system on the chip - SOC).

1.3. Архитектура компьютеров фон Неймана

Типичное программно-управляемое ядро компьютера, который функционирует по принципу фон Неймана, состоит из блока управления (БК) – управляющего автомата и блока обработки данных (БОД) - операционного автомата. БК обеспечивает выборку и декодирование команд и выдачу соответствующих управляющих сигналов. В свою очередь, БОД состоит из трех групп схем: блока регистров данных, арифметико-логического устройства (АЛУ) и блока локальной памяти (см. рис. 1.6). Часто ядро компьютера называют центральным процессорным элементом (ЦПЭ).

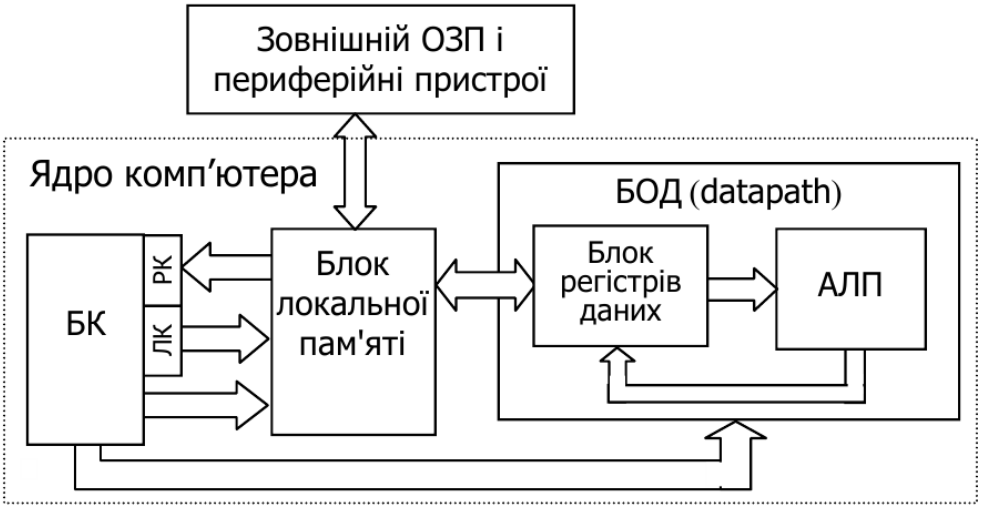


Рис.1.6. Структура процессора фон Неймана

Блок обработки данных обеспечивает выполнение каждой команды в ядре процессора. Блок регистров данных - это группа регистров общего назначения, которые хранят данные для вычислений при выполнении команд. АЛУ выполняет все арифметические и логические операции, а также операции сдвига. Блок локальной памяти сохраняет данные и программы и обеспечивает их быструю выборку. В современных процессорах – это ассоциативное оперативное запоминающее устройство (ОЗУ), то есть, кэш-ОЗУ.

Функционирование модели компьютера фон Неймана основывается на повторении четырехэтапной процедуры выполнения команды.

На первом этапе, который называется выборка команды, ядро процесора посылает к внешнему ОЗУ адрес команды счетчика команд (ЛК) и сигнал, что нужна следующая команда. ОЗУ отвечает тем, что высылает эту команду, которая попадает в блок управления, а именно - в регистр команды (РК).

На втором этапе - этапе декодирования - блок управления решает, какая это команда и какие действия надо сделать для ее выполнения. Полученная информация пересылается в блок обработки данных.

Третий этап - это этап выполнения команды. Блок обработки данных получает необходимые данные из блока регистров данных или из локальной памяти, или внешнего ОЗУ, обрабатывает их и выдает результаты.

Четвертый этап - этап запоминания результатов в локальной памяти или внешнем ОЗУ. При этом также определяется адрес следующей команды, как правило, в регистре - счетчике команд.

Каждая команда выполняется с такой же последовательностью этапов. Ядро процессора повторяет эти четыре этапа во время выполнения всей программы. Отличие лишь в том, что разные команды неодинаково управляют блоком обработки данных и выборкой данных и последующей команды. Все вышеприведенные функции этапов выполняются в блоке обработки данных и блоке управления. Эти функции подразделяются на категории в зависимости от их цели.

Функции выборки команды управляют потоком команд. Они обеспечивают выборку следующей команды (вычисления ее адреса) в зависимости от условий. Это условия команд перехода, вызова процедур и т.д.

Блок управления выполняет ***функцию дешифрования***, чтобы определить, какую функцию следует вычислять по определенной команде. ***Функции управления блоком обработки данных*** обеспечивают соответствующую последовательность управляющих сигналов для выполнения арифметических и логических действий, сдвига в АЛУ, а также пересылки данных.

Функция адресации данных обеспечивает подачу необходимых адресов данных в соответствии с заданным типом адресации. Некоторые из функций выполняет роль управления памятью, включая поддержку виртуальной памяти, отображения страниц памяти и управления кэш-ОЗУ.

Функции управления процессами обеспечивают одновременное выполнение в компьютере нескольких вычислительных процессов, как, например, в режиме мультипрограммирования. Такой функцией является функция прерывания.

Прерывание – это остановка нормального хода выполнения команд. Есть две ситуации, когда это выполняется. Первая, называется ***ловушка*** (***trap***), возникает когда процессор определяет, что выполнение команды вызывает некоторую ошибку (переполнение, доступ к защищенной области и т.д.). Вторая, называется собственно ***прерыванием***, (***interrupt***) возникает, когда внешнее устройство подает сигнал, что какое-то событие требует специальной

обработки. Неймановские архитектуры можно классифицировать по ряду признаков. К ним относятся:

– система команд;
– форматы данных;
– конфигурация пространств памяти данных и программ;
– временная диаграмма выполнения команд;
– структура процессора;
– интерфейсы процессора;
– система прерываний;
– механизм защиты памяти;
– набор периферийных устройств;
– программное обеспечение и другие.

1.4. Система команд компьютеров и ее выбор

Слово команды процессора может быть разделено на операционную часть - код операции (КОП) и адресную часть:

КОП	Адресная часть команды

В поле КОП помещается код операции - функции команды, которую выполняет ядро процессора при выполнении этой команды. Очевидно, что чем больше битов содержит это поле, тем больше определенных команд можно в нем закодировать.

Простейший подход к кодированию - это когда используется постоянное число битов КОП во всех командах. На практике, при разработке системы команд, можно определить, что некоторые команды более общие, чаще используются, и их КОП может быть короче.

Например, пусть мы выбрали 4-разрядный код операции для важнейших команд. Тогда возможны 16 комбинаций таких кодов. 15 из них могут использоваться для самых распространенных команд, а шестнадцатая комбинация - для указания, что это остальные команды. Тогда дополнительные биты в этой комбинации кодируют остальные менее важные команды.

Другой пример. Коды 0000, ..., 1101 кодируют 14 наиболее употребляемыех команды, а коды 111X - имеют префиксы длиной 4 и 8 бит, то есть отвечают шестнадцати кодам операций 11100000 ,..., 11101111 и еще 256 кодам 111100000000, ..., 111111111111.

По этому принципу построены системы команд процессоров фирмы Intel, которые имеют КОП длиной от 5 до 19 бит. Из-за этого, а также того, что команды могут выполнять сложные действия, такую систему команд называют ***системой сложных команд***. Такая система команд используется в CISC-процессорах (complex instruction set computer – **CISC**).

С другой стороны, выбор системы команд с малой сложностью, которые распространены в RISC-поцесорах (reduced instruction set computer – **RISC**), строго определяет, что должны быть КОП одинаковой длины и малое количество форматов команд, которые имеют одинаковую разрядность. Регулярность таких форматов упрощает декодирование команд и распараллеливание их выполнения.

В адресной части команды находится информация об адресах операндов и результата, а также о местонахождении следующей команды. Различают **4, 3, 2, 1 и 0-*адресные команды***. 0-адресная команда предполагает, что адрес задан неявно, например, он находится в заранее определенном регистре, таком, как аккумулятор, регистр флажков или по неявно заданному адресу, который хранится в указателе стека, счетчике команд. При этом адрес используемого регистра, закодирован в КОП. 0-адресная команда другого типа содержит в себе, так называемый, непосредственный операнд, то есть константу.

1-адресная команда предполагает, что один операнд или результат находится по указанному адресу, а другой – получается так же, как в 0- адресной команде. В четырехадресной команде есть четыре адресных поля, кроме трех адресов операндов и результата указывают адрес следующей команды.

Количество адресов в команде зависит от пространства адресов процессора. Оно оптимизируется при разработке системы команд. С одной стороны, одна команда с многими полями адресов заменяет ряд коротких команд, обеспечивая высокое быстродействие. Так, например, в компьютерах первых поколений была команда, которая обеспечивала переход по одному из трех адресов в зависимости от того, результат больше, меньше или равен нулю.

С другой стороны, в современных компьютерах пространство адресов требует длину слова адреса до 4 и даже 8 байт. Поэтому команды с большими полями адресов занимают значительный объем памяти и увеличивают время выборки команды. Поэтому выбор формата команд - это сложная задача оптимизации.

Эффективность системы команд проверяется путем моделирования работы процессора с выбранной системой команд и сравнение результатов моделирования с параметрами других процессоров (время выполнения тестовых программ, их длина после компиляции и т.д.).

Одной из идей создания RISC-процессоров была идея разделения команд на такие, которые работают только с блоком обработки данных, и такие, которые нет. Команды первого типа – это 3- или 4-адресные команды. В них используются адреса регистровой памяти, которые являются кодами длины 4-5-бит (иногда – до 7 и 8 бит). Команды второго типа – это 0-, 1- и 2-адресные команды. В двухадресных командах один из адресов – это ссылка на ячейку в ОЗУ, а вторая – это адрес регистра. Такая команда обеспечивает загрузку данного в регистр или пересылка данного из регистра в ОЗУ.

Адресное поле необязательно указывает адрес одного операнда. Существуют команды, адресные поля которых содержат информацию о множестве или диапазон адресов. Тогда такая команда выполняет обработку многих операндов с заданного множества или диапазона. Например, по команде LDM архитектуры ARM выполняется загрузка до 16 регистров ОЗУ, номера которых позиционно указаны в 16-разрядном адресном поле.

Поле адреса может представлять или прямой код адреса, или код, который играет существенную роль в получении такого адреса, например, код косвенной адресации. Подробнее информация в адресных полях будет рассмотрена при рассмотрении видов адресации, используемых в компьютерах.

1.5. Форматы данных и операции с ними

Данные в компьютерах – конструктивные объекты архитектуры – всегда представлены в двоичном виде в некотором формате. Вопрос того, как представлять данные, осложняется тем, что в компьютерах от различных производителей те же данные часто представлены в различных форматах. В таких компьютерах используется разная длина машинного слова и особенные наборы символов. Соответственно, используются различные способы сохранения целых чисел и чисел с плавающей запятой. Но в архитектуре большинства микропроцессоров соблюдены одинаковые правила представления данных – одинаковые их форматы.

Представление символов

Годами использовался один и тот же подход: выбиралось некоторое число бит для представления символа и составлялась таблица соответствия кода и символа (буквы, цифры, знака, специального знака). Например, во многих компьютерах первых поколений использовались 6-разрядные символы для задания больших букв, цифр, знаков операций. Теперь для кодирования используют 7-8 разрядные коды, задающие также и малые буквы. 8-разрядные

коды задают как латиницу, так и кириллицу. В мире таблица кодов символов стандартизирована. Наиболее распространенный стандарт – American Standard Code for Information Interchange – **ASCII**. У нас этот стандарт немного видоизменен и принят как стандарт 8-разрядных Кодов Обмена Информацией – **КОИ-8**.

8-разрядные коды не могут быть приспособлены для отображения любого алфавита. Поэтому в разных странах используют, так называемые, ***кодировки*** – производные ASCII таблицы, которые полностью поддерживают латинские буквы и отличаются в рисунке национального алфавита. То есть 128 первых кодов таблицы – те же во всех кодировках, а различие – в 128 кодах второй половины таблицы.

Так, таблица ASCII с украинскими буквами имеет номер 866. В операционной системе Windows используются другие таблицы, например, с кириллицей под номером 1251.

С целью поддержки таких алфавитов, как японский, китайский, а также для получения международной таблицы кодирования, буквы которой выглядят одинаково во всех операционных системах, сейчас распространяется 16-разрядное кодирование ***Unicode***.

Некоторые архитектуры поддерживают представления букв тем, что они имеют специальные команды, которые отличают кодирования букв. Так, например, в архитектуре Intel 80x86 есть команды, которые обеспечивают арифметические действия с байтами, представляющими десятичные цифры, например команды aaa, aas, aam, aad, которые выполняют десятичную коррекцию после сложения, вычитания, умножения и деления чисел, представленных ASCII-кодами.

Представление строк

Строки букв представляются в виде последовательности байтов, один или два байта на букву. Система команд микропроцессора, как правило, поддерживает обработку строк, такую как пересылка, сравнение, сравнение с шаблоном, заполнение, ввод-вывод строк. Максимальная длина строки, которая обрабатывается по одной команде, зависит от архитектуры определенного процессора.

Пересылка строк, как правило, происходит байт за байтом. Следует внимательно программировать пересылку строки, начальные и конечные адреса которой перекрываются, так как при этом можно испортить эту строку, переписывая байты с одной части строки в другую. Строки имеют произвольную длину, так что язык программирования должен обеспечить способ определения длины строки. Так, например, в языке Си нулевой байт означает конец строки.

Представление целых чисел и операции с ними

В первых компьютерах, а также в спецпроцессорах основным представлением чисел было представление двоичными дробями с запятой, которая стоит между знаковым разрядом и старшим разрядом числа. При таком представлении удобно выполнять все арифметические операции, округляя результаты с количеством разрядов разрядной сетки. Но при этом адреса в компьютере представляются целыми числами, для которых следует задействовать другие операции умножения и деления, чем для дробных чисел, что неудобно. Поэтому сейчас в компьютерах арифметика дробных чисел непосредственно не используется.

Целые числа представляются сейчас в компьютерах только как двоичные коды с дополнением до двух. Хотя было время, когда такие числа представлялись десятичными и даже троичными кодами. Также были попытки их кодирования числами Фибоначчи или кодами остатков.

Для положительных чисел представление двоичным кодом очевидно. Каждый бит представляет величину 2 в степени номера бита. По условной договоренности ученых-компьютерщиков, старший бит при написании стоит слева. Так, число 130 представляется байтом 1000 00102 = 82h или полусловом 0000 0000 1000 00102 = 0082h, где h - суффикс, обозначающий ***шестнадцатиричное число*** (hexadecimal).

Есть несколько способов представления отрицательных чисел, но во всех современных микропроцессорах отрицательные целые числа представлены с дополнением к двух. При таком представлении два одинаковых по модулю и противоположные по знаку числа при сложении дают число 2^k, где k - разрядность чисел, бит. Например,

A = 01012 = 5 и -A = 10112 = -5; A + (-a) = 100002 = 24.

При этом самым большим четырехразрядным числом является 0111 = 7, а наименьшим - 1000 = -8.

Пусть складываются два кода: 0111 + 0111. Результат должен быть +14, но получим 1110 = -2. Итак, мы имеем результат, который с точки зрения представления чисел со знаком является переполнением разрядной сетки. В большинстве процессоров такое переполнения фиксируется в флажке переполнения V = 1 (***oVerflow***), а также в флажке переноса C = 1 (***Carry***). В противном случае при выполнении операции, например, 0011 - 0111 получаем правильный результат - 1100, хотя с переносом, поэтому для него V = 0, C = 1. В каждом компьютере есть также флажок нулевого результата Z (***Zero***), а также отрицательного результата N (***Negative***).

При умножении целых чисел различают операции умножения целых без знака и умножения целых со знаком. Например, в первом и втором случаях имеем: 0001 * 1111 = 00001111 и 0001 * 1111 = 11111111.

Соответственно, различаются команды умножения со знаком и без него. То же относится к операциям деления. Часто в компьютерах реализованы умножение и деление без знаков. Тогда знак учитывается с помощью дополнительных вычислений, которые требуют соответствующего избытка времени.

Аналогично следует различать операции сравнения, потому что 1110 > 0111 (без знака) и 1110 < 0111 (со знаком).

Операции сдвига

Операцию сдвига на k разрядов можно считать умножением на коэффициент 2^k при сдвиге влево или на коэффициенты 2^{-k} - при сдвиге вправо.

Различают арифметический и логический сдвиги вправо, соответственно, целых чисел со знаком и без знака. Например, следующие команды процессора I80x86 выполняют различные сдвиги:

```
mov DX, 8C00h; загрузки отрицательного числа
sar DX, 4; сдвиг вправо со знаком на 4 разряда, результат - F8C0h
shr DX, 4; сдвиг вправо без знака на 4 разряда, результат - 0F8Ch
shl DX, 4; сдвиг влево на 4 разряда, результат - F8C0h
```

Также используют операции циклического сдвига, когда разряды, выдвигаются поступают в тот самый операнд с противоположной стороны.

Последний разряд выдвигается и запоминается в флажке переноса C для дальнейшего использования в качестве условия в команде условного перехода. При многоразрядном сдвиге выдвигаемые разряды можно запомнить в отдельном регистре, например, в процессоре I80x86:

```
shld AX, DX, 4; 4 разряда, выдвигаемых слева с DX, задвигаются в AX
```

В простых микропроцессорах операция сдвига на k разрядов используются для нормализации мантиссы при выполнении подпрограмм вычислений с плавающей запятой. Для ускорения этих вычислений в системе команд есть специальная команда, которая возвращает число k разрядов, на которые необходимо сдвинуть операнд при нормализации.

Операция распространения знака

Часто выполняются операции преобразования форматов чисел меньшей разрядности в числа большей разрядности, или используются короткие числа в расчетах. Если первоначальное число без знака, то к нему слева добавляются нули. А если со знаком, то добавляются знаковые разряды, и такая операция называется распространением знака. Большинство процессоров имеют специ-

альные команды для этого. Если они отсутствуют, то необходимо число дополнить нулями, сделать сдвиг влево на соответствующее количество разрядов, а затем – вправо, выполняя сдвиг со знаком. Например: В памяти хранится байт: AAh байт загружен в регистр полуслова: 00AAh, полуслово сдвинуто влево на 8 разрядов: AA00h, оно же сдвинуто вправо на 8 разрядов: FFAAh.

Хранение чисел в памяти

Если в процессорном ядре обрабатываются и хранятся многобайтовые слова, есть два варианта порядка их записи во внешнюю память: старшими байтами вперед (big-endian), то есть старшие байты записываются по меньшим адресам, или младшими байтами вперед (little-endian). Примеры записи слова старшими и младшими байтами вперед для четырехбайтных слова показано на рис. 1.7.

Многобайтовую слово адресуется, как правило, за младшей адресу б этого слова. Итак, в архитектуре "старшими байтами вперед", такой, как Apple, MIPS, первым читается старший байт, а в архитектуре "младшими байтами вперед" первым читается младший байт, так как в архитектуре i80x86.

Следует отметить, что для обеспечения быстрого доступа к слову долгоной 2k, оно записывается по адресу, в коде которой k младших битов является нулевыми. В этом случае говорят, что адрес слова (полуслова) выровнена по границей слова (полуслова). При этом, как правило, можно читать 2k байтов одновременно. В противном случае необходимо читать слово побайтно или по частям с последующим соединением прочитанных байтов в целое слово.

Для обеспечения совместимости архитектур, архитектуры современных микро- ропроцесорив, как правило, позволяют перед запуском операционной сис- темы переключать порядок записи байтов из старших на младших байта вперед и наоборот.

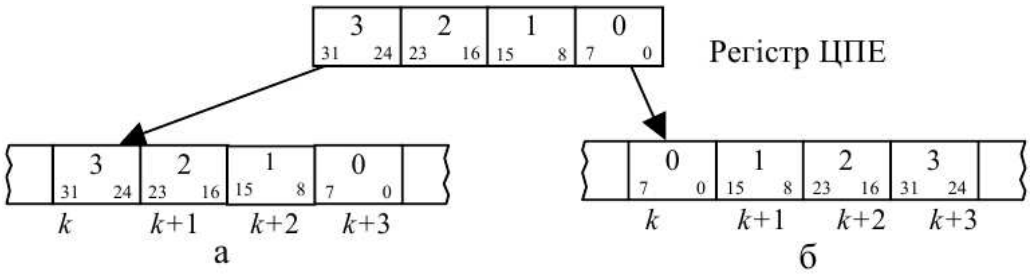


Рис.1.7. Запись слова в память со старшими байтами вперед (а) и младшими байтами вперед (б), начиная с адреса k

Арифметика расчета адресов

Арифметика целых чисел без знака, как правило, используется для расчета адресов. При этом действует семантика, когда переполнения не принимается во внимание. То есть, вычисления выполняются по модулю 2^k.

Например, пусть исчисляется базовый адрес плюс смещение. Тогда если код смещения D больше, чем 2^{k-1}, то такое смещение действует так, как от базовой адреса отняли код (2^k – D).

Например, при k = 16, если D = 0FFFFh = 10000h – 1, то 3456h + D = 3455h.

По этой причине здесь важно также использовать операцию распространения знака, если адрес дается краткое отрицательное число.

Представление десятичных чисел

Десятичные числа могут быть представлены как строки букв. При этом строки для расчетов необходимо преобразовывать в целые двоичные числа, а после расчетов – преобразовать их обратно. В кодах ASCII цифра и представлена байтом, в младшей тетраде которого стоит двоичный код и, а в старшей – код 3. Например, число 25 кодируется как 3235h. Система команд архитектуры I80x86 имеет команды для обработки двоично-десятичных неупа- кованых чисел, цифры которых представлены такими кодами.

Рациональным является представление цифры 4-разрядным двоично-десятичным кодом, так что в одном байте помещается две цифры. Такой стандартный формат данных называется ***упакованным десятичным кодом***. Некоторые архитектуры компьютеров, такие как IBM370, имеют команды арифметики десятичных упакованных чисел. Все процессоры могут оперировать с упакованными десятичными кодами по соответствующим подпрограммам.

Представление чисел с плавающей запятой

Такое представление широко используется в научных расчетах, так как оно обеспечивает вычисления с малой погрешностью без заботы о масштабировании данных. Формат числа с плавающей запятой состоит из бита знака S, поля порядка E и поля мантиссы M, следовательно, число равно $(-1)^S \cdot K^E \cdot$M, где K - основание порядка. В разных компьютерах K бывает равным 2, 4, 8, 10 и 16.

До применения стандартного представления чисел с плавающей запятой вычисления в компьютерах были несовершенными. Если два целых числа делятся нацело, то в некоторых архитектурах соответствующие числа с плавающей запятой нацело не делились. В других архитектурах результат A + A не равен 2.0·A. Также равенство (A – B) = – (B – A) не исполнялось. А малое

ненулевое число – делитель X в операции A/X могло вызвать прерывание "деление на 0".

В 1985 году был опубликован стандарт IEEE-754 – стандарт двоичной арифметики с плавающей запятой. Он предусматривает два основных формата: ***single*** (одинарный) разрядностью 32 и ***double*** (двойной) разрядностью 64. Одинарный и двойной форматы показаны на рис. 1.8. В одинарном формате минимальный порядок равен 0000 0001, что соответствует значению 2^{-126}, максимальный порядок равен 1111 1110 и соответствует значению 2^{127}. 23-разрядный код мантиссы равен 24-разрядной мантиссе, потому что старшая единица в нормализованной мантиссе не запоминается.

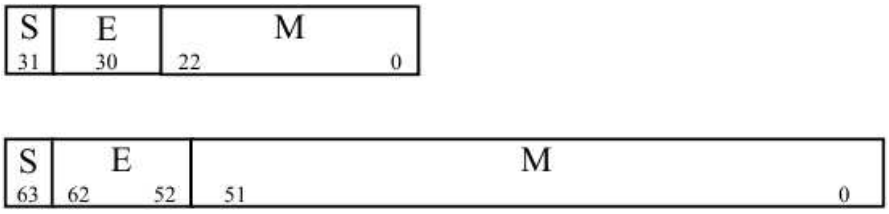


Рис.1.8. Форматы чисел с плавающей запятой

Например, код 1 1000 0000 100 ... 00 имеет знак минус, E = 128, то есть представляет порядок $2^{128-127} = 2$ и мантиссу 1.10 ... 0 = 1.510, то есть это число равно десятичному -3.0.

Стандарт предусматривает четыре режима округления результата:

– округление до ближайшего числа;

- усечение, лишние младшие разряды мантиссы отбрасываются, то есть это округление к нулю;

- округление до большего (к + ∞). Результат – ближайшее больше число.

- округление к меньшему (к – ∞). Результат – ближайшее меньшее число.

Режим округления записывается в управляющем регистре сопроцессора с плавающей запятой, таком как FPCR в архитектуре сопроцессора i80x87.

Стандарт также предлагает форматы с расширенной точностью. В одинарном расширенном формате мантисса имеет разрядность 32, порядок – 11 или больше, а в двойном расширенном формате – 64 и 15, соответственно. Эти форматы используются в аппаратных ускорителях вычисления математических функций, чтобы их результаты имели все точные разряды с одинарной или двойной точностью.

Сейчас внедряется стандарт, согласно которому возможны вычисления с 128-разрядными числами с плавающей запятой.

Сложилась традиция, что АЛУ с плавающей запятой предупреждает о переполнении разрядной сетки, что как правило, вызывает прерывание. Стандарт предлагает автоматически заменять переполнение специальным кодом E = 1111 1111 и M = 00 ... 0, что означает + или – бесконечность (+ INF, - INF). Тогда, когда такое значение встречается в последующих вычислениях, то оно ведет себя логичным образом, например: $1.0 + \infty = +\infty$; $1.0 / \infty = 0.0$.

Но в некоторых случаях результатом операций не может быть какое-то значение. Это, например, результат действий: $\infty - \infty$; $0.0 / 0.0$. В таких случаях возможен переход к исключительному состояния или замена результата на специальное значение с названием "***не-число***" – **NaN** (not a number), закодированное как E = 1111 1111, $M \neq 00 \ldots 0$.

Стандарт требует использования двух типов не-числа – сигнализирующего (signaling) и несигнализирующего (nonsignaling) не-числа. Если один из операндов операции является ***сигнализирующим*** не-числом, то необходимо перейти в прерывание. Не-числом, как правило, задаются неинициализированные исходные данные для вычислений. Тогда вычисления, не имеющие смысла, не начнутся.

Если один или оба операнда – ***несигнализирующие*** не-числа, то результатом будет также несигнализирующее не-число, прерывание не происходит и вычисления будут продолжены. В конце таких вычислений можно по признакам не-числа определить, какие из полученных результатов не имеют численного смысла или в каком месте возникают неправильные результаты.

Особая ситуация возникает, когда результат операции по абсолютной величине меньше наименьшего числа, которое может быть представлено с плавающей запятой. Эта ситуация называется "***недополнением***" (underflow) или потеря точности.

Например, наименьший нормализованный код – это 0 00000001 0000 ... 0, равный 2^{-126}. Если в результате вычислений возникло число 2^{-128}, то оно будет представлено кодом 0 00000000 010000 ... 0. При этом код порядка E = 00000000 означает, что это число с недополнением (разрядов мантиссы). Таким образом, вычисления могут вестись более точно.

Например, когда вычисляется число $x - y + y$, то если $x \approx y$, при выполнении действия $x - y$ в формате обычной плавающей запятой будет нулевой результат и конечный результат будет y. При получении $x - y$ как недополнения, результат будет правильный и равен x.

1.6. Способы адресации

Адресное пространство процессора

Архитектура каждого процессора характеризуется его адресным пространством. Срок адресное пространство означает числовой диапазон адресов памяти, к которым может обращаться ядро процессора. Отсчет адресов памяти, как правило, выполняют в байтах, начиная с нулевого байта. Итак, процессор, который имеет k-разрядную шину адресов имеет предельное пространство адресов от 0 до $2^{k}-1$.

Как правило, адресное пространство разделено на определенные области, которые используются для хранения данных, программ, операционной системы, а также ввода-вывода и другие. Конкретное разделение адресного пространства называют ***картой памяти*** (***memory map***).

Каждый процессор обеспечивает ряд режимов адресации данных. При выборе набора режимов адресации решается проблема соотношения сложности системы команд и их эффективности. Часто существует непосредственная связь между тем, какая адресация памяти и тем, на каком языке выполняется программирование. С одной стороны, существующие архитектуры влияют на создание языка и соответствующего компилятора. С другой стороны, разработка новых архитектур ведется с учетом особенностей алгоритмических языков.

Прямая (абсолютная) адресация

Большинство языков, таких как Паскаль, Си, Фортран, дают возможность программисту декларировать статическое данное, то есть такое данное, которому назначается единая ячейка в памяти в течение выполнения программы. В этом случае размер и адрес данного можно определить до запуска программы на выполнение, то есть во время ее компиляции. Это означает, что действительный (логический) адрес известен до запуска программы. Поэтому многие процессоры имеют прямую адресацию, что позволяет ставить действительный адрес непосредственно в команде.

Во многих процессорах прямая адресация напрямую не реализована. Но тогда адрес статического данного – это адрес в пределах некоторого определенного блока или сегмента памяти, начало которого задано базовым адресом, то есть прямой адрес – это на самом деле смещение. Поэтому, например, в архитектуре I80x86 прямая адресация – это когда 16- или 32-разрядное смещение d установлено непосредственно в команде, а исполнительный адрес (физический) равен сумме d и кода адреса, хранящегося в сегментном регистре.

Например, ассемблерная команда этой архитектуры mov AX, data выполняет загрузку данного по символьному адресу data в регистр AX. При этом чтение данного выполняется с прямой адресацией, а запись – с ***прямой регистровой адресацией*** (регистр AX задается в команде адресом 000). То есть, регистровая адресация – это прямая адресация регистровой памяти. Следует отметить, что идентификатор data, который является символьным адресом, получает конкретное числовое значение при компиляции.

Индексная адресация

Массив данных, как особая структура, используется почти во всех языках программирования. Поэтому в современных компьютерах аппаратно поддерживается индексация массивов. Индексная адресация реализуется в аппаратуре как масштабируемое смещение (хранится в индексной регистре), которое прибавляется к заданному начальному адресу массива, который известен во время компиляции. В массивах могут храниться байты, полуслова, слова и т.д. Это означает, что адрес является кратным 1, 2, 4, ...

Если в индексном регистре хранится индекс массива, то его нужно умножить на 1, 2 или 4, то есть смасштабировать. Масштабирование выполняется сдвигом адреса влево, что обеспечивается компилятором или аппаратной реализацией команд с индексной адресацией.

В архитектуре I80x86 индексная адресация выполняется при добавлении смещение установленного в команде, к содержимому индексного регистра SI, DI, например на рис. 1.9.

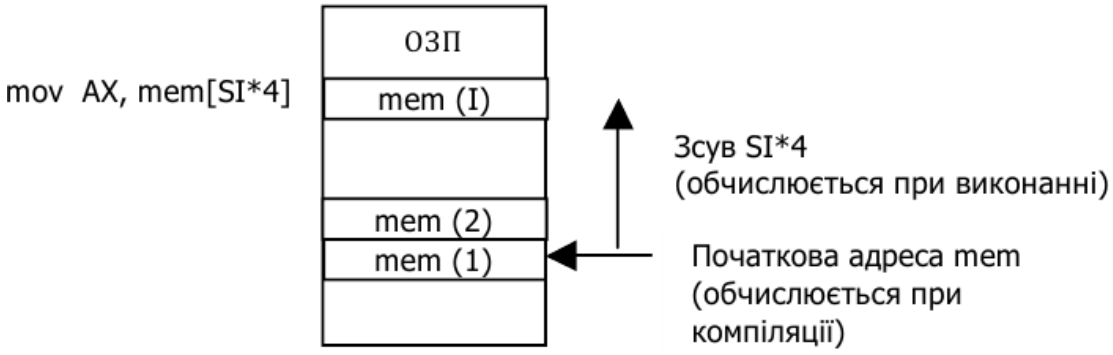


Рис. 1.9. Индексная адресация

Здесь ассемблер обеспечивает в объектном коде смещение массива mem, который состоит из четырехбайтовых элементов. Так как используется масштабный коэффициенты 4, то регистр SI содержит индекс элемента

массива, который загружается в регистр. Например, если читается 10-й элемент массива, то в SI находится число 9, так как индексация начинается с нуля.

Базовая адресация

Эта адресация похожа на индексную адресацию. Но вместо индексного регистра используется регистр базы, содержащий адрес блока памяти. Это абсолютный адрес и его не нужно надо масштабировать. В отличие от статического данного, адрес динамического данного неизвестен на период компиляции и определяется в процессе выполнения программы. Базовая адресация используется, когда нужно найти адрес динамического данного. Тогда в регистр базы загружается адрес данного, который вычисляется во время выполнения программы.

Во многих компьютерах не используется прямая адресация из-за необходимости иметь перемещаемые программы. В таком случае базовая адресация реализуется на аппаратном уровне. При этом в регистр базы помещается начальный адрес программного модуля, а абсолютный адрес вычисляется как базовый адрес плюс прямой (или другой) адрес из команды.

Также базовый адрес используется при вызовах процедур. При таком вызове параметры в процедуру можно передать как базовый адрес памяти, где они хранятся. Чаще всего – это адрес вершины стека с этими параметрами.

В архитектуре I80x86 базовая адресация выполняется при прибавлении смещения X установленного в команде, к содержимому базовых регистров BX, BP, как например:

```
mov AX, [BP + X]
```

Базовая плюс индексная адресация

Если массив – динамический, или если данные представлены в стеке, то адресация требует как базовой, так и индексной адресации. Тогда абсолютный адрес равен $A = B + I*C$, где B – базовый адрес, I – индекс, C – масштабный коэффициент.

Многие процессоры обеспечивают аппаратную реализацию такой адресации, которая часто называется двойной индексной адресацией. Тогда выборка, например, элемента массива $X(I)$ выполняется одной командой загрузки-чтенияния. В других процессорах такая операция требует выполнения нескольких команд.

Как правило, параметры процедуры находятся в области стека. Поэтому при вызове процедуры будут задействованы указатель области стека, базовый адрес вершины стека с параметрами и масштабируемый индекс операнда.

Такая адресация используется в архитектуре I80x86 путем добавления базового (AX, BX, CX, DX, BP) и масштабируемого индексного (SI, DI) регистров, а также возможного числового смещения X, например, в команде

```
mov AX, [BX + SI * 4 + X]
```

Косвенная адресация

Косвенная адресация происходит, когда команда обращается к данному не напрямую, а сначала читает ячейку, где записан адрес этого данного. Например, когда параметры передаются в процедуру, то чаще всего значение параметра является адресом фактического параметра, а не копия самого параметра.

Рассмотрим процедуру Q (рис. 1.10). При ее вызове значение параметра Dat, передаваемый процедуре – это не значение, а адрес, где находится Dat. Такая косвенная адресация выполняется за 2 шага.

Рис. 1.10. Пример косвенной адресации

Сначала адрес параметра Dat загружается в базовый регистр. Затем выполняется базовая адресация, например:

```
mov BX, OFFSET Dat
mov AX, [BX]
```

где по первой команде в базовый регистр BX загружается адрес операнда Dat в памяти, а по второй команде по этому адресу этот операнд записывается в регистр AX.

При косвенном переходе адрес перехода задается косвенно, как значение, хранящееся или в определенной ячейке памяти, или в регистре. Поэтому можно написать команду перехода

```
jmp CX
```

если в регистре CX хранится абсолютный адрес перехода.

В некоторых компьютерах косвенная адресация выполняется аппаратно. Хотя тогда надо только одна команда для доступа, но необходимость двух последовательных доступов к памяти остается.

Косвенная адресация с индексной адресацией

Если параметр, передается при вызове процедуры, это - массив, то косвенная адресация должна комбинироваться с индексной адресацией, как в примере на рис. 1.11.

Рис. 1.11. Пример косвенной адресации с индексной

С помощью косвенной индексной адресации организуется условный переход по нескольким адресам, как в операторе

```
Case: mov AX, Njump; загрузка номера перехода - индекса таблицы переходов
      mov BX, AX;
      shl BX, 3 ;          индекс, смасштабированный умножением на 8
      jmp Jump_table [BX]; переход в таблицу Jump_table с номером Njump
```

Здесь оператор jmp выполняет косвенный переход по адресу, который является базовым адресом Jump_table плюс индексный адрес, хранящийся в регистре BX. Только некоторые процессоры имеют встроенной такую адресацию. Остальные процессоров выполняет последовательность двух или более команд для выполнения такой адресации.

Косвенная адресация с базовой

В предыдущих примерах считалось, что указатель косвенной адресации - статический адрес. Но в языках, ориентированных на операции со стеком (Си, Паскаль), указатель может сам быть размещенным в стеке. Поэтому для доступа к нему необходимо добавить базовую адресацию, как на рис. 1.12.

Рис. 1.12. Косвенная адресация с базовой

Теперь адресация Dat включает в себя добавление смещения указателя на область стека, чтобы получить указатель на Dat, а затем с этим указателем добраться до собственно Dat. Такая адресация может быть выполнена как последовательность команд, хотя некоторые процессоры имеют встроенную такую адресацию.

Косвенная адресация с базовой и индексной

Наконец, рассмотрим пример, когда массив, размещенный в динамическом стеке, передается в процедуру в качестве параметра. Тогда доступ к элементу массива *D* (*I*) выполняется за три шага (рис. 1.13):

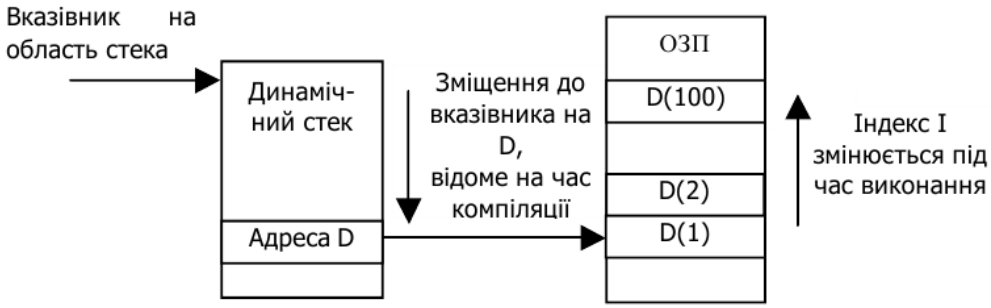


Рис.1.13. Косвенная адресация с базовой и индексной

1.7. RISC- и CISC - процессоры

История появления RISC-процессоров

В 60-80-е годы архитекторы новых процессоров пытались добавлять в систему команд такие команды, которые должны были способствовать ускорению выполнения некоторых вычислений, обработки текста, компиляции. Эти команды были сложными в исполнении и выполнялись по многотактовым микропрограммам. Поэтому системы команд компьютеров каждого нового поколения характеризовались все большим количеством сложных команд.

С другой стороны, интересы программистов компьютеров по многим вопросам расходились с идеями разработчиков, заложенными в систему команд. Так, многие команды (например, команда проверки строки TRT, команды десятичной арифметики в IBM-360) вообще не принимали участие в компиляции, потому что они могли использоваться только в отдельных случаях или для их выполнения были необходимы дополнительные преобразования форматов

данных. Также оказалось, что для процессоров с такой системой сложных команд трудно выполнять автоматическую оптимизацию программ.

С целью поддержки переносимости матобеспечения система команд усложнялась с каждым новым поколением процессоров. Для минимизации оборудования редко используемые команды реализуются с помощью подпрограмм с помощью команды-ловушки TRAP.

Таким образом, процессор с системой сложных команд (CISC-процессор) всегда имеет избыточную сложность управляющих блоков и, как следствие, сравнительно невысокие быстродействие и отношение производительность – аппаратные расходы. Через неоптимальность транслированных программ, ему также присущи большие простои оборудования.

Идея RISC-процессора основывается на том, чтобы заменить сложные многотактовые команды на цепочки простых однотактныхо команд. Поэтому их название – ***Reduced Instruction Set Computer*** – переводится как ***компьютер с системой команд сокращенной сложности***.

Впервые принципы RISC-процессоров были внедрены в компьютере CDC 6600 в начале 60-х годов. При создании этого компьютера С. Крей обнаружил, что в научных расчетах можно обойтись упрощенной системой команд, которые несложно декодировать. При этом можно создать эффективный компилятор с Фортрана. Новинкой было то, что:

– к ОЗУ обращаются только команды записи-чтения данных, вместо непосредственного адреса указывается номер регистра с адресом;

– арифметические команды оперируют с тремя регистрами;

– формат команд упрощенный и однородный;

– в блоке обработки данных много функциональных модулей, которые обрабатывают данные параллельно, а выполнение команд – конвееризованное, для организации параллельного выполнения команд регистрируется занятость модулей.

Проект IBM 801 был первой попыткой разработать RISC-архитектуру специально. Целью проекта было создание компьютера, который может быстро переключать контекст при прерываниях. Разработка началась в 1974 и в 1980 г. был готов прототип. Эта научная разработка имела большое влияние на становление RISC-архитектур в мире.

В 1980-1983 гг. в Беркли под руководством Д.Патерсона были разработаны первые RISC-микропроцессоры RISC I и RISC II. В то же время в Стэнфорде был разработан другой RISC-микропроцессор – MIPS (Multiprocessor without Interlocked Pipeline Stages). Идеи, которые были заложены в эти микропроцессоры, были затем внедрены в сериях процессоров SPARC і MIPS, соответственно.

Принципы построения архитектуры RISC-процессоров

Принципы построения RISC - архитектуры следующие.

Одна команда за один такт. Простая команда должна выполняться за один такт. Этот принцип расходится с порядком функционирования процессоров последовательного действия неймановской архитектуры. Согласно ему, начало выполнения следующей команды обусловлено результатом выполнения предыдущей команды. Команда не может начинаться ранее окончания предыдущей, должна пройти последовательные четыре этапа своего выполнения (см. архитектуру фон Неймана) и поэтому она не может быть выполнена за один такт, от начала до конца. Но следующие архитектурные особенности таких процессоров призваны поддерживать выполнение каждой команды в RISC-процессорах за один такт.

Конвейеризация. Выполнение каждой команды проходит несколько стадий: выборка команды (ВК), дешифрация команды (ДК), выборка операндов (ВО), операция с операндами (ОП), запись результатов (ЗР). Если соседние команды не зависящих друг от друга, то эти стадии можно выполнять в конвейере команд - одну стадию за такт, как на временной диаграмме на рис 1.14.

Рис. 1.14. Выполнение команд в 4-ступенчатом конвейере команд

Регистровая память. Если операнды соседних команд находятся в регистровой памяти, то такие команды могут выполняться за один такт, не приемля во внимание выборку и дешифрацию команд. При этом ре- пап, полученный в данном такте, может принимать участие как операнд уже в следующем такте. Такое явление называют ***прямая передача операнда*** (***forwarding***) (рис. 1.15).

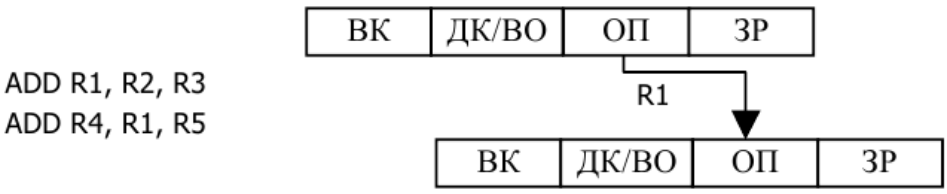


Рис. 1.15. Прямая передача операнда между командами через регистровую память

Поэтому множество команд RISC-процессора, как правило, делится на подмножество команд обработки данных в регистрах, подмножество команд пересылки данных между регистровой памятью и ОЗУ и команды управления.

Пусть имеем программу:

```
L R2, A
L R3, B
NOP
ADD R4, R2, R3
L R5, C
L R6, D
NOP
ADD R7, R5, R6
```

Здесь пустые команды NOP (No OPeration) вставлены для выполнения задержки, в течение которой данные A, B, C, D успевают считаться с внешней памяти по командам загрузки L. Вместо команд NOP, можно переставить команды так, чтобы задержки исполнились естественным образом:

```
L R2, A
L R3, B
L R5, C
L R6, D
ADD R4, R2, R3
ADD R7, R5, R6
```

Здесь выполнения действий проходит ***с перекрытием*** (***interlocking***). Компилятор так оптимизирует порядок команд в цепочках команд, чтобы выполнялась ***программная конвейеризация*** – данные заранее загружаются в регистры, а результаты выгружаются из них на фоне вычислений. Также расписание операций составляется таким образом, чтобы данные, которые обрабатываются и промежуточные результаты дольше задерживались в регистрах и не переписывались назад в ОЗУ, чтобы минимизировались обмены между регистрами и ОЗУ.

Выполнение команд во время обработки команды перехода. При обработке команды перехода после ее дешифрации конвейер должен быть очищен, то есть команды, которые следуют за ней и были загружены в конвейер, должны быть отменены, так как это не те команды, которые выполняются в соответствии с программой. При этом возникает задержка перехода и приостановка конвейера, пока в него не загрузится и не будет выполняться команда, на которую происходит переход. Но в некоторых RISC-процессорах конвейер при переходах не очищается, а начинают выполняться до *N* команд, стоящих после команды перехода (рис.1.16).

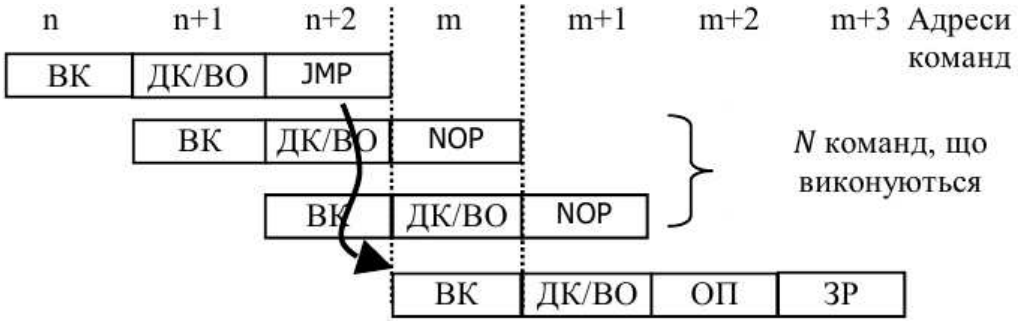


Рис.1.16. Выполнение команды перехода в конвейере команд RISC-процессора

Для сохранения семантики программ по этим *N* командами ничего не происходит, то есть это команды NOP. Но во многих случаях компилятор подставляет вместо них цепочку команд, выполняющих полезные действия. Например, после команды вызова подпрограммы можно вставить *N* первых команд из подпрограммы, вызываемой, если это не команды перехода и выполнять переход через эти команды в подпрограмме.

Возврат из подпрограммы выполняется на команду, стоит после цепочки N команд. Или по команде вызова вставить команды, которые передают параметры в подпрограмму:

```
L R1, par1                CALL proc
L R2, par2        ⇒       L R1, par1
CALL proc                 L R2, par2
NOP
NOP
```

Следовательно, за счет выполнения команд во время задержки перехода команды перехода также могут выполняться, в среднем, за один такт.

Упрощенная адресация. У упрощенных команд – упрощенная адресация. Прежде всего, в большинстве RISC-процессоров – обязательное выравнивание данных на границу адресов. Так, адреса 4-байтных слов всегда имеют два нулевых младших бита. Тогда чтение такого слова можно выполнить по одной командой, выполняя только одну пересылку по шине данных.

Кроме того, упрощаются механизмы обслуживания виртуальной адресации и кэш-ОЗУ. При этом не происходит случай, когда половина слова находится в кэш-ОЗУ, а вторая половина – во внешнем ОЗУ. Косвенная адресация, когда результирующий адрес хранится в ячейке ОЗУ, выполняется парой простых команд. Эти команды выполняются даже быстрее, чем единственная команда в CISC-процессорах, которая делает то же действие.

Так как прямой адрес не может поместиться в команде целиком (32-разрядный адрес в 32-разрядной команде), то чаще всего используется косвенная адресация, когда адрес хранится в регистре.

Простой формат команд. Простой, пусть даже логически избыточный, формат команд, включая код операции, поля адресов, непосредственных операндов, обеспечивает несложную и следовательно, быструю дешифрацию команд и адресов. За счет этого минимизируется период тактового интервала.

Все команды в RISC-процессорах, как правило, имеют одинаковую длину и минимум форматов команд. В этом случае положение полей кода операции и адресов в соседних командах известно заранее, что дает возможность декодировать несколько команд одновременно. Это упрощает прогнозирование доступа к ОЗУ, благодаря чему можно заранее выполнять чтение операндов и команд из него.

У RISC-процессоров прогрессивная архитектура, которая обеспечивает высокие отношение производительность – энергопотребление, производительность – аппаратные расходы и поэтому она идеально подходит для своей реализации в современных микросхемах.

Сравнение RISC- и CISC-процессоров

Так как декодирование сложных команд – затрудненное, такие команды нельзя быстро выбирать из ОЗУ и распараллелить их выполнения. Поэтому CISC-процессоры, как правило, имеют низкое быстродействие в сравнении с RISC-процессорами. В результате, наиболее совершенная CISC-архитектура Intel 432, как и архитектура DEC VAX, не выдержали конкуренции с RISC-архитектурами в середине 80-х годов.

CISC-архитектура Intel 80x86 – исключение, подтверждающее правило. Эта архитектура проигрывает RISC-архитектурам, таким как PowerPC, PARISC, MIPS, SPARC по быстродействию выполняемых программ. Но, во-первых, во время, когда RISC-процессоры появились на рынке, персональные компьютеры с процессорами I80x86 уже успели завоевать лидирующую роль на рынке персональных компьютеров и было наработано десятки тысяч приложений для них. Поэтому подавляющее большинство пользователей компьютеров выбрали архитектуру I80x86, так как было неудобно и невыгодно переходить на RISC-архитектуры. Исключением были пользователи рабочих станций, которых было немного. Во-вторых, архитектура I80x86, начиная с I486, развивалась с приобретением признаков RISC-процессоров, благодаря увеличению числа регистров, добавлению упрощенных команд, конвейеризации их выполнения.

Важно оценить также влияние компиляторов на выбор архитектуры. Наиболее быстрые программы – это программы, написанные на языке ассемблера. Но сейчас очень редко используется программирование на этом языке. Следовательно, производительность компьютера зависит не столько от его быстродействия, как от эффективности компилятора. Принцип действия

компилятора заключается в распознавании в тексте программы некоторой операции (или идиомы) и подстановке в исходный код библиотечной процедуры или макрооперации выполнения этой операции, описанных машинными кодами. Оптимизация программы компилятором заключается в подстановке таких процедур и макроопераций или их комбинаций, которые должны выполнить минимальное количество команд. То есть, задача оптимизации программы является сложной комбинаторной задачей, сложность которой существенно возрастает с увеличением множества объектов, т.е. вариантов цепочек команд, которые необходимо выбирать.

Такая задача оптимизации упрощается, если используется малое количество команд в системе команд. Так, установлено, что компилятор Си для CISC-процессора 68020 использует лишь 30% всех команд из системы команд. Кроме того, в CISC-процессорах разные команды используют различные типы операндов, и компилятор должен все время следить за согласованностью типов и иметь большое множество библиотечных процедур с различными входными и выходными типами данных.

В RISC-архитектуре простой формат команд, минимизировано количество команд, много регистров данных, однородное их использование и т.д. Все это дает возможность сконструировать компилятор проще, а оптимизацию программ сделать более эффективной и более глубокой.

1.8. Система прерываний

Назначение и функции системы прерываний. Основные понятия

В 60-х годах прошлого века внедрение системы прерываний в архитектуру компьютеров было чрезвычайно важным этапом развития вычислительной техники. Благодаря использованию системы прерываний, стало возможным не только отрабатывать реакции на внешние события, но и внедрить автоматическое управление вычислительным процессом с помощью операционной системы, а также реализовать в компьютерах режим работы со многими вычислительными процессами.

Система прерываний – это аппаратно-программный комплекс, обеспечивает реакцию компьютера на некоторое множество событий (сигналов прерываний) которые являются внутренними или внешними относительно выполняемого вычислительного процесса. Такая реакция сводится к переходу к другой последовательности команд, выполнение ее и возвращение к прерванному процессу.

Операционная система (**ОС**) – это набор программ, который служит для организации эффективного выполнения множества вычислительных задач в компьютере и поддержки вычислительной системы в рабочем состоянии.

Диспетчер – комплекс программ операционной системы, предназначенный для управления обработкой сигналов прерываний, анализа причин прерывания и запоминания информации, необходимой для восстановления работы прерванной программы. Диспетчер, особенно в многопроцессорных компьютерах, также занимается динамическим распределением задач или их программных потоков между вычислительными ресурсами.

Работа процессора в реальном времени – выполнение задачи пользователя таким образом, чтобы его запросы (сигналы прерывания) были отработаны немедленно или с допустимой задержкой. Здесь под пользователем имеется в виду не только оператор компьютера, но и объект управления (если компьютер включен в контур системы управления), периферийное устройство (например, контроллер дисковода), резидентный процесс (например, драйвер обслуживания клавиатуры, WEB-браузер) и тому подобное.

Режим разделения времени – это одновременная обработка процессором нескольких задач, инициированных различными пользователями. При этом пользователи ждут реакции на свои запросы в согласованности с работой в реальном времени. То есть, процессор "разделяет" время своей работы между пользователями.

Итак, работа в реальном времени предусматривает, что обслуживание запроса на прерывание *i*-го типа требует времени обслуживания не менее допустимой задержки $T_{\text{доп}i}$ с учетом того, что несколько запросов могут возникнуть одновременно. Так, например, $T_{\text{доп}i}$ при прерываниях ввода с клавиатуры не может быть меньше 0,1 с, а при чтении с жесткого магнитного диска – 10 мкс. При одновременных равноприоритетных запросах на прерывания эти запросы ставятся в очередь, которая может быть обслужена, в среднем за время $T_{\text{ч}}$. Тогда требованием к быстродействию компьютера, работающему в реальном времени, является $T_{\text{доп}i} \le T_{\text{ч}}$.

Классификация сигналов прерываний

В компьютере могут быть задействованы сотни и тысячи сигналов, вызывающих прерывания. Все сигналы не являются равноправными и требуют различные алгоритмы своей отработки. Ниже дана классификация сигналов прерываний.

1. Сигналы о событиях, возникающих в самом компьютере (как в его аппаратуре, так и программах). Это – сигналы возникновения сбоев, отказов в различных устройствах, фиксируются специальным схемам. Например, это

событие уменьшения напряжения питания, которое вызывает прерывание сохранения состояния компьютера перед его выключением. При сбое в канале передачи данных или при ошибке чтения памяти по прерыванию выполняется программа, которая повторяет передачу данных или чтения памяти.

Прерывания, связанные с некоторыми событиями, причинами которых является выполнения программы, например, ошибками в программах, называются ***программными***. Они вызываются сигналами: неправильного кода команды, неправильной адресации, деления на ноль, переполнения порядка или мантиссы числа, нарушения условий защиты памяти, использования в программе запрещенных команд (например, привилегированных команд) и др.

Программируемые прерывания - это программные прерывания, которые вызваны выполнением специальных команд (***traps***), предназначенных для выявление выполнения некоторых заданных условий. Например, такие прерывания выполняются при отладки программы, выполнении команд, которые не реализованы аппаратно. К программируемым прерываниям относится также обращение пользовательской программы к ресурсам ОС с помощью команды прерывания. В ассемблерных программах для архитектуры I80x86 наиболее распространено прерывание по команде INT 21h, используемое, например, для ввода-вывода.

2. Сигналы с датчиков времени. Они используются для обеспечения работы компьютеров в реальном времени с разделением времени, а также алгоритмов управления объектами. Датчиками служат микросхема часов, генераторы импульсов. Для предсказания зависания программ используют датчик слежения за зависанием (***watchdog***), который при условии зависания (когда основная программа не обращается в него через промежуток времени, меньше заданного) вызывает прерывание, выполняющее перезагрузку компьютера.

Программно-реализованные датчики времени – это ячейки памяти, которые изменяются ОС по прерыванию от аппаратного датчика времени и которые вызывают прерывания при равенстве своего содержания заданной величине. Программа-***планировщик*** планирует выполнение программ в системах с разделением времени, а именно, какая из рабочих программ должна выполняться в данный промежуток времени и сколько времени необходимо ей выделить.

3. Сигналы с устройств ввода-вывода. Эти сигналы генерируются при приеме сигналов синхронизации устройств ввода-вывода, а также сигналов занятости, готовности или сбоя, или вмешательства оператора в работу компьютера. На эти сигналы реагируют прерывания, вызывающие программы обработки, каждая из которых соответствует определенному устройству. Такие программы принято называть драйверами устройств ввода-вывода.

4. Сигналы запроса. Эти сигналы приходят в компьютер с различных внешних источников. Для систем управления – это сигналы о события, происходящие в управляемых объектах, для многопроцессорных систем – синхронизирующие сигналы от других процессоров.

Алгоритм функционирования системы прерываний

Особыми элементам системы прерываний является регистр прерываний, вмещающий флажки прерываний и приоритетный шифратор, которые определяются как следующие.

Индикатор событий (флажок прерывания) – это аппаратный элемент, выполняющего фиксацию отдельного сигнала прерывания, устанавливается в 1 при появлении сигнала прерывания и устанавливается в 0 после отработки прерывания.

Регистр прерываний – регистр, объединяющий в себе выходы всех индикаторов событий. В отличие от обычного регистра, состояние его разрядов меняется в произвольные моменты времени независимо друг от друга.

Приоритетный шифратор – аппаратный или аппаратно-программный блок, который решает, какое из зафиксированных прерываний следует обслуживать первым.

Общий алгоритм функционирования системы прерываний показан на рис.1.17. При реализации этого алгоритма выполняются следующие действия.

1) Сначала один или несколько разрядов регистра прерываний фиксируют наличие сигнала прерывания.

2) Исполняемая программа останавливает свое выполнение. Момент разрешенного прерывания действующей программы может быть задан одним из способов: код команды указывает, можно ли команду прерывать; прерывание в конце выполнения любой команды; прерывание в любом такте выполнения команды.

При возникновении прерываний в RISC-процессорах необходимо останавливать конвейер команд и хранить дополнительно состояния его ступеней или давать исполниться до конца последней команде, которая обрабатывается. То есть при реализации системы прерываний в RISC-процессорах механизм управления конвейером команд существенно усложняется.

3) В момент прерывания программы запоминаются состояния основных регистров, в которых хранятся данные программы, то есть ее ***контекст***, для того, чтобы потом можно было восстановить ее работу.

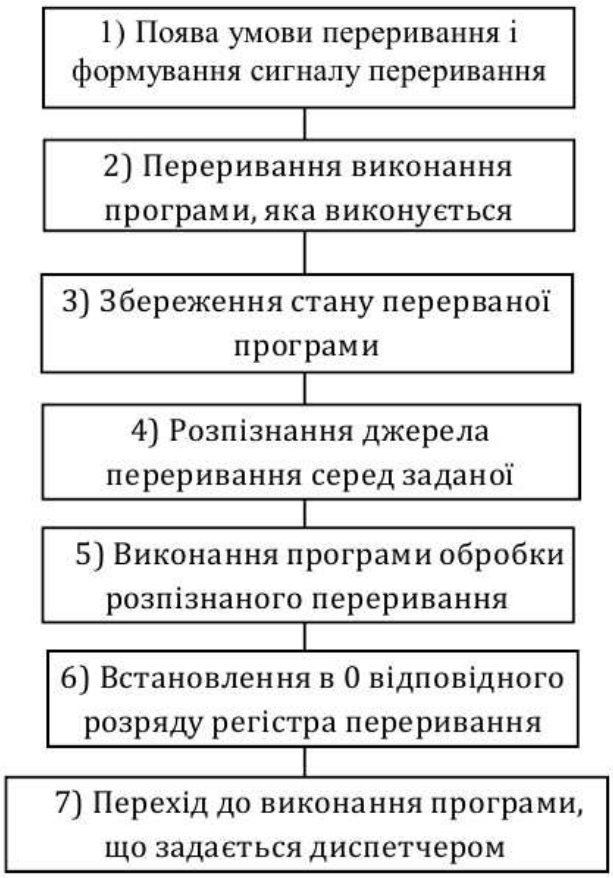


Рис. 1.17. Алгоритм выполнения прерывания

4) Определяется причина прерывания и выбирается программа обслуживания конкретного прерывания. Если источников прерываний много (сотни и тысячи), то аппаратно, с помощью приоритетного шифратора выделяется группа источников, среди которых программным способом определяется конкретный источник прерывания.

Если возникло несколько запросов на прерывание, то они должны обслуживаться в порядке, определенном приоритетами. Решение, которое прерывание обслуживать первым, готовит приоритетны шифратор, управляемый диспетчером. Дисциплина обслуживания прерывания, которая заложена в диспетчер и приоритетный шифратор, устанавливает последовательность обработки нескольких флажков прерывания в регистре прерываний.

5, 6) Выполнение программы обработки распознанного прерывания. Установка в ноль соответствующего разряда регистра прерывания, то есть причина прерывания устраняется.

7) Диспетчер обеспечивает продолжение прерванной программы с помощью восстановления состояния регистров или обработку следующего запроса на прерывание (в зависимости от обстоятельств, дисциплины обслуживания прерывания и т.д.).

Способы сохранения состояния прерванной программы

При использовании программного способа, содержание всех регистров – контекст программы – сохраняется диспетчером в отдельной области оперативной памяти – каждому программе – своя область. Возвращение содержимого регистров выполняется также программно.

Для ускорения этого процесса область сохранения делают в ОЗУ с повышенным быстродействием. Также можно ускорить этот процесс, если сохранять состояние не всех регистров, а только тех, которые могут измениться при отработке прерывания.

Аппаратный способ сохранения заключается в том, что содержание регистров переписывается автоматически в аппаратный стек или в сегмент ОЗУ, предназначенный для сохранения состояния. В архитектуре IA-32 во время прерывания 104 байтов данных из основных регистров автоматически сохраняются в сегменте состояния задачи (TSS) (см. Разд. 4.4). Во многих RISC-системах регистры данных и адресов продублированы и при прерывании одна группа регистров, в которых сохраняется контекст прерванной программы, переключается на другую группу регистров, используемых для обработки прерывания. Преимущества этого способа в высоком быстродействии, а недостатки – в отсутствии гибкости, ограниченном количестве прерываний, увеличении аппаратных затрат.

Аппаратно-программный способ означает аппаратное сохранение основных регистров (аккумулятора, регистра состояния, счетчика команд) и программное сохранения остальных регистров.

Основные характеристики системы прерываний

Система прерываний определяет общее быстродействие копмьютера. Поэтому для оптимизации архитектуры по быстродействию следует определить характеристики и параметры системы прерываний. Временная диаграмма отработки прерывания показана на рис. 1.18.

На ней определены следующие временные характеристики системы прерываний:

T_{pi} – время реакции;

T_{zapi} – время запоминания состояния;

T_{Pni} – время распознавания источника сигнала прерывания;

$T_{Пi}$ – время подготовки к обработке прерывания, характеризующее абсолютную задержку при обработке прерывания, то есть ее быстродействие;

T_{Oi} – время непосредственной обработки прерывания;

$T_{заг}$ – общее время обработки прерывания.

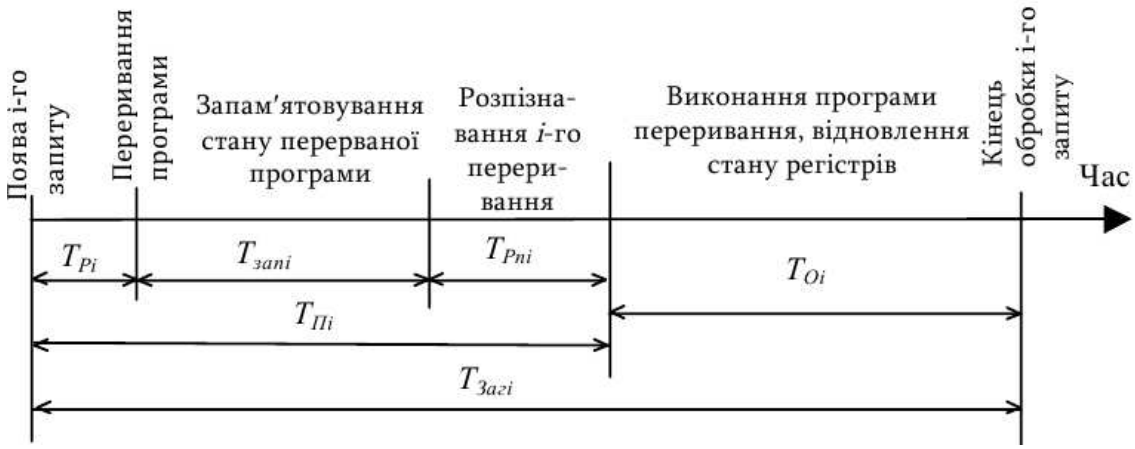


Рис. 1.18. Временная диаграмма отработки прерывания

Отсюда следует такой параметр, как

$K_B = T_{Пі} / T_{заг}$ – коэффициент потерь, характеризующий относительную быстродействие системы прерываний.

Состояние насыщения системы прерываний – это такое состояние, когда поток запросов на прерывание переходит грань, когда запросы начинают обслуживаться несвоевременно, то есть когда $T_{заг} > T_{доп}$.

Дисциплина обслуживания прерывания

Система прерываний должна иметь возможность выделять один флажок прерывания среди множества одновременно установленных флажков согласно установленной дисциплине обслуживания прерывания. Запросы на прерывание могут иметь разную степень важности в зависимости от конкретных обстоятельств, которые могут меняться. В идеальном случае система выделяет тот запрос (флажок), который является наиболее важным. ***Оптимальное обслуживание*** – это такое обслуживание, когда система прерываний правильно и быстро реагирует на степень важности запросов и переключает процессор на обработку наиболее важной ситуации.

В компьютерах наиболее распространены следующие дисциплины.

1. Дисциплина обслуживания с ***циклической очередью*** означает циклический опрос всех разрядов регистра прерываний. Обрабатывается первый попавшийся единичный флажок. После обработки *i*-го флажка обрабатывается *i* + 1-й флажок, если он установлен. Если время опроса достаточно мало, то синоним дисциплины – ***первым пришел – первым обслужен***.

2. Дисциплина обслуживания ***с относительными приоритетами***. При этом из множества запросов выбирается один наиболее важный запрос согласно с присвоенными приоритетами. Если во время его обслуживания приходит еще более важный запрос (более приоритетный), то он ждет, пока

закончится обработка предыдущего запроса. Следующим запросом, который будет обработан, будет тот запрос, который является наиболее приоритетным среди зафиксированных запросов на момент окончания обработки ранее выбранного запроса.

3. Дисциплина обслуживания ***с абсолютными приоритетами*** означает, что в каждый момент времени обрабатывается наиболее важный запрос, независимо от того, сколько менее важных запросов ждут обработки или доработки, то есть здесь возможно прерывание в прерывании.

Приоритеты назначаются или отдельным запросам, или группам однотипных запросов. Наивысший приоритет предоставляется запросам отказов аппаратуры, далее – прерываниям от таймера. Запросам от более быстродействующих объектов предоставляется более высокий приоритет.

Если компьютер работает ***с динамическими приоритетами***, то приоритеты могут переоцениваться и переназначаться в зависимости от ситуации.

Защита от прерываний – это механизм временного запрета перываний от выбранных источников. Он выполняется с помощью ***регистра маски прерываний***, код 0 в *i*-м разряде которого означает нечувствительность *i*-го флажка прерывания. Такая защита используется для запрета прерываний при выполнении ответственных приложений, например, записи в НЖМД, при выполнении привилегированной программы операционной системы.

Как правило, есть отдельный бит регистра маски, который управляет запретом всех прерываний, то есть бит ***глобального запрета прерываний***.

Классификация систем прерываний

Наиболее характерным признаком системы прерываний есть правило организации порядка выбора запроса прерывания на обслуживание. По этому признаку эти системы делятся на системы:

– с опросом (например, циклическим или приоритетным опросом);

– с абсолютным приоритетом;

– с относительными приоритетами;

– со смешанными приоритетами.

По реализации системы распознавания источника прерывания:

– с программной реализацией схемой приоритета;

– с аппаратной реализацией схемой приоритета;

– с аппаратно-программной схемой приоритета;

– многоступенчатая система прерывания – когда источников прерывания очень много.

Аппаратная реализация прерываний

Аппаратная реализация прерываний – это неотъемлемое условие обеспечения высокого быстродействия системы прерываний. Типичная схема аппаратной реализации прерываний показана на рис. 1.19.

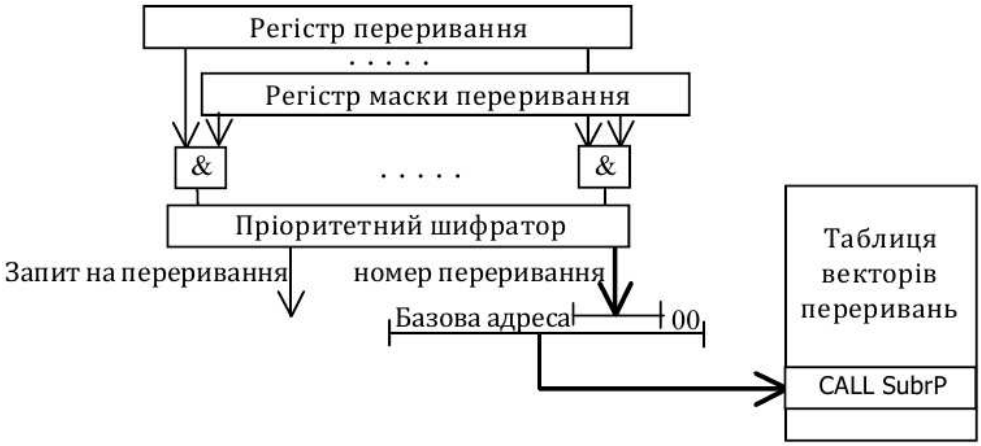


Рис. 1.19. Схема аппаратной реализации прерываний

В регистре прерывания фиксируется событие запроса в флажке *p*. Код в регистре маски прерывания маскирует некоторые или все биты флажков прерываний. Приоритетный шифратор среди нескольких установленных и незамаскированных флажков выделяет наиболее приоритетный номер *p* и выдает сигнал запроса прерывания. Выделенный номер прерывания *p*, который нужно обслужить, подается на формирование адреса входа в ***таблицу векторов прерываний***. Это формирование состоит в добавлении базового адреса таблицы векторов прерываний к масштабированому коду *p*, то есть к коду $2^k p$.

В *p*-й строке таблицы векторов прерываний сохраняется короткая 2^k-байтовая подпрограмма отработки прерывания или команда вызова подпрограммы прерывания, которая находится в резидентной области ОС, или абсолютный адрес этой подпрограммы. В последнем случае вызов подпрограммы прерывания реализуется аппаратно, а адрес подпрограммы имеет код, собственно, вектора прерывания. Во время окончания подпрограммы прерывания по команде iret происходит возврат в программу, которая была прервана. Действие этой команды отличается от действия обычной команды возврата из подпрограммы ret по крайней мере тем, что после нее *p*-й флажок, который вызвал прерывание, автоматически устанавливается в 0.

В архитектуре I80x86 при работе в реальном режиме таблица векторов ров прерываний имеет нулевой базовый адрес. В ней хранятся адреса – векторы прерываний размером 4 байта. Количество векторов равно 256.

Запросы на прерывание от внешних источников поступают в процессор с этой архитектурой через вывод INTR. Для ввода до восьми запросов на прерывание к входу INTR присоединяется аппаратный программируемый контроллер прерываний I8259.

Этот контроллер имеет в своем составе регистр команд прерывания ICR и регистр маски прерывания IMR с адресами 20h и 21h соответственно. Каждый единичный бит в IMR запрещает прерывания. Система ввода-вывода BIOS инициализирует контроллер, задавая фиксированные приоритеты, а также задавая наименьший номер прерываний для входа IRQ0, причем источник на входе контроллера IRQ0 имеет наивысший приоритет, а источник IRQ7 – самый низкий. Если несколько запросов на прерывание приходят на входы контроллера, он посылает эти запросы в процессор последовательно в порядке их приоритетов.

О конце отработки очередного прерывания процессор сообщает контроллеру, записывая код 20h в регистр ICR. Итак, программа отработки прерывания должна заканчиваться командами:

```
mov AL, 20H
out 20H, AL; сообщение контроллера об отработке прерывания
iret
```

В таблице 1 представлены источники стандартных аппаратных прерываний архитектуры I80x86.

Таблица 1. Аппаратные прерывания DOS

№	Источник прерывания
IRQ 0	Системный таймер
IRQ 1	Контроллер клавиатуры
IRQ 2	синхросигнал видео, связан с 9
IRQ 3	COM2 / COM4
IRQ 4	COM1 / COM3
IRQ 5	Свободный
IRQ 6	Контроллер FDD
IRQ 7	LPT1
IRQ 8	Часы реального времени с автономным питанием
IRQ 9	Параллельно с 2
IRQ 10	Свободный
IRQ 11	Свободный
IRQ 12	Контроллер мыши PS / 2
IRQ 13	Математический сопроцессор
IRQ 14	Контроллер IDE HDD (первый канал)
IRQ 15	Контроллер IDE HDD (второй канал)

Выводы

Системному программисту часто необходимо разрабатывать особые программы обслуживания прерываний, специфические для данного компьютера при оптимизации его производительности, надежности, расширении функциональности, если таких программ не окажется в средствах ОС. Поэтому необходимо знать систему прерываний и уметь ее использовать и изменять.

2. Архитектура окружения процессорного ядра

2.1. Основы организации интерфейсов

Обмен данными в компьютерах и интерфейсы

Компьютеры первого поколения были выполнены с централизованной структурой, в которой все блоки, такие как память на магнитных дисках, лентах, барабанах, терминалы ввода-вывода, ОЗУ были присоединены непосредственно к центральному процессору (ЦП) и работали под его управлением. Так как быстродействие как ЦП, так и периферийных устройств были относительно небольшие, то ЦП бо'льшую часть своего времени тратил на обмены с периферийными устройствами. При разработке компьютеров второго и третьего поколений стало понятно, что быстродействующий ЦП большую часть времени не выполняет полезной работы, обслуживая медленные периферийные устройства. Возникла задача разделить между собой ЦП и периферийные устройства и дать им возможность работать более независимо. Эту проблему решило внедрение интерфейса и системы прерываний.

Интерфейс общая шина впервые был внедрен в компьютерах PDP-8 в 1965 году. ***Общая шина*** давала возможность работать периферийным устройствам и ЦП независимо, а также пересылать данные друг другу. В этом интерфейсе все устройства одинаковым образом подключены к одной шине через стандартный аппаратный интерфейс. Эти устройства должны подчиняться одинаковым правилам при подаче сигналов в шину и приеме их с шины. Эти правила называются ***протоколом интерфейса***.

При внедрении архитектуры IBM-360 применение мультиплексного и селекторного интерфейсов (каналов) позволило десятку разных моделей ЦП этой архитектуры присоединять одинаковым образом четырех десятков различных типов периферийных устройств по желанию пользователя. Это было одной из причин высокого коммерческого успеха этой архитектуры.

Итак, история первых трех поколений компьютеров показала необходимость применения интерфейсов. Это позволило:

– эффективно использовать ресурсы ЦП и периферийных устройств;

– изменять структуру компьютера (иногда даже во время его работы) в зависимости от потребностей;

– благодаря стандартизации, разрабатывать и производить различные устройства компьютеров в произвольное время различными производителями, а также комплектовать компьютеры оптимальным образом.

Под ***интерфейсом*** понимают аппаратные средства, служащие для однородного соединения нескольких блоков, которые через него могут надежно обмениваться данными и быть связанными в систему. Описание интерфейса состоит из описания его аппаратной части и протокола.

Аппаратная часть интерфейса описывается как структура, включающая набор аппаратных средств интерфейса и линий связи между этими средствами и блоками, подключаемыми к интерфейсу, а также требований к разъемам и электрическим сигналам, которые распространяются в шинах.

Линия связи получила название ***интерфейсной шины***. Ее описание, как правило, является описанием разъема интерфейса. Также существуют физические ограничения на конструкцию интерфейсной шины – конструкцию ее проводников и их максимальную длину. Другие физические ограничения распространяются на конструкцию генераторов сигналов шины. Это уровни напряжений и токов в проводниках, представляющих биты информации, а также временные показатели их изменения.

Интерфейсная шина состоит из: адресной шины, шины данных, шины управления, шин питания, резервной шины.

Протокол интерфейса представляет собой систему правил поведения с интерфейсом во времени: правила подключения к интерфейсу, арбитража и передачи данных по нему. Он задается алгоритмами подачи (приема) сигналов на адресную шину, шины данных, управления и питания.

Системные и локальные интерфейсы

Общая шина – это один из самых простых интерфейсов. Доступ к ней разделяется между всеми устройствами, подключаемыми к ней. Преимущества этого в ее низкой цене и универсальности. Одновременно к ней может быть подключены единственный задатчик – мастер и один или несколько исполнителей – слейвов. Мастер посылает в интерфейс запрос на один акт использования интерфейса (на ***транзакцию***) и при подтверждении запроса захватывает управление интерфейсом. После выполнения пересылок чтения или записи мастер отключается от шины – заканчивает управление ею, т.е. завершает транзакцию. Если к общей шине подключено несколько мастеров, то в интерфейсе используется блок ***арбитра***, который решает которому мастеру отдать управление при одновременных запросах от нескольких мастеров.

Главным недостатком общей шины является ее ограниченная пропускная способность, так как невозможно, чтобы несколько мастеров выполняли транзакции одновременно.

Традиционно интерфейсные шины делятся на шины, обеспечивающие организацию связи процессора с памятью и шины ввода-вывода. ***Шины ввода-вывода*** могут иметь большую длину, поддерживать подключение многих типов устройств, и как правило, соответствуют одному из шинных стандартов. ***Шины процессор – память*** напротив, сравнительно короткие, часто специализированные и обеспечивают максимальную пропускную способность канала память – процессор. На этапе разработки системы заранее известны все типы и параметры устройств, подключаемых к шине процессор – память. Поэтому необходима максимальная пропускная способность может быть обеспечена. В Вместе с тем, разработчик шины ввода-вывода имеет дело с устройствами, которые имеют различные, часто заранее неизвестные задержки и другие параметры.

С целью уменьшения своей стоимости некоторые компьютеры имеют единую шину для памяти и устройств ввода-вывода. Такая шина часто называется ***системной***. Персональные компьютеры первых поколений строились на основе одной системной шины в стандартах ISA или EISA.

Необходимость сохранения баланса производительности при увеличении быстродействия микропроцессоров привела к двухуровневой организации шин в персональных компьютерах на основе локальной шины. ***Локальной шиной*** называется шина, электрически подключенная к ядру процессора. Она объединяет процессор, память, схемы буферизации для системной шины и ее контроллер, а также некоторые вспомогательные схемы. Типичными примерами локальных шин является VLBus и PCI.

Рассмотрим типичную транзакцию шины. Шинная транзакция имеет две части: посылка мастером адреса в блок-исполнитель и прием (или передача) данных.

Шинные транзакции задаются характером взаимодействия с памятью: транзакция типа "чтение" передает данные из памяти (или в ЦП, или в устройстве ввода-вывода), транзакция типа "запись" записывает данные в память или в регистр периферийного устройства. В транзакции типа "чтение" по шине сначала пересылается в память адрес вместе с сигналами управления, указывающими, что это транзакция чтения. Память отвечает выдачей на шину требуемых данных с соответствующими сигналами управления. Для транзакции типа "запись" необходимо, чтобы ЦП или устройство вода-вывода послал в память адрес и данные и не ожидал возвращение данных. При этом ЦП может не ждать завершения записи.

Разработка интерфейсной шины связана с целевыми параметрами стоимости и производительности компьютера. Так, отдельные линии адреса и данных, многоразрядные шины данных и режим групповых пересылок дают увеличение производительности за счет увеличения стоимости интерфейса.

Используются два типа интерфейсных шин в зависимости от способа их коммутации: шины с коммутацией сетей (circuit-switched bus) и шины с коммутацией пакетов (packet-switched bus).

В шине ***с коммутацией сетей*** мастер запрашивает шину, после арбитража выставляет на ней адрес и блокирует шину до конца обслуживания запроса. Система коммутации такой шины прокладывает маршрут от источника сигнала до приемника на все время транзакции. Большая часть времени при этом может тратиться, например, на задержку выборки из памяти. То есть, в шинах с коммутацией сетей время используется нерационально.

При передаче сообщения с данными через шину ***с коммутацией пакетов*** оно разделяется на два или более пакетов, которые передаются последовательно. Причем маршруты пересылки пакетов могут не совпадать.

Минимальное сообщение состоит из адресной части и данных. Транзакция пересылки данных может разделяться на две части: запрос шины с пересылкой адреса и ответ с пересылкой данных. Такая методика называется ***расщепление транзакции*** (***split transaction***). Так, транзакция чтения делится на подтранзакцию запроса чтения, содержащей адрес, и подтранзакцию ответа памяти, содержащей данные.

Каждая из транзакций должна быть помечена (тэгирована), чтобы ЦП и память могли распознавать, что есть что.

Шина часто выполняется как ***конвейерная шина*** (pipelined bus). При этом вдоль маршрутов пересылки данных и адресов устанавливаются конвейерные регистры. Благодаря этому, тактовую частоту шины можно поднять до экстремальных значений.

Расщепленные транзакции позволяют выполнять доступ к шине от других мастеров между подтранзакциями, пока активный мастер ждет данное от медленного устройства. Если передается массив данных несколькими пакетами, то его передача может быть приостановлена, например, прерыванием для передачи более приоритетного данного, а затем передача массива восстанавливается. В этом случае транзакция расщепляется явным образом.

Итак, шина с расщеплением транзакций имеет более высокую пропускную способность, но она имеет и бо'льшую латентную задержку, чем шина с коммутацией сетей, которая захватывается на все время выполнения транзакции.

Шины с коммутацией пакетов и коммутацией сетей являются частными случаями системы межпроцессорного связи с коммутацией пакетов и коммутацией сетей, соответственно, которые будут рассмотрены в следующем разделе.

Если ***шина синхронная***, то она включает в себя сигналы синхронизации, передаваемые по линиям управления шины и фиксированный протокол, задает положение сигналов адреса и данных относительно сигналов синхронизации. Такие шины могут быть быстрыми и дешевыми. Но они имеют два недостатка. Во-первых, все действия в шине должны выполняться с одной частотой синхронизации, то есть с максимальным быстродействием, на которую рассчитана шина. Во-вторых, по причине возникающего перекоса фазы сигналов в соседних разрядах многоразрядных шин, синхронные шины не могут быть длинными и к ним не может быть подключено много устройств. Как правило, шины процессор-память – синхронные.

В современных скоростных последовательных шинах, таких как PCI-E, SATA, USB, Ethernet, сигнал синхронизации задан неявно и воспроизводится специальной схемой в приемнике шины.

Асинхронная шина не тактируется – в ней используется старт-стопный режим передачи и протокол ***квитирования*** (***handshaking***). Квитирование означает, что приемник данного в момент приема посылает передатчику обратный сигнал ("квитанцию"), который сообщает, что приемник принял данное и готов для приема следующего данного. Эта схема дает возможность проще приспособить устройства с различным быстродействием и увеличить длину шины без проблем системы синхронизации.

Асинхронная последовательная шина COM, а также шина PS2 персонального компьютера работает в старт-стопном режиме. При этом источник сигнала передает один байт в виде десятибитного символа, первый и последний бит которого – это символы старта (нулевой уровень) и стопа (единичный уровень), а остальные – биты байта, начиная с младшего. Продолжительность каждого импульса символа определяется заданной частотой передачи. Эта частота установлена одинаковой во всех интерфейсах COM с погрешностью не более 3%. Так, при скорости 9600 бод (т.е. 9600 битовых символов в секунду) продолжительность одного импульса составляет $1/9600 \approx 104$ мкс. То есть, максимальная скорость передачи составляет 960 байт в секунду.

Синхронная шина, как правило, имеет бо’льшую пропускную способность чем асинхронная из-за отсутствия накладных расходов на синхронизацию шины (квитирование), а также из-за возможности конвейеризации. Выбор типа шины (синхронной или асинхронной) влияет не только на пропускную способность, но также на емкость системы ввода-вывода (длина и количество

устройств, которые могут быть подключены к шине). Рассмотрим далее распространенные системные шины, которые имели усовершенствования с развитием персональных компьютеров.

Системные шины

Одной из популярных шин персональных компьютеров была системная шина **XT-Bus** – шина архитектуры XT, которая была в первых персональных компьютерах IBM PC. Она поддерживает обмен 8-разрядными данными в 20 разрядном адресном пространстве, работает на частоте 4,77 МГц.

ISA (Industry Standard Architecture – архитектура промышленного стандарта) – основная шина на компьютерах типа PC AT. Это расширение шины XT-Bus, ее разрядность – 16/24 (адресное пространство 16 МБ), тактовая частота – 8 МГц, максимальная пропускная способность –5,55 МБ/с.

EISA (Enhanced ISA – расширенная ISA) – это функциональное и конструктивное расширение ISA. Разрядность – 32/32 (адресное пространство – 4 ГБ), работает на частоте 8 МГц. Максимальная пропускная способность – 32 МБ/с. В отличие от предыдущих шин, которые управлялись только процессором, эта шина поддерживает режим ***Bus Mastering***.

Этот режим обеспечивает управление шиной со стороны произвольного устройства на шине, благодаря системе арбитража и возможности разделения каналов прерывания IRQ и ***прямого доступа памяти*** (**DMA**). Такой режим используется для высвобождения процессора от операций пересылки команд и/или данных между двумя устройствами на одной шине. Например, режим DMA в архитектуре PC AT выполняет контроллер DMA процессора, который является общим для всех устройств. При Bus Mastering каждое устройство – мастер имеет собственный такой контроллер.

VLB (VESA Local Bus) – 32-разрядное дополнение к шине ISA. Разрядность - 32/32, тактовая частота – 25-50 МГц, максимальная скорость обмена – 130 МБ/с.

PCI (Peripheral Component Interconnect - соединение внешних компонент) – развитие VLB. Разрядность – 32/32 (расширенный вариант – 64/64), тактовая частота –до 33 МГц (PCI 2.1 - до 66 МГц), пропускная способность – до 132 МБ/с (264 МБ/с для 32/32 на 66 МГц и 528 МБ/с для 64/64 на 66 МГц), поддержка Bus Mastering и автоматической настройки. Количество соединений шины на одном сегменте ограничено четырьмя. Сегментов может быть несколько, они соединяются через ***мосты*** (***bridges***). Сегменты могут соединяться в структуры с разной топологией (дерево, звезда и т.д.). Это на сегодня самая популярная системная шина. 64-разрядное соединение имеет дополнительную 64-

контактную секцию. Все соединения и устройства к ним делятся на те, которые поддерживают уровни сигналов 5В, 3.3 В и универсальные.

PCI-X – PCI eXpanded – быстродействующая параллельная шина с коммутацией пакетов, которая является модернизацией шины PCI. Устройства PCI-X можно вставлять в соединение PCI и наоборот, так как эти интерфейсы взаимозаменяемы аппаратно и программно. Сигналы в PCI-X проходят через конвейерные регистры, а фронті сигналов воспроизводятся с помощью схем фазового слежения (PLL), благодаря чему получается высокая скорость передачи данных.

Предельная пропускная способность интерфейса PCI-X с 64-разрядной шиной равна 1064 МБ/с. Она достигается благодаря высокой загрузке шины: как только транзакция началась, то вставка тактов ожидания не допускается. Большинство транзакций – это пересылки блоками длиной не менее 128 байтов.

Система, которая требует скорости более 1 ГБ/с, реализуется в новой версии шины - PCI-X 2.0 с протоколами Dual Data Rate (DDR) или Quad Data Rate (QDR) по каналу связи PCI-X на частоте 133 МГц. Благодаря применению дополнительных стробов импульсов за один такт передается два или четыре данных и достигается скорость до 4256 МБ/с. Но при высокой скорости количество потребителей шины уменьшается до двух, что превращает эту шину в соединение точка-точка. Дополнительная генерация и проверка контрольных разрядов позволяет исправлять единичные ошибки в передаче, что позволяет поддерживать высокие скорость и надежность передачи.

PCMCIA (Personal Computer Memory Card International Association – ассоциация производителей плат памяти для персональных компьютеров) – внешняя шина компьютеров класса NoteBook. Другое название модуля PCMCIA - PC Card. Максимально простая шина, разрядность – 16/26 (адресное пространство – 64 МБ), поддерживает автоматическую настройку, возможно подключение и отключение устройств в процессе работы компьютера.

Интерфейсы периферийных устройств (шины ввода-вывода)

Шина AGP. Шины устройств вывода графических данных персональных компьютеров (ПК) постоянно совершенствуется. Так, ускоренный графический порт (AGP) - это расширение шины PCI с целью обработки больших массивов данных 3D-графики. Intel разрабатывала AGP для решения двух проблем. Во первых, для воспроизведения 3D-графики необходимо как можно больше памяти информации текстурных карт (texture maps) и z-буфера (z-buffer). При нормальных обстоятельствах z-буфер, содержит информацию о трехмерном

изображении и использует ту же память, что и текстуры. Это создает конфликты с ЦП. Также производительность графической подсистемы и системной памяти ограничиваются физических характеристиками шины PCI.

Чтобы решить эти проблемы, Intel разработала шину AGP. Получилось прямое соединение или соединение точка-точка (point-to-point) между графической подсистемой и системной памятью. При этом разделяется доступ к памяти с центральным процессором компьютера.

Шина USB. Шина USB (Universal Serial Bus) предназначена для обеспечения обмена данными между компьютером и подключенным к нему периферийным устройством (ПУ) в условиях динамического (горячего) изменения конфигурации системы. При проектировании новой шины принимались во внимание:

• простота изменения конфигурации системы;

• низкая стоимость решения;

• адаптация к существующей инфраструктуре ПК и матобеспечения;

• стимулирование разработки новых классов устройств для ПК.

Возможности USB происходят из ее технических характеристик:

• высокая скорость обмена –12 Мбит/с;

• максимальная длина кабеля для высокой скорости обмена – 5 м;

• максимальное количество подключенных устройств (включая размножители) – 127;

• возможно подключение устройств с различными скоростями обмена;

• напряжение питания для периферийных устройств – 5 В;

• максимальный ток питания на устройство – 500 мА.

На физическом уровне топология шины представляет собой древовидную структуру – многоуровневую звезду. В верхушке дерева – "корне" шины USB – находится корневой концентратор, который обеспечивает связь периферии с компьютером (хостом). Он имеет набор соединений точка-точка с другими узлами и/или функциями, находящимися внизу по потоку. Узел состоит из двух функциональных элементов – повторителя для управления коммутацией потоков информации между входным и выходным портами узла, – контроллера для управления состоянием узла и его портов.

Хост-контроллер отвечает за выполнение операций:

• нахождение подключение/отключение устройств;

• направление потоков управляющей информации между хостом и ПУ;

• управление потоками данных по шине, выполнение протокола шины;

• сбор информации о статусе и активности ПУ системы, а также формирование отчетов о состоянии системы USB;

• выделение ПУ энергоресурсов системы.

Разработчики нестандартных ПУ должны включать свои драйверы в уровень программного обеспечения шины USB. Система конфигурирования шины и идентификации ПУ поставляется разработчиком ОС или другими производителями программного обеспечения. Эта система управляет всеми узлами сети. Драйвер USB от разработчика ОС выполняет диспетчеризацию активности шины. Драйвер хост-контроллера следит за выполнением запросов доступа к шине и обеспечивает выделение ресурсов шины.

В шине USB используется ***мультиплексирование данных с временным уплотнением*** (time division multiplexing - TDM). При этом данные передаются пакетами. Размер пакета переменный, зависит от многих факторов. Хост-контроллер объединяет пакеты, передаваемые одновременно нескольким ПУ, в кадры длительностью по 0,001 с. Порядок пакетов в кадре задается драйвером хост-контроллера.

Шина **USB2** – модернизированная шина USB с возможностью пересылки данных со скоростью до 480 Мбит/с.

Сейчас постепенно внедряется третье поколение шины USB – USB3, которая увеличит пропускную способность еще в десять раз.

Шины IDE и SCSI. Под термином **IDE** (Integrated Drive Electronics – электроника, которая вмонтирована в привод), или ATA (AT Attachment – то, что подключается к PC AT) понимают простой и недорогий интерфейс для PC AT. Все функции по управлению накопителем обеспечивает встроенный контроллер, а 40-проводный кабель является фактически упрощенным сегментом 16-разрядной шины AT-Bus (ISA). Наипростейший адаптер IDE имеет только адресний дешифратор – остальные сигналы заводятся прямо на разъем ISA. Адаптер IDE не имеет системы BIOS – все функции поддержки IDE встроены в системный BIOS PC AT. Но контроллер IDE может иметь собственную BIOS, которая загружается в ЦП и подменяет часть функций системной BIOS.

Основной режим работы устройств IDE – програмний обмен (PIO) под управлением центрального процесора, но все винчестеры EIDE поддерживают обмен в режиме DMA, а большинство контроллеров – в режиме Bus Mastering.

Под термином **SCSI** – Small Computer System Interface (Интерфейс малых вычислительных систем) понимают стандарты, разработанные Национальным институтом стандартов США (ANSI) и которые задают механизм реализации магистрали передачи данных между системной шиной компьютера и периферийными устройствами. Главной особенностью подсистемы SCSI является размещение в периферийном устройстве интеллектуального контроллера. Стандарт SCSI-1 – периферийная шина на основе 50-жильного экранированного кабеля. Шина данных SCSI-1 имеет разрядность 8 бит, а максимальная скорость передачи – 5 Мбайт/ с.

Fast SCSI отличается только тем, что допускает передачу 10 Мбайт/сек в синхронном режиме.

Wide SCSI увеличивает разрядность шины данных (16 или 32 бит) и дает скорости 10 или 20 Мбайт/с. В комбинации Fast-and-Wide SCSI возможны скорости 20 и 40 Мбайт/с.

PCI-E – PCI Express – представляет третье поколение шин ввода-вывода для подключения периферийных устройств. К первому поколению принадлежат ISA, EISA, VESA и Micro Channel, ко второму – PCI, AGP, и PCI-X. Но в отличие от многоразрядных шин предыдущих поколений, в которых несколько потребителей подключаются параллельно, PCI-E организована как соединение точка-точка с последовательной передачей битов данных. Благодаря высокой тактовой частоте и специальному кодированию, типичная скорость передачи достигает 2,5 Гбит/с.

Несколько устройств подключаются к шине помощью встроенных мультиплексоров. Последовательная передача данных дает возможность существенно уменьшить количество контактов в разъеме и проводов в кабеле до четырех, а также выполнять СБИС для подключения периферийных устройств, которая имеет много портов. Соединение PCI-E состоит из соединений точка-точка кратности x1, x2, x4, x8, x12, x16 или x32, которые называются трассами (lanes). Одна трасса состоит из одной пары проводов, которая передает сигнал в прямом направлении и другой пары, которая передает сигнал в обратном направлении.

Во время передачи байта через трассу он кодируется десятью битами, которые передаются последовательно. Такое кодирование характерно тем, что в битовом потоке не встречаются последовательности одинаковых битов длиной более четырех. Благодаря этому, а также дополнительной полиномиальной свертке, сигнал в трассе становится шумоподобным, имеет небольшую мощность, высокую помехоустойчивость и обеспечивает надежное воспроизведение в приемнике тактовой частоты передачи.

Кроме того, передатчик и приемник не соединены электрически – сигнал не имеет постоянной составляющей и поэтому проходит через емкости конденсаторов, установленных на выходах приемо-передатчиков. Это увеличивает надежность связи и дает возможность выполнить горячее соединение периферийных устройств (без выключения питания). Некоторые комбинации битов запрещены и свидетельствуют об ошибке передачи, а некоторые – являются служебными и несут информацию о виде транзакции или о том, что шина свободна. Таким образом, в шине PCI-E сигнал передается непрерывно, что позволяет быстро синхронизироваться приемнику на прием данных.

Передача данных через последовательный интерфейс выполняется с использованием протокола передачи пакетов. Во время аппаратной инициализации, соединение инициализируется с автоматическим выбором количества трасс и тактовой частоты без вмешательства операционной системы.

SATA – Serial ATA – стандартная шина, которая заменяет шины IDE, SCSI и ATA для подключения НЖМД. Ее строение на физическом и логическом уровнях очень похоже на строение шины PCI-E. В отличие от шины USB, шина SATA обеспечивает значительно бо'льшую скорость отклика периферийного устройства и соответственно, большую пропускную способность из-за того, что она не имеет временного мультиплексирования данных, как в шине USB.

2.2 Организация памяти компьютеров

Общие свойства памяти компьютеров

Все компьютеры требуют запоминания переменных в ячейках памяти и их считывания оттуда по мере необходимости. Эти ячейки являются регистрами, или они образуют блоки, называемые блоками памяти (БП), которые могут запоминать миллионы битов данных.

Бывают разные типы БП с различными характеристиками в зависимости от их применения. Память с записью и чтением, где можно запоминать значение, хранить их неопределенный промежуток времени и читать при необходимости, является памятью ***с произвольным доступом*** или ***оперативным запоминающим устройством*** (**ОЗУ**, Random Access Memory, **RAM**).

В памяти типа ***постоянное запоминающее устройство*** (**ПЗУ** или Read-Only Memory, **ROM**) информация запоминается надолго и часто до того, как ПЗУ будет использовано в системе. Существует разновидность этой памяти, называется программируемым ПЗУ (programmable ROM – PROM), где пользователь может запоминать желаемые данные, но процедура записи требует использование специального электронного устройства и выполняется в течение определенного времени. Если в PROM стирание выполняется электрически, то такая память называется программируемым ПЗУ с электрическим стиранием (**EEPROM**).

Увеличение объема БП приводит к увеличению его задержки и стоимости. Поэтому в компьютерах используются БП, имеющие различные объем, быстродействие, которые формируют иерархическую систему. Главный БП данных в компьютере это ОЗУ. ОЗУ построен на микросхемах ***динамического*** **ОЗУ** (**DRAM**) или ***статического*** **ОЗУ** (**SRAM**). Цены на микросхемы памяти

уменьшаются примерно на 30% в год. DRAM обычно в пять раз дешевле SRAM такого же объема.

Также энергопотребление DRAM примерно в четыре раза меньше. Но как правило, SRAM имеет в 2–3 раза больше быстродействие чем DRAM. В современной синхронной DRAM (SDRAM) скомбинированы большой объем DRAM и высокое быстродействие буфера типа SRAM, и она, благодаря конвейерному режиму групповой пересылки (***burst mode***), обеспечивает среднее время доступа меньше 2-5 нс.

До сих пор существовала тенденция увеличения быстродействия ОЗУ вдвое каждые пять лет. Но быстродействие центрального процессорного элемента (ЦПЭ) увеличивается более динамично. Благодаря закону Мура, такой двойной рост выполняется за два года. Это приводит к увеличению промежутка между быстродействями ЦПЭ и ОЗУ. Компромиссное решение этой проблемы заключается в использовании промежуточного ОЗУ малого объема с большим быстродействием, где запоминаются данные, которые часто используются. Это ОЗУ называется ***сверхоперативною памятью*** (СОЗУ). Представителем СОЗУ является кэш-ОЗУ, в которой хранятся недавно использованные страницы виртуальной памяти. Быстродействие СОЗУ зависит от расстояния до ЦПЭ. Наиболее эффективный путь – это размещение СОЗУ у ЦПЭ на той же микросхеме. Объем такого СОЗУ ограничен технологии микросхем и равен 0,1 – 10 мегабайт. Чтобы увеличить эффективность СОЗУ, оно выполняется как иерархическое СОЗУ. СОЗУ, которое интегрировано в микросхеме ЦПЭ, является СОЗУ первого уровня.

СОЗУ, расположенное за СОЗУ первого уровня, является СОЗУ второго уровня. Часто используются СОЗУ третьего уровня. В последние годы СОЗУ первого и второго уровней размещают на одном кристалле вместе с ЦПЭ.

Когда ЦПЭ требует данные и он не находит их в СОЗУ первого уровня, то эти данные переписываются туда с СОЗУ второго уровня. Чтобы увеличить быстродействие передачи данных, СОЗУ разных уровней работают параллельно – когда порция данных читается с СОЗУ первого уровня, то одновременно в него другая порция переписывается с СОЗУ второго уровня.

Чаще всего команды и данные размещены в разных областях памяти. После чтения они загружаются в различные части ЦПЭ: команды попадают в блок управления, а данные заходят в АЛУ. Поэтому часто СОЗУ разделяются на две части: СОЗУ данных и СОЗУ команд. Такое разделение позволяет читать как данные, так и команды параллельно. Кроме того, защита данных и программ происходит по-разному. Архитектура, в которой СОЗУ данных и СОЗУ команд выполнены как отдельные блоки, исторически приобрела название ***манчестерской архитектуры***.

Большие массивы данных запоминают во внешних БП с хранением после отключения питания (***non-volatile* RAM – NVRAM**), которые имеют большой объем (10^{11} –10^{16} байт), но с малым быстродействием. Они часто основаны на магнитных дисках и иногда – на магнитных лентах. Многие современных NVRAM в портативных устройствах, таких как мобильный телефон, цифровой фотоаппарат, основанные на flash EEPROM.

Накопитель на жестких магнитных дисках - НЖМД (hard disc driver - HDD), который еще называются винчестером, является наиболее распространенным NVRAM. Его время доступа равно 1–10 мс, а скорость передачи данных – 2–100 МБ/с. Чтобы уравнять быстродействие СОЗУ и быстродействие NVRAM, в НЖМД используется буферная память среднего объема и средней производительности, которая называется HDD-кэш.

Система всех БП, которые используются ЦПЭ, называется компьютерной памятью. Память, состоящая из нескольких уровней, рассматривается пользователем как виртуальная память. Каждый уровень такой памяти обслуживается специальным блоком управления, который обеспечивает автоматическую передачу данных между блоками памяти разных уровней. Этот блок, как правило, использует некоторую стратегию, которая минимизирует среднее время доступа данных.

В соответствии с некоторой стратегией доступа, виртуальная память ведет себя как БП с объемом NVRAM (~10^{10} байтов) и временем доступа СОЗУ (~10 нс).

Интегральная схема памяти БП может быть частью ЦПЭ или системы на кристалле. В ней ячейка памяти реализована как триггер. Поэтому такой БП не может иметь большой объем (более 10^6 – 10^7 байтов), и он тратит большую часть энергии. Это является причиной того, что БП большого объема изготавливаются как отдельные микросхемы. Благодаря специальной технологии, особой конструкции ячейки памяти, усилителей чтения-записи, дешифраторов и мультиплексоров, такой БП имеет минимизированное энергопотребление, высокое быстродействие и объем до ~ 10^9 байтов. Микросхемы памяти делятся на SRAM, DRAM и EEPROM, согласно их технологии и свойствами.

Название RAM означает, что доступ к памяти - случайный, то есть по любому адресу можно иметь доступ в любой момент времени. Вспомним, что в компьютерах первого поколения использовались блоки оперативной памяти на линиях задержки, то есть с последовательным режимом доступа.

Название статического ОЗУ (SRAM) означает, что как только данные запомнятся в ОЗУ, они там остаются храниться, пока не будет отключено питание.

Напротив, в динамическом ОЗУ (DRAM) необходимо организовать периодическое ***восстановления состояния памяти*** (***refresh***), чтобы сохранить данные от потери. Существуют SRAM, которые запоминают до ~ 10^8 байтов данных. На рис. 2.1 показан пример КМОП SRAM с объемом 2 Кбайт, принципы строения которой присущи другим SRAM.

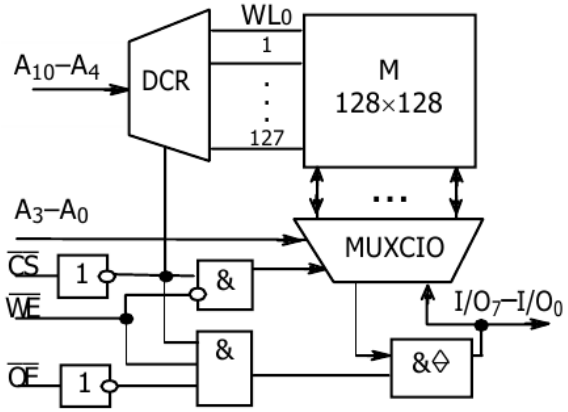


Рис.2.1. Структура SRAM объемом 4 кбит

Динамическая память

DRAM похожа на SRAM тем, что данные запоминаются по тому же закону адресации ячеек. Различие между двумя типами БП заключается в конструкции ячейки запоминающего матрицы М. Схемы таких ячеек показаны на рис. 2.2 и 2.3. Ячейка динамической памяти состоит только из одного транзистора и запоминающего конденсатора C_S. Это позволяет получить высокую плотность интеграции и получить микросхемы, которые имеют до ~ 10^{10} ячеек.

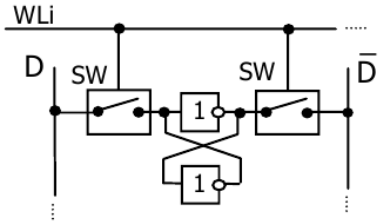


Рис. 2.2. Схема ячейки SRAM

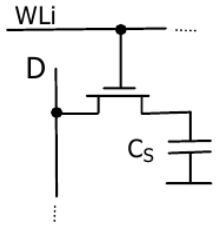


Рис. 2.3. Схема ячейки DRAM

Когда сигнал выборки строки WLi = 1, транзистор работает как замкнутый ключ и позволяет выполнять запись или чтение. Состояние хранения включается, когда сигнал WLi = 0 и путь между шиной данных и конденсатором CS прерывается. Закрытый транзистор, имеющий малый, но ощутимый ток утечки, способствует разряду конденсатора. Поэтому данные могут быть

сохранены в DRAM только в течение короткого периода времени (<1000 мс), и поэтому они должны периодически восстанавливаться.

Операция восстановления (refresh) выполняется специальной схемой усиления заряда. Период восстановления, как правило, должен быть не больше миллисекунды. За один цикл восстанавливается одна строка матрицы памяти. Чтобы восстановить DRAM, нужно перебрать все строки. Поэтому строки в большой DRAM перебираются с частотой несколько мегагерц. Чтобы обеспечить автоматическое обновление, DRAM имеет ***refresh counter*** – счетчик восстанавливаемых строк, как часть регистра адреса строки.

При доступе к DRAM часто используется отмена в адресации строк и колонок запоминающей матрицы. Пусть имеем DRAM объемом 65 Кбит. На рис. 2.4 показана временная диаграмма при записи в нее. RAS и CAS означают сигналы выбора строки (row address select) и выбора колонки (column address select). Когда $\overline{RAS}$ = 1, схема хранит данные. Когда $\overline{RAS}$ = 0, $\overline{CAS}$ = 1, разряды адреса строки запоминаются во внутреннем регистре адреса. Одновременно все ячейки соответствующей строки восстанавливаются. Когда $\overline{CAS}$ = 0, разряды адреса колонки декодируются, одновременно выполняется запись или чтение в зависимости от сигнала $\overline{WE}$, которые заканчиваются когда $\overline{CAS}$ = 1.

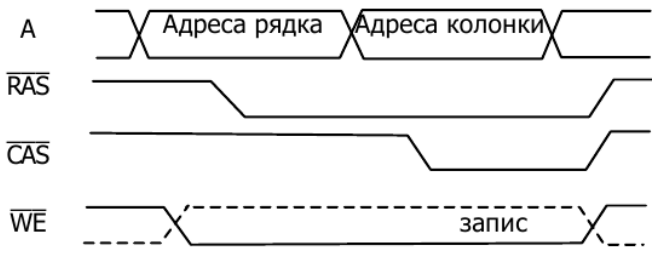


Рис. 2.4. Временная диаграмма записи в DRAM A

Как видим, адрес в DRAM загружается за два такта. Благодаря этоме, число выводов адресов микросхемы вдвое меньше, чем в SRAM. Время доступа к DRAM минимизируют, когда адрес строки запоминается, и тогда все ячейки этой строки доступны по адресу колонки. Недостаток такого режима заключается в необходимости внешнего мультиплексора адресов, который формирует адрес строки и колонки.

Микропроцессоры, как правило, имеют специальный контроллер DRAM, который обеспечивает как подачу адресов, сигналов управления, так и управления временными интервалами, инициализации микросхем, контроль битов. Поэтому недостатки DRAM "невидимые" для пользователя.

Из-за больших схемы дешифрации адреса, задержки усилителей сигналов и двухтактный режим доступа, DRAM имеет время доступа десятки наносекунд. Современная DRAM обеспечивает бо'льшее быстродействие доступа благодаря следующим улучшениям. Ряд банков памяти, размещенных в микросхеме, имеют независимые дешифраторы и схемы управления. Слово адреса является объединением адресов строк, колонки и банка. Адрес банка может изменяться скорее, чем время доступа к нему, так как в соседних тактах происходит доступ к различным банкам. Микросхемы DRAM имеют двунаправленные шины данных с разрядностью до 36 и более битов. Чтобы обеспечить доступ к части этих данных (байт, тетрада), используется ***битовая маска*** (DQM).

В синхронной DRAM (SDRAM) выполнено конвейерный доступ, то есть входные данные, адрес, считанные данные и даже массив данных, читается по адресу строки, запоминается в буферных регистрах на время такта синхросерии. Поэтому операция чтения требует 2–10 тактов. Но благодаря этому, частоту синхросерии можно увеличить до сотен мегагерц.

Если выполняется набор одинаковых операций, например, последовательность чтений, то один доступ может выполняться, в среднем, за один такт. Для этого служит режим ***быстрого страничного доступа*** (***fast page mode*** – FPM) или режим ***групповой пересылки*** (режим ***burst***). В этом режиме адрес строки запоминается и доступ выполняется для различных адресов колонок. Это обеспечивает регистр адреса колонки, который реализован как счетчик. Поэтому когда ЦПЭ записывает или читает набор данных по последовательности адресов, обычно используется режим групповой пересылки.

Чтобы управлять режимом быстрого страничного доступа, временные зазадержки программируются в DRAM заданием четырех чисел. Например, набор 5-1-1-1 означает, что для выполнения первого доступа необходимо 5 тактов, для остальных доступов - по одному такту. Блок управления DRAM должен различать, что DRAM работает в разных режимах (инициализация, чтение, запись, групповая пересылка, восстановление, загрузка и др.), для которых следует подать соответствующие последовательности управляющих сигналов. Сигналы RAS, CAS, WE и DQM формируют слово команды (запись адреса, чтение и т.д.), которое запоминается в DRAM по фронту синхросигнала или по его фронту и спаду, если DRAM в режиме DDR. Во время загрузки режима биты адреса считаются битами управления.

Постоянная память

EPROM - это электрически программируемое ПЗУ (Erasable-Programmable ROM). Программирование EPROM выполняется передачей заряда в простран-

ство затвора МОП-транзистора, который служит запоминающим элементом, благодаря повышенному напряжению на затворе и эффекту туннелирования электронов.

Если этот заряд присутствует, то транзистор открыт. Этот заряд сохраняется долгое время, потому что он "застревает" в диэлектрике затвора или во вспомогательном затворе, который погружен в диэлектрик, так называемом, плавающем затворе. В устройствах такого типа стирание данных происходит при его облучении ультрафиолетовым светом в течение нескольких минут.

ПЗУ с электрическим стиранием (electrically erasable EPROM - **EEPROM, E2PROM**) имеют то преимущество, что данные могут быть стертыми электрической схемой. Чтобы стереть бит, на затвор запоминающего транзистора подается напряжение обратной полярности. Новая технология дает возможность выполнять большое число стираний и записей (до миллиона) за короткое время. За высокую скорость стирания, по сравнению с EPROM, эти устройства называются flash EPROM, где "flash" – "вспышка" - означает быстродействие, с которым массив данных может быть стертым.

Сегнетоэлектрические **ПЗУ** (***ferroelectric*** **RAM – FRAM**) сохраняют преимущества DRAM (большие объем и быстродействие) и NVRAM (данные сохраняются после отключения питания). Ячейка ПЗУ напоминает ячейку DRAM. Но бит данных в ней запоминается не как заряд, а как знак поляризации диэлектрика.

Для этого конденсатор сделан из сегнетоэлектрика. Когда выполняется чтения, то конденсатор пытаются перезарядить. И если конденсатор при этом меняет поляризацию, то считается, что был записан единичный бит. Но этот процесс приводит к изменению поляризации. Поэтому процесс чтения заканчивается записью прочитанного бита назад.

Блоки сверхоперативной памяти

Время доступа СОЗУ меньше, чем в обычных ОЗУ. СОЗУ подключают к ЦПЭ, чтобы уменьшить поток данных от RAM или средний период доступа. При этом увеличивается быстродействие ЦПЭ, потому что он непосредственно обращается к быстродействующему СОЗУ. Кроме того, доступ к СОЗУ и операции в АЛУ могут быть реализованы параллельно.

Различают СОЗУ с режимами непосредственной, ассоциативной, конвейерной и стековой адресации. При непосредственной адресации ячейки СОЗУ имеют адреса с 0 по m–1. Адрес ячейки размещается непосредственно в поле адреса команды. Когда m имеет малое значение (до 32–128), СОЗУ называют регистровой памятью.

В режиме ассоциативной адресации, операнд имеет не адрес, а ярлык, тэг (tag) или ключевое слово. Когда тэг подается на вход ассоциативного СОЗУ, то оно выдает одно или несколько слов с тэгами, которые совпадают с входным тэгом, или ничего, если тэг не совпадает. Этот режим адресации используется в кэш-памяти, которая будет рассмотрена далее.

Два или более процессорных блока часто соединяются через буферное СОЗУ с конвейерным режимом адресации. Такое СОЗУ представляет собой набор регистров, которые соединены в конвейер. Блок-источник записывает данные в СОЗУ, а блок-приемник читает эти данные в том же порядке, в котором данные поступают в СОЗУ. Этот режим называется первый зашел - первый вышел (***first in – first out* – FIFO**). Поэтому такой СОЗУ называется как FIFO.

FIFO часто используется, когда блок-источник выдает данные через неравномерные промежутки времени, или с другой частотой, чем блок-приемник их принимает, например, в коммуникационных системах.

В режиме стековой адресации СОЗУ рассматривается как регистр вершины стека. Пользователь имеет доступ только к регистру вершины стека и может "запихнуть" (***push***) данные в него или вынуть (***pull***) данные из него. Считанное данное - это данное, которое было занесено в стек последним, и поэтому этот режим называется "последний используется первым" (***least recently used* – LRU**). Часто есть доступ не только к последнему регистру, но и к предпоследнему, или произвольному – по адресу. Например, таким образом построен стек сопроцессора с плавающей запятой i8x87 микропроцессоров компании Intel.

Кэш-память

Автоматический обмен данными между СОЗУ и ОЗУ достигается в режиме ассоциативной адресации. Структура такого СОЗУ показана на рис. 2.5. Он имеет блок памяти информации БП (M), ассоциативный БП (CAM), компаратор адреса CMP, блок управления CU, регистр адреса RGA, и регистр данных RGD.

Каждое данное, записывается в M, имеет тэг, сохраняет адрес данного, который хранится в CAM. Когда выполняется доступ СОЗУ, в RGA записывается адрес, и сигналы чтения RD и записи WR поступают в CU. Адрес, который поступил, сравнивается с адресами в CAM с помощью компараторов CMP. Если адрес равен некоторому адресу в CAM, тогда говорят о ***попадании в кэш*** (***hit***). В этой ситуации данное читается с M в RGD или записывается с RGD в M. Если адрес не равен какому-либо адресу в CAM, говорят о ***непопадание в кэш*** (***miss***). Тогда CU организует доступ к внешнему ОЗУ.

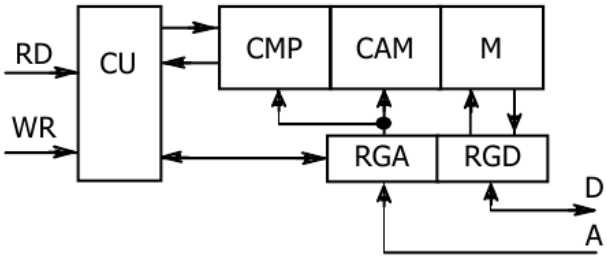


Рис. 2.5. Структура кэш-ОЗУ

Данное, прочитанное с ОЗУ, записывается в RGD и в пустую ячейку блока M, а его адрес записывается в соответствующую ячейку в CAM. При следующем чтении этого данного чтения выполняется с СОЗУ, а не из внешнего ОЗУ. Когда выполняется запись данного, то оно записывается как в M, так и во внешнее ОЗУ.

Когда кэш-ОЗУ в действии, то все его ячейки заполнены. Поэтому, чтобы доступ по новому адресу состоялся, какая-нибудь ячейка кэш-ОЗУ должна быть освобождена. Естественно, когда высвобождается ячейка, в которую был доступ наименьшее количество раз. Чтобы ее определить, блок CU имеет схему, которая выбирает ее по некоторой стратегии.

Самая распространенная стратегия следующая. Пусть триггер Ti привязан к *i*-й ячейке CAM. В начале Ti = 0 для всех *i*. Если доступ к *i*-й ячейке состоялся с попаданием, тогда Ti = 1. Это означает, что по крайней мере, один доступ к этой ячейке состоялся. Если произошло непопадание, тогда выбирается для очистки первая ячейка, в которой Ti = 0. Когда при этом Ti = 1 для всех *i*, тогда очищаются все ячейки и устанавливаются все Ti = 0, кроме одной, которая выбрана для доступа. Также используют дисциплину случайного выбора ячейки для очистки, или выбор ячейки по кругу.

Больше об особенностях дисциплины обращения к кэш-памяти – в разделе 3.6. «Параллелизм доступа памяти».

Недостаток ассоциативного ОЗУ заключается в сложности аппаратуры, когда количество адресов велико. Ее эффективность увеличивается, когда используется адресно-ассоциативный режим. Рассмотрим пример блока кэш-ОЗУ объемом 8 кбайт микропроцессора i486. Он имеет 32-разрядную шину адреса и 4-байтовую шину данных. Одна ассоциативная ячейка содержит четыре 16-байтовые строки данных и 91-битный тэг (рис. 2.6). Блок памяти M имеет только 128 ассоциативных ячейки.

Тэг имеет четыре 21-разрядных поля адресов, 4 бита кода правильности строки и 3 бита, которые используются чтобы находить ячейку для освобождения. 4 младших бита адреса выбирают один байт в строке. 7 промежуточных разряда адреса выбирают ассоциативнуя ячейку. Остальные разряды кода адреса сравниваются с соответствующим 21-разрядным полем тэга.

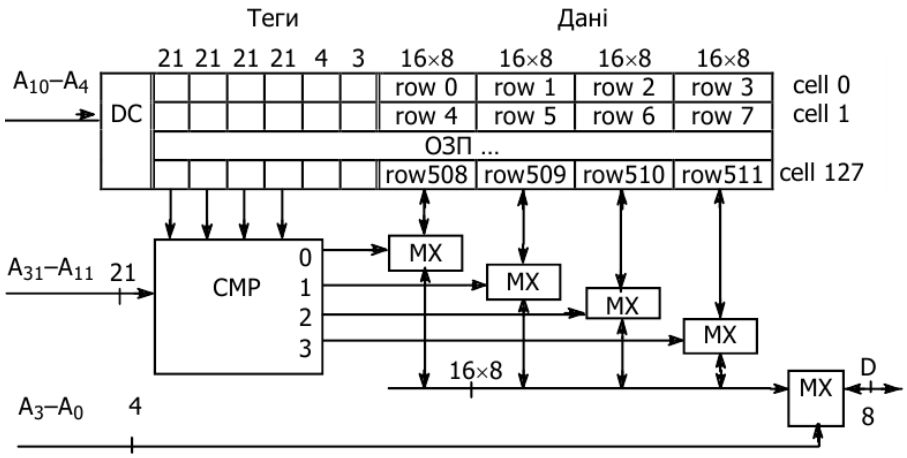


Рис. 2.6. Структура адресно-ассоциативного кэш-ОЗУ

Когда выполняется доступ памяти, 7-битное поле адреса выбирает одну ассоциативную ячейку, четыре 21-битовых компаратора сравнивают старшие разряды адреса с соответствующими полями тэга. Если сравнение удачное и бит правильности равен 1, то данное считывается. Иначе в выбранной ячейке строка отменяется, и если к нему не происходил доступ долгое время, в нее записывается данное с внешней памяти. Одновременно корректируется соответствующее поле тэга. Следовательно, такое кэш-ОЗУ построено на обычном ОЗУ и лишь на четырех компараторах адресов.

Многие микропроцессоры имеют кэш-ОЗУ подобной структуры. Они отличаются объемом, числом ассоциативных ячеек, длиной строки и т.д. Когда произошло непопадание, то заменяется не один байт, а целая строка данных. Поэтому, если последовательность адресов случайная, то среднее время доступа может уменьшиться во много раз из-за многих повторений пересылок строк между кэш-ОЗУ и внешним ОЗУ. Но в чаще последовательность адресов – это растущие адреса, например, адреса цепочек команд или соседних элементов массивов. Поэтому следующие команды или данные зачастую появляются в той же строке, и непопадание в кэш возникает реже.

2.3. Внешние запоминающие устройства

Периферийные устройства

Периферийные устройства компьютеров делятся на устройства ввода-вывода и внешние запоминающие устройства. Общей характеристикой устройств ввода-вывода служит скорость передачи данных (максимальная скорость, с которой данные могут передаваться между устройством ввода-вывода и основной памятью или ЦПЭ). В табл. 2.1. представлены основные устройства ввода-вывода, которые применяются в современных компьютерах,

а также указаны приблизительные скорости обмена данными, которые обеспечиваются ими. Рассмотрим наиболее быстрые из этих устройств – магнитные диски.

Таблица 2.1. Примеры устройств ввода-вывода

Тип устройства	Направление передачи данных	Скорость передачи данных (Кбайт/с)
Клавиатура	Ввод	0.01
Мышь	Ввод	0.10
Голосовой ввод-вывод	Ввод-вывод	20.00
Сканер	Ввод	200.00
Лазерный принтер	Вывод	500.00
Графический дисплей	Вывод	100000.00
Магнитная лента	Ввод-вывод	2000.00
Оптический диск	Ввод-вывод	5000.00
Магнитный диск	Ввод-вывод	50000.00

Магнитные диски

Накопитель на жестких магнитных дисках (НЖМД) состоит из одного или нескольких, чаще всего, алюминиевых дисков с магнитными слоями с двух сторон. Дорожкой называется круговая последовательность битов, записанных на диск за его полный оборот. Каждая дорожка делится на секторы фиксированной длины. Обычно, один сектор содержит 512 байтов данных. Теперь бывают 2048-байтовые секторы, обеспечивающих повышенное быстродействие НЖМД.

Перед данными в секторе размещен заголовок, который позволяет магнитной головке синхронизироваться перед чтением или записью данных. В заголовке записан адрес сектора и флажки, в том числе флажок испорченного сектора. После данных в секторе идет код циклического контроля. Это, как правило, код свертки с 32-разрядным полиномом, который позволяет надежно определить, испорчены данные в секторе, или нет. Также в современных НЖМД в конце сетора размещен код исправления ошибок. Это код Рида-Соломона, который дает возможность исправлять несколько испорченных байтов. Между соседними секторами находится межсекторный интервал.

Структура дорожки формируется при форматировании НЖМД, чаще всего, во время его изготовления. Из-за наличия преамбулы, контрольных кодов и межсекторного интервала, емкость форматирования НЖМД примерно на 15% меньше емкости неформатированного.

Магнитные диски размещены на одном шпинделе, который раскручивается мотором со скоростью 5400, 7200 или 10000 об./мин. в зависимости от конструкции НЖМД (рис.2.7).

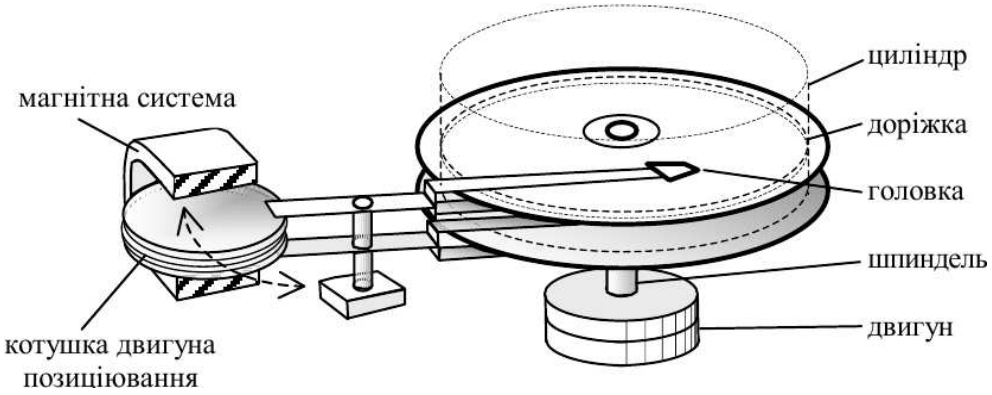


Рис. 2.7. Конструкция НЖМД с четырьмя головками

Магнитные головки закреплены на блоке из легких консолей, который может точно и быстро позиционировать головки над определенной дорожкой с помощью электромагнитного двигателя (в среднем по 5-15 мс, переход на соседнюю дорожку – менее 1 мс). Итак, дорожки представляют собой ряд концентрических кругов, центр которых совпадает с центром шпинделя. Ширина каждой дорожки составляет долю микрона. Совокупность дорожек, расположенных на одном расстоянии от центра, называются ***цилиндром***.

Плотность записи битов на дорожках различна и зависит от расстояния от центра диска. Плотность записи также зависит от качества поверхностного слоя диска, конструкции головки и чистоты воздуха и достигает 10 000 бит/мм. Чтобы обеспечить долговечность НЖМД, он выполняется в жестком герметичном корпусе, внутреннее пространство которого связано с наружным воздухом через плотный фильтр.

Со стороны операционной системы НЖМД воспринимается как фиксированный набор дисков с числом головок $N_г$, числом цилиндров $n_ц$, на каждом из которых находятся $N_С$ секторов. Причем номера головки, цилиндра, сектора записываются в преамбулы соответствующих секторов во время форматирования на низком уровне. Нумерация цилиндров и головок начинается с 0, а секторов – с 1.

В системе LBA (Logical Block Addressing) на нумерацию цилиндра отводится 16 бит, головки – 4 бита и сектора – 8 бит, что позволяет изготавливать НЖМД с объемом до $2^{16+4+8+9}$ = 131 Гбайт. С целью существенного удаления границы максимального объема НЖМД в 2002 г. утвержден протокол UltraDMA доступа к НЖМД, согласно которому на адресацию сектора отводится до 48 бит.

НЖМД имеет собственный контроллер, который воспринимает через интерфейс НЖМД (ATA, SATA или SCSI) команды чтения, записи, форматирования и другие. Также контроллер выполняет исправление ошибок чтения. Кроме того, контроллер имеет буферное кэш-ОЗУ объемом несколько мегабайт и устройство DMA, благодаря которым достигается уменьшение среднего времени доступа к выбранному сектору и увеличение скорости потока между НЖМД и основным ОЗУ до нескольких десятков мегабайт в секунду.

Для уплотнения сохранения информации количество секторов на внешних цилиндрах увеличивают по сравнению с количеством секторов на внутренних. Также количество физических секторов на одной дорожке достигает нескольких тысяч, что находится в противоречии с адресацией по системе LBA. Поэтому НЖМД с физическим диском может иметь более десятка логических дисков или несовпадение адресов логических и физических секторов.

Отображение адресов логических секторов в адреса физических секторов выполняется контроллером НЖМД. Кроме того, этот контроллер периодически проверяет целостность секторов. В случае нахождения поврежденного сектора он незаметно для пользователя заменяет его на сектор из зарезервированной области диска с соответствующей коррекцией таблицы перекодировки физических адресов секторов в логические. Такая таблица хранится в специальном EEPROM.

Компакт-диски

Компакт-диск (***Compact Disc*** **– CD**) или оптический диск разработала корпорация Philips вместе с Sony в 1980 году для хранения и широкого распространения аудиоинформации. Подавляющее большинство CD имеет диаметр 120 мм, толщину 1,2 мм, а диаметр отверстия в середине – 15 мм. Тонкий слой алюминия, нанесенный на прозрачную поликарбонатную основу, имеет спиралевидную дорожку. На ней находятся углубления (pits) размером меньше микророна и области уровня поверхности (lands), которые кодируют единицы и нули информации. При воспроизведении записи сфокусированный луч лазерного диода освещает дорожку, а фоторегистрирующая матрица воспринимает значительно меньше света от углубления, чем от области поверхности. Также она следит за тем, чтобы луч был сфокусирован в центре дорожки. Положительный и отрицательный переходы регистрируемого уровня освещенности фиксируются как 0 и 1, соответственно.

С 1984 CD начали использоваться для хранения компьютерных данных под названием **CD-ROM** (***Compact Disc – Read Only Memory***). Во время записи данных каждый байт кодируется помехоустойчивым 14-битным кодом Хэминга. Каждые 24 байта запоминаются в 588-битовом фрейме, 396 битов

которого служат для исправления ошибок и контроля. Каждые 98 фреймов формируют один сектор. Сектор начинается с 16-байтовой преамбулы, которая вмещает синхронизирующую последовательность, трехбайтовый номер сектора, по которому драйвер находит нужный сектор и байт кодирования типа сектора. Всего сектор вмещает 2 048 байтов данных и 228 дополнительных байтов для исправления ошибок по алгоритму Рида-Соломона.

Кроме того, байты из соседних секторов перетасованы. Следовательно, если на поверхности диска появится широкая царапина, которая испортит несколько десятков соседних байтов, то при чтении после обратного перемешивания поврежденные байта более равномерно распространятся среди байтов неиспорченных секторов и смогут быть восстановлены с помощью алгоритма декодирования Рида Соломона. Но из-за выполнения трехуровневой системы исправления ошибок (в байтах, фреймах и секторах) объем записываемой информации увеличен в 3,6 раза.

Для реализации возможности самостоятельной записи на CD-ROM, получили распространение CD-ROM с записью - **CD-R** (***CD-Recordable***). Они имеют дополнительную канавку шириной 0,6 мм для направления лазера при записи. Между слоем металла для отражение и основой размещен слой красителя, который под воздействием нагрева лазером темнеет.

В компакт-дисках с возможностью перезаписи - в **CD-RW** (**CD- *Rewritable***) вместо красителя используется легкоплавкий сплав серебра, индия, сурьмы и теллура, который имеет кристаллический и аморфный состояния с разной степенью отражения.

CD-ROM имеют емкость около 700 Мбайт. Для увеличения их емкости в несколько раз были разработаны диски **DVD** (***Digital Versatile Disc***). Уплотнение записи в них до 4,7 Гбайт и более достигается за счет уменьшения вдвое размеров углублений, шага спирали дорожки, а также изготовления многослойного диска. Для этого пришлось заменить инфракрасный лазер на лазер красного цвета. В последнее время получили распространение диски ***Blue Ray***, объем которых до 70 Гбайт достигнут благодаря использованию голубого лазера, что и было причиной происхождения их названия.

Логическая структура НЖМД системы FAT

Большинство модулей ОС и пользовательские программы выполняют доступ к НЖМД, не адресуя определенные секторы, цилиндры и головки, а как к файловой системе. Логическая структура НЖМД задает отображение файла в множество кластеров.

Кластер – это некоторое заранее заданное количество соседних секторов (1,2,4, ..., 128), которое определяет минимальный объем файла.

Файл состоит из одного и более кластеров, необязательно размещенных подряд.

Каталог файлов – файл, который указывает имя файла, его атрибуты и номер начального кластера файла, а также дочерние ветви каталога.

Корневой кластер – кластер, содержащий корень каталога файлов.

Таблица размещения файлов – File Allocation Table (**FAT**) – таблица, количество записей в которой равно количеству кластеров в логическом диске НЖМД, каждая запись соответствует одному кластеру и может быть меткой конца файла, если это конечный кластер файла, номером следующего кластера файла, может указывать, что кластер незанятый, или то, что он испорчен.

Наиболее распространены стандартные логические структуры НЖМД – это системы FAT и NTFS. Существуют три разновидности системы FAT с различными размерами записи в таблице: FAT12 – разрядность записи – 12, то есть можно задать максимум 4093 кластера; FAT16 – разрядность записи – 16, то есть максимум – 65533 кластера; FAT32 – разрядность записи – 32, то есть максимум – 2^{32} –3 кластера.

При этом один НЖМД может содержать от одного до четырех логических диска (раздела) со своими, возможно, различными таблицами FAT.

Структура логического диска

Первый сектор в физическом диске – ***бутовский сектор*** (***master boot sector***) – содержит параметры физического диска. Он находится на стороне 0, цилиндре 0 и в секторе 1 и содержит следующую информацию:

– размер сектора (обычно 512, бывает 2048);

– количество секторов в кластере;

– количество секторов на дорожке;

– число цилиндров;

– размер таблицы FAT;

– сообщения об ошибках;

– таблицу деления на логические диски (partitition table);

– размещения начальных точек разделов, указание на место, где хранится операционная система.

После бутовского сектора следуют от одного до четырех логических дисков.

Логический диск включает:

– бутовская запись;

– резервные секторы;

– первую и вторую таблицы FAT;

– кластер корня каталога;
– кластеры данных (файлов);
– еще резервные секторы.

Бутовская запись (***boot record***) включает следующие данные:
– команда перехода на программу начального загрузчика;
– версия ДОС;
– количество байтов на сектор;
– количество секторов в кластере;
– количество резервных секторов перед таблицей FAT;
– число таблиц FAT;
– максимальное число строк в корневом кластере;
– число секторов в которых хранится ДОС;
– число секторов, в которых хранится таблица FAT;
– число секторов в дорожке;
– число головок;
– число скрытых секторов.

Из-за меньшей плотности записи на нулевой дорожке, надежность сохранение данных на ней выше. Поэтому бутовский сектор, бутовская запись диска С: и его таблицы FAT размещаются именно на этой дорожке.

Поиск местоположения файлов производится по первой таблице FAT. Если эта таблица оказалась испорченной, то используется дублирующая ее вторая таблица. Но если будет поврежден первый сектор, содержащий таблицу деления на логические диски, то ОС не будет способна воспроизвести информацию о логическом строении НЖМД, и запись-чтение файлов становится невозможным.

Логическая структура НЖМД системы NTFS

Файловая система ***New Technology File System*** (**NTFS**) появилась вместе с Windows NT в 1993г. и переняла все положительные свойства файловой системы Unix. В системе NTFS все данные представлены в виде файлов с одинаковой структурой. Файл меньшего размера занимает один кластер и обязательно имеет такие поля, как заголовок, поле стандартной информации, имя файла, хранящиеся данные, а в конце файла – дескриптор безопасности.

В основу системы положена система служебных файлов, главным из которых является каталог файлов, называемый Master File Table (MFT). Все служебные файлы хранятся в MFT-зоне в начале диска, которая занимает 1/8 всего объема диска. При переполнении диска эта зона также заполняется файлами пользователя, приводя к замедленному доступа.

Основной файл $MFT представляет собой массив файловых записей, каждый из которых описывает соответствующий файл или папку. Одна запись имеет длину 1 килобайт и содержит: имя файла, его атрибуты и индексы файла в виде номера первого кластера файла, длины цепочки кластеров файла. Если файл размещается в цепочках кластеров, то указывается ссылка на первый кластер цепочки продолжения с указанием длины цепочки. Если описание файла не помещается в одной записи, то дается ссылка на запись продолжения. Возможно продолжение файла в нескольких цепочках, так что файл представляется в виде дерева. Это дает возможность считывать файл параллельно несколькими потоками. В записи подкаталога есть ссылки на файлы из него.

Максимальный номер кластера равен 2^{64}, хотя реально длина файла не превышает 2 Тбайт. Для уплотнения кодирования координат кластеров, они кодируются кодами Хафмена. Итак, каталог файлов представляет собой иерархический отсортированный список записей в виде дерева. Поиск адресов кластеров ведется не по таблице FAT, а проходом по дереву цепочек кластеров.

В отдельных служебных файлах $Bitmap и $BadClust отмечаются свободные и сбойные кластеры. Причем таблица свободных кластеров занимает столько битов, сколько кластеров находится на НЖМД. Следовательно, то, что данный кластер является свободным, отмечается одним битом. В файле $Boot сохраняется системный загрузчик.

Для надежности функционирования НЖМД, четыре основных файла с MFT-зоны хранятся в двух экземплярах – первый в начальных кластерах, а второй – где-то посредине диска со стартовыми адресами, указанными в бутовском секторе.

При выполнении любой файловой операции ОС записывает параметры этой операции в служебный файл журнала $LogFile. Тогда, если отказ компьютера произойдет при выполнении файловой операции, то при восстановлении работоспособности компьютера, ОС по записям в этом журнале отремонтирует поврежденные файлы и записи. Кроме того, право доступа к служебным файлам имеют только программы ОС с наивысшим приоритетом. Благодаря таким особенностям, файловая система NTFS существенно надежнее системы FAT.

Система накопителей RAID

Скоростной НЖМД высокой надежности существенно дороже обычных НЖМД. Из-за этого в 1988 г. была предложена архитектура массива недорогих НЖМД, которая из-за своей избыточности обеспечивает как надежность хранения данных, так и высокую скорость доступа. Из-за этого она называется **RAID** (Redundunt Array of Inexpensive Disks). В накопителе RAID от 4 до 15

НЖМД параллельно подключены к одному контроллеру таким образом, что ОС воспринимает его как один большой НЖМД.

Существует несколько способов организации RAID-массива, которые называются ***уровнями***. Например, в одном из уровней все секторы НЖМД разделяются на множества ***полос*** (***strips***), причем полосы дублируются на НЖМД. Из-за этого запись файла занимает то же время, как у обычного НЖМД, а чтение ускоряется, так как выполняется параллельно с нескольких НЖМД. При повреждении одной полосы она восстанавливается по ее копии. При отказе одного из НЖМД он не останавливает работу накопителя в целом и может быть заменен на исправный НЖМД без выключения питания.

Теперь во многих чипсетах ПК встроены контроллеры накопителя RAID, что позволяет их широкое использование.

2.4.Архитектура консоли компьютера

Понятие консоли

Операторская консоль появилась в первых компьютерах и была пультом управления и индикации компьютера, который был консольно закреплен на шкафу с майнфреймом. Через консоль вводились данные и отображались результаты, компьютер запускался и останавливался.

С тех пор традиционно набор устройств для непосредственного управления компьютером, ручного ввода-вывода данных и программ называют консолью. Все алгоритмические языки имеют отдельное множество функций и процедур для ввода числовых и текстовых данных с консоли и их вывода на консоль. Также традиционно строковые команды ОС вводятся с консоли а ОС выводит результаты реакции на консоль.

В первых компьютерах консоль состояла из набора переключателей и лампочек или электрической пишущей машинки. Теперь это, как правило, дисплей с клавиатурой и манипулятором типа мышь.

Клавиатура и мышь

Клавиатура используется для ручного ввода и данных. Клавиатура различается по положению букв на ее поле. Стандартное розположени латинского клавиатуры – это расположение "QWERTY" (по расположению букв в верхней строке), а кириллической – "ЙЦУКЕНГ". Стандартная клавиатура имеет 101 клавиши.

Сейчас клавиатура подключается к процессору через соединение PS/2 или USB. Соответственно, такая клавиатура называется клавиатурой типа PS/2 или

USB. Когда кнопка нажата, генерируется код соответствующей буквы, который передается в процессор. В клавиатуре типа PS / 2 код буквы не являются ее ASCII-кодом, а является, так называемым, ***скан-кодом***.

Как только буква нажата, то передается ее скан-код, который заносится в порт BIOS с номером 60h. Если она нажата и удерживается, то скан-код выдается примерно каждые 100 мс. Каждая клавиша имеет единственный скан-код, который посылается, как только она нажимается. После отпускания клавиши клавиатура посылает код отпуска "F0", после которого выдается код клавиши, которую отпустили. При нажатии некоторых клавиш, называемых расширенными клавишами, перед скан-кодом посылается код "E0".

Мышь – это ручное устройство, которое в своей традиционной конструкции состоит из резинового шарика с нижней стороны и нескольких кнопок – с верхней. Мышь изобрел Д.Енджелбарт в 1967 г. во время исследований по поиску эффективных интерфейсов между человеком и ЭВМ.

Если мышь движется, шарик делает вращения, которые пропорциональны дистанции, прошедшей мышью. По четырехжильному кабелю через разъем PS/2 или USB коды перемещения и события нажатия или отпускания кнопок посылаются в ПК.

В оптической мыши шарик заменен на оптическую систему из светодиода и светочувствительной матрицы. Спецконтролер мыши вычисляет ее перемещения, если луч светодиода при движении отражается от поверхности с различным пространственным поглощением света.

В каждый момент, когда мышь движется, она посылает три байта последовательным кодом, который при подключении к PS/2 удовлетворяет стандарту соединение RS-232 с битом четности. Эти байты задают состояние мыши и ее перемещение по осям *x* и *y*. В байте состояния задаются направление перемещения, то, что скорость перемещения больше некоторой границы, а также события нажатия на кнопки мыши. Если мышь движется без остановок, то она посылает сигналы примерно каждые 50 мс.

Видеодисплей

Видеодисплей или монитор построен на основе электронно-лучевой трубки или жидкокристаллической панели, а также блока управления.

В электронно-лучевой трубке электронная пушка генерирует луч электронов. Система вертикального и горизонтального отклонения формирует магнитное поле на пути электронного луча таким образом, чтобы он отклонялся и сканировал люминесцентный экран согласно растру – вдоль строк, слева – направо и сверху – вниз. Электроны, попавшие на экран, подсвечивают люминофор с яркостью, которая пропорциональна току луча. Модуляция тока

луча во времени соответствует яркости изображения на экране. Для получения цветного изображения используют три электронных пушки и растр из люминофорных пятен трех цветов – красного, зеленого и синего (R, G, B). Специальная маска перед экраном обеспечивает попадание электронов только из одной пушки на люминофор определенного цвета.

Одна цветная точка на экране называется ***пикселом*** (picture element - ***pixel***). Один пиксел чаще всего кодируется трехбайтовым словом, байты которого задают яркость соответствующих цветов. Подача информации на экран управляется тактовым синхросигналом с частотой F_c.

Итак, слова пикселей подаются на модуляторы дисплея с частотой F_c. Этот поток сопровождается синхроимпульсами (СИ) вертикальной VS и горизонтальной HS развертки. Параметры СИ для стандартного VGA (video graphic adapter) дисплея с размерами изображения 640 на 480 точек для частоты смены кадров 60 Гц и Fc = 25 МГц представлены в табл. 2.2.

Таблица 2.2. Временные параметры дисплея с разрешением 640·480

	СИ вертикальной развертки VS			СИ горизонтальной развертки HS	
Параметр	Время	Тактов Fc	Строк	Время такта	Fc
Период синхросигнала	16.7 мс	416800	521	32 мкс	800
Время отображения	15.36 мс	384000	480	25.6 мкс	640
Ширина импульса	64 мкс	1 600	2	3.84 мкс	96

Блок управления дисплея обеспечивает точные горизонтальную и вертикальную развертки луча, которые синхронизируются согласно СИ вертикальной и горизонтальной развертки. Он также обеспечивает корректную форму изображения и четкость и правильное отражение цветовой гаммы.

Как правило, возможности подстройки параметров дисплея. В современных дисплеях блок управления имеет возможность передать ЦПЭ информацию о состоянии дисплея по шине I^2C, чтобы ОС смогла выполнить соответствующую настройку контроллера дисплея.

Жидкокристаллическая (ЖК) панель дисплея состоит из источника света и жидкокристаллического модулятора света. Источник света – это тонкая люминесцентная лампа, светодиодный или электролюминесцентный осветитель с однородной светимостью. Модулятор света имеет многослойную конструкцию, основные слои которой – это поляризационные фильтры, слой жидкокристаллических ячеек и слой цветного фильтра в виде растра из троек

сегментов красного, синего и зеленого цветов, количество которых равно количества пикселей.

Поляризационные фильтры, между которыми размещены ЖК ячейки, имеют ортогональные плоскости поляризации. Жидкие кристаллы между ними в состоянии покоя вращают плоскость поляризации, благодаря чему свет проникает через ячейки. При подаче управляющего сигнала на жидкокристаллическую ячейку жидкий кристалл под действием электрического поля перестает вращать плоскость поляризации, и фильтры в области ячейки не пропус кают свет. Если ширина и амплитуда импульсов управляющего сигнала модулированные, то средняя попускная способность ячейки регулируется в широких пределах, в результате чего можно задавать произвольную яркость и цвет соответствующего пиксела.

Для параллельного управления всеми ячейками управляющие сигналы на них подаются с распределенного дешифратора, выполненного в слое тонкопленочных транзисторов на основе органических полупроводников.

Поэтому жидкокристаллическая панель соответствующей конструкции называется **TFT-*дисплеем***. (thin film transistor – TFT).

В последние годы получают распространение дисплеи на основе органических светодиодных матриц, которые называются **OLED-*дисплеями*** (organic light emitting diode – OLED). В них каждый пиксел составлен из разноцветных миниатюрных органических светодиодов.

На дисплей каждого вида графический сигнал подается через стандартные соединения. Наиболее распространенное соединение – соединение VGA (video graphic array), через которое в дисплей передаются три аналоговых сигналы яркости цветов R, G, B, два синхросигналы вертикальной VS и горизонтальной HS развертки и сигнал наличия пикселей DE. При приеме видеосигнала через соединение VGA жидкокристаллическим дисплеем он преобразуется в цифровую форму с помощью аналого-цифровых преобразователей с тактовой частотой Fc.

Другим стандартным соединением является разъем **DVI** (digital video interface). Это соединение представляет собой цифровой последовательный интерфейс с принципом функционирования таким же, как у интерфейса PCI-E. В отличие от VGA, разъем DVI обеспечивает четкое воспроизведение изображения с произвольными размерами (до 4000·2000) и высокой частотой кадров из-за того, что цифровой сигнал не превращается в промежуточную аналоговую форму.

Разъем **HDMI** (high definition multimedia interface) по электрической схеме подобен соединению DVI, но способен передавать не только изображения, но и

закодированное звуковое сопровождение и управляющие сигналы. Кроме того, изображение ограничено прямоугольником 1920 на 1080 пикселей.

Контроллер дисплея

Контроллер дисплея – это блок, который поставляет информацию для отображение на дисплее в соответствии с выбранным режимом работы дисплея. Структура типового контроллера дисплея показана на рис. 2.8.

При работе в графическом режиме, в буфере кадров хранятся изображения, которые отображаются на дисплее в виде массивов с размерами, равными размерам растра дисплея, например, 640·480. В одном слове массива хранится информация о яркости цветов пиксела с соответствующими координатами. Согласно настройкам контроллера дисплея, разрядность слова меняется от 8 до 32 бит.

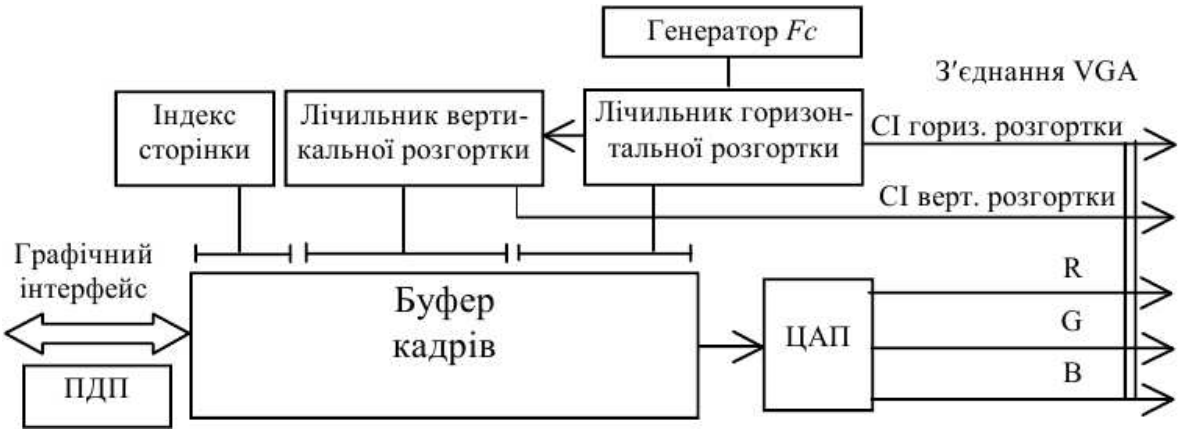


Рис.2.8. Структура контроллера дисплея

Светимость пикселей дисплея является, как правило, нелинейной. Для линеаризации светимости в простейшем случае используют гамма-корекцию. Она заключается в возведении кода V яркости пиксела в степень γ. Например, значение яркости, которое подается на жидкокристаллический дисплей при кодировании 8 битами равно $256 \cdot (V/256)^{\gamma}$, где γ = 2,0 – 2,6 задается отдельно для каждого цвета и конструкции дисплея.

Счетчик горизонтальной развертки с частотой F_c перебирает адреса пикселов в строке изображения. Считанная информация о яркости пиксела в цифро-аналоговом преобразователе (ЦАП) превращается в сигналы яркости трех цветов. Также по состоянию этого счетчика формируются СИ горизонтальной развертки и импульсы счета счетчика вершины тикальнои развертки. Последний адресует строки изображения, выводятся пиксел за пикселом через ЦАП. Он генерирует СИ вертикальной развертки.

Буфер кадров представляет собой двухпортовое скоростное ОЗУ и содержит несколько кадров изображения, сохраненные в соответствующих

страницах. Динамическое изменение этих страниц обеспечивает непрерывное отображение сложных мультимедийных сцен на экране дисплея. Поэтому объем буфера может составлять сотни мегабайт. Независимо от отражения кадров на дисплее, к нему обеспечивается быстрый доступ от ЦПЭ через графический интерфейс. Новые контроллеры дисплея имеют графический интерфейс на основе PCIe с высокой пропускной способностью, благодаря использованию до 16 трасс (lanes). Параллельная пересылка больших массивов графических данных поддерживается механизмом ПДП.

Благодаря тому, что системная шина современных компьютеров имеет высокую пропускную способность, в некоторых моделях компьютеров большая часть буфера кадров реализована как часть основного ОЗУ. Это облегчает работу драйвера дисплея, которому становится ненужным организовывать пересылки между ОЗУ и контроллером дисплея.

Контроллер дисплея может работать в текстовом режиме, например, во время начальной загрузки ОС. При этом отдельная текстовая страница буфера содержит строки текста в кодировке ASCII, как правило, по 80 символов в строке. Слово одного символа вмещает байт кода буквы и байт информации о его цвете, цвете фона, наличии курсора в положении данной буквы. В отдельной области памяти (ПЗУ и ОЗУ) сохраняется прошивка генератора изображений символов, в которой по адресу символа в таблице ASCII сохраняется битовое изображение этого символа с учетом межстрочного интервала, например, размером 16·8.

Функционирование генератора изображений символов состоит в том, что во время отображения страницы на экране вместо слов из буфера кадров подставляются биты изображения выбранной буквы из ПЗУ или ОЗУ. ОЗУ заполняется при загрузке ОС в зависимости от заданного номера таблицы ASCII, например, с кириллическим видом букв.

Современные контроллеры дисплея имеют сложное строение. Для ускорения мультимедийных приложений, а именно для ускорения выполнения стандартизированных графических функций из библиотек, таких как DirectX, DivX, в их состав входят параллельные спецпроцессоры, структура которых будет рассмотрена ниже. Также буфер кадров дополнен ОЗУ большого объема (несколько сотен мегабайт) для хранения промежуточных данных, сцен и текстур, необходимых для быстрого выполнения этих библиотечных функций.

2.5. Структура однопроцессорного компьютера

Соединение ЦПЭ и окружения компьютера через систему интерфейсов называют его структурой. Чтобы рассмотреть такое подключение, целесообразно рассмотреть общую структуру типичного однопроцессорного компьютера, которая показана на рис. 2.9. ЦПЭ связан с буфером быстрых периферийных устройств (ББПУ) через скоростную локальную шину, которая традиционно называется FSB (Front Side Bus). Шина FSB – это полнорозрядная шина, включает в себя шину адреса и двунаправленный шину данных, разрядность которой, как правило, 64.

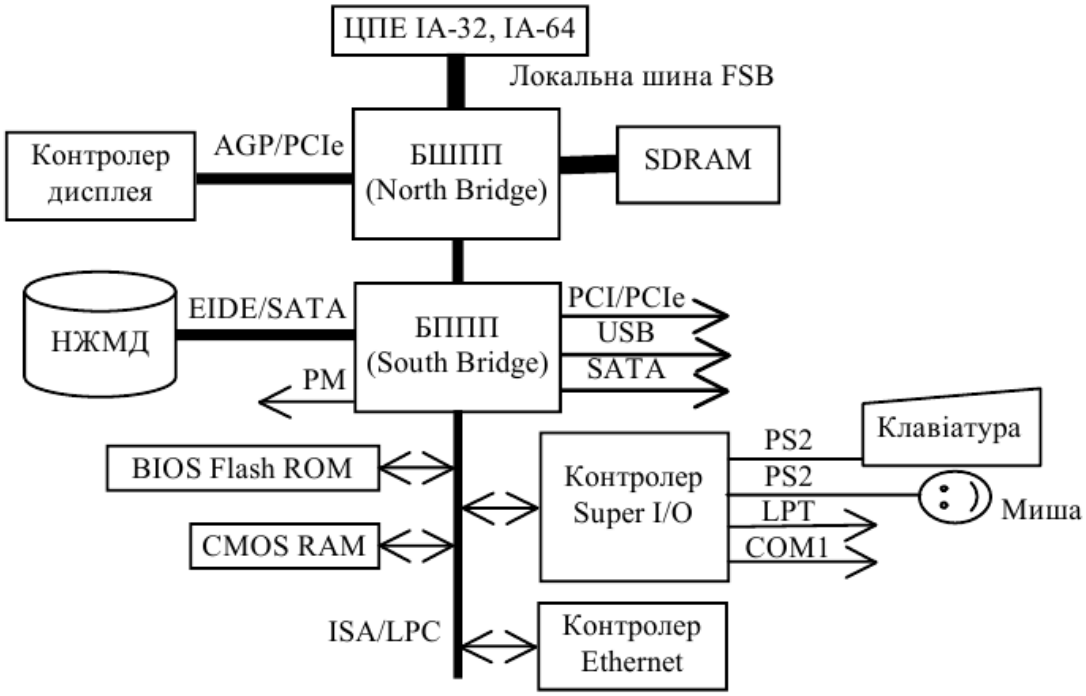


Рис. 2.9. Типичная структура однопроцессорного компьютера

ББПУ из-за своего положение на структурной схеме получил название "северный мост" (North Bridge). ББПУ связывает ЦПЭ с ОЗУ и контроллером дисплея, которые требуют наибольших потоков информации, а также с буфером медленных периферийных устройств (БМПУ). Тактовая частота F_T ББПУ определяет быстродействие всего компьютера. Для повышения быстродействия шины FSB в тактовом периоде выделено четыре фазы, во время которых выполняются пересылки данных. Таким образом, шина FSB имеет пропускную способность $4 \cdot 8 \cdot F_T$ байт/с. Собственная тактовая частота ЦПЭ также получается умножением F_T на коэффициент, который кратный 0,5. Так, например, при F_T = 400 МГц пропускная способность FSB – 12800 Мбайт / с, а тактовая частота ЦПЭ при коэффициенте умножения 9 равна 3,6 ГГц.

БМПУ или "южный мост" (South Bridge) имеет в своем составе несколько стандартных интерфейсов, таких как PCI, PCIe, USB, SATA, EIDE, через которые компьютер соединяется с НЖМД и рядом периферийных устройств. Отдельным интерфейсом выступает шина LPC (Low Pin Count bus), которая предназначена для подключение таких медленных устройств, как ПЗУ с программами базовой системы ввода-вывода (BIOS), энергонезависимое ОЗУ, в котором хранятся основные настройки структуры компьютера (Setup). В состав БМПУ входят также часы реального времени, контроллер DMA, контроллер прерываний, а также блок управления энергопитанием с выходом PM (Power Management). Последний руководит включением-выключением компьютера, установлением величин напряжений питания всех микросхем материнской платы, экономией энергопотребления компьютера, вызовом прерывание по отказом питания.

Микросхемы БШПП и БППП, как правило, разрабатываются и выпускаются вместе и служат главным набором микросхем (***Chipset***) материнской платы компьютера. Из-за их большой аппаратной сложности и высокого быстродействия они, как и ЦПЭ, рассеивают много энергии. Поэтому для их надежной работы они имеют системы охлаждения.

К шине LPC подключается также микросхема интерфейсов медленных периферийных устройств, которая называется контроллером Super I/O. Ее выходами имеются интерфейсы PS/2, LPT (шина принтера старого образца), COM1, COM2 (последовательные порты по стандарту RS232), через которые подключаются мышь, клавиатура и другие устройства. Иногда в ее состав входит звуковой мультимедийный интерфейс.

С увеличением степени интеграции микросхемы процессора и количества ее выводов все больше функций контроллера дисплея, ББПУ и БМПУ реализуются в этой микросхеме. Это приводит к уменьшению габаритов компьютера, его энергопотребления и себестоимости, увеличению быстродействия и надежности.

3. Параллелизм в однопроцессорных компьютерах

3.1. Конвейерная и векторная обработка информации

Основные виды параллелизма в компьютерах

Выше рассматривались архитектуры неймановская процессоров, основной чертой которых является ***атомарность*** выполнения всех действий. Это означает, что все выполнения команд, микрокоманд являются неделимыми и реализуются строго последовательно. Неделимость команды означает, что нельзя затормозить ее выполнения на определенном этапе и выполнить во время этого какие-то действия с элементами памяти компьютера. Так, следующая команда не начинает выполняться, прежде чем закончится выполнение предыдущей команды, данное читается с ОЗУ строго после того, как оно будет туда записано предыдущей командой. Несколько команд не могут быть выполнены одновременно в нескольких АЛУ, то есть параллельно. Поэтому граница быстродействия неймановская процессоров определяется периодом выполнения команды, который ограничен суммой задержек дешифратора команд, АЛУ, чтение ОЗУ программ, чтения-записи ОЗУ данных.

В данном разделе рассмотрены архитектуры, в которых сделан отход от атомарности действий, что позволило ускорить вычисления, благодаря их распараллеливанию. Прежде всего, вспомним некоторые принципы построения компьютеров.

Принцип параллельной обработки информации. Для достижения высокой производительности и (или) надежности компьютеров независимые этапы вычисления и управления распределяются между несколькими операционными и управляющими автоматами (или ЦПЭ), связанными некоторой системой коммутации.

Принцип конвейерной обработки информации означает, что сложная операция разделяется на последовательность нескольких последовательно выполняемых ступеней (микроопераций, этапов) таким образом, чтобы они могли выполняться параллельно для потока таких операций.

Добавим еще несколько определений: ***Параллелизм*** вычислений – это возможность одновременного выполнения независимых арифметико-логических или служебных операций. Можно выделить три основные формы параллелизма:

– естественный или векторный параллелизм;

– параллелизм независимых ветвей;

– параллелизм смежных операций или скалярный параллелизм.

Суть ***параллелизма независимых ветвей*** заключается в том, что в программе решения большой задачи могут быть выделены независимые программные части – ветви, которые можно обрабатывать параллельно.

Под ***естественным параллелизмом*** понимают, прежде всего, возможность обработки независимых данных по одному и тому же алгоритму. Это, например, возможность одновременной обработки элементов векторов данных (***векторный параллелизм***), выполнение ряда одинаковых или однородных задач для различных наборов исходных данных.

Программа для неймановского процессора представляет собой список последовательно выполняемых операций. ***Скалярный параллелизм*** означает, что подмножества этих операций можно выполнять параллельно, если между ними нет зависимости.

Акт синхронизации – это момент, когда одна порция вычислений передает управления другой порции, например, момент, когда одна команда передает управление другой, один вычислительный процесс запускает другой, одна подпрограмма передает свои результаты следующей и запускает ее на выполнение.

Зернистость вычислений – это средний промежуток времени между актами синхронизации параллельных вычислений или период синхронизации.

Относительно скалярного параллелизма часто используют термин ***мелкозернистый параллелизм*** (период синхронизации – 1 – 100 тактов). Этим он отличается от ***средне-*** (примерно 100 – 10^5 тактов) и ***крупнозернистого параллелизма*** (более 10^5 тактов), к которым чаще всего относят векторный параллелизм и параллелизм независимых ветвей.

Каждый вид архитектуры компьютера может эффективно выполнять только те алгоритмы, которые имеют присущую зернистость. Далее рассмотрим архитектуру конвейерных RISC-процессоров, которая реализует мелкозернистый параллелизм на уровне микроопераций.

Конвейер команд RISC-процессора

Основным моментом для развития конвейерных компьютеров было обоснование академиком С.А.Лебедевым в 1956 г. метода, который был назван "принципом водопровода" (позже он стал называться принципом конвейерной обработки). Прежде всего, был реализован конвейер команд, на основе которого практически одновременно были построены советский компьютер БЭСМ-6 (1957-1966 гг.) и английская машина ATLAS (1957-1963 гг.). Конвейер команд позволил получить в компьютере БЭСМ-6 быстродействие в 1 млн оп/с. Далее конвейеры команд совершенствовались и становились необходимым элементом всех быстродействующих компьютеров, таких как компьютеры IBM 360/370 и ЕС10хх.

Начиная с середины 80-х годов прошлого века, все микропроцессоры стали строиться по архитектуре RISC - микропроцессоров, характерным признаком которых является конвейер команд.

Выше рассматривались архитектуры RISC-процессоров с точки зрения их системы команд. Напомним, что один из принципов построения этих процессоров является выполнение одной команды за 1 такт. Именно благодаря конвейеризации выполнение команд, этот принцип становится возможным.

Для конвейерного выполнения команд выполнения конкретной команды разделяется на последовательность отдельных фаз (ступеней). Разделение на фазы реализуется в различных архитектурах по-разному. Но обобщенный разделение выглядит следующим образом:

– Выборка команды (ВК). Загрузка команды из памяти в регистр команды ядра процессора.

– Дешифрация команды (ДК). Благодаря простоте команды RISC-процессора, она происходит очень быстро.

– Выборка операндов (ВО). Благодаря тому, что операнды RISC-процессора чаще всего находятся в регистровой памяти, ПО происходит очень быстро.

– Операция (ОП). Собственно вычисления команды.

– Запись результатов (ЗР) обратно в регистр или другую память.

Если выполняется последовательность команд без условных переходов, то указанные фазы соседних команд можно выполнять параллельно в ступенях конвейера команд, как на рис.1.9.

Благодаря тому, что формат команды простой, ее длина известна (32 бита), выборка команды оказывается простой – это чтение слова из памяти без необходимости определять длину команды, как в случае CISC-процессора. Декодирование команды также упрощенное, так как количество форматов – минимальное, поля кода операции и адресов операндов всегда находятся в одном месте команды.

Команды разделены на множества команд обращения к ОЗУ и арифметические команды. Это упрощает фазу выборки операндов. В арифметических командах выборка операндов из регистровой памяти выполняется почти мгновенно с прямой адресацией. Благодаря тому, что все фазы имеют почти одинаковую и небольшую сложность, их можно выполнять в конвейере команд.

Например, такой фрагмент программы

```
ADD R1, R2, R3; R1 = R2 + R3 - команда 1
ADD R4, R5, R6; - команда 2
ADD R7, R8, R9; - команда 3
ADD R10, R11, R12; - команда 4
```

выполняется следующим образом. Как только команда 4 выбрана, команда 3 дешифруется и выбираются для нее операнды, команда 2 выполняет сложение, результат команды 1 записывается в регистр.

Хотя выполнение одной команды продолжается четыре такта, средняя производительность процессора приближается к одной команды за такт (при непрерывном потоке независимых команд).

В предыдущем примере все команды независимы. Более типичным примером является следующий, где соседние команды зависят по данным.

```
ADD R1, R2, R3; R1 = R2 + R3 - команда 1
ADD R4, R1, R6; - использует R1, записывает R4
ADD R7, R4, R9; - использует R4, записывает R7
```

В этой программе каждая команда нуждается в результате предыдущей команды. Согласно фазам конвейера команд, такое невозможно, так как фазы чтения и записи соседних команд не стыкуются (рис. 1.9). Но эта проблема решается, благодаря способу перенаправления результата (***forwarding***). Во время этого результат команды сразу направляется как операнд на следующую команду, как на рис. 1.10.

Команды загрузки данных и сохранения их в ОЗУ отличаются от команд АЛУ тем, что, на самом деле, выборка данных выполняется не за один, а за несколько тактов из-за низкой быстродействие ОЗУ и интерфейса связи.

Например, команды чтения и сложения

```
L R3, OPER
ADD R4, R3, R2
```

не могут быть выполнены в соседних тактах, так как операнд OPER будет считываться больше, чем за один такт, например, 2 (если ОЗУ – кэш-память).

Но если между ними вставить пустую команду:

```
L R3, OPER
NOP
ADD R4, R3, R2
```

то это увеличит задержку между командами на один такт, что гарантирует своевременное появление операнда в регистре, который адресуется в арифметической команде. Эти пустые команды вставляются компилятором автоматически.

Часто компилятор переставляет команды так, чтобы минимизировать число дополнительных пустых команд, например:

Начальная программа	После оптимизации
L R2,OP1	L R2,OP1
L R3,OP2	L R3,OP2
NOP	L R5,OP3
ADD R4,R2,R3	L R6,OP4
L R5,OP3	ADD R4,R2,R3
L R6,OP4	ADD R7,R5,R6
NOP	
ADD R7,R5,R6	

После такой перестановки команд смысл программы остается тем самым, но задержка между чтением операндов из ОЗУ и использованием их в арифметических командах становится допустимой, и поэтому вставка пустых команд не нужна.

Реальные программы содержат большое количество (до 25%) команд условных переходов и вызова подпрограмм. Выполнение таких команд имеет два аспекты. Во-первых, в том такте, когда процессор распознал такую команду, команды, которые являются следующими за ней (по следующим возрастающим адресам), уже попали в конвейер и начали обрабатываться. Во-вторых, команда, которая действительно должна быть выполнена по команде перехода, не может поступить в конвейер сразу в следующем такте. Для этого конвейер нужно останавливать и очищать от команд, которые не следует выполнять.

Самый простой способ ускорить выполнение таких команд – это способ задержанного перехода. При этом когда команда перехода распознана, то дается возможность исполниться одной или нескольким командам, которые последовательно следуют за ней, перед тем, как будет сделан сам переход.

Пусть выполняется следующая последовательность команд:

```
L R9, # 4
JMP M1
ADD R9, # 1
. . .
M1: L R3, R9
```

Когда команда JMP выполняется, команда ADD будет уже в конвейере и ей предоставляется право выполняться. Когда будет сделано собственно переход

на метку M1 и выполнена команда L, в регистр R3 загружается число 5, а не 4, как было бы при выполнении программы в обычном процессоре.

Место после команды перехода называется промежутком задержки перехода (branch delay slot). Оно заполняется компилятором таким образом, чтобы не ставить пустых команд. Например, при передаче параметров в подпрограмму, одну из команд передачи можно задать после команды вызова:

Начальная программа	После оптимизации
L R2,OP1	L R2,OP1
L R3,OP2	CALL SUBPR
CALL SUBPR	L R3,OP2
NOP	

Так же можно ставить первую команду вызываемой подпрограммы, соответственно увеличив адрес начала подпрограммы, чтобы «обойти» эту команду во время вызова.

Программирование задержанного перехода – это необычная вещь для программиста. Но она вполне ясно и корректно выполняется компилятором. Так, компилятор для архитектуры MIPS способен так переставить команды в программе, чтобы заполнить от 70 до 90% промежутков задержанного перехода.

Конвейер команд CISC-процессора

Современные CISC-процессоры также имеют конвейер команд, благодаря которому в них достигается высокая тактовая частота. Рассмотрим для примера эволюцию конвейера команд в процессорах Intel Pentium, ..., Pentium-4, которые имеют синонимы P5, P6, P7.

Структура, точнее, ***микроархитектура*** процессора P5 имела пятиступенчатый конвейер команд со ступенями: выборка команды, две ступени дешифратора, выполнение операции и запись результата. Это позволило иметь тактовую частоту около 100 МГц.

В микроархитектуре P6 (PentiumPro, Pentium II, Pentium III) конвейер команд выполняет как RISC, так и CISC команды, хотя последние – менее эффективно.

Она имеет увеличенное до 12 число ступеней. Благодаря конвейеризации и улучшению технологии СБИС, в процессорах с этой микроархитектурой удалось довести тактовую частоту до 1,4 ГГц.

В микроархитектуре P7 при выполнении CISC-команд они заменяются на цепочки RISC-команд, которые выполняются в конвейерном режиме (см. далее «Кэш трасс»). При этом на считывание и преобразование CISC-команд на RISC-

команды отведено 8 ступеней, а на выполнение RISC-команд – 20 ступеней. Благодаря этому, в процессоре Pentium-4 в 2005 году тактовая частота достигла 3,8 ГГц. Интересная деталь микроархитектуры – две с 28 ступеней конвейера вообще ничего не выполняют. Они предназначены только для выравнивания задержки сигналов при их передаче из одной части кристалла в другую. Это вызвано тем, что площадь кристалла CISC-процессора слишком велика, чтобы передавать сигналы через нее без промежуточной буферизации в конвейерных регистрах.

Другие недостатки, которые выявлены в микроархитектуре P7 – это большие задержки на очистку длинного конвейера при неудачной попытке предсказать направление перехода, что приводит к существенному уменьшению производительности (см. "Механизмы предсказания условных переходов") и задержка конвейера при выполнении чтения ячейки ОЗУ после записи в нее, который состоялся недавно (см. "Согласованность последовательного доступа памяти").

Итак, тактовая частота процессора не является надежным показателем производительности процессора с определенной архитектурой. Количество ступеней конвейера команд должно подбираться так, чтобы микроархитектура обеспечивала максимальную производительность (например, при решении тестовых задач) при максимальной тактовой частоте, которая приемлема для нее. В отличие от CISC-процессоров, RISC-процессоры имеют меньшую площадь кристалла и более коротким конвейером команд, благодаря чему они имеют как высокую тактовую частоту, так и более высокую общую производительность, а также меньшее энергопотребление.

Программная конвейеризация

При аппаратной конвейеризации выполнение команды разбивается на ступени, такие как выборка, исполнения и запись. Только после заполнения конвейера процессор будет выполнять по одной команде за такт. При программной конвейеризации операции одной итерации программного цикла делятся на S степеней (такие как выборка из ОЗУ, выполнение и запись в ОЗУ) и в цикле выполняется первая ступень с i-й итерации, вторая ступень с i-1-й итерации и т.д. Команды разгона программного конвейера должны быть сформированы в цикл, чтобы инициализировать первые S-1 итераций, а команды торможения должны стоять после цикла для очистки конвейера на S-1 последних итерациях. Ниже показан пример программы сложения двух массивов до и после программной конвейеризации.

Начальная программа (цикл 11 тактов)

```
    L   R2,ARRA ;  адрес массива A
    L   R3,ARRB ;  адрес массива B
    L   R4,#(N*4)  ; длина массива
    L   R5, #4  ;  инкремент адреса
M1: L   R6, [R2]  ;загрузка A(i)
    L   R7, [R3]  ;загрузка B(i)
    NOP
    NOP
    ADD R6,R6,R7 ;  A(i)+B(i)
    S   R6, [R2]   ; сохранение суммы
    ADD R2, R2,R5; модифікація адреси
    ADD R3, R3,R5
    SUB R4,R4,R5 ; конец цикла?
    JNE M1  ; переход в начало цикла
    NOP
```

После оптимизации (цикл 8 тактов)

```
    L   R2,ARR1 ;  адрес массива 1
    L   R6, [R2]      ; загрузка A(0)
    L   R3,ARR2     ;  адрес массива 2
    L   R7, [R3]      ; загрузка B(0)
    L   R4, #(N*4)  ; длина массива
    L   R5, #4        ;  инкремент адреса
M1:  ADD R3,R3,R5
    L   R7, [R3]      ; загрузка B(i+1)
    ADD R8, R6, R7   ;A(i)+B(i)
    S   R6, [R2]      ; сохранение суммы
    ADD R2,R2,R5
    SUB R4,R4,R5
    JNE M1        ; конец цикла?
    L     R6, [R2];  загрузка A(i+1)
```

Часто программная конвейеризация используется при расчетах с плавающей запятой, когда латентная задержка конвейера очень большая (более 6-8 тактов), а также в сигнальных процессорах, в VLIW-микропроцессорах, в процессоре Itanium, в которых за один такт вы- ется большое количество действий и нужно минимизировать продолжительность итерации.

3.2. Векторно - конвейерные компьютеры

В обработке структурированных данных наиболее распространенной является векторная операция, которая имеет натуральный параллелизм. ***Вектор*** – это одномерный массив, имеющий размер, который известен на этапе компиляции. Часто вектор формируется из многомерного массива. На последовательных языках, таких как Fortran, некоторая векторная операция выражается с помощью цикла DO:

```
      DO 10 I = 1, N
10 C (I) = A (I) + B (I + 16);
```

Векторные операции (сложение, умножение, деление, сравнение и т.д.) носят универсальный характер и часто входят в параллельные языка программирования. Так, вышеприведенной векторная операция языке MATLAB выглядит более наглядно:

```
C = A (1: N) + B (16: N + 16);
```

Области использования векторных операций над массивами большие: цифровая обработка сигналов и изображений, решение задач линейной алгеб-

ры, моделирование физических сред, метеорология, оптимизация, автоматизированного ция проектирования и т.д.

В конце 60-х гг. внедрен арифметический конвейер, который стал центральным устройством в суперкомпьютерах CRAY-1 (1976г.).

Арифметический конвейер основан на разделении цикла выполнения операции на ряд этапов, для каждого из которых выделен отдельная аппаратная ступень. Одновременно на разных этапах обработки в конвейере находятся несколько данных. Такой арифметический конвейер позволяет эффективно обрабатывать векторы чисел, так как при обработке отдельных чисел его ступени будут простаивать, как это будет показано ниже.

Суперкомпьютеры решают, как правило, многочисленные задачи, используя операции с плавающей запятой двойной точности. Такие операции можно разделить на несколько десятков этапов, каждый из которых выполняется за минимизированный промежуток времени, не превышающий период тактового сигнала T_C. Благодаря минимальному периоду тактового сигнала (T_C = 0,2 – 2 нс), арифметический конвейер позволил получить быстродействие в сотни миллионов операций в секунду с плавающей запятой, то есть в сотни ***MFlops*** (Mega Floating point operations per second).

Подавляющее большинство суперкомпьютеров в 70-80-х и в начале 90-х годов составляли векторно-конвейерные компьютеры. Но потом они не выдержали конкуренции с многопроцессорными компьютерами.

АЛУ с плавающей запятой современных микропроцессоров представляют собой арифметический конвейер. Сигнальные микропроцессоры с плавающей запятой, а также графические сопроцессоры можно отнести к разряду векторно-конвейерных компьютеров. Поэтому следует внимательно рассмотреть свойства архитектуры векторно-конвейерных компьютеров.

Векторно-конвейерные компьютеры имеют особую архитектуру, которая адаптирована к выполнению векторных операций. Признаками такой архитектуры являются:

– несколько арифметических конвейеров для выполнения действий сложения, умножение, деление с плавающей запятой;

– векторная память, которая является аналогом регистровой памяти, в которой в качестве регистра служит память объемом в один вектор (как правило, длиной 64);

– ОЗУ с быстрым групповым доступом, как правило, расслоенное; N блоков такого ОЗУ позволяют выполнять доступ к N соседним ячейкам одного вектора;

– аппаратные устройства для генерации адресов элементов векторов (генераторы адресов);

– возможность соединения выхода одного конвейера со входом другого для выполнения групповой операции или нескольких соседних арифметических команд над одним данным без записи промежуточных результатов в память (***сцепление конвейеров***).

Типичной командой групповой операции конвейерной ЭВМ является команда SAXPY (Single precision A multiplied by X Plus Y), которая выполняет действия A * X + Y над векторами с одинарной точностью. При ее выполнении сцепляются конвейеры умножения и сложения.

Векторно-конвейерные компьютеры удовлетворяют принципу конвейерной обработки. Этот принцип заключается в том, что вычисления сложной функции делится на ряд последовательных этапов, выполняемых в вычислительных ресурсах, которые называются ступенями конвейера. Причем поток данных проходит последовательно эти ступени и благодаря непрерывности потока, эти данные обрабатываются параллельно, а соседние данные одновременно обрабатываются на соседних ступенях. Благодаря специализации ступеней конвейера, увеличению количества ступеней и соответственно, уменьшению задержки выполнения каждой ступени, которая меньше T_C, получается высокая пропускная способность конвейера (в MFlops) при минимизированных аппаратных затратах.

На рис. 3.1 показана временная диаграмма загрузки некоторого конвейера при выполнении векторной операции для векторов длиной *N*. По диаграмме видно, что использование векторных операций связано с определенными временными накладными расходами.

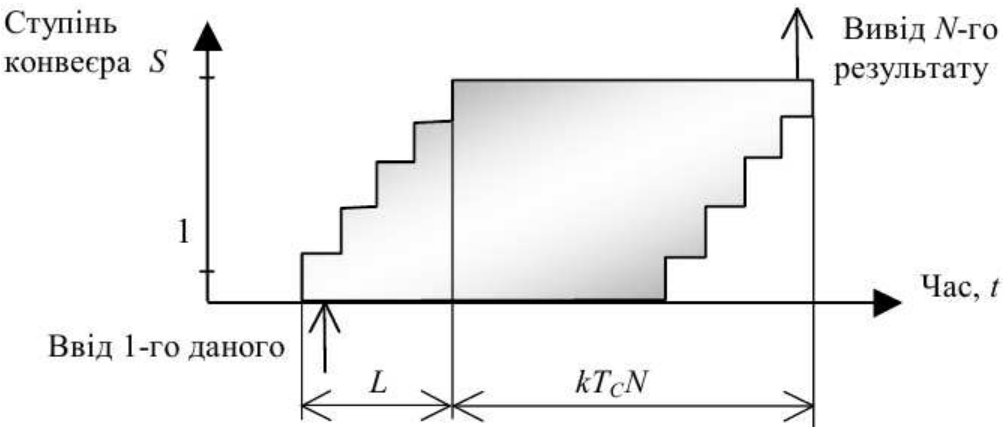


Рис. 3.1. Временная диаграмма загрузки конвейера

Согласно рис. 3.1, время выполнения векторной операции можно оценить формуле:

$$T = L + kT_C N, \qquad (*)$$

где T_C – продолжительность тактового интервала, k – среднее количество тактов в периоде выдачи результатов, L – время запуска конвейера, то есть время, необходимое для заполнения конвейера, включая время для подготовки векторных операндов. В простейшем случае его можно оценить как $L \approx SkT_C$. Как правило, L больше для компьютеров, в которых векторы перед обработкой хранятся в ОЗУ, чем для компьютеров с векторными регистрами.

Период выдачи результатов kT_C связан с продолжительностью тактового интервала или периода вычислений. Во многих компьютерах k = 1 и этот период совпадает с тактовым периодом. В высокопроизводительных компьютерах с несколькими конвейерами, например, в современных графических акселераторах, количество конвейерьеров M может достигать сотен, и тогда при максимальном их загрузке k = 1/M.

Из формулы (*) следует, что средняя продолжительность получения одного результата равна

$$T_R = kT_C + L/N.$$

Производительность конвейерного компьютера можно характеризовать числом результатов, которые он выдает за единицу времени:

$$R = \frac{1}{T_R} = \frac{N}{kT_C N + L} \qquad (**)$$

Если L = 0 или когда $N \to \infty$, то из этой формулы следует $R_\infty = 1 / (kT_C)$. Эта величина называется ***асимптотической производительностью***. Она является максимальной производительностью при отсутствии накладных расходов на запуск конвейера. Если N = 1, получим минимальную производительность, которая будет значительно меньше производительности обычного скалярного компьютера.

Интерес представляет величина $N_{½}$, равной длине вектора, для которой достигается половина асимптотической производительности. Пусть kT_C = 1 нс, L = 40 нс. Тогда, согласно (**), $R_\infty = 10^9$ Flops и

$$10^9/2 = \frac{N_{½}}{10^{-9}N_{½} + 40\cdot10^{-9}}.$$

Решив это уравнение, получим $N_{½}$ = 20. Считая, что $L \approx SkT_C$, то есть число ступеней конвейера для этого примера S = 40, можно сделать следующий вывод. Если длина вектора меньше половины числа ступеней конвейера, производительность конвейерного компьютера будет меньше половины асимптотической производительности. Этот вывод следует принимать во внимание при программировании обработки векторов данных в современных процессорах.

3.3. Суперскалярные процессоры

Суперскалярная архитектура

Неймановская архитектура почти всегда была в фарватере русла развития вычислительной техники. Появление каждого нового поколения компьютеров можно связать с очередной попыткой расширить узкое место этой архитектуры – линию связи между памятью команд и данных и ядром ЦПЭ.

Так, в микропроцессорах i486 внедрено кэш-ОЗУ. Это позволило ядру ЦПЭ выполнять одновременное чтение 16 байт команд, запись и чтение 4 байт данных из кэш-ОЗУ. При этом поток информации между ядром и кэш-ОЗУ стал в пять раз больше, чем поток из внешней памяти.

Новый этап развития неймановской архитектуры начался с появлением суперскалярніх процессоров. Основным признаком ***суперскалярной архитектуры*** является одновременная выборка нескольких команд с кэш-ОЗУ, их дешифрация и разделение на несколько потоков команд, которые выполняются в отдельных обрабатывающих блоках. Условия такого параллельного выполнения команд: 1) соседние команды должны быть независимыми по данным, адресам, управлению; 2) для команд должны быть свободны аппаратные ресурсы для их выполнения.

Итак, в суперскалярном процессоре могут одновременно выполняться несколько команд на нескольких обрабатывающих блоках. Иначе говоря, в нем аппаратно реализовано параллелизм на уровне команд.

Другим признаком суперскалярной архитектуры является разделение кэш-ОЗУ на ОЗУ команд и ОЗУ данных, параллельный доступ к которым также расширяет узкое место неймановской архитектуры. Так, в суперскалярном процессоре Pentium за счет этого разделения и увеличения разрядности шины кэш-ОЗУ, поток информации между ядром ЦПЭ и кэш-ОЗУ увеличен втрое по сравнению с процессором i486.

Третий признак суперскалярных процессоров – это то, что у них есть достаточное количество операционных ресурсов, а также аппаратура для распознавания соседних команд, которые можно выполнять одновременно.

К процессорам с суперскалярной архитектурой, кроме IA-32, IA-64 и их клонов, относятся архитектуры семейств микропроцессоров Power PC, SuperSPARC, PA-RISK, IBM Power, MIPS, ARM и др. Сейчас эти процессоры имеют не более шести потоков команд.

Рассмотрим для примера архитектуру процессора SuperSPARC. Специалистыи фирмы SUN Microsystems определили частоту появления команд в типичных задачах, которые решаются на системах SPARC (см. табл. 3.1).

Таблица 3.1. Частота выполнения команд в различных приложениях

Тип команды	Приложение с целочисленными вычислениями	Приложение с плавающей запятой
Арифметическая с фиксированной запятой	50%	25%
Арифметическая с плавающей запятой	0	30%
Чтение ОЗУ	17%	25%
Запись в ОЗУ	8%	15%
Условный переход	25%	5%

Анализ статистики выполнения команд показывает целесообразность разделения потока команд на три части: команды арифметики с фиксированной запятой, с плавающей запятой и команды обращения к ОЗУ, что и было реализовано в процессоре SuperSPARC. При этом с кэш-ОЗУ одновременно выбираются четыре соседние команды, из которых выделяетя группа из не более чем трех выполняемых команд. Эти команды удовлетворяют следующим условиям: должно быть не более двух целочисленных результата, одно обращение к ОЗУ данных, одна операция с плавающей запятой и одна команда управления, которая должна быть последней в группе. Измерение производительности процессора SuperSPARC показали, что, в среднем, в нем одновременно выполняются от 1,4 до 2 команд при пиковом значении – 3 команды.

Развитие суперскалярной архитектуры в направлении увеличения числа потоков команд связано с ростом сложности логической схемы, которая распознает независимые команды в потоке команд. С другой стороны, возможности параллелизма на уровне команд для произвольных практических задач ограничены и поэтому нецелесообразно увеличивать число потоков команд более шести – восьми.

Параллелизм суперскалярных процессоров ограничен:

– аппаратными ресурсами – сейчас можно реализовать до десяти потоков команд;

– уровнем имеющегося скалярного параллелизма команд в программе, в существующих программах средний уровень параллелизма – 2–3 команды;

– сложностью схем распознавания параллелизма;

– независимостью команд по данным, управлению.

Повысить производительность суперскалярного процессора можно следующими путями:

– увеличить суперскалярный параллелизм в программе с помощью приемов программирования и компиляции, чтобы увеличить параллелизм на уровне команд;

– автоматически выполнять увеличение параллелизма при дешифрации потока команд;

– указывать параллелизм прямо в скомпилированной программе – чтобы упростить распознавание параллелизма аппаратурой;

- выполнять два или более независимых программных вычислительных процесса, что вдвое или более раз увеличит количество независимых команд в потоке (multithreading);

– уменьшить простои конвейера команд, например, предугадать условные преходы;

– уменьшить глубину конвейера, благодаря чему быстрее заполняется конвейер после переходов, короче циклы зависимости по данным (получение результата D, запись D в память, считывание D из памяти).

VLIW-архитектура

Другим путем внедрения параллелизма на уровне команд является применение ния архитектуры процессора с длинной командой – **VLIW**-архитектуры (Very Large Instruction Word - VLIW). В длинной команде VLIW-процессора стоят несколько независимых полей управления различными операционными блоками.

Если принять, что суперскалярный процессор в каждом такте выполняет строго одинаковое число команд, пусть даже пустых, то получим архитектуру VLIW-процессора.

По сравнению с суперскалярным процессором, в VLIW-процессоре поток команд из ОЗУ программ увеличен, зато отпадает необходимость в схеме распознавания независимых команд. При этом ответственность за загруженность процессора лежит на компиляторе и программисте.

Некоторые сигнальные процессоры можно отнести к VLIW-процессорам, например, SHARC. Процессор Itanium компании Intel с архитектурой EPIC (explicit parallel instruction computer) является VLIW-процессором с длиной слова команды 16 байт, имеет три команды в слове.

3.4. Повышение быстродействия суперскалярных процессоров

Механизмы предсказания условных переходов

При выполнении команд условного перехода следующей командой является команда, которая находится в конвейере команд или команда, адрес которой указан в команде перехода. В последнем случае конвейер команд необходимо очистить. Выполнение ветвей разветвлений имеет неодинаковую вероятность. Если в конвейер команд загружать команды из более вероятной ветви, то необходимость очистки конвейера возникать значительно реже, а производительность процессора – будет выше. Для реализации этого в суперскалярных процессорах задействованы механизмы предсказания условных переходов.

Наиболее распространенный переход, требующий предсказания – это переход по команде проверки окончания цикла. При этом вероятность возврата к началу цикла приближается к единице. Такие команды можно выявлять на стадии компиляции. В первых RISC-процессорах, в которых было задействовано предсказание перехода (IBM Power), в командах перехода компилятором устанавливался флаг направлении вероятного перехода.

В современных процессорах предсказание перехода выполняется специальной аппаратурой во время выполнения программы согласно некоторой стратегии.

Для предсказания адреса перехода используют ***таблицу адресов перехода*** (Branch Target Buffer – **BTB**). Эта таблица функционирует как кэш-ОЗУ и сохраняет адреса команд, на которые раньше выполнялись переходы. Например, в процессоре Pentium-4 таблица BTB имеет размер 4096 элементов.

Для предсказания направления условного перехода используется механизм, основанный на изучении поведения переходов в программе в процессе ее выполнения. Этот механизм учитывает как локальное поведение команды перехода (например, «как правило, переходит», «как правило, не переходит»), так и глобальные закономерности («меняется по определенному закону» и т.д.). История поведения команды перехода записывается в специальныой ***таблице истории переходов*** (Branch History Table – **BHT**).

Современные механизмы предсказания переходов обеспечивают правильное предсказания более чем в 90% случаев. В совершенных системах предсказания переходов используют:

- сочетание локального и глобального механизмов для предсказания обычных команд перехода с учетом истории поведения;

– статическое предсказание для команды, выполняемой в первый раз, основанный на эмпирических закономерностях. Например, «переход назад» предусматривается как сделанный, так как может начать переход по цикла, а «переход вперед» – как несделанный;

- предсказание коротких циклов, которое распознает переходы в конце короткого цикла и вычисляет число итераций цикла, что позволяет правильно предсказать момент выхода из цикла;

- предсказание косвенных переходов, то есть переходов по адресу в ячейке памяти, с определением целевого адреса для различных исполнений команды перехода даже с учетом чередования этих адресов;

- предсказание целевых адресов для команды выхода из подпрограммы, которое использует небольшой аппаратный стек для хранения адресов возврата (Return Address Stack) для отработки команд Call – Return.

Блок предсказания переходов работает одновременно с дешифратором команд и независимо от него. Благодаря эффективной реализации предсказания адреса перехода, в процессорах Pentium-III, Pentium-4 и K8 при правильном предсказании теряется лишь один такт. Это означает, что минимальные временные накладные расходы, например, на выполнение итерации цикла или на один переход в цепи переходов, составляет два такта. По сути, предсказание переходов в такой цепи команд выполняется в собственном цикле, который состоит из двух стадий: предсказание и считывания строки команд из кэш-ОЗУ и декодирование команд из этой строки.

Когда команда перехода попадает в функциональное устройство для решения о ее выполнении, то будет предсказано, произойдет этот переход или нет. В момент завершения выполнения этой команды при неправильном предсказании перехода выполнение всех последующих команд отменяется, и начинается считывания команд из кэш-ОЗУ по правильным адресам.

Такую процедуру называют очисткой конвейера, а время (в тактах) на выполнение команды неправильно предсказанного перехода с момента ее считывания с кэш-ОЗУ называют ***длиной конвейера непредсказанного перехода***. Это время характеризует чистую потерю в идеальных условиях, когда команда проходит через все ступени конвейера и нигде не задерживается по внешним причинам (прерывания, непопадание в кэш-ОЗУ и т.д.). В реальных условиях потеря на неправильно предусмотренный переход может быть больше. Длина конвейера непредсказанного перехода принимает значения от 11 тактов для Pentium-III до 30 тактов для Pentium-4E.

Кэш трасс и раскрутка циклов

Кэш-ОЗУ команд в процессорах типа Pentium-4 сильно отличается от кэш-ОЗУ команд (I-кэш) в процессорах-предшественниках. В нем многие команды из внешнего ОЗУ перед записью в кэш-ОЗУ перекодируются в микрокоманды (МК), причем в кэш-ОЗУ размещаются не отдельные команды а трассы последовательностей команд и МК. Поэтому такое кэш-ОЗУ в процессоре Pentium-4 называется ***кэш трасс*** (Trace-cache, ***Т-кэш***).

Трасса – это последовательность команд или микрокоманд, которая выполняет те же действия, что и заданный участок программы. Трассы формируются в соответствии с динамическим порядком выполнения дешифрированных команд. В момент дешифрации выполняется первичное предсказание перехода, и если предусматривается сделанный переход, то целевая команда помещается в трассу непосредственно вместо команды перехода. Трасса может содержать множество таких «встроенных» переходов.

При формировании трассы в нее записываются команды, выполняемые в коротком программном цикле без команд, которые управляют его выполнением. Такой процесс называется ***раскруткой цикла*** (***loop unrolling***). Если команды цикла не помещаются в одну трассу, то они разделяются среди нескольких трасс или строится цикл с большим количеством команд в своем ядре, но с меньшим числом итераций.

Т-кэш имеет ряд преимуществ:

– размещение кэш-ОЗУ после дешифратора команд и хранение в нем сформированных МК позволяет обойти сложное параллельное декодирование нескольких команд в условиях повышенной тактовой частоты, а также сократить временные затраты (уменьшить число ступеней конвейера) на обработку МК после ее выборки из кэш-ОЗУ;

– динамическая замена CISC-команд на эквивалентную последовательность RISC-команд или МК позволяет использовать положительные свойства RISC-команд – выполнение одной команды за один такт, увеличение тактовой частоты;

– «встраивания» предусмотренных переходов в трассу позволяет снизить потери на их выполнение – теперь в одном такте может быть выполнена операция, предшествующая переходу, сам переход и операция за ним. В классических процессорах такая последовательность выполняется не менее, чем за три такта;

– раскрутка цикла в трассу уменьшает временные затраты на условные переходы и слежения за концом цикла;

– упрощение адресации команд – ассоциативный адрес предоставляется только первому блоку МК, а не каждой строке кэш-ОЗУ;

– возможность распознавания и выполнения ***спекулятивных*** (***speculative***) вычислений – предварительных вычислений, которые не будут выполняться в операционных блоках. Это, например, выявление и невыполнение команды сложения с нулевым операндом, вычисления конечного значения счетчика циклов.

Т-кэш в Pentium-4 состоит из блоков размером в 6 ячеек. Одна МК занимает одну ячейку. Темп последовательного чтения с Т-кэш составляет один блок за два такта, или три МК за такт обработки. Объем Т-кэш составляет 12К ячеек, или 2 048 блоков. Максимальная длина трассы составляет 64 блока (384 ячейки для МК).

Итак, Т-кэш увеличивает скалярный параллелизм программы непосредственно перед выполненных ее команд.

Принципиальной особенностью Т-кэш является ***отсутствие прямой связи между адресом команды и местом соответствующей МК*** (или последовательности МК) в кэш-ОЗУ в связи с тем, что команда архитектуры IA-32 переменной длины превращается в одну или несколько МК фиксированной длины. Кроме того, хранение МК в виде трасс в порядке выполнения операций нарушает монотонность и непрерывность соответствия между адресами команды и положением МК в кэш-ОЗУ. И наконец, возможность раскрутки циклов в Т-кэш (то есть размещение в ней нескольких итераций цикла) и построение перекрещивающихся трасс, нарушает взаимную однозначность между командами программы и МК – теперь одна команда может отробазиться в несколько МК, соответствующих различным итерациям цикла или в МК из нескольких трасс.

Механизм отображения адреса команды в позиции МК необходим не для каждой команды, а только для тех команд, на которые выполняется переход, то есть для первой МК в каждой трассе. Все остальные МК формируют цепочку блоков Т-кэш до конца трассы.

При сходстве схемы адресации строк в Т-кэш с аналогичной схемой в классических кэш-ОЗУ, между ними есть принципиальная разница. В классических кэш-ОЗУ ассоциативно адресуется каждый блок, в то время как в Т-кэш ассоциативно отражается только первый блок трассы. Остальные блоки трассы размещены в наборах ячеек с увеличивающимися номерами, (после набора 255 следует набор 0). Все блоки в трассе связаны в двунаправленный список – в служебной информации каждого блока указаны номера следующего и предщего блоков. В случае, если какой-то блок удаляется с Т-кэш, трасса «сокращается», то есть предыдущий блок трассы замечается как последний.

Если команда, на которую выполняется переход, отсутствует в Т-кэш, то есть в нем нет трассы с таким начальным адресом, начинается считывание

команд непосредственно с кэш-ОЗУ второго уровня со следующими их декодированием и исполнением. Одновременно выполняется формирование новой трассы и ее размещения в Т-кэш. Если встречается команда условного перехода, то трасса, начиная с этой команды, строится в соответствии с режимом работы блока предсказания перехода.

Дешифратор обрабатывает входящий поток команд со скоростью не более одной команды за такт, в зависимости от сложности формата команды и наличия у нее префиксов. Если команда не может быть преобразована в последовательность из небольшого числа МК (до четырех), она заменяется вызовом «микрокодовой подпрограммы», которая при своем исполнении генерирует МК и пересылает их на выполнение.

Есть ряд ограничений, которые не дают возможность заполнить микро-командами все шесть ячеек блока трассы и требуют формировать следующий блок. Это требование размещения всех МК, которые отражают одну команду, в одном блоке и ограничения на число МК перехода – их может быть не более двух.

При появлении команды перехода построение трассы не прерывается, а продолжается в соответствии с предусмотренным направлением перехода.

Если переход будет предсказан как сделанный, то после МК перехода в трассе будет размещена МК, на которую выполняется переход. При предсказа-нии перехода, как несделанного, формирование трассы продолжается в естественном порядке. Но если во время первого выполнения операции перехода выяснится, что переход был предсказан неправильно, формирование трассы останавливается, и она будет прервана после этой МК. Также формирование трассы останавливается, если встречается команда косвенного перехода, вызова подпрограммы или возврата из подпрограммы.

Может случиться, что правильно предусмотренный переход является переходом по циклу. В таком случае есть возможность «раскрутить» этот цикл и разместить в трассе несколько его итераций, чтобы повысить скорость выборки и выполнения команды за счет снижения накладных расходов на переход по циклу и слежения за его концом.

Вот так формируются трассы в Т-кэш. При последующем выполнении кода МК считываются с Т-кэш последовательно до завершения трассы в темпе один блок с 3 МК за 2 такта.

Если при выполнении трассы встречается операция условного перехода, то выполняется новое предсказание ее направления и следующие действия:

– если направление совпадает с тем, который был предусмотрен при построении трассы, то выполнение трассы продолжается без потери тактов;

– если направление не совпадает, а трасса по новому предсказанному адресу найдена в Т-кэш, то начинает выполняться найденная трасса;

– если направление не совпадает, а трасса по новому адресу не найдена, то начинается формирование новой трассы с одновременным выполнением порожденных МК.

При неправильном прогнозе перехода выполнения всех последующих МК останавливается, результаты их работы аннулируются, и выполняется поиск трассы по другому (правильному) адресу.

Переименование регистров

Переименование регистров – это способ удаления зависимостей по данных – запись после чтения и записи после записи, возникающих в последовательной программе между регистровыми переменными. Это уменьшает количество жестких условий предшествования и увеличивает количество команд, которые могут выполняться одновременно в суперскалярном процессоре.

При таком переименовании процессор не выполняет запись результата в указанный регистр, а делает запись в специальный буфер, регистры которого имеют адреса, которые динамично изменяются. Например, вместо

```
ADD R1, R2, R3
MUL R2, R4, R5
```

регистр R2 во второй команде переименуется, например, на R33, и запись выполняется в R33, а не в R2. Это разрешает зависимость "запись после чтения" между этими двумя командами, и они могут выполняться одновременно. В следующих командах зависимость по данным через регистр R2 должна быть восстановлена – чтобы чтение вместо R2 выполнялось с R33.

Впервые этот способ был использован в 1967 году в IBM 360/91 для ускорения выполнение операций с плавающей запятой. Этот компьютер также известен тем, что в нем были внедрены конвейеризация и динамическая выборка команд. ***Динамическая выборка*** означает, что выбранные команды не выполняются, а ждут в очереди высвобождения ресурсов и диспетчеризации, образно говоря, кладутся на полку – ***shelving***. Через большую сложность своей аппаратной реализации этот способ широко не использовался до 90-х годов.

Первые суперскалярные процессоры не задействовали этот способ. Он внедрялся по частям. Сначала как частичное переименование в IBM Power, PowerPC. Затем переименование стало использоваться во всех суперскалярных процессорах. ***Частичное переименование*** касается некоторого множества команд, например, с плавающей запятой, а ***полное*** – всех.

Буферные переименованные регистры могут быть:

– частью регистровой памяти (присоединенняа память),

– отдельным блоком регистровой памяти,

– буфером изменения порядка доступа (кольцевой буфер),

– как часть буфера динамической выборки команд.

Каждый буферный регистр может быть 1) свободным, 2) архитектурным регистром (с адресом как в архитектуре), 3) переименованным регистром без результата, 4) переименованным регистром с корректным результатом.

Сначала первые *n* регистров становятся ***регистрами архитектуры***. Затем, когда станет необходимым, дополнительный регистр выбирается из свободных регистров и становится ***переименованным регистром*** без результата. После записи в него результата, он становится регистром с корректным результатом. После того, как предыдущий архитектурный регистр высвободился, переименованный регистр переходит в состояние архитектурного. Наконец, когда архитектурный регистр долго не используется, он переходит в состояние свободного.

Буфер изменения порядка (***reorder buffer***) используется в процесорах Pentium, начиная с Pentium-IIp. При этом каждой выполняемой команде отвечает отдельный указатель на регистр буфера. Этот буфер также может выполнять функции ***буфера динамической выборки команд*** - **DRIS** (deferred scheduling register renaming instruction shelf).

Количество регистров в буфере находится как компромисс между аппаратными затратами и ускорением выполнения программ. В большинстве архитектур число регистров в 1,5–8 раз больше числа *n* архитектурных регистров.

Наибольшую сложность представляет схема определения свободных регистров. Как правило, она использует анализ таблицы отображения регистров для потока команд и определяет, какие регистры долгое время не используются.

3.5. Многопотоковая работа процессора

Программный поток

Программный поток (***thread*** – нить) – это основная программная единица, отражающая средне- и крупнозернистый параллелизм. Это как правило, относительно независимый вычислительный процесс, выполнение подпрограммы, модуля средней или большой сложности. Нескольким потокам, формирующим одну программу, ОС выделяет одни и те же ресурсы памяти. В

рамках однопроцессорной архитектуры существуют следующие возможности параллельного выполнения потоков:

– ***последовательное выполнение потоков по квантам времени*** – Time-Slice Multithreading. При этом процессор переключается между программными потоками через фиксированные промежутки времени. Этому способу присуще нерациональное использование ресурсов, особенно если несколько процессов с разной важности находятся в ожидании;

– ***последовательное выполнение потоков по событиям*** – Switch-on-Event Multithreading. Таким событием может быть момент синхронизации связанных процессов. Часто – это переключение задач при возникновении вынужденных длинных пауз, например, непопадание в кэш-ОЗУ (cache miss). В этом случае процесс ждет загрузки данных из медленной памяти в кэш ОЗУ, приостанавливается, высвобождая ресурсы ЦПЭ для других процессов. Благодаря этому, возможно ***сокрытие латентной задержки*** (latent time hiding) обращения к медленной памяти или к другому процессору.

Недостаток последовательного выполнения потоков заключается в в необходимости изменения контекста, который требует много машинных тактов. Также он не дает возможности эксплуатировать параллелизм нескольких операционных блоков.

– ***параллельное выполнение потоков*** – ***Simultaneous Multithreading*** (**SMT**). В этом случае два или более программных потоков выполняются на одном процессоре одновременно, то есть без переключения между ними. Тогда ресурсы ЦПЭ распределяются динамически по принципу: "не используешь – отдай другому". При этом ресурсы нескольких операционных блоков могут быть задействованы совершеннее.

В архитектуре Power5 также была применена дисциплина SMT. Полная реализация SMT довольно сложная. Так, логические схемы для организации SMT занимают 24% процессорного ядра Power5. По данным фирмы IBM, применение SMT улучшает производительность серверов примерно на 28–35%.

Более упрощенный подход положен в основу технологии Intel Hyper-Threading. При этом для реализации многопоточности используется только 5% аппаратуры кристалла.

Особенности технологии Hyper-Threading

Впервые технология Hyper-Threading была реализована в процессоре Pentium-4 – Intel Xeon. В одном ***физическом процессоре*** Intel Xeon формируются два ***логических процессора*** (**LP** – Logical Processor), которые разделяют вычислительные ресурсы ЦПЭ. ОС и приложения "видят" именно два логических ЦПЭ и могут распределять работу между ними.

Одна из целей внедрения Hyper-Threading – при наличии только одного активного потока позволить ему выполняться с тем же быстродействием, как и на обычном ЦПЭ. Для этого у процессора предусмотрены два основных режима работы: Single-Task (ST) и Multi-Task (MT). В режиме ST активен только один логический процессор, который безраздельно пользуется доступными ресурсами (режимы ST0 и ST1), а другой LP остановлен командой HALT. При появлении второго программного потока бездействующий LP активизируется и физический ЦПЭ переводится в режим MT. Остановка командой HALT неиспользованных LP выполняется операционной системой.

Состояние или контекст каждого из двух LP, так называемый, Architecture State (AS) хранится в отдельном блоке. Он включает в себя состояние регистров различного типа – общего назначения, управляющих, прерываний и служебных.

У каждого LP есть свой APIC (контроллер прерываний) и набор регистров, для корректной работы с которыми вводится ***таблица псевдонимов*** (Register Alias Table – **RAT**). RAT отслеживает соответствие между восемью регистрами общего назначения архитектуры IA-32 и 128 регистрами физического ЦПЭ (по одной таблице RAT на каждый LP).

При выполнении двух программных потоков поддерживаются два набора счетчиков команд - Next Instruction Pointers. Два активных LP получают доступ к общему Т-кэш поочередно, то есть через такт. В то время, когда активен только один LP, он получает монопольный доступ к Т-кэш. Аналогичным образом выполняется доступ к ПЗУ микропрограмм. Также дублированы блоки, поддерживающие виртуальную адресацию ITLB (Instruction Translation Look-aside Buffer). Блок декодирования команды архитектуры IA-32 также разделяется между обоими программными потоками опять же через такт.

Пять блоков назначения на ресурсы (Schedulers) обрабатывают очереди декодирования команд независимо от их принадлежности к процессорам LP0 или LP1 и направляют команды на выполнение соответствующим блокам обработки – в зависимости от готовности к выполнению первых и доступности вторых.

Кэш-ОЗУ всех уровней полностью разделяются между двумя LP. В результате, большинство приложений, которые имеют ускорение в многопроцессорных системах, могут также ускоряться и на ЦПЭ с включеным режимом Hyper-Threading без каких-либо модификаций. Но есть и проблемы: например, если один процесс находится в цикле ожидания, он может занять все ресурсы физического ЦПЭ, не давая возможности работать второму LP. Таким образом, производительность при использовании Hyper-Threading может падать (до 20%). Для минимизации этого, компания Intel рекомендует вместо пустых

циклов ожидания использовать команду PAUSE (появилась в архитектуре IA-32, начиная с Pentium-4). Также ведется работа по автоматической оптимизации кода при компиляции.

Поддержка многопоточной обработки совершенствуется в направлении увеличение числа виртуальных процессоров, улучшения динамической выборки команд, чтобы увеличить загруженность операционных блоков. Так, в процессоре Sun UltraSPARC T1, появившемся в конце 2006г., каждое из восьми ядер ЦП, способно обрабатывать четыре программных потока, то есть в нем общее число потоков равно 32. Причем число ступеней конвейера уменьшено до шести, что способствовало упрощению схем предсказания переходов и диспетчеризации загрузки блоков обработки. В 2013 г. число ядер в следующей архитектуре UltraSPARC T5 увеличено до 16, а общее количество параллельных программных потоков – до 128.

3.6. Параллелизм доступа памяти

Согласованность последовательного доступа памяти

Большинство алгоритмических языков имеют простую семантику последовательного доступа памяти. Это позволяет программисту считать, что все операции с памятью выполняются по одному порядку, задаваемому программой. На самом деле, в современной вычислительной системе могут выполняться несколько доступов к памяти одновременно, например от соседних команд программы, от устройства ПДП, от нескольких программных потоков, от другой программы, которая выполняется параллельно. Кроме того, порядок доступов, заданный программистом, может быть искажен как при трансляции программы, так и при ее выполнении аппаратурой в современном процессоре. Поэтому очень важно изучить проблемы, связанные с согласованностью доступов к памяти.

Анализ доступов к памяти удобно изучать на мультипроцессорной модели вычислителя, в которой несколько процессорных элементов (ПЭ) имеют параллельный и независимый доступ к одной, так называемой, ***разделенной памяти.*** Аналогичные ситуации случаются в одном ПЭ, который работает в режиме многопотоковой обработки.

Программист при программировании, как правило, подразумевает модель согласованного последовательного доступа памяти или просто последовательной согласованности. Вычислительная система ***последовательно согласована***, если порядок любых вычислений в этой системе такой же, как у вычислений,

которые выполняются на одном процессоре, то есть как это задано в программе.

Есть два требования к последовательной согласованности:

– выполнение команд программы в том же порядке, как и в одном процессоре;

– последовательный порядок команд неизменный, что обеспечивает единый порядок доступа к памяти.

В последовательно согласованной системе память подключена к ПЭ через коммутатор, который последовательно подключает ее к разделяемой памяти в порядке, который зависит только от начальной программы.

Рассмотрим пример (алгоритм Деккера):

```
ПЭ1: F1 = 0                          ПЭ2: F2 = 0
       If F2 = 0 then                       If F1 = 0 then
           F1 = 1                               F2 = 1
        {критическая процедура}              {критическая процедура}
```

Когда ПЭ1 пытается начать критическую процедуру, он проверяет чужой флажок на 0 и устанавливает свой флажок в 1, чтобы предупредить ПЭ2 от выполнения критической процедуры. То же самое делает ПЭ2. Если система последовательно согласована, то только ПЭ1 будет первым начинать критическую процедуру, независимо от различных обстоятельств. Далее обстоятельства реализации согласованности будут рассматриваться на этом примере.

Нарушения последовательной согласованности

Рассмотрим несколько типичных примеров улучшения архитектуры, которые приводят к нарушению согласованности. Сложность реализации последовательной согласованности зависит от того, есть ли кэш-ОЗУ или нет.

Архитектура без кэш-ОЗУ, но с буфером записи. Все современные процессоры имеют промежуточный буфер записи, в который при записи помещается записываемое данное, что позволяет закончить выполнение команды до завершения операции записи в ОЗУ. Структура системы из двух ПЭ с буферами записи показана на рис. 3.2.

Процессор может положить операцию записи флажка в буфер записи и продолжить работу, не дожидаясь конца операции записи. Чтение для ускорения выполняется без буферизации. При этом наличие буферов может нарушить согласованность – в то время, когда запись флажка F1 = 1 проходит через буфер, в котором задерживается до момента t3, ПЭ2 может прочитать нулевой флажок и начать критическую процедуру вовремя.

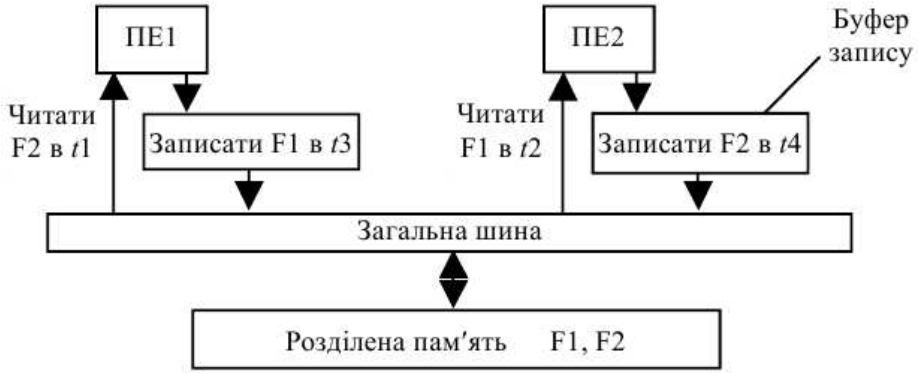


Рис. 3.2. Нарушение согласованности доступа в буферах записи

Архитектура с операциями записи-чтения, имеющих разное время. Система на рис. 3.3 имеет несколько блоков памяти, доступ к которым имеет разную задержку. Такая ситуация также бывает, например, если различные коди выполняются в конвейерном режиме и имеют разное время исполнения и, соответственно, разный момент записи в буфер памяти. Первый ПЭ записывает новое значение D и устанавливает флажок F, для того, чтобы предупредить второй ПЭ, то должен прочитать новое данное. Но через различные задержки доступа к двум ОЗУ, второй ПЭ на самом деле прочитает старое данное D.

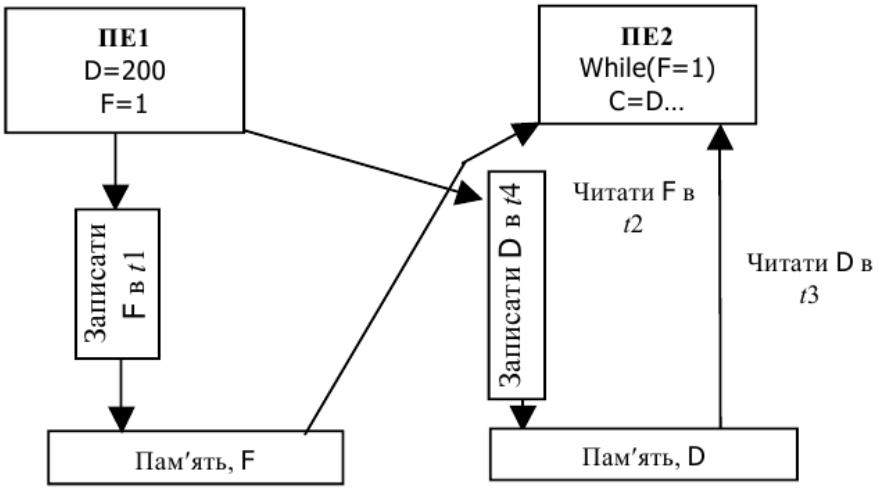


Рис. 3.3. Нарушение согласованности доступа в памяти с различными задержками

Неблокируемая операции чтения. В согласованных процессорах операция чтения, которая следует после записи, блокируется, пока читаемая ячейка получит новое значение. Для упрощения работы ЦПЭ, блокировки отменяют, например, при использовании динамического планирования. Это также может привести к нарушению согласованности.

Вышеприведенные примеры показывают, что процессор должен обеспечить, чтобы его предыдущая операция доступа памяти завершилась до того,

как начнется следующая операция в соответствии с порядком команд в программе. Это называется требованием порядка команд.

Последовательная согласованность в архитектурах с кэш-ОЗУ

Кэширование памяти, или дублирование данных в основном ОЗУ и кэш-ОЗУ, используемых несколькими процессами, приводят к ситуациям, подобным описанным выше. Поэтому в системах с кэш-ОЗУ должны выполняется такие же требования, чтобы сохранялся порядок, заложенный в программе. Показанным выше ситуациям прилагаются следующие, связанные с дублированием данных.

Протокол когерентности кэш-ОЗУ. ***Протокол когерентности кэш-ОЗУ*** – это дисциплина замены старого значение в памяти на новое во всех его копиях. Новое значение распространяется способом ***отмены*** (***invalidating***) или ***перезаписи*** (***updating***) всех его копий.

Когерентность (***consistency***) кэш-ОЗУ выполняется при условиях:

– операция записи должна распознаваться всеми ПЭ, которые могут в негго записывать;

– операции записи в одну ячейку воспринимаются всеми ПЭ в одинаковом порядке.

Правда, эти условия недостаточны для гарантии согласованности доступа памяти. Для реализации когерентности кэш используют различные способы.

Детектирование конца записи. При отсутствии кэш-ОЗУ подтверждение записи может быть сгенерировано, когда данное для записи дойдет до памяти назначения. Но этот момент слишком ранний для систем с кэш-ОЗУ. Пусть к каждому ПЭ на рисунке 3.2 добавлено кэш-ОЗУ. Пусть ПЭ2 имеет операнд D в кэш-ОЗУ. Теперь ПЕ1 записывает F после записи D в ОЗУ, но до того момента, когда D дойдет до кэш-ОЗУ ПЭ2. Есть возможность, что ПЭ2 будет читать новое значение F из общей памяти и все еще читать старое значение из своего кэш-ОЗУ, что нарушает согласованность.

Поэтому ПЭ2 должен ждать, пока данное D перепишется в кэш-ОЗУ или будет отменено, перед тем, как будет запись в F. Вот почему запись в ячейку, которая имеет копию в кэш-ОЗУ, требует прихода сигнала подтверждения отмены или перезаписи. Более того, все такие сигналы должны собираться или в общем ОЗУ или в ПЭ, который выполняет запись. Только после приема сигнала подтверждения, ПЭ вправе считать, что запись завершена.

Выполнение неразделимости операций записи. Распространение изменений в несколько кэш-ОЗУ – это операция, которая длится во времени и может быть разделенной (не атомарной). Поэтому следует выполнять условия неразделенности.

Во-первых, операции записи в одну ячейку должны выполняться последовательно, причем для всех ПЭ последовательность записей должна восприниматься одинаково.

Во-вторых, запрещено читать данное, пока не получен сигнал, что это данное отменено во всех кэш-ОЗУ или оно в них перезаписано.

Оптимизация когерентности кэш-ОЗУ

Оптимизация протокола когерентности кэш-ОЗУ направлена на ускорение обращения к этой памяти. Во-первых, можно выполнять несколько записей параллельно, с перекрытием, если они относятся к независимым ветвям программы. Тогда данное задерживается в буфере записи нужное количество тактов (для выравнивания задержек) и заранее рассылается сигнал отмены данного в кэш-ОЗУ.

Во-вторых, операции чтения задерживаются в соответствии с потребностями программы. При этом есть возможность в промежуток времени задержки прочитать данное еще раз, если пришел сигнал подтверждения записи.

Ослабление требований согласованности доступа памяти

Как видим, удовлетворение требований согласованности доступа памяти, с одной стороны, требуют больших дополнительных аппаратных затрат в параллельных вычислительных системах, с другой стороны, приводит к замедлению вычислительного процесса даже до уровня, когда вычислительная система имеет производительность ниже производительность обычного процессора.

Поэтому, как правило, используются архитектурные модели с ослаблением требований согласованности доступа памяти. В таких моделях команды разделяются на те, которые должны удовлетворять требованиям согласованности доступа и такие, которые могут нарушать эти требования.