Федеральное агентство по образованию

САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

С. А. МОЛОДЯКОВ

АРХИТЕКТУРА ЭВМ
Часть 2. Современные процессоры и мультипроцессорные системы. Периферийные устройства

УЧЕБНОЕ ПОСОБИЕ

Санкт-Петербург
Издательство Политехнического университета
2014

УДК 004.2: 004.35: 004.431.4

Молодяков С.А.

Архитектура ЭВМ. Часть 2: учебное пособие. СПб.: Изд. СПбГПУ, 2014.- с.

В учебном пособии представлен материал лекций, которые читаются по дисциплинам «Архитектура ЭВМ» и «ЭВМ и периферийные устройства» на кафедре информационных и управляющих систем СПбГПУ. В первой части пособия рассмотрены вопросы, связанные с базовой организацией ЭВМ, а также ее программирования на языке Ассемблера. Во второй части рассмотрены современные процессоры и мультипроцессорные системы, а также периферийные устройства.

Учебное пособие предназначено для студентов, изучающих дисциплины «Архитектура ЭВМ», «ЭВМ и периферийные устройства» по специальностям 231000 Программная инженерия, 230100 Информатика и вычислительная техника.

Печатается по решению редакционно-издательского совета Санкт-Петербургского государственного политехнического университета.

Оглавление

Глава 1. Процессоры. Архитектурные пути обеспечения параллельности уровня команд

1.1. Пути повышения производительности

Повышение производительности компьютера и вычислительной системы связано с тремя элементами:

- Повышение тактовой частоты процессора.
- Увеличение пропускной способности подсистемы памяти.
- Повышение степени параллелизма вычислений в компьютере/вычислительной системы.

Повышение тактовой частоты в первую очередь определяется успехами технологии. Сегодня практически все производители используют технологию КМОП. Максимальная частота работы процессора связана в основном с процессом перезаряда паразитных емкостей. Величина паразитных емкостей определяется проектными нормами (минимальным размером элемента на кристалле). Уменьшение проектных норм приводит одновременно к уменьшению требуемых напряжений питания, уменьшению мощности, потребляемой одним логическим вентилем и к уменьшению времени переключения вентиля. Основное соотношение для экстенсивного увеличения скорости определяется выражением:

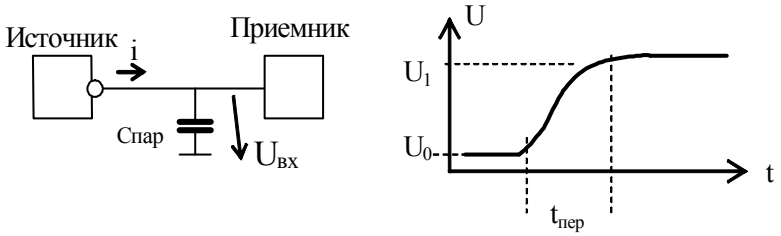
$t_{пер} = k\,\Delta U_{01}\,L_{пр} / \sqrt{P}$

где $t_{пер}$ - время переключения сигнала с одного уровня на другой (см. рис.);

ΔU_{01} – диапазон переключения;

$L_{пр}$ – проектная норма, паразитная емкость пропорциональна квадрату величины проектной нормы $C_{пар} \sim \varepsilon * L_{пр}^2/d$

P – мощность переключения.



Из этой (качественной) формулы видно, каким путем и за какую цену можно повышать быстродействие. Время переключения элемента будет тем меньше, чем меньше промежуток ΔU_{01} между интервалами сигнала (напряжения), кодирующими значения логической переменной, чем меньше проектные нормы $L_{пр}$ (размеры элементов на кристалле) и чем больше мощность P, потребляемая логическим элементом от источника питания, и затрачиваемая на перезаряд паразитных емкостей и в конечном итоге, нагревающая кристалл БИС.

В настоящее время одним из основных показателей эффективности процессора является величина, показывающая производительность на единицу затраченных ресурсов. Такой величиной служит, например величина (МГц/Ватт).

Технологический предел линейных размеров транзисторов на кристалле, обусловленный физическими ограничениями. На пути дальнейшей миниатюризации кроме физических ограничений имеются и экономические. Для каждого следующего

поколения микросхем стоимость технологии удваивается. Технология развивается в направлении увеличения плотности транзисторов на кристалле, роста числа слоев металлизации и повышения тактовой частоты наряду с уменьшением напряжения питания и удельной (на один транзистор) потребляемой электрической и выделяемой тепловой энергии. С каждым поколением линейные размеры элементов уменьшаются в 0.7 раза. Смена поколений микропроцессоров происходит каждые 2-3 года. Увеличение числа слоев металлизации экспоненциально повышает процент брака при производстве, увеличение площади кристалла также приводит к снижению выхода годных кристаллов.

На рис.1.1 показан график уменьшения характерного размера проектной нормы в компании Intel. На рис.1.2 приведена эволюция технологии в компании Intel. Компания вводит новую технологию каждые два года. Видно, что происходит непрерывное повышение тактовых частот.

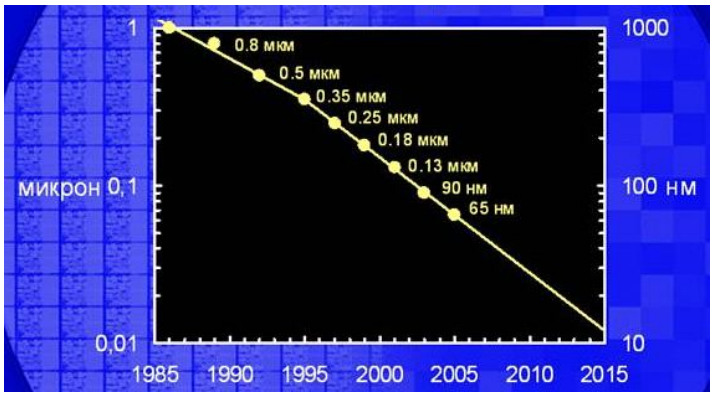


Рис.1.1. График уменьшения характерного размера проектной нормы в компании Intel.

Процесс	P1262	P1264	P1266	P1268
Литография	90 нм	65 нм	45 нм	32 нм
Начало произв.	2003	2005	2007	2009

Рис.1.2. Эволюция технологии в компании Intel.

Ограничение Лэндауэра

При необратимом процессе обработки информации, когда возможна потеря битов, затраты на обработку одного бита не могут быть меньше величины kT ln2, где: k — константа Больцмана, а T — температура по Кельвину

Обработка одного бита — это результат выполнения какой-либо операции над двумя битами (AND, OR, XOR). Можно считать, что при выполнении операции один из битов-операндов превращается в результат операции и энергия не теряется, а вот другой бит-операнд теряется. Энергия выделяется в виде тепла, что и считается минимальной энергетической платой за выполнение элементарной операции.

Технологиям в 22 нм соответствует уровень энергетических затрат на обработку одного бита не менее 100 000 – 1 000 000 kT, и Лэндауэр, исследуя динамику улучшения технологий, показал, что достижение уровня kT произойдет в районе 2020 года (5 нм).

Закон Мура перестанет работать не позднее 2024 года

Инновации, не масштабирование – ключ к развитию микроэлектроники

Таково мнение док. Бернарда Мейерсона – вице-президента по стратегическим альянсам и директора по технологиям компании IBM. Вскоре промышленность уже не сможет улучшать производительность систем за счет увеличения рабочей частоты, поскольку, чем выше рабочая частота и уровень интеграции, тем больше потребляемая мощность приборов, уже сегодня достигающая своего предела. Чтобы увеличить производительность необходимы новые принципы построения компонентов, схем и их архитектуры.

По мнению Мейерсона, в истории развития полупроводниковой электроники можно выделить две поворотные точки. В 60–70-е годы на первых этапах микроэлектроники разработчики продвигали биполярную технологию и сумели довести число транзисторов на кристалле до десятков тысяч. Дальнейшее увеличение плотности элементов столкнулось с проблемой слишком возросших энергозатрат. Но к этому времени уже появилась КМОП-технология, обещавшая улучшить характеристики и степень интеграции микросхем, выполненных на ее основе. И все это при меньшей потребляемой мощности. Поэтому с 70-х годов разработчики непрерывно увеличивают число вентилей на кристалле и улучшают их характеристики. Однако сегодня КМОП-микросхемы так же "голодают", как и их ранние биполярные "сестры". Современная микроэлектроника, считает Мейерсон, подошла ко второй поворотной точке в своей истории, хотя сейчас нет новой технологии, которая позволила бы в ближайшем будущем заменить КМОП микросхемы. Такие перспективные устройства, как приборы на углеродных нанотрубках появятся лишь через 10–15 лет.

Увеличение пропускной способности подсистемы памяти достигается сбалансированным выбором многоуровневой структуры КЭШ-памятей и пропускной способности шин между этими уровнями. Следует помнить, что эффективность использования (быстрой) кэш-памяти основана на принципе «один раз медленно из основной памяти в кэш, а затем много раз быстро из кэша в процессор». Иерархическая структура памяти рассмотрена в части 1 пособия.

В зависимости от специфики конкретной программы эффективность кэша может варьировать от очень высокой - 5…10 кратное ускорение (для задач с малым объемом кода и данных, которые целиком помещаются в кэш и с высокой степенью их повторного использования – типичный пример –итеративная вычислительная задача) до отрицательной, когда значительная часть данных (чаще) или кода (реже) используется лишь однократно.

Повышение степени параллелизма вычислений в компьютере в первую очередь связано с появлением дополнительных элементов (транзисторов) на кристалле, которые возникли за счет успехов в технологии. Стало возможным использовать и развивать различные пути обеспечения параллельности уровня команд и уровня потоков команд. Появляются новые архитектуры процессоров. Можно выделить два этапа развития архитектуры процессора:

- Развития архитектуры ядра процессора. Суперскалярность и конвейерность.
- Развития многоядерных архитектур.

В данной части и следующей рассмотрим проблемы и направления развития процессора, которые обеспечивают параллельное выполнение команд и их этапов в процессоре. Многоядернность позволяет не только параллельно выполнять потоки команд, но и снижает потреблюемую процессоров мощность. Это является основным

направлением развития архитектур быстродействующих процессоров. Многоядерные архитектуры будут рассмотрены далее при описании многопроцессорных систем.

На рис.1.3. показано развитие параллелизма в рамках единого процессора на примере процессоров Intel x86.

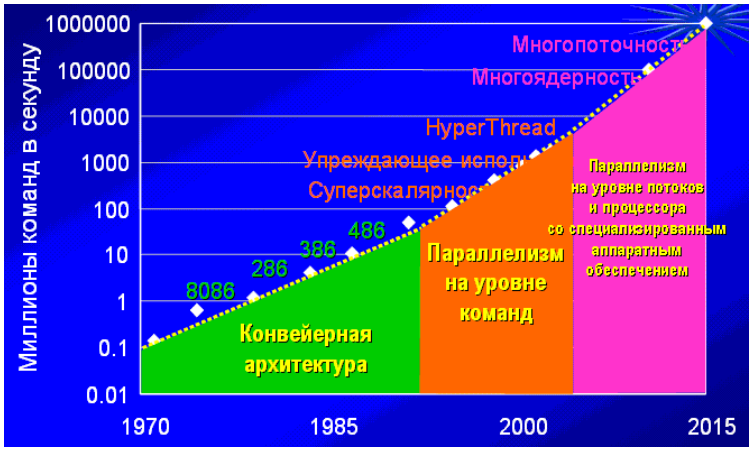
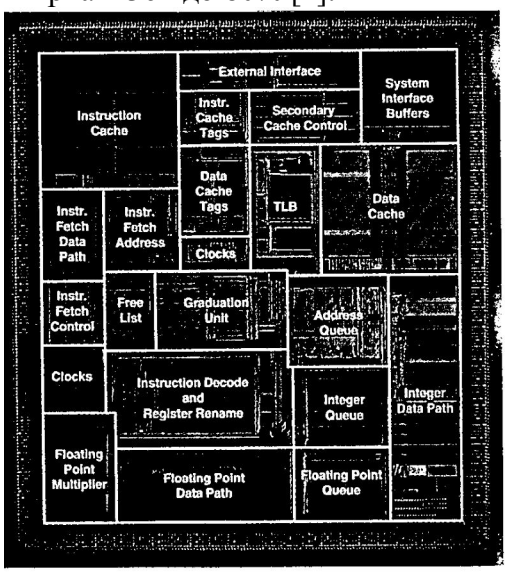


Рис. 1.3. Развитие параллелизма в рамках единого процессора Intel x86.

С каждым поколением микропроцессоров все меньшую долю поверхности кристалла занимают исполнительные элементы (само вычислительное ядро) и большую часть можно отдать на дополнительные к ядру элементы. Специализация современных микропроцессоров сводится к специализации этих элементов. Так в универсальных микропроцессорах дополнительные площади поверхности кристалла отдаются под КЭШ память, а в микроконтроллерах - под коммуникационные и другие устройства. На рис. 1.4 приведен пример распределения поверхности кристалла для процессора R10000 (Процессор 5-го поколения; 64 регистра целочисленных+64 регистра чисел с плавающей запятой; каждый регистр 64 разряда; число конвейеров – 5). В процессорах шестого поколения доля КЭШ первого и второго уровня возрастает: в Pentium4 до 40 %, а в Alpha21364 до 80% [1].



Twin FPUs in the R10000 processor mean exceptional

Интерфейс шины (5-8%)			
КЭШ команд и данных(10-20%)			
TLB и транслятор адреса (5-10%)			
Д/Ш	Буферы	Регистры	. . .
Очередь		Очередь	
IntegU		FPU	

Рис.1.4. Пример распределения транзисторов процессора R10000

1.2. Способы оценки производительности средств вычислительной техники

Производительность средства ВТ (компьютера) можно оценивать различными способами. В таблице 7.1. приведены способы, в разной степени учитывающие влияние на производительность «окружения» процессора. В первом (грубом) приближении принято считать, что производительность определяется скоростью (тактовой частотой) процессора. Однако, суперскалярные процессоры выполняют более одной операции за такт, поэтому более точным показателем возможности процессора выполнять команды, является пиковая производительность. В этом случае для суперскалярного процессора с тактовой частотой 1МГц, выполняющего 3 операции за такт, пиковая производительность составит 3 MIPS или 3MFLOPS. (MFLOPS = MFlop/sec. - миллионы операций с плавающей точкой в секунду).

Табл 7.1. Разные способы оценки производительности/эффективности

N	Способ оценки	Влияющие факторы
1.	Время переключения элементов	Скорость переключения транзисторов и скорость перезаряда паразитных емкостей
2.	Тактовая частота – пиковая производительность	Логическая «глубина» тактируемых слоев
3.	Времена выполнения команд	Аппаратная/ микропрограммная реализация
4	Интенсивность потока результатов на выходе АЛУ	Степень конвейеризации/ запараллеливания команд, зависимости (dependencies), относительная частота использования команд
5.	Benchmarks (тестовые последовательности команд)	Относительное быстродействие подсистем: процессора, а также слоев памяти
6.	Тестовые задачи	Быстродействие дисковой и видео подсистем, а также организационная сторона и взаимодействие с обьектами вне ВС (оператор, обьект уп-равления, сеть)

Для полной оценки производительности ЭВМ используются системы тестов. В целом выделяют тесты разработчиков и тесты потребителей.

Один из распространенных тестов при оценке быстродействия ЭВМ – это тест/индекс Intel iCOMP(Intel COmpatible Microprocessor Performance). Этот тест - индекс в 1992г. фирма Intel предложила для оценки производительности своих процессоров; тест содержит 67% операций с 16-разрядными целыми числами, 25% - с 32-разрядными целыми числами, 5% - с 32-разрядными действительными числами и 3% - с 16-разрядными действительными числами

SPEC (Standard Performance Evaluation Corporation) - это корпорация, созданная в 1988 году, объединяющая ведущих производителей вычислительной техники и программного обеспечения. SPEC имеет целью разработку и стандартизацию методов оценки производительности современных компьютеров. Разработанные SPEC тестовые пакеты являются де-факто стандартами для оценки производительности современных микропроцессоров, компьютеров и системного ПО.

SPEC выполняет две основные функции: Разрабатывает тестовые пакеты. Собирает и публикует официальные результаты тестов. Корпорация разработала целый ряд тестов для разных систем. Выделим только тестовый пакет для оценки производительности микропроцессоров (ЦП) и вычислительных систем - CPU2000.

Пакет CPU2000 состоит из двух групп тестов - CINT2000 для оценки производительности на целочисленных операциях и CFP2000 для оценки производительности на операциях с плавающей точкой. Группы тестов CINT2000 и CFP2000 ориентированы на оценку работы микропроцессоров, подсистемы кэш-памяти и оперативной памяти, а также компиляторов. Эти тесты не имеют отношения к оценке производительности сети, дисков или графической подсистемы.

В набор CINT2000 входят 11 тестовых приложений, написанных на языке C, и один тест (252.eon) на C++.

Название	Краткое описание задачи
164.gzip	Утилита сжатия данных (gzip)
175.vpr	Приложение для расчета FPGA-кристаллов
176.gcc	Компилятор C
181.mcf	Приложение для решения задачи потока минимальной стоимости в сети
186.crafty	Программа для игры в шахматы
197.parser	Синтаксический разбор для естественного языка
252.eon	Трассировка лучей
253.perlbmk	Интерпретатор языка Perl
254.gap	Вычислительная задача из теории групп
255.vortex	Объектно-ориентированная база данных
256.bzip2	Утилита сжатия данных (bzip2)
300.twolf	Задача позиционирования и маршрутизации

В набор CFP2000 входят 14 тестовых приложений, из которых 6 написаны на языке Fortran 77, 4 на языке Fortran 90 и 4 на C++.

Название	Краткое описание задачи
168.wupwise	Задача квантовой хромодинамики
171.swim	Гидродинамическая задача моделирования для "мелкой" воды
172.mgrid	Многосеточная решалка для трехмерного потенциального поля
173.applu	Решение параболических/эллиптических дифференциальных уравнений
177.mesa	Трехмерная графическая библиотека (Mesa3D)
178.galgel	Гидродинамическая задача: анализ колебательной нестабильности
179.art	Моделирование нейронной сети
183.equake	Моделирование землетрясения методом конечных элементов
187.facerec	Задача распознавания лиц на графических изображениях
188.ammp	Задача вычислительной химии
189.lucas	Задача теории чисел (проверка простоты)
191.fma3d	Моделирование crash-тестов методом конечных элементов
200.sixtrack	Моделирование ускорителя элементарных частиц
01.apsi	Атмосферная задача с учетом температуры, ветра и загрязнений

Результаты CPU2000 (CINT2000 и CFP2000) предоставляются в вариантах ("метриках"): время выполнения одной итерации теста, количество итераций за фиксированное время. Метрики типа "rate" (SPECint_rate2000, SPECfp_rate2000) позволяют оценить суммарный объем вычислений, который компьютер может выполнить за определенное время. Метрики "non-rate" (SPECint2000) оценивают просто "скорость" вычислений.

Тестовые наборы CINT2000 и CFP2000 состоят соответственно из 12 и 14 независимых тестовых программ, каждая из которых дает отдельный показатель производительности. Для усреднения результатов по отдельным тестам в данном тестовом наборе используется среднее геометрическое от нормализованных результатов всех тестов. Для устранения влияния случайных факторов на результаты тестирования, каждый тест прогоняется нечетное число раз (как минимум, 3), результаты по запускам располагаются по возрастанию и в качестве окончательного берется результат из середины ряда ("медианный").

Показатель SPECint2000, например, вычисляется следующим образом:

SPECint2000 = RefTime / MeasuredTime,

где RefTime - время исполнения теста на эталонной машине, а MeasuredTime - время исполнения на тестируемой машине. Таким образом, смысл этого показателя - в относительном ускорении по сравнению с эталонной машиной.

Linpack benchmark

Тест используется для оценки производительности мультипроцессорных систем, включая суперкомпьютеры. Он состоит в решении системы линейных арифметических уравнений $Ax = f$, A – *матрица.*

Единицей измерения является *1 флопс*. Количество операций, необходимое для решения задачи Linpack, известно с самого начала и зависит от ее размерности. Измеряемая характеристика производительности получается как простое частное от деления известного числа операций на время, затраченное на решение задачи .

Имеется версия для тестирования кластерных систем.

Business Applications Performance Corporation BAPCo SYSmark® 2007

BAPCo – это консорциум независимых тестовых лабораторий, разработчиков компьютеров, полупроводниковой техники и программного обеспечения. В настоящее время членами консорциума являются фирмы Adaptec, Amdahl Corporation, Compaq, Dell, Hewlett-Packard, IBM, Intel, Microsoft, Motorola, NEC и издания Federal Computer Week, InfoWorld, VNU Business Publications Limited (UK). http://www.bapco.com

Набор тестов SYSmark работает под управлением всех 32-битных операционных систем семейства Windows и, используя настоящие приложения (в состав тестов входит 12 приложений, называемых Internet Content Creation and Productivity), позволяет определить производительность компьютеров в режиме имитации реальной работы. При тестировании используются следующие офисные приложения: CorelDRAW , Microsoft Excel , Dragon Systems NaturallySpeaking Preferred, Corel Paradox 9, Microsoft PowerPoint и Microsoft Word и др., а также приложения для разработки содержимого Web-узлов: MetaCreations Bryce, Adobe Photoshop и др.

Полученные в результате тестирования коэффициенты отражают производительность тестируемого компьютера по отношению к базовой конфигурации.

Помимо уменьшения времени переключения логических элементов, существует фактор, приводящий к увеличению скорости работы процессоров, и за 20 лет, благодаря этому фактору, удалось достигнуть дополнительного увеличения скорости более чем на порядок. Этот дополнительный фактор – совершенствование структуры процессоров, вследствие чего удается строить более быстрые процессоры без увеличения скорости переключения элементов. Основное улучшение связано с параллельностью. Параллельное выполнение команд в ядре процессора обеспечивается двумя процессами: конвейеризацией и параллельным выполнением команд - суперскалярностью.

1.3. Архитектуры реализации параллельности выполнения команд. Конвейеризация. Суперскалярная архитектура. Процессоры с длинным командным словом.

Конвейеризация.

Конвейеризация (См часть 1.) в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры (см. рис.1.5.). Количество блоков может быть различным, например их может быть 5. Команда по тактам в процессе выполнения продвигается по блокам. Одновременно вслед за командой (раньше нее) по этим же блокам продвигаются другие команды. В результате, если каждый этап команды выполняется отдельным блоком за один такт, то на выходе процессора каждый такт, появляется результат очередной команды. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько команд. Конвейерная обработка такого рода широко применяется во всех современных быстродействующих процессорах.

Рис.1.5. Разбиение процессора на блоки выполнения команд.

Конвейеризация - способ обеспечения параллельности выполнения команд.

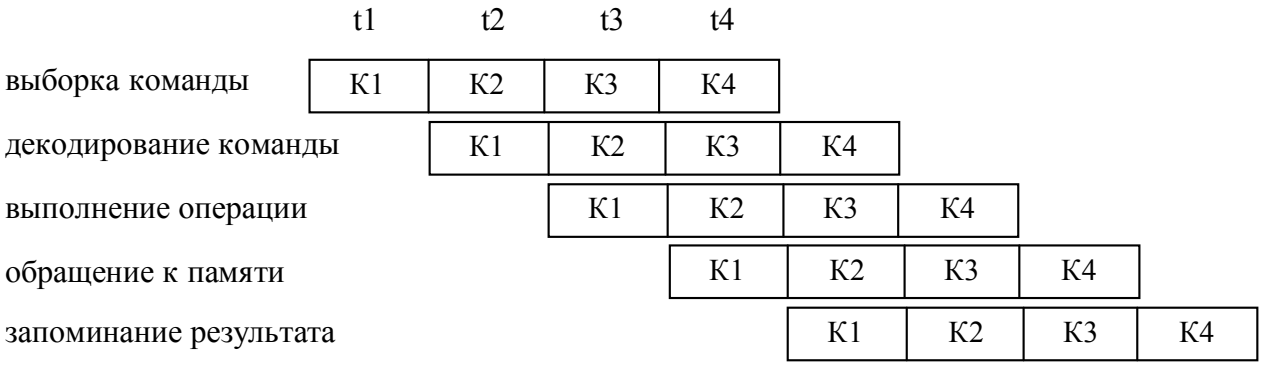
Суперконвейер. Если обычный конвейер разбить на более мелкие этапы, то получим длинный или супер-конвейер. Суперконвейер позволяет поднять тактовые частоты процессора, но при этом действует ряд ограничений на длину конвейера. В данной части будут рассмотрены основные ограничения, прежде всего связанные с зависимостями между командами, которые в ряде случаев тормозят прохождение команд по конвейеру.

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

Выполнение типичной команды можно разделить на следующие этапы:

- выборка команды - IF (по адресу, заданному счетчиком команд, из памяти извлекается команда);
- декодирование команды / выборка операндов из регистров - ID;
- выполнение операции / вычисление эффективного адреса памяти - EX;
- обращение к памяти - MEM;
- запоминание результата - WB.

В зависимости от типа команды, вида способа адресации время выполнения команды сильно варьируется. Дольше всего выполняются этапы, связанные с обращением к памяти. Работа конвейера представлена на диаграмме, на которой изображены выполняемые команды, номера тактов и этапы выполнения команд.



Конвейеризация увеличивает пропускную способность процессора (количество команд, завершающихся в единицу времени), но она не сокращает время выполнения отдельной команды. Имеются некоторые накладные расходы на конвейеризацию, возникающие в результате несбалансированности задержки на каждой его ступени. Частота синхронизации (такт синхронизации) не может быть выше, чем время, необходимое для работы наиболее медленной ступени конвейера.

Поток команд - естественная последовательность команд, проходящая по конвейеру процессора. Процессор может поддерживать несколько потоков команд (суперпроцессоры 5 и 6 поколения), если для каждого потока и каждого этапа есть исполнительные элементы.

Конвейеризация эффективна только тогда, когда загрузка конвейера близка к полной, а скорость подачи новых команд и операндов соответствует максимальной производительности конвейера (характерно для RISC-процессоров). Если произойдет задержка, то параллельно будет выполняться меньше операций и суммарная производительность снизится. Такие задержки могут возникать в результате возникновения конфликтных ситуаций. Рассмотрим различные типы конфликтов, возникающие при выполнении команд в конвейере, и способы их разрешения.

Существуют три класса конфликтов:

1. Структурные конфликты, которые возникают из-за конфликтов по ресурсам, когда аппаратные средства не могут поддерживать все возможные комбинации команд в режиме одновременного выполнения с совмещением.
2. Конфликты по данным, возникающие в случае, когда выполнение одной команды зависит от результата выполнения предыдущей команды.
3. Конфликты по управлению, которые возникают при конвейеризации команд переходов и других команд, которые изменяют значение счетчика команд.

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

Среднее количество тактов выполнения команды в конвейере	**CPI конвейера = CPI идеального конвейера + + Приостановки из-за структурных конфликтов + + Приостановки из-за конфликтов по данным + + Приостановки из-за конфликтов по управлению**

Суперскалярная архитектура. Процессоры с длинным командным словом.

Иерархия параллельной обработки данных включает четыре уровня параллельности, каждый из которых должен быть поддержан аппаратурой.

1. Слабо связанные задачи общего проекта - решаются на своих процессорах (процессорная сеть).

2. Отдельные ветви или потоки программ, которые выполняются параллельно (мультискалярный процессор, многоядерный процессор или мультипроцессорная система).

3. Одновременное выполнение нескольких команд (суперскалярный процессор, процессоры с длинным командным словом.).

4. Разбиение процесса выполнения команд на части, которые можно выполнять параллельно (конвейерный процессор).

Обеспечение параллельности на первых двух уровнях определяется не только аппаратурой, а в первую очередь решаемыми задачами и алгоритмами разбиения этих задач на параллельные потоки. Вторые два уровня - параллельность уровня команд. На этих уровнях параллельное выполнение команд напрямую не зависит от выполняемых программ. Обеспечение параллельности уровня команд является основным средством достижения высокого быстродействия микропроцессора.

Есть два крайних подхода, при возможных промежуточных, к отображению присущего микропроцессору внутреннего параллелизма обработки данных на архитектурном уровне в системе команд. Первый подход более консервативен и состоит в том, что никакого указания на параллельную обработку внутри процессора система команд не содержит. Такие процессоры относятся к классу суперскалярных. Термин суперскалярный появился в 1987 году. Во втором подходе в формате команд уже заложена параллельной. В специально отведенных полях команды каждому из параллельно работающих обрабатывающих устройств предписывается действие, которое устройство должно совершить. Такие процессоры называются процессорами с длинным командным словом (VLIW). Предполагается, что существуют компиляторы с языков высокого уровня, которые готовят программы для загрузки их в VLIW-микропроцессоры или уже на низком уровне языка Ассемблера формируются длинные команды. В мультискалярных процессорах реализуется третий подход, связанный с обеспечением параллельности выполнения команд. Мультискалярный микропроцессор содержит несколько процессорных элементов со своими счетчиками команд, которые выполняют свои ветви программ. Такой микропроцессор представляет собой однокристальную многопроцессорную систему с разделяемой памятью и работающей на 2 уровне параллельности.

Суперскалярность только на уровне аппаратуры	Процессоры с длинным командным словом (very long instruction word - VLIW)	Мультискалярные процессоры Многоядерные процессоры
Параллельность уровня команды.		

Суперскалярность состоит в способности независимо выполнять команды параллельно на разных конвейерах. Очередная команда в такой структуре поступает на первый освободившийся конвейер (если конвейеры одинаковы). В реализациях структура, как правило, более сложная (см. напр. Pentium Pro), когда невозможно выделить отдельные “нити” нескольких конвейеров, но, тем не менее, для каждой стадии выполнения команды существует более одного исполнительного блока.

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

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

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

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

Динамическая последовательность команд структурирована (разделена) зависимостями по данным и управлению. Зависимости по управлению (которые проявляются как переходы по условию) представляют главное препятствие высоко-параллельному выполнению потому, что эти зависимости должны быть установлены прежде чем будут выполнены все последующие команды.

1.4. Структурные конфликты, зависимости по данным и по управлению

1.4.1. Структурные конфликты

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

не могут поступать в него в каждом такте. Другая возможность появления структурных конфликтов связана с недостаточным дублированием некоторых ресурсов, что препятствует выполнению произвольной последовательности команд в конвейере без его приостановки. Например, машина может иметь только один порт записи в регистровый файл, но при определенных обстоятельствах конвейеру может потребоваться выполнить две записи в регистровый файл в одном такте. Это также приведет к структурному конфликту.

Структурные зависимости – конфликт из-за недостаточности ресурсов.

Структурные конфликты возникают, например, и в машинах, в которых имеется единственный конвейер памяти для команд и данных. В этом случае, когда одна команда содержит обращение к памяти за данными, оно будет конфликтовать с выборкой более поздней команды из памяти. Чтобы разрешить эту ситуацию, можно просто приостановить конвейер на один такт, когда происходит обращение к памяти за данными. Подобная приостановка часто называются "конвейерным пузырем" (pipeline bubble) или просто пузырем, поскольку пузырь проходит по конвейеру, занимая место, но не выполняя никакой полезной работы. Пример структурного конфликта при реализации памяти с одним портом представлен на рис.7.6.

Возникает вопрос: почему разработчики допускают наличие структурных конфликтов? Для этого имеются две причины: снижение стоимости и уменьшение задержки устройства. Конвейеризация всех функциональных устройств может оказаться слишком дорогой. Машины, допускающие два обращения к памяти в одном такте, должны иметь удвоенную пропускную способность памяти, например, путем организации раздельных кэшей для команд и данных.

Команда	**Номер такта**								
	1	2	3	4	5	6	7	8	9
Команда 1	IF	ID	EX	MEM	WB				
Команда 2		IF	ID	EX	MEM	WB			
Команда 3			IF	ID	EX	MEM	WB		
Команда 4				stall	IF	ID	EX	MEM	WB
Команда 5						IF	ID	EX	MEM
Команда 6							IF	ID	EX

Рис. 1.6. Диаграмма работы конвейера при структурном конфликте

1.4.2. Конфликты по данным и методы их преодоления

Зависимости по данным обусловлены использованием одних и тех же ресурсов памяти (регистров, ячеек памяти) в разных командах. Рассмотрим конвейерное выполнение последовательности команд на рисунке 7.7.

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

Рис. 1.7. Последовательность команд в конвейере

В этом примере все команды, следующие за командой ADD, используют результат ее выполнения. Команда ADD записывает результат в регистр R1, а команда SUB читает это значение. Если не предпринять никаких мер для того, чтобы предотвратить этот конфликт, команда SUB прочитает неправильное значение и попытается его использовать. На самом деле значение, используемое командой SUB,

является даже неопределенным: хотя логично предположить, что SUB всегда будет использовать значение R1, которое было присвоено какой-либо командой, предшествовавшей ADD, это не всегда так. Если произойдет прерывание между командами ADD и SUB, то команда ADD завершится, и значение R1 в этой точке будет соответствовать результату ADD.

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

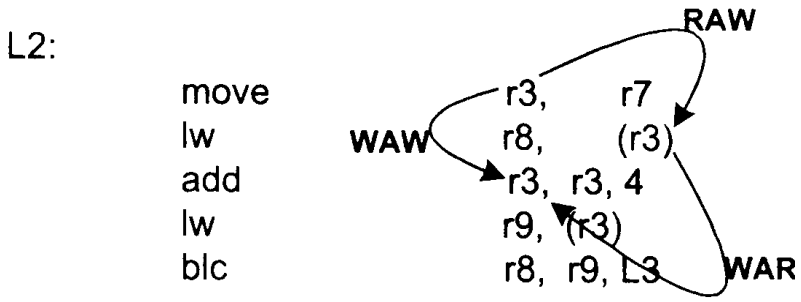
Классификация всех видов зависимостей представлена в таблице и на рис.7.8.

RAR - "чтение после чтения"	WAR - "запись после чтения"	WAW - "запись после записи"	RAW - "чтение после записи"
Отсутствие зависимостей	Лишние зависимости		Действительная зависимость

Рис.1.8. Зависимости по данным

Возможны три конфликта по данным в зависимости от порядка операций чтения и записи (Рассмотрим две команды i и j, при этом i предшествует j):

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



- WAR (запись после чтения) - j пытается записать результат в приемник прежде, чем он считывается оттуда командой i, так что i может некорректно получить новое значение. Особенно часто конфликты такого рода могут возникать в системах, допускающих выполнение команд не в порядке их расположения в программном коде (6 поколение процессоров).
- WAW (запись после записи) - j пытается записать операнд прежде, чем будет записан результат команды i, т.е. записи заканчиваются в неверном порядке, оставляя в приемнике значение, записанное командой i, а не j. Этот тип конфликтов присутствует только в конвейерах, которые выполняют запись со многих ступеней.

К сожалению не все потенциальные конфликты по данным могут обрабатываться с помощью механизма "обходов". Рассмотрим следующую последовательность команд (рис.7.9):

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

Рис. 1.9. Последовательность команд в конвейере

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

Многие современные компиляторы используют технику планирования команд для улучшения производительности конвейера. В простейшем алгоритме компилятор просто планирует распределение команд в одном и том же базовом блоке. **Базовый блок** представляет собой линейный участок последовательности программного кода, в котором отсутствуют команды перехода, за исключением начала и конца участка (переходы внутрь этого участка тоже должны отсутствовать). Планирование такой последовательности команд осуществляется достаточно просто, поскольку компилятор знает, что каждая команда в блоке будет выполняться, если выполняется первая из них, и можно просто построить граф зависимостей этих команд и упорядочить их так, чтобы минимизировать приостановки конвейера.

Однако, требуются более сложные алгоритмы планирования. Они базируются на следующих методах.

1. Методы динамической оптимизации ("out-of-order execution" - неупорядоченное выполнение).

Основными средствами динамической оптимизации являются:

Размещение схемы обнаружения конфликтов в возможно более низкой точке конвейера команд.

Буферизация команд, ожидающих разрешения конфликта, и выдача последующих, логически не связанных команд, в "обход" буфера. В этом случае команды могут выдаваться на выполнение не в том порядке, в котором они расположены в программе.

Соответствующая организация коммутирующих магистралей, обеспечивающая засылку результата операции непосредственно в буфер, хранящий логически зависимую команду, задержанную из-за конфликта, или непосредственно на вход функционального устройства до того, как этот результат будет записан в регистровый файл или в память.

2. Метод переименования регистров (register renaming).

Выделяются **логические регистры**, обращение к которым выполняется с помощью соответствующих полей команды, и **физические регистры**, которые размещаются в аппаратном регистровом файле процессора. Номера логических регистров динамически отображаются на номера физических регистров посредством таблиц отображения, которые обновляются после декодирования каждой команды. Каждый новый результат записывается в новый физический регистр. Однако предыдущее значение каждого логического регистра сохраняется и может быть

восстановлено в случае, если выполнение команды должно быть прервано из-за возникновения исключительной ситуации или неправильного предсказания направления условного перехода.

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

Программист имеет дело только с логическими регистрами. Реализация физических регистров от него скрыта. Метод переименования регистров упрощает контроль зависимостей по данным.

1.4.3. Конфликты по управлению

Существует две разновидности переходов: безусловные, всегда передающие управление по новому указанному адресу, и условные, которые меняют или не меняют ход выполнения программы в зависимости от результата сравнения или выполнения какого-либо другого условия. (Когда условный переход не выполняется, программа просто продолжает выполнение следующей по порядку команды.)

В типичной программе до 10% команд могут быть безусловными переходами, и еще 10-20% - представлять собой условные переходы. Безусловные переходы проблем не вызывают; процессор "уверен", что они будут выполнены и поэтому просто начинает выборку команд по указанному адресу. Команды условных переходов представляют гораздо большие трудности, потому что процессор не может "знать", будет ли переход выполнен или нет до тех пор, пока команда не пройдет исполнительную ступень конвейера.

Однако ожидание, пока команда ветвления покинет исполнительную ступень, означает отказ от возможности выборки и обработки многих команд. Процессору нужен некий алгоритм, который позволит ему "угадать", будет ли выполняться переход или нет. Если предсказание окажется верным, то исполнение продолжится с малой задержкой либо вовсе без нее. Если же предположение ошибочно, то частично выполненные команды придется удалить из конвейера, а новые команды выбрать из области памяти с правильным адресом, декодировать и выполнить их. Это повлечет за собой существенное снижение производительности - особенно ощутимое в процессорах, которые, как P6, имеют многочисленные и глубокие конвейеры. Если P6 ошибается в предсказании перехода, он может потерять от 4 до 15 тактов.

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

Простейший метод работы с условными переходами заключается в приостановке конвейера, как только обнаружена команда условного перехода до тех пор, пока она не достигнет ступени конвейера, которая вычисляет новое значение счетчика команд. Такие приостановки конвейера из-за конфликтов по управлению встречаются часто и должны реализовываться иначе, чем приостановки из-за конфликтов по данным, поскольку выборка команды, следующей за командой условного перехода, должна

быть выполнена как можно быстрее, как только мы узнаем окончательное направление команды условного перехода.

Команда перехода	IF	ID	EX	MEM	WB					
Следующая команда		IF	stall	stall	ID	EX	MEM	WB		
Следующая ком. +1			stall	stall	stall	IF	ID	EX	MEM	WB
Следующая ком. +2				stall	stall	stall	IF	ID	EX	MEM
Следующая ком. +3					stall	stall	stall	IF	ID	EX
Следующая ком. +4						stall	stall	stall	IF	ID
Следующая ком. +5							stall	stall	stall	IF

Рис. 1.10. Последовательность команд в конвейере

Если конвейер будет приостановлен на три такта на каждой команде условного перехода, то это может существенно отразиться на производительности машины. При частоте команд условного перехода в программах, равной 30% и идеальном CPI машина с приостановками условных переходов достигает примерно только половины ускорения, получаемого за счет конвейерной организации. Число тактов, теряемых при приостановках из-за условных переходов, может быть уменьшено на каждом этапе выполнения команды следующими способами: раннее распознание команды условного ветвления, проверка и предсказание выполнения условия перехода, ускоренное (раннее) вычисление целевого адреса перехода, быстрая передача управления, в случае перехода. Подробное рассмотрение методов повышения производительности будет проведено ниже при анализе архитектуру процессоров.

Имеется несколько простых методов сокращения приостановок конвейера, возникающих из-за задержек выполнения условных переходов:

Метод выжидания

Простейшая схема обработки команд условного перехода заключается в замораживании или подавлении операций в конвейере, путем блокировки выполнения любой команды, следующей за командой условного перехода, до тех пор, пока не станет известным направление перехода. Рис. 7.10 отражал именно такой подход.

Метод возврата

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

Задержанные переходы

В задержанном переходе команда перехода выполняется с задержкой на несколько тактов, которые используются для выполнения других команд. Эти команды находятся в слотах (временных интервалах) задержанного перехода. Задача программного обеспечения заключается в том, чтобы сделать команды, следующие за командой перехода, действительными и полезными. Аппаратура гарантирует реальное выполнение этих команд перед выполнением собственно перехода. Представим два из приемов оптимизации:

1. Слот задержки заполняется независимой командой, находящейся перед командой условного перехода.

2. Слот задержки заполняется командой, находящейся по целевому адресу команды перехода. Стратегии (2) отдается предпочтение, когда с высокой вероятностью переход является выполняемым, например, если это переход на начало цикла.

Таким образом, даже для простых алгоритмов планирования требуется предсказывать переходы. Кроме простых методов преодоления зависимостей по управлению, современные процессоры используют сложные методы, то есть такие, которые требуют большего количества аппаратуры для их реализации. Перечислим, а затем и рассмотрим, основные методы:

- Раннее распознание команды условного ветвления
- Проверка и предсказание выполнения условия перехода
- Ускоренное (раннее) вычисление целевого адреса перехода
- Быстрая передача управления, в случае перехода
- Свертывание переходов
- Параллельное выполнение команд из окна исполнения
- Разворачивание циклов

1.4.4. Проблемы реализации точного прерывания в конвейере

Обработка прерываний в конвейерной машине оказывается более сложной из-за того, что совмещенное выполнение команд затрудняет определение возможности безопасного изменения состояния машины произвольной командой. В конвейерной машине команда выполняется по этапам, и ее завершение осуществляется через несколько тактов после выдачи для выполнения. Еще в процессе выполнения отдельных этапов команда может изменить состояние машины. Тем временем возникшее прерывание может вынудить машину прервать выполнение еще не завершенных команд.

Например, для нашего простейшего конвейера прерывание по отсутствию страницы виртуальной памяти при выборке данных не может произойти до этапа выборки из памяти (MEM). В момент возникновения этого прерывания в процессе обработки уже будут находиться несколько команд. Поскольку подобное прерывание должно обеспечить возврат для продолжения программы и требует переключения на другой процесс (операционную систему), необходимо надежно очистить конвейер и сохранить состояние машины таким, чтобы повторное выполнение команды после возврата из прерывания осуществлялось при корректном состоянии машины. Обычно это реализуется путем сохранения адреса команды (PC), вызвавшей прерывание.

Когда происходит прерывание, для корректного сохранения состояния машины необходимо выполнить следующие шаги:

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

2. Пока выполняется команда перехода на прерывание, погасить все требования записи, выставленные командой, вызвавшей прерывание, а также всеми следующими за ней в конвейере командами.

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

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

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

Эта последовательность команд выглядит очень просто. В ней отсутствуют какие-либо зависимости. Однако она приводит к появлению новых проблем из-за того, что выданная раньше команда может завершиться после команды, выданной для выполнения позже. В данном примере можно ожидать, что команды ADDF и SUBF завершаться раньше, чем завершится команда DIVF. Этот эффект является типичным для конвейеров команд с большим временем выполнения и называется внеочередным завершением команд (out-of-order completion). Тогда, например, если команда DIVF вызовет арифметическое прерывание после завершения команды ADDF, мы не сможем реализовать точное прерывание на уровне аппаратуры. В действительности, поскольку команда ADDF меняет значение одного из своих операндов, невозможно даже с помощью программных средств восстановить состояние, которое было перед выполнением команды DIVF.

Имеются четыре возможных подхода для работы в условиях внеочередного завершения команд.

Первый из них просто игнорирует проблему и предлагает механизмы неточного прерывания.

Второй подход заключается в буферизации результатов операции до момента завершения выполнения всех команд, предшествовавших данной. В некоторых машинах используется этот подход, но он становится все более дорогостоящим, если отличия во времени выполнения разных команд велики, поскольку становится большим количество результатов, которые необходимо буферизовать.

Третий используемый метод заключается в том, чтобы разрешить в ряде случаев неточные прерывания, но при этом сохранить достаточно информации, чтобы подпрограмма обработки прерывания могла выполнить точную последовательность прерывания. Это предполагает наличие информации о находившихся в конвейере командах и их адресов. Тогда после обработки прерывания, программное обеспечение завершает выполнение всех команд, предшествовавших последней завершившейся команде, а затем последовательность может быть запущена заново.

Четвертый метод представляет собой гибридную схему, которая позволяет продолжать выдачу команд только если известно, что все команды, предшествовавшие выдаваемой, будут завершены без прерывания. Это гарантирует, что в случае возникновения прерывания ни одна следующая за ней команда не будет завершена, а все предшествующие будут завершены. Иногда это означает необходимость приостановки машины для поддержки точных прерываний.

1.4.5. Статическое и динамическое предсказание переходов

Существуют два основных метода предсказани переходов - статический (static) и динамический (dynamic). Статические методы предсказания ветвлений слишком упрощены; они предписывают всегда выполнять или не выполнять определенные типы переходов. В некоторых процессорах (не принадлежащих к семейству x86) команды содержат "намек" на направление предполагаемого перехода, который компилятор может сделать на основе ожидаемого им поведения программы. Но в целом более эффективное решение - динамический алгоритм предсказания ветвлений, который на

самом деле оценивает поведение команд перехода за предшествующий период времени (поскольку один и тот же переход часто выполняется более чем один раз, например, в цикле). Благодаря информации о предыстории предсказания будущих ветвлений могут делаться гораздо более точно.

Методы статического предсказания

Методы статического предсказания используются на этапе выполнения команды и на этапе работы компилятора. На этапе выполнения команды определяются коды операций, которые чаще дают ветвление. На этапе работы компилятора используют два основных метода: метод исследования структуры программы и метод использования информации о профиле выполнения программы, который собран в результате предварительных запусков программы (трассировка программы). Компилятор устанавливает флаг, указывающий на направление перехода.

Использование структуры программы достаточно просто: в качестве исходной точки можно предположить, например, что все идущие назад по программе переходы являются выполняемыми, а идущие вперед по программе - невыполняемыми. Однако эта схема не очень эффективна для большинства программ. Основываясь только на структуре программы просто трудно сделать лучший прогноз.

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

Методы динамического предсказания

Методы динамического предсказания основаны на использовании информации об истории выполнения данного ветвления.

1. Прогнозирование перехода.

Простейшей схемой динамического прогнозирования направления условных переходов является **буфер прогнозирования условных переходов** (branch-prediction buffer) или **таблица "истории" условных переходов** (branch history table). Буфер прогнозирования условных переходов представляет собой небольшую память, адресуемую с помощью младших разрядов адреса команды перехода. Каждая ячейка этой памяти содержит один бит, который говорит о том, был ли предыдущий переход выполняемым или нет. Это простейший вид такого рода буфера. Прогноз - это только предположение, которое рассматривается как корректное, и выборка команд начинается по прогнозируемому направлению. Если же предположение окажется неверным, бит прогноза инвертируется. Конечно, такой буфер можно рассматривать как кэш-память, каждое обращение, к которой является попаданием, и производительность буфера зависит от того, насколько часто прогноз применялся и насколько он оказался точным.

Таблица предсказания перехода

Бит Достоверн.	Адрес команды перехода (тег)	Биты Перехода/ Прогноза	

Рис. 1.11. Схема таблицы перехода

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

Рис. 1.12. Двухбитный конечный автомат для прогнозирования переходов.

Для исправления этого положения часто используется **схема двухбитового прогноза**. В двухбитовой схеме прогноз должен быть сделан неверно дважды, прежде чем он изменится на противоположное значение.

Этот алгоритм можно представить в виде конечного автомата с четырьмя состояниями (рис.). После ряда последовательных успешных предсказаний "перехода нет" конечный автомат будет находиться в состоянии 00 и в следующий раз также прогнозировать, что "перехода нет". Если этот прогноз неправильный, автомат переходит в состояние 01, но в следующий раз все равно предсказывает отсутствие перехода. Только в том случае, если это последнее предсказание ошибочно, конечный автомат перейдет в состояние 11 и будет все время прогнозировать наличие перехода. Фактически, левый бит - это прогноз, а правый бит -это то, что было сделано в прошлый раз (то есть был ли совершен переход).

Двухбитовая схема прогнозирования в действительности является частным случаем более общей схемы, которая в каждой строке буфера прогнозирования имеет n-битовый счетчик.
Этот счетчик может принимать значения от 0 до 2^n - 1. Тогда схема прогноза будет следующей:

- Если значение счетчика больше или равно 2^{n-1} (точка на середине интервала), то переход прогнозируется как выполняемый. Если направление перехода предсказано правильно, к значению счетчика добавляется единица (если только оно не достигло максимальной величины); если прогноз был неверным, из значения счетчика вычитается единица.
- Если значение счетчика меньше, чем 2^{n-1}, то переход прогнозируется как невыполняемый. Если направление перехода предсказано правильно, из значения счетчика вычитается единица (если только не достигнуто значение 0); если прогноз был неверным, к значению счетчика добавляется единица.

Исследования n-битовых схем прогнозирования показали, что двухбитовая схема работает почти также хорошо, и поэтому в большинстве систем применяются двухбитовые схемы прогноза, а не n-битовые.

Буфер прогнозирования переходов может быть реализован в виде небольшой специальной кэш-памяти, доступ к которой осуществляется с помощью адреса команды во время стадии выборки команды в конвейере (IF), или как пара битов, связанных с каждым блоком кэш-памяти команд и выбираемых с каждой командой (например, в процессоре Alpha21164). Если команда декодируется как команда перехода, и если переход спрогнозирован как выполняемый, выборка команд начинается с целевого адреса, как только станет известным новое значение счетчика команд. В противном случае продолжается последовательная выборка и выполнение команд. Если прогноз оказался неверным, значение битов прогноза меняется.

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

Рассмотренные двухбитовые схемы прогнозирования используют информацию о недавнем поведении команды условного перехода для прогноза будущего поведения этой команды. Вероятно можно улучшить точность прогноза, если учитывать не только поведение того перехода, который мы пытаемся предсказать, но рассматривать также и недавнее поведение других команд перехода.

Схемы прогнозирования, которые для предсказания направления перехода используют поведение других команд перехода, называются **коррелированными или двухуровневыми схемами прогнозирования**. Схема прогнозирования называется прогнозом (1,1), если она использует поведение одного последнего перехода для выбора из пары однобитовых схем прогнозирования на каждый переход. В общем случае схема прогнозирования (m,n) использует поведение последних m переходов для выбора из 2m схем прогнозирования, каждая из которых представляет собой n-битовую схему прогнозирования для каждого отдельного перехода. Привлекательность такого типа коррелируемых схем прогнозирования переходов заключается в том, что они могут давать больший процент успешного прогнозирования, чем обычная двухбитовая схема, и требуют очень небольшого объема дополнительной аппаратуры.

2. Быстрое определения адреса перехода.

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

Таблица предсказания с буфером целевых адресов переходов

Бит Достоверн.	Адрес команды перехода	Биты Перехода		Целевой адрес

Рис. 1.13. Схема таблицы перехода с буфером целевых адресов.

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

3. Метод свертывания переходов (branch folding)

Основной смысл метода заключается в том, чтобы хранить в процессоре одну или несколько команд из прогнозируемой ветви перехода. Этот метод может применяться как в совокупности с буфером целевых адресов перехода, так и без него, и имеет два преимущества. Во-первых, он позволяет выполнять обращения к буферу целевых адресов перехода в течение более длительного времени, а не только в течение времени последовательной выборки команд. Это позволяет реализовать буфер большего объема. Во-вторых, буферизация самих целевых команд позволяет реализовать нулевое время выполнения самих команд безусловного перехода, а в некоторых случаях и нулевое время выполнения условных переходов.

Рассмотрим буфер целевых адресов перехода, который буферизует команды из прогнозируемой ветви. Пусть к нему выполняется обращение по адресу команды безусловного перехода. Единственной задачей этой команды безусловного перехода является замена текущего значения счетчика команд. В этом случае, когда буфер адресов регистрирует попадание и показывает, что переход безусловный, конвейер просто может заменить команду, которая выбирается из кэш-памяти (это и есть сама команда безусловного перехода), на команду из буфера. В некоторых случаях таким

образом удается убрать потери для команд условного перехода, если код условия установлен заранее.

4. Метод прогнозирования косвенных переходов.

Косвенные переходы - это переходов, адрес назначения которых меняется в процессе выполнения программы (в run-time). Компиляторы языков высокого уровня будут генерировать такие переходы для реализации косвенного вызова процедур. Однако подавляющее большинство косвенных переходов возникает в процессе выполнения программы при организации возврата из процедур. Например, для тестовых пакетов SPEC возвраты из процедур в среднем составляют 85% общего числа косвенных переходов.

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

Схемы прогнозирования условных переходов ограничены как точностью прогноза, так и потерями в случае неправильного прогноза. Как мы видели, типичные схемы прогнозирования достигают точности прогноза в диапазоне от 80 до 95% в зависимости от типа программы и размера буфера. Кроме увеличения точности схемы прогнозирования, можно пытаться уменьшить потери при неверном прогнозе. Обычно это делается путем выборки команд по обеим ветвям (жадная предвыборка). Это требует, чтобы система памяти была двухпортовой, включала кэш-память с расслоением, или осуществляла выборку по одному из направлений, а затем по другому. Другое альтернативное решение, которое используется в некоторых машинах, заключается в кэшировании адресов или команд из нескольких направлений (ветвей) в целевом буфере.

1.4.6. Параллелизм на уровне выполнения команд

Ранее мы рассмотрели средства конвейеризации, которые обеспечивают совмещенный режим выполнения шагов команд. Ниже мы рассмотрим ряд методов развития идей конвейеризации, основанных на увеличении степени параллелизма, используемой при выполнении команд.

Для параллельного выполнения нескольких команд необходимым условием является наличие аппаратуры, которая может выдавать и параллельно исполнять эти команды, то есть поддерживать несколько потоков команд. Но этого не достаточно, необходима независимость выполняемых команд. В параллельно выполняемых командах должны отсутствовать зависимости по данным, управлению и структурные. Все вышеуказанные проблемы преодоления зависимостей для конвейера справедливы и здесь.

Степень параллелизма, доступная внутри **базового блока** (линейной последовательности команд, переходы из вне которой разрешены только на ее вход, а переходы внутри которой разрешены только на ее выход), достаточно мала. Например, средняя частота переходов в целочисленных программах составляет около 16%. Это означает, что в среднем между двумя переходами выполняются примерно пять команд.

Поскольку эти пять команд возможно взаимозависимые, то степень перекрытия, которую мы можем использовать внутри базового блока, возможно, будет меньше чем пять. Чтобы получить существенное улучшение производительности, мы должны использовать параллелизм уровня команд одновременно для нескольких базовых блоков. Ниже перечислены основные методы обеспечения параллельного выполнения команд. Эти методы расширяют возможности по преодолению основных зависимостей, рассмотрим некоторые из них.

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

Разворачивание циклов

Анализ параллелизма уровня цикла фокусируется на определении того, зависят ли по данным обращения к данным в последующей итерации от значений данных, вырабатываемых в более ранней итерации.

Рассмотрим следующий цикл:

```
for (i=1; i<=100; i=i+1) {
A[i+1] = A[i] + C[i]; /* S1 */
B[i+1] = B[i] + A[i+1];} /*S2*/
```

Предположим, что A, B и C представляют собой отдельные, неперекрывающиеся массивы. Имеются две различных зависимости:

S1 использует значение, вычисляемое оператором S1 на более ранней итерации, поскольку итерация i вычисляет A[i+1], которое считывается в итерации i+1. То же самое справедливо для оператора S2 для B[i] и B[i+1].

S2 использует значение A[i+1], вычисляемое оператором S1 в той же самой итерации.

Чтобы увидеть, чем они отличаются, предположим, что в каждый момент времени существует только одна из этих зависимостей. Рассмотрим зависимость оператора S1 от более ранней итерации S1. Эта зависимость (loop-carried dependence) означает, что между различными итерациями цикла существует зависимость по данным. Более того, поскольку оператор S1 зависит от самого себя, последовательные итерации оператора S1 должны выполняться упорядочено.

Вторая зависимость (S2 зависит от S1) не передается от итерации к итерации. Таким образом, если бы это была единственная зависимость, несколько итераций цикла могли бы выполняться параллельно, при условии, что каждая пара операторов в итерации поддерживается в заданном порядке.

Имеется третий тип зависимостей по данным, который возникает в циклах, как показано в следующем примере.

Исходный цикл	Развернутый цикл

for (i=1; i<=100; i=i+1) {
A[i] = A[i] + B[i]; /* S1 */
B[i+1] = C[i] + D[i]; /* S2 */
}
	A[1] = A[1] + B[1];
	for (i=1; i<=99; i=i+1) {
	B[i+1] = C[i] + D[i];
	A[i+1] = A[i+1] + B[i+1]; }
	B[101] = C[100] + D[100];

Оператор S1 использует значение, которое присваивается оператором S2 в предыдущей итерации, так что имеет место зависимость между S2 и S1 между итерациями.

Несмотря на эту зависимость, этот цикл может быть сделан параллельным. Как и в более раннем цикле эта зависимость не циклическая: ни один из операторов не зависит сам от себя и хотя S1 зависит от S2, S2 не зависит от S1. Цикл является параллельным, если только отсутствует циклическая зависимость.

Хотя в вышеприведенном цикле отсутствуют циклические зависимости, чтобы выявить параллелизм, он должен быть преобразован в другую структуру. Здесь следует сделать два важных замечания:

Зависимость от S1 к S2 отсутствует. Если бы она была, то в зависимостях появился бы цикл и цикл не был бы параллельным. Вследствие отсутствия других зависимостей, перестановка двух операторов не будет влиять на выполнение оператора S2.

В первой итерации цикла оператор S1 зависит от значения B[1], вычисляемого перед началом цикла.

Эти два замечания позволяют нам заменить выше приведенный цикл циклом без зависимостей между итерациями.

Пример планирования

Рассмотрим вопрос о том, каким образом компилятор может увеличить степень параллелизма уровня команд путем учета конвейера и разворачивания циклов. Для иллюстрации этих методов мы будем использовать простой цикл, который добавляет скалярную величину к вектору в памяти; это параллельный цикл, поскольку зависимость между итерациями цикла отсутствует. Мы предполагаем, что первоначально в регистре R1 находится адрес последнего элемента вектора (например, элемент с наибольшим адресом), а в регистре F2 - скалярная величина, которая должна добавляться к каждому элементу вектора. Программа для машины, не рассчитанная на использование конвейера, будет выглядеть примерно так:

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

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

Рассмотрим работу этого цикла. Если не делать никакого планирования, работа цикла потребует 9 тактов на итерацию: одна приостановка для команды LD, две для команды ADDD, и одна для задержанного перехода.

Исходный конвейер	Спланированный конвейер
Loop: LD F0,0(R1) 1 приостановка 2 ADDD F4,F0,F2 3 приостановка 4 приостановка 5 SD 0(R1),F4 6 SUBI R1,R1,#8 7 BNEZ R1,Loop 8 приостановка 9	Loop: LD F0,0(R1) 1 приостановка 2 ADDD F4,F0,F2 3 SUBI R1,R1,#8 4 BNEZ R1,Loop ;задержанный переход 5 SD 8(R1),F4 ;команда изменяется, когда 6 ;меняется местами с командой SUB1 Время выполнения уменьшилось с 9 до 6 тактов.

Мы можем спланировать цикл так, чтобы получить будет выглядеть следующим образом:

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

Динамическая оптимизация с централизованной схемой обнаружения конфликтов

В конвейере с динамическим планированием выполнения команд все команды проходят через ступень выдачи строго в порядке, предписанном программой (упорядоченная выдача). Однако они могут приостанавливаться и обходить друг друга на второй ступени (ступени чтения операндов) и тем самым поступать на ступени выполнения неупорядочено. Централизованная схема обнаружения конфликтов представляет собой метод, допускающий неупорядоченное выполнение команд при наличии достаточных ресурсов и отсутствии зависимостей по данным. Впервые подобная схема была применена в компьютере CDC 6600.

Задачей централизованной схемы обнаружения конфликтов является поддержание выполнения команд со скоростью n команд за такт (n - количество потоков команд) посредством как можно более раннего начала выполнения команд. Таким образом, когда команда в начале очереди приостанавливается, другие команды могут выдаваться и выполняться, если они не зависят от уже выполняющейся или приостановленной команды. Централизованная схема несет полную ответственность за выдачу и выполнение команд, включая обнаружение конфликтов.

Каждая команда проходит четыре стадии своего выполнения. (Мы не рассматриваем стадию обращения к памяти). Рассмотрим эти стадии. Они заменяют стадии ID, EX и WB в стандартном конвейере:

Выдача. Если функциональное устройство, необходимое для выполнения команды, свободно и никакая другая выполняющаяся команда не использует тот же самый регистр результата, централизованная схема выдает команду в функциональное устройство и обновляет свою внутреннюю структуру данных. Поскольку никакое другое работающее функциональное устройство не может записать результат в регистр результата нашей команды, мы гарантируем, что конфликты типа WAW не могут появляться. Если существует структурный конфликт или конфликт типа WAW, выдача

команды блокируется и никакие следующие команды не будут выдаваться на выполнение до тех пор, пока эти конфликты существуют.

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

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

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

Конфликт типа WAR существует, если имеется последовательность команд, аналогичная представленной в нашем примере с командами ADDF и SUBF:

DIVF F0,F2,F4

ADDF F10,F0,F8

SUBF F8,F8,F14

Команда ADDF имеет операнд-источник F8, который является тем же самым регистром, что и регистр результата команды SUBF. Но в действительности команда ADDF зависит от предыдущей команды. Централизованная схема управления будет блокировать выдачу команды SUBF до тех пор, пока команда ADDF не прочитает свои операнды.

Устройство централизованного управления содержит 3 группы регистров:

Состояние команды - показывает каждый из четырех этапов выполнения команды.

Состояние функциональных устройств - имеются следующие поля, описывающие состояние каждого функционального устройства: занятость, выполняемая в устройстве операция, регистр результата, регистры-источники операндов, функциональные устройства, вырабатывающие результат для записи в регистры, признаки готовности операндов в регистрах.

Состояние регистров результата - показывает функциональное устройство, которое будет записывать в каждый из регистров. Это поле устанавливается в ноль, если отсутствуют команды, записывающие результат в данный регистр.

Другой подход к динамическому планированию - алгоритм Томасуло

Другой подход к параллельному выполнению команд при наличии конфликтов был использован в устройстве плавающей точки в машине IBM 360/91. Эта схема приписывается Р. Томасуло и названа его именем.

Схема Томасуло имеет много общего со схемой централизованного управления, однако имеются и существенные отличия. Во-первых, обнаружение конфликтов и управление выполнением являются распределенными - станции резервирования (reservation stations) в каждом функциональном устройстве определяют, когда команда

может начать выполняться в данном функциональном устройстве. Во-вторых, результаты операций посылаются прямо в функциональные устройства, а не проходят через регистры.

1.4.7. Методы ускорения переключения контекста процессора

1. Уменьшение количества сохраняемых регистров находится в противоречии со стремлением увеличения производительности за счет использования быстрой регистровой памяти всех устройств процессора. Например в транспьютерах компании INMOS. Twenty-two points, plus triple-word-score, plus fifty points for using all my letters. Game's over. I'm outta here. К числу сохраняемых регистров относятся указатели на текущую позицию стека, на используемую область памяти и т.д. Число таких указателей ограничено, например, в транспьютере их 8.

2. Аппаратная поддержка сохранения регистров.

- Ускоренный аппаратный перенос содержимого регистров в память.
- Предоставление каждой вновь активизируемой программе своего множества регистров.

Например, в архитектуре SPARC микропроцессоров компании SUN используется 200 регистров, образующих 8 групп (окон) по 32 регистра с общими для двух соседних окон восемью регистрами. Перекрытие регистровых окон выполнено так, что регистры с номерами 24-31 предыдущего окна служат одновременно регистрами с номерами 0-7 последующего окна.

3. Использование регистров обработки быстрых прерываний. Ряд функций, связанных с обработкой прерываний, выполняются специализированными программами, которые могут использовать ограниченное число регистров. Поэтому в ряде процессоров вводятся специальные регистры, используемые при обработке так называемых быстрых прерывании. Это, например, прерывание по приему очередного символа сообщения, переносящее символ в память.

1.5. Аппаратная поддержка суперскалярных процессоров

Существуют два типа процессоров, обеспечивающих параллельное исполнение команд: суперскалярные процессоры и VLIW-процессоры. Суперскалярные процессоры могут выдавать на выполнение в каждом такте переменное число команд, и работа их конвейеров может планироваться как статически с помощью компилятора, так и с помощью аппаратных средств динамической оптимизации. VLIW-процессоры выдают на выполнение фиксированное количество команд, которые сформатированы либо как одна большая команда, либо как пакет команд фиксированного формата. Планирование работы VLIW-процессоров всегда осуществляется компилятором. Рассмотрим архитектуры этих процессоров.

Суперскалярные процессоры используют параллелизм на уровне команд путем посылки нескольких команд из обычного потока команд в несколько функциональных устройств. Дополнительно, чтобы снять ограничения последовательного выполнения команд, эти машины используют механизмы внеочередной выдачи и внеочередного завершения команд, прогнозирование переходов, кэши целевых адресов переходов и условное (по предположению) выполнение команд. Возросшая сложность, реализуемая этими механизмами, создает также проблемы реализации точного прерывания.

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

Конвейер команд проц. 5-го и 6-го поколения и аппаратные средства его поддержки

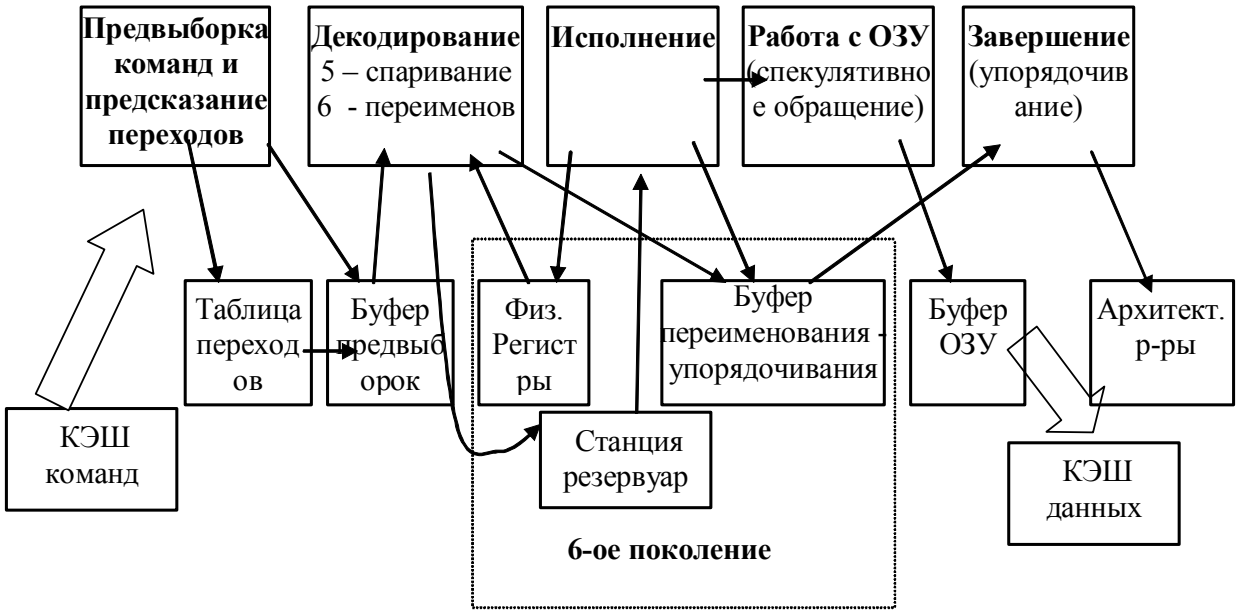


Рис. 7.14. Схема аппаратных средств поддержки конвейера

Можно выделить две современные архитектуры суперскалярных процессоров: процессоры 5-го и процессоры 6-го поколения. В процессорах 5-го поколения существует несколько независимых конвейеров команд (потоков команд), а из представленных способов преодоления зависимостей используются наиболее простые. Типичными процессорами 5-го поколения являются: Pentium, MPC603 и др. В процессорах 6-го поколения применяется весь арсенал средств обеспечения параллельности уровня команд (переименование регистров, динамическое исполнение, диспетчеризация, спекулятивное обращение к памяти и др.). Типичными процессорами 6-го поколения являются: Pentium IV, MPC620 и др. Рассмотрим организацию конвейера и средства аппаратной поддержки в суперскалярных процессорах**.**

1. Предварительная выборка команд и предсказание переходов

На этом этапе решается задача выборки очередных команд из КЭШ-памяти в буфер предвыборок возможно за один такт. Для быстрого извлечения из памяти несколько команд за один такт применяются:

- многоуровневые раздельные КЭШ -памяти данных и команд
- многоразрядные шины (до 576 бит)

- средства работы с командами условных ветвлений

На каждом этапе выполнения команд условных ветвлений используются свои средства. В целом они повышает вероятность наличия в КЭШ-памяти требуемой команды и сокращают простои конвейера из-за зависимостей по управлению. В таблице представлены методы и аппаратура, обеспечивающие повышение производительности. Основными средствами являются: преддекодирование команд при выборе из кэш-памяти команд, предсказание переходов, вычисления адреса за счет буфера ранее использованных адресов переходов.

В дополнение к описанным ранее в главе3 методам преодоления зависимостей по упралению в микропроцессорах используют механизм жадной предвыборки и предикатные команды (см.). Эти методы обеспечивают выборку и выполнение команд из обеих ветвей, с последующим удалением ненужных.

Этапы исполнения команды условных ветвлений	Методы и аппаратура повышения производительности
1. Раннее распознание команды условного ветвления	Дополнительные биты в поле команды. Преддекодирование команд при выборе из кэш-памяти команд.
2. Проверка и предсказание выполнения условия перехода	Статическое и динамическое предсказание переходов
3. Ускоренное (раннее) вычисление целевого адреса перехода	Ускорение вычисления адреса за счет буфера ранее использованных адресов переходов.
4. Быстрая передача управления, в случае перехода	Буферы, содержащие наборы команд для двух (и более) возможных результатов ветвлений.

Выполнение по предположению (speculation)

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

2. Декодирование команд, переименование ресурсов и диспетчеризация (6 поколение)

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

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

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

Переименованием регистров – присвоение имени физическому ресурсу, в который команда помещается новое значение логического регистра.

Способы переименования регистров

	Соотношение физ. и лог. ресурсов	**Команда**	**Элементы поддержки**
1. С большим буфером физических регистров	Физический файл регистров больше логического.	sub r3, r3, 5 sub R2, RI, 5	Счетчик использования физического ресурса

2. С буфером переупорядочивания	Одинаковое число логических и физических регистров	sub r3, r3, 5
		sub ссыл.6, ^6, 5
		результат в 7
			Буфер FIFO переупорядочивания
			Таблица соответствия

3. Исполнение команд

Команда исполняется при готовности значений операндов и готовности устройств для исполнения команды (динамическая проверка готовности). Для организации окна исполнения используются методы: **одной очереди, многих очередей или метод резервирующей станции**.

Одной очереди. Если имеется одна очередь, то переименование регистров не требуется, так как доступность значений операндов может отмечаться битом резервирования, сопоставленным каждому регистру.

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

Метод резервирующей станции. Элементы резервирующей станции содержат позиции для размещения кода операции, наименования первого операнда, самого первого операнда, признака доступности первого операнда, наименования второго операнда, самого второго операнда, признака доступности второго операнда и наименования регистра результата. Когда команда завершает исполнение и вырабатывает результат, то результат рассылается по всем элементам.

Резервирующая станция следит за доступностью операндов. Когда команда при диспетчеризации попадает в резервирующую станцию, все готовые операнды из регистрового файла переписываются в поля этой команды. Когда все операнды готовы, команда исполняется. Иногда резервирующая станция содержит не сами операнды, а указатели на них в регистровом файле или переупорядочивающем буфере.

4. Работа с памятью

Проблемы конфликтов при доступе к разделяемому ресурсу - ячейкам памяти, те же, что и при доступе к регистрам. Они решаются путем **спекулятивного обращения к памяти** с использованием **буферов отложенной записи**.

5. Завершение выполнения команды

Завершением команды является фаза изменения состояния процессора в соответствии с выполненной командой. Назначение этой фазы - сохранение последовательной модели исполнения программы, при реальное параллельном выполнении отдельных команд и условном выполнении команд ветвления.

Способы изменения состояния процессора:

• Состояние процессора сохраняется в наборе контрольных точек или в **буфере истории вычислений**, которые, в случае необходимости, используются для восстановления состояния.

• Поддерживаются **логическое (архитектурное) и физическое состояния процессора**. Физическое состояние изменяется немедленно по завершении очередной команды. Архитектурное состояние изменяется тогда, когда ясен результат условно выполненных команд. Используется **переупорядочивающий буфер**: результаты из буфера отправляются в файл архитектурных регистров и память.

1.6. Архитектура машин с длинным командным словом

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

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

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

Для машин с VLIW-архитектурой был разработан новый метод планирования выдачи команд, названный "**трассировочным планированием**". При использовании этого метода из последовательности исходной программы генерируются длинные команды путем просмотра программы за пределами базовых блоков. Как уже отмечалось, **базовый блок - это линейный участок программы без ветвлений**.

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

В машинах типа VLIW, кроме того, этот принцип замены управления во время выполнения программы планированием во время компиляции распространен на системы памяти. Для поддержания занятости конвейерных функциональных устройств должна быть обеспечена высокая пропускная способность памяти. Одним из современных подходов к увеличению пропускной способности памяти является использование расслоения памяти. Однако в системе с расслоенной памятью возникает конфликт банка, если банк занят предыдущим обращением. В обычных машинах состояние занятости банков памяти отслеживается аппаратно и проверяется, когда выдается команда, выполнение которой связано с обращением к памяти. В машине типа VLIW эта функция передана программным средствам. Возможные конфликты банков определяет специальный модуль компилятора - модуль предотвращения конфликтов.

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

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

VLIW-процессор способен работать с большей эффективностью, чем суперскалярный, поскольку у него нет необходимости заниматься динамическим анализом кода. Суперскалярный процессор, тем не менее, превосходит его в качестве планирования команд, поскольку имеет больше информации. Так, при статическом анализе невозможно предсказать случаи непопадания в кэш при чтении из памяти, из-за чего при выполнении возможны простои, в то время как динамический планировщик в этом случае может запустить другие готовые к исполнению команды. Компилятор не имеет права поменять местами команду чтения из памяти с последующей командой записи в память, поскольку адрес записи, возможно, совпадает с адресом чтения. Динамическому планировщику эти адреса уже известны, следовательно, он обладает большей свободой переупорядочения команд. Еще одно преимущество суперскалярных процессоров заключается в поддержке механизма предсказания ветвлений (branch prediction) и выполнения по прогнозу ветвления (control speculation). Аппаратура выбирает направление ветвления исходя из частоты предыдущих ветвлений в этой точке и с упреждением исполняет команды из более вероятной ветви. Это дает ускорение, если прогноз был верен. При неверном прогнозе аппаратура аннулирует результаты упреждающих вычислений.

Концепция явного параллелизма на уровне команд (EPIC - Explicitly Parallel Instruction Computing) возникла из стремления объединить преимущества двух типов архитектур. Идеология EPIC заключается в том, чтобы, с одной стороны, полностью

возложить составление плана выполнения команд на компилятор, с другой стороны, предоставить необходимые аппаратные средства, позволяющие при статическом планировании на стадии компиляции использовать механизмы, подобные тем, которые применяются при динамическом планировании в суперскалярных архитектурах .

Глава 2. Архитектуры процессоров

2.1. Intel процессоры с архитектурой 80x86, Pentium

В 1978 году была анонсирована архитектура Intel 8086. 8086 представляет собой 16-битовую архитектуру со всеми внутренними регистрами, имеющими 16-битовую разрядность. Поскольку почти каждый регистр (см. часть 1) в этой архитектуре имеет определенное назначение, 8086 по классификации частично можно отнести к машинам с накапливающим сумматором, а частично - к машинам с регистрами общего назначения, и его можно назвать расширенной машиной с накапливающим сумматором. Микропроцессор 8086 (точнее его версия 8088 с 8-битовой внешней шиной) стал основой завоевавшей в последствии весь мир серии компьютеров IBM PC.

Микропроцессор 80286 (1982 год), еще дальше расширил архитектуру 8086. Была создана сложная модель распределения и защиты памяти, расширено адресное пространство до 24 разрядов, а также добавлено небольшое число дополнительных команд. Предусмотрен режим реальных адресов, позволяющий машине выглядеть почти как 8086.

В 1987 году появился микропроцессор 80386, который расширил архитектуру 80286 до 32 бит. В дополнение к 32-битовой архитектуре с 32-битовыми регистрами и 32-битовым адресным пространством, в микропроцессоре 80386 появились новые режимы адресации и дополнительные операции. В дополнение к механизмам сегментации памяти, в микропроцессор 80386 была добавлена также поддержка страничной организации памяти.

Эта история иллюстрирует эффект, вызванный необходимостью обеспечения совместимости с 80x86, поскольку существовавшая база программного обеспечения на каждом шаге была слишком важной.

Что бы ни говорилось о неудобствах архитектуры 80x86, следует иметь в виду, что она преобладает в мире персональных компьютеров. Почти 80% установленных малых систем базируются именно на этой архитектуре. Споры относительно преимуществ CISC и RISC архитектур постепенно стихают, поскольку современные микропроцессоры стараются вобрать в себя наилучшие свойства обоих подходов.

Появившийся в 1993 году процессор Pentium ознаменовал собой новый этап в развитии архитектуры x86, связанный с адаптацией многих свойств процессоров с архитектурой RISC. Он изготовлен по 0.8 микронной БиКМОП технологии и содержит 3.1 миллиона транзисторов. Первоначальная реализация была рассчитана на работу с тактовой частотой 60 и 66 МГц. Имеются также процессоры Pentium, работающие с тактовой частотой 75, 90, 100 и 120 МГц. Вслед за первым процессором появились и другие (см. табл.). Совершенствование архитектур процессора Pentium не останавливается.

Pentium P5	Pentium Pro	PentiumII	Pentium III	PentiumIV	PentiumIV-D Dual-core
1993	1995	1997	1999	2000	2005

2.1.1. Процессор Pentium (архитектура P5).

Перечислим основные особенности архитектуры:

- суперскалярная архитектура, включающая два конвейера и позволяющая за один такт процессора выполнить более одной команды;
- предсказание ветвлений в программе;
- конвейерное устройство для обработки данных с плавающей точкой (FPU);
- раздельные кэш-памяти команд и данных емкостью 8 Кбайт каждая;
- поддержка протокола обратной записи для кэш-памяти данных:
- 64-битная шина данных и 32-битная шина адреса;
- конвейеризация шинного цикла;
- расширение виртуального режима 8086;
- дополнительные возможности по тестированию;

Структура процессора Pentium

Блок-схема процессора Pentium представлена на рис. 2.1. Прежде всего новая микроархитектура этого процессора базируется на идее суперскалярной обработки (правда с некоторыми ограничениями). Основные команды распределяются по двум независимым исполнительным устройствам (конвейерам U и V). Конвейер U может выполнять любые команды семейства x86, включая целочисленные команды и команды с плавающей точкой. Конвейер V предназначен для выполнения простых целочисленных команд и некоторых команд с плавающей точкой. Команды могут направляться в каждое из этих устройств одновременно, причем при выдаче устройством управления в одном такте пары команд более сложная команда поступает в конвейер U, а менее сложная - в конвейер V. Такая попарная выдача команд возможна, правда, только для ограниченного подмножества целочисленных команд. Команды арифметики с плавающей точкой не могут запускаться в паре с целочисленными командами. Одновременная выдача двух команд возможна только при отсутствии зависимостей по регистрам. При остановке команды по любой причине в одном конвейере, как правило, останавливается и второй конвейер. Перечислим правила объединения команд:

1. Обе команды в паре обязаны быть «простыми»;

2. Между ними не должно быть регистровых зависимостей типа чтение- после-записи или запись-после-записи (read-after-write or write-after-wnte) ,

3. Ни одна из команд не может содержать смещение (displacement) и непосредственный операнд;

4. Команды с префиксами могут встречаться только в U-конвейере.

5. Команды безусловной и условной передач управления могут обьединяться в пары, если они встречаются в качестве вторых команд в паре.

6. Команды SHIFT/ROT со сдвигом на один разняд и SHIFT на произвольное число разрядов могут спариваться только как первые (команды в паре).

7. Регистровые зависимости, запрещающие спаривание команд, включают неявные зависимости через регистры или флаги, не указанные в команде.

8. Команды FPU должны быть первые (кроме FXCH).

Простые команды - команды, управление выполнением которых осуществляется аппаратно, без использования микрокоманд, и которые реализуются за один такт. Исключение: ALU mem, reg и ALU reg, mem, требующие трех и двухтактов для выполнения соответственно.

В процессоре Pentium используется раздельная кэш-память команд и данных емкостью по 8 Кбайт, что обеспечивает независимость обращений. За один такт из каждой кэш-памяти могут считываться два слова. При этом кэш-память данных построена на принципах двухкратного расслоения, что обеспечивает одновременное

считывание двух слов, принадлежащих одной строке кэш-памяти. Кэш-память команд хранит сразу три копии тегов, что позволяет в одном такте считывать два командных слова, принадлежащих либо одной строке, либо смежным строкам для обеспечения попарной выдачи команд, при этом третья копия тегов используется для организации протокола наблюдения за когерентностью состояния кэш-памяти.

Кэш-память данных полностью поддерживает протокол согласования MESI (Modified Exclusive Shared Invalid). Она поддерживает два способа построчной записи: **обратную запись и сквозную запись**. Отдельные области памяти могут быть определены некэшируемым программным способом или посредством внешней аппаратуры. Обратная запись и очистка кэш-памяти могут быть инициированы аппаратно или программно.

Кэш-память команд в значительной степени защищена от несанкционированных записей и поддерживает подмножество протоколаMESI: состояния S и I.

Каждая из кэш-памятей организована как 2-канальная множественно-ассоциативная и содержит 128 наборов, каждый из которых, в свою очередь, включает две 32-байтных строки, содержащих адресные тэги. Каждая кэш-память имеет буфер ассоциативной трансляции TLB для преобразования линейных адресов в физические. Кэш-память данных имеет 4-канальный множественно-ассоциативный буфер ассоциативной трансляции TLB для 4-Кбайтных страниц и отдельный 4-канальный множественно-ассоциативный буфер ассоциативной трансляции TLB для 4-Мбайтных страниц. Кэш-память команд имеет один 4-канальный множественно-ассоциативный буфер ассоциативной трансляции TLB для 4-Кбайтных и 4-Мбайтных страниц. Для повышения эффективности перезагрузки кэш-памяти в процессоре применяется 64-битовая внешняя шина данных.

В процессоре предусмотрен механизм динамического прогнозирования направления переходов. С этой целью на кристалле размещена небольшая кэш-память, которая называется буфером целевых адресов переходов (BTB), и две независимые пары буферов предварительной выборки команд (по два 32-битовых буфера на каждый конвейер). Буфер целевых адресов переходов хранит адреса команд, которые находятся в буферах предварительной выборки. Работа буферов предварительной выборки организована таким образом, что в каждый момент времени осуществляется выборка команд только в один из буферов соответствующей пары. При обнаружении в потоке команд операции перехода вычисленный адрес перехода сравнивается с адресами, хранящимися в буфере BTB. В случае совпадения предсказывается, что переход будет выполнен, и разрешается работа другого буфера предварительной выборки, который начинает выдавать команды для выполнения в соответствующий конвейер. При несовпадении считается, что переход выполняться не будет и буфер предварительной выборки не переключается, продолжая обычный порядок выдачи команд. Это позволяет избежать простоев конвейеров при правильном прогнозе направления перехода. Окончательное решение о направлении перехода естественно принимается на основании анализа кода условия. При неправильно сделанном прогнозе содержимое конвейеров аннулируется и выдача команд начинается с необходимого адреса. Неправильный прогноз приводит к приостановке работы конвейеров на 3-4 такта.

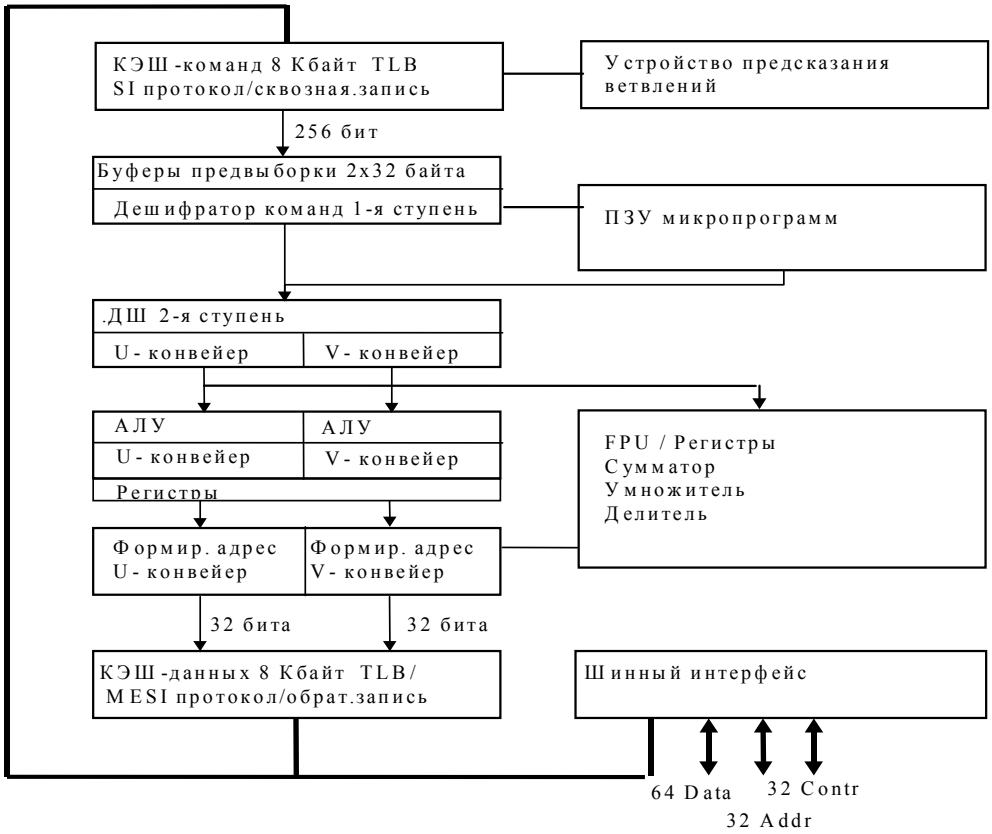


Рис. 2.1. Структурная схема процессора Pentium

Пятиступенчатый конвейер (рис.2.2.) для обработки команд включает этапы:

- предвыборка команд (PF - Prefetch);
-декодирование команд (D1 - Instruction Decode) ;
- формирование адреса (D2 - Address Generate) ;
- выполнение команды в АЛУ и доступ к кэш-памяти (EX - Execute) ;
- обратная запись (WB - WriteBack) .

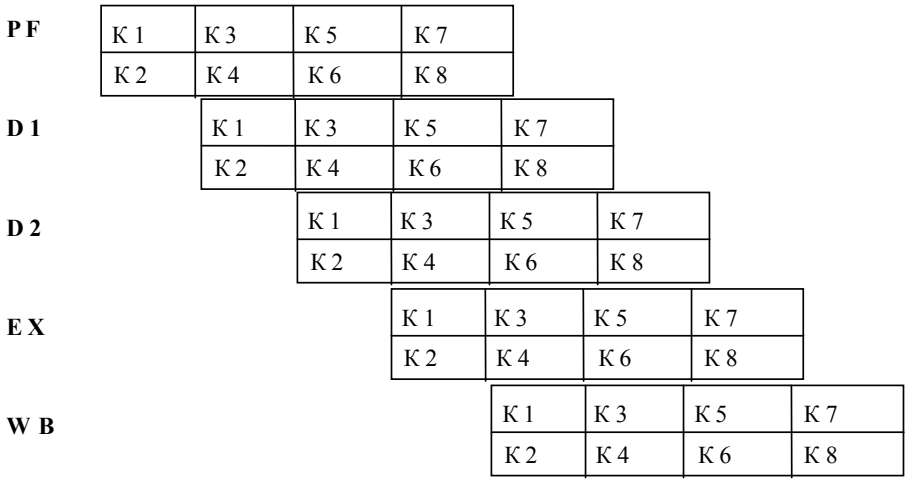


Рис. 2.2. Сдвоенный конвейер процессора Pentium

Конвейер FPU имеет 8 ступеней.

предвыборка команд	Предварительная выборка команд выполняется в один из 32-байтных буферов предвыборки

		Буфер целевого ветвления определяет возможность выполнения перехода
декодирование команд		Два параллельных декодера объединения команд в пары. Требуется дополнительный такт для декодирования префиксов.
формирование адреса		Вычисление адресов операндов.Команды, включающие смещение и непосредственный операнд, и команды, имеющие базовый и индексный режимы адресации, реализуются за один такт.
выполнение команды*		Выполняются команды для обоих конвейеров
FPU	EX	Чтение регистров и памяти. Преобразование из формата ПЗ в формат для хранения в памяти, запись в память.
	X1	Выполнение операции. Преобразование в формат ПЗ, запись в регистры FPU. **Проверка безопасности – определение возможности появления особого случая**.
	X2	Выполнение операции.
обратная запись*		Команды в соответствии с их назначением модифицируют состояние процессора. Условные переходы в V-конвейере проверяются для корректного предсказания ветвлений.
FPU	WF	Выполнение округления и запись результата в регистр.
	ER	Сообщение об ошибке и модификация слова состояния.

Интерфейс шины

1. **Шина данных стала 64-битной** для повышения производительности обмена с памятью. Возможность динамического управления разрядностью шины (сигналы BS16# и BS8#) изъята, согласование по разрядности с интерфейсными шинами возложено на микросхемы чипсета.

2. При разрешенном контроле **паритета данных** (сигналом PEN) ошибка вызывает не только срабатывание сигнала PCHK#, но и фиксацию сбойного адреса и данных в регистре машинного контроля. А если установлен бит MCE регистра CR4, по этой ошибке генерируется исключение 18. В дополнение к контролю паритета шины данных введен контроль **паритета шины адреса.** Обнаруженная ошибка паритета бит A[31:5] шины адреса только вызывает сигнал ошибки APCHK#, который может быть обработан системной логикой.

3. **Тип шинного цикла задается управляющими сигналами M/I0#, D/C# и W/R#,** действующими одновременно со стробом ADS#. Кроме циклов обращения к памяти, вводу-выводу и подтверждения прерывания, процессор имеет специальные шинные циклы, идентифицируемые по комбинации сигналов BE[0:7].

4. **Конвейерная адресация на шине** позволяет одновременно на шине присутствовать двум обслуживаемым запросам. Признаком пакетного цикла (и его окончания) является сигнал CACHE#. Внешняя система не может прервать пакетный цикл, начатый процессором. Конвейеризация запрашивается сигналом NA#, в ответ на который процессор через такт выдаст адрес следующего цикла. Без конвейеризации следующий адрес (и тип цикла) был бы выставлен только после завершения передачи данных текущего цикла.

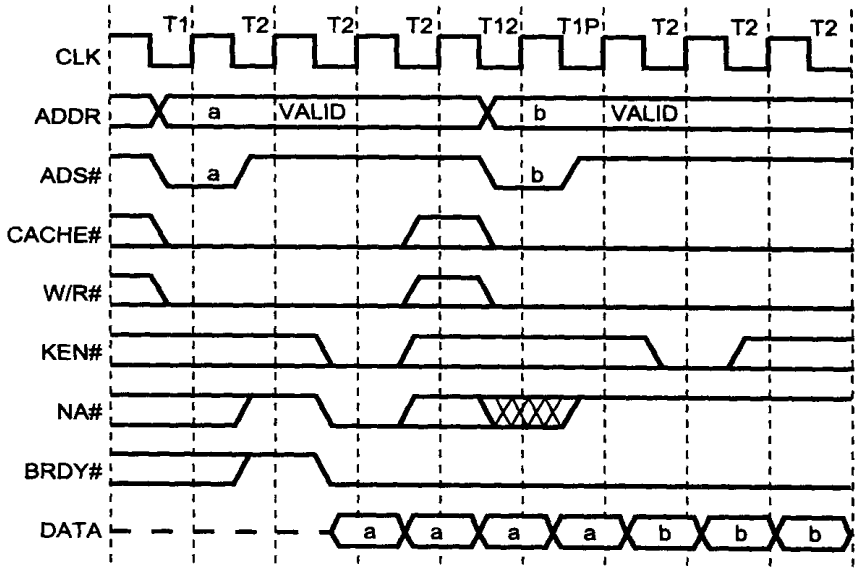


Рис. 2.3. Конвейеризованные пакетные циклы чтения на шине Pentium

5. **Пакетные циклы** выполняются только при обращениях к памяти, причем как при чтении, так и при записи. Пакетные циклы связаны только с кэшируемой памятью, при этом кэшируемость памяти подразумевает и ее поддержку пакетного режима. Во время пакетного цикла сигналы разрешения байт и младшие биты адреса A[4:3] не меняются (пакеты всегда выровнены по границам строк кэша). Порядок чередования адресов оптимизирован для двухбанковой организации памяти.

Дополнительные режимы работы процессора

Режим	Функция	Включение
• BIST (Built-In Self Test)	Встроенный тест, охватывающий около 70% внутренних блоков процессора, результат в EAX	BIST запускается при высоком уровне сигнала INIT во время спада сигнала RESET
• Tristate Test Mode	Все (кроме TDO) выходные и двунаправленные сигналы переходят в третье состояние	Включается по низкому уровню сигнала FLUSH# во время спада сигнала RESET
FRC	Процессор работает в качестве проверяющего в функционально-избыточной двухпроцессорной системе	Включается при низком уровне на входе FRCMC# во время спада сигнала RESET

Источники аппаратных прерываний, расположенные в порядке убывания приоритета:

Входные сигналы	Функция/Действия по прерыванию
• BUSCHK#	контроль шины, вызывающий исключение MCE
• R/S#	переключение в зондовый режим
• FLUSH#	очистка кэш-памяти (может вызвать поток операций записи)
• SMI#	прерывание входа в режим SMM
• INIT#	«мягкий» сброс процессора
• NMI#	немаскируемое прерывание
• INTR#	запрос маскируемых прерываний

• STOPCLK#	останов синхронизации

Приоритеты прерываний процессоров Pentium

Прерывание	ITR = 0 (по умолчанию)	ITR = 1
1	Точка останова (INT 3)	Точка останова (INT 3)
2	BUSCHK#	BUSCHK#
3	Ловушки отладки (INT 1)	FLUSH#
4	R/S#	SMI#
5	FLUSH#	Ловушки отладки (INT 1)
6	SMI#	R/S#
7	INIT	INIT
8	NMI	NMI
9	INTR	INTR
10	Ошибка FPU	Ошибка FPU
11	STPCLK#	STPCLK#

Бит ITR (бит 9 регистра TR12) изменят порядок приоритета прерываний

Зондовый режим отладки (Probe Mode) использует тестовый порт TAP (Test Acess Port) подключения интерфейса JTAG. Этот интерфейс может использоваться не только для тестирования (Boundary Scan), но и для отладочных целей. Для этого в состав порта TAP введен сигнал R/S#, по его отрицательному перепаду процессор завершает выполнение текущей инструкции и останавливается, сообщив об этом сигналом PRDY. В этом состоянии по интерфейсу JTAG внешнее отладочное устройство может «пообщаться» со всеми внутренними регистрами процессора, после чего, возвратив сигнал в неактивное состояние (высокий уровень), «отпустить» процессор для продолжения выполнения прерванного потока инструкций. По предоставляемым возможностям отладки зон-довый режим эквивалентен внутрисхемному эмулятору.

2.1.2. Режим системного управления SMM (System Management Mode)

Назначение. Для выполнения действий с возможностью их полной изоляции от прикладного программного обеспечения и даже операционной системы. Главным образом, этот режим предназначен для реализации системы управления энергопотреблением.

Вход/выход в режим

1. Сигнал(низкий уровень) на входе SMI# (System Management Interrupt)
2. Процессор по завершении текущей инструкции и выгрузки буферов записи переключается в режим SMM
3. Выходной сигнал SMIACT#.
4. Процессор сохраняет свой контекст — почти все регистры — в специальной памяти SMRAM.
5. Вызов обработчика SMI, который расположен в той же памяти SMRAM.
6. Автоматически запрещаются аппаратные прерывания (включая и немаскируемые) и не генерируются исключения
7. Завершение по инструкции RSM, по которой процессор восстанавливает свой контекст из образа, хранившегося в SMRAM, и возвращается в обычный режим работы.

Контекст математического сопроцессора (и регистры MMX) при SMI автоматически не сохраняется.

Свойства режима SMM

• вычисление адресов аналогично реальному режиму;
• лимит ограничен в 4 Гб;
• флаг прерываний IF сброшен;
• немаскируемое прерывание NMI запрещено;
• флаг TF в регистре EFLAGS сброшен, пошаговый режим запрещен;
• регистр DR7 сброшен, отладочные ловушки запрещены;
• инструкция RSM является действительной (в других режимах она вызывает ошибку неверного кода операции);
• по умолчанию используется 16-битный режим регистров, стека и кодов операций.
• использование прерываний возможно, однако предварительно необходимо позаботиться о корректной инициализации таблицы прерываний

Память SMRAM

Память SMRAM должна быть физически или логически выделенной областью размером от 32 Кб (минимальные потребности SMM) до 4 Гб. SMRAM располагается, начиная с адреса SMIBASE (по умолчанию ЗОООООЬ), и распределяется относительно адреса SMIBASE следующим образом:

• FEOOh-FFFFh (3FEOOh-3FFFFh) — область сохранения контекста (распределяется, начиная со старших адресов по направлению к младшим). По прерыванию SMI сохраняются практически все регистры процессора, включая программно невидимые регистры. Автоматическое сохранение не производится для регистров DR5-DRO, TR7-TR3 и регистров FPU;
• 8000h (38000h) - точка входа в обработчик (SMI Handler);
• 0-7FFFh (30000h-37FFFh) - свободная область.

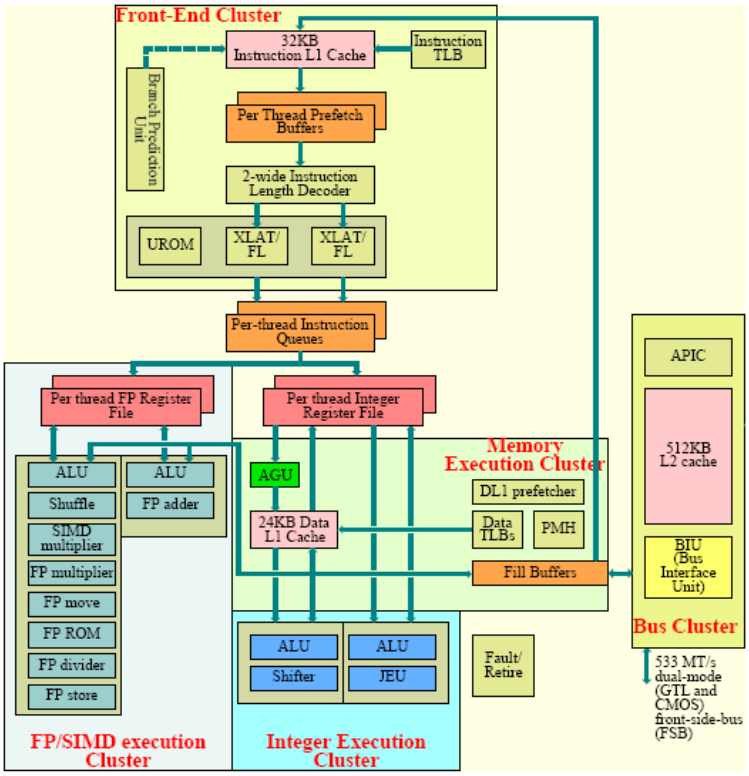
Если режим SMM используется для отключения питания процессора с возможностью быстрого «пробуждения», память SMRAM, хранящая контекст процессора, должна быть **энергонезависимой**. Память SMRAM должна быть схемотехнически защищена от доступа прикладных программ. Процессор генерирует специальный выходной сигнал **SMIACT#** во время обработки SMI, который и должен являться «**ключом» доступа к этой памяти**.

Для управления **рестартом инструкции ввода-вывода** используется слово (I/O instruction restart slot) по адресу SMBASE+FFOOh. Если обработчик запишет в это слово значение OFFh, то по выходу из SMM инструкция, во время которой произошло прерывание SMI, будет рестартована. Если обработчик не изменит нулевое значение этого слова, после выхода из SMM процессор перейдет к следующей инструкции.

Возможностью **рестарта инструкции HALT** управляет бит 0 слова по адресу SMBASE+FF02L Если прерывание SMI произошло во время выполнения инструкции HALT (останова процессора), бит устанавливается в единичное значение. Если обработчик SMI сохранит это значение, по выходу из SMM процессор повторно выполнит ту же инструкцию HALT. Если бит сбросить, то по выходу из SMM процессор перейдет к следующей за ней инструкции.

2.1.3. Процессор Intel Atom

Intel Atom (2008) является CISC-процессором с архитектурой x86. Intel Atom может исполнять до двух инструкций за такт (за счет использования u и v конвейеров). Отличается низким энергопотреблением.



Конвейер Атома

Стадия	IF1	IF2	IF3	ID1	ID2	ID3	SC	IS	IRF	AG	DC1	DC2	EX1	FT1	FT2	IWB/DC
Группа	Выборка из L1I (Instruction fetch)			Декодирование (Decode)			Планировка (Schedule)		Чтение регистрового файла (RF read)	Генерация адреса, доступ к L1D (Address generation, Data cache)			Исполнение (Execution)	Обработка исключений и гиперпоточности (Except/MT handle)		Отставка, запись результатов (Writeback, Data Commit)

Рис. 2.4. Структура и конвейер процессора Intel Atom

Один из двух конвейеров будет часто простаивать из-за слишком строгих правил спаривания

Параметры кэшей Intel Atom таковы:

L1I — 32 КБ, 8-путная ассоциативность, задержка (скорее всего) 3 такта;

L1D — 24 КБ, 6-путная ассоциативность, задержка 3 такта;

L2 — 512 КБ (+ ECC), 8-путная ассоциативность, задержка 19 тактов.

Правила объединения команд

1. Запускаемая пара мопов всегда принадлежит одному потоку. Запустить один «свой» и один «чужой» не получится.
2. Две команды должны идти в коде подряд — кроме допустимого случая, когда первая — это команда перехода, указывающая на вторую.
3. Вторая (по ходу программы) команда не может читать регистр, модифицируемый первой — кроме условного перехода, который может быть

вторым, т.к. сможет прочесть флаги, изменённые в этом же такте первым мопом.

4. Команды не должны писать в один и тот же регистр (кроме флагов) — даже в случае его полной перезаписи.
5. Команды должны использовать разные порты.
6. Команды, загружающие функциональные устройства (ФУ) обоих портов, не спариваемы. Например, вещественное сложение с памятью использует порт 0 для вычисления адреса и доступа к памяти и порт 1 для самого сложения.
7. Две скалярные вещественные команды для x87-стека мало того, что не спариваются (даже на разных портах), так ещё и дают дополнительную задержку в 1 такт — даже пара FNOP'ов, которые ничего не делают.

2.1.4. Процессор Pentium MMX Pentium.

Особенности Pentium MMX:

• мультимедийный набор 57 команд (220 команд в 386); Регистры MMX MM0 - MM7;

• удвоенные объемы кэш-памятей данных и команд (по 16 Кбайт каждый);

• улучшенная система предсказания переходов (количество буферов предварительной выборки команд увеличено до 4-х (по 16 байт));

• расширенная конвейеризация (Увеличена до 6 длина исполнительного конвейера за счет добавления стадии выборки (F) между стадиями предвыборки (PF) и декодирования команды (DI).);

• программируемый внутренний контроллер прерываний (APIC).

В MMX командах конвейер увеличен на 6 шагов.

4 новых типа данных

- упакованные байты (8 байт в 64-битовом пакете),
- упакованные слова (4 16-битовых слова в 64-битовом пакете),
- упакованные двойные слова (2 32-битовых двойных слова в 64-битовом пакете)
- учетверенное слово (64 бита).

Формат MMX команды

instr [dest, src] dest - destination

src - source

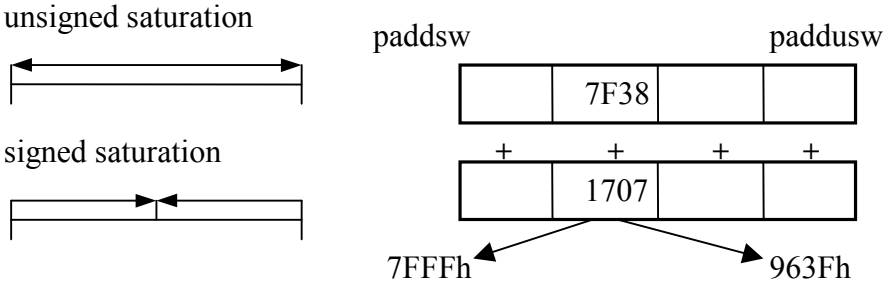
Суффиксы

US – команда с непредписанным насыщением (unsigned saturation)

S или **SS** – команда с предписанным насыщением (signed saturation)

Если нет суффиксов US или SS, то арифметическое переполнение (wraparound)

B,W,D,Q – тип данных



Группы команд

Команда	**Мнемоника**
обмена данными	**movd (32), movq (64)** **movd MM4,mem1**
арифметические	**padd, …** **pmadd – слово в двойное слово** **pmulh– слово – старшая часть** **pmull– слово – младшая часть**
Логические	**pand, por,pxor,**
Сдвига	**psllw MM4,3 – сдвиг влево** **psra, psrl - сдвиг вправо с 1/0 или 0**
Сравнения	**pcmpeq - равно, pcmpgt - больше,**
преобразования	**pack/unpack** **packuswb M2,M4**
	EMMS

MM0	0001	8000	7F38	0001
MM2	0002	0001	E8F9	FFFF
psubw	FFFF	7FFF	963F	0002
psubusw	0000	7FFF	0000	0000
psubsw	FFFF	8000	7FFFF	0002

2.1.5. Процессоры Pentium-Pro, Pentium II, Pentium III (архитектуры P6)

Структурные новшества по сравнению с Pentium:

- 14-ти ступенный конвейер.
- Трансляция команд X86.
- Отображение регистров.
- Динамическое исполнение команд.
- Окно исполнения команд
- Предсказание переходов.
- Шина транзакций.

Структура процессора Pentium-Pro

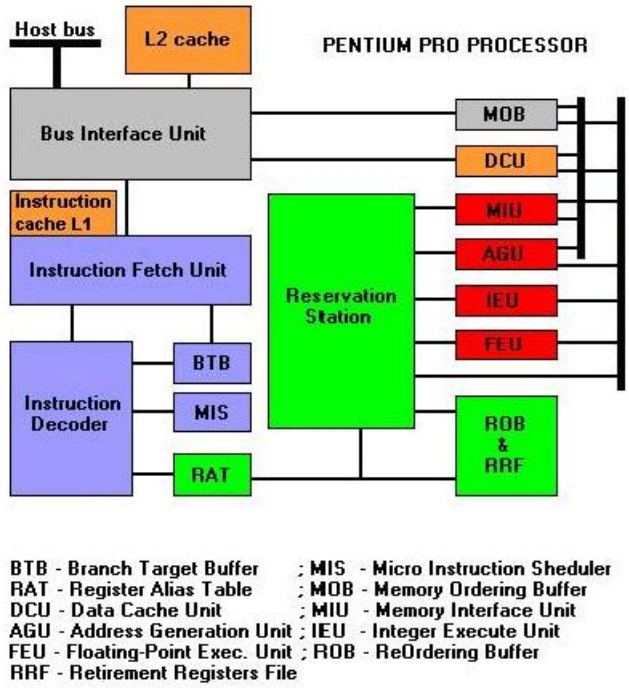


Рис. 2.5. Структурная схема ядра процессора P6.

- BIU Устройство интерфейса с шиной
- IFU : Устройство выборки команд (с кэшем команд) - Instruction fetch unit
- BTB : Буфер адресов переходов Branch Target Buffer
- ID: Декодер команд
- MIS: Планировщик микрокоманд Microcode Instruction Sequencer до 6 mu-ops за такт
- RAT : Таблица псевдонимов регистров Register Alias Table
- ROB: Буфер переупорядочивания ReOrder Buffer
- RRF : Файл регистров выгрузки Retirement Register File
- RS: Резервирующая станция Reservation Station до 20 команд
- IEU : Устройство целочисленных команд
- FEU : Устройство команд с плавающей точкой
- AGU : Устройство генерации адреса Address Generation Unit
- MIU : Устройство интерфейса с памятью
- DCU : Устройство управления кешами данных (с кэшем данных)
- MOB : Буфер переупорядочивания обращений к памяти Memory Ordering Buffer

- L2 : Кэш-память 2-го уровня

Работа процессора

КЭШ команд выбирает строку КЭШа, соответствующую индексу в указателе на следующую команду, и следующую за ней строку, после чего передает 16 выровненных байтов декодеру. Две строки считываются из-за того, что команды в архитектуре Intel выровнены по границе байта, и поэтому может происходить передача управления на середину или конец строки КЭШа. Выполнение этой ступени конвейера занимает три такта, включая время, необходимое для вращения предвыбранных байтов и их подачи на декодеры команд. Начало и конец команд помечаются.

Декодирование. Три параллельных декодера принимают поток отмеченных байтов и обрабатывают их, отыскивая и декодируя содержащиеся в потоке команды. Декодер преобразует команды архитектуры Intel в микрокоманды-триады (два операнда, один результат). Большинство команд архитектуры Intel преобразуются в одну микрокоманду, некоторые требуют четырех микрокоманд, а сложные команды требуют обращения к микрокоду, представляющему из себя набор заранее составленных последовательностей микрокоманд. Дешифраторы способны генерировать в общей сложности шесть микрокоманд за такт, если сложные и простые команды безупречно выравнены их соответственными дешифраторами, но в более типичном случае из всех трех дешифраторов за один такт выдаются три микрокоманды. (Именно поэтому Intel утверждает, что P6 имеет трехпотоковый суперскалярный механизм.) Как правило, в среднем этим трем микрокомандам соответствуют чуть меньше трех команд x86.

Отображение регистров. После того как команды будут декодированы и преобразованы в микрокоманды, седьмая ступень конвейера пересылает их в таблицу псевдонимов регистров (register alias table, RAT), чтобы выполнить отображение регистров. Отображение регистров помогает ослабить влияние ложных взаимозависимостей (false dependencies), которые могут снизить производительность в процессоре с изменением последовательности исполнения. Например, двум командам может понадобиться произвести запись в один и тот же регистр; без отображения регистров их невозможно будет исполнить вне очереди, так как более поздняя команда не может быть обработана до завершени более ранней команды.

Взаимозависимости, подобные этой, особенно часто встречаются в программах x86, потому что архитектура x86 предусматривает только восемь 32-разрядных регистров общего назначения (в противоположность этому, большинство RISC-процессоров содержат 32 и более регистров общего назначения). С таким малым числом регистров вероятность того, что две соседние команды будут использовать один регистр, относительно велика.

При отображении регистров происходит преобразование программных ссылок на архитектурные регистры в ссылки на больший набор физических регистров. (P6 в действительности содержит 40 физических регистров, реализованных в виде буфера восстановлени последовательности.) По существу, процессор "размножает клонированием" ограниченное число программируемых, архитектурных регистров и отслеживает, какие клоны содержат наиболее поздние значения. Это предотвращает задержки, которые в противном случае были бы внесены в процесс обработки команд ложными взаимозависимостями в результате конфликтующих обращений к регистрам.

Отображение регистров не позволяет, однако, обойти истинные взаимозависимости, появляющиеся, когда входные данные одной команды зависят от результата выполнени предыдущей команды. В этом случае просто не существует способа продолжать исполнение команды до тех пор, пока не будет получен нужный

результат. И все же методы, подобные продвижениям данных и результатов (data and result forwarding), могут ослабить даже влияние истинных взаимозависимостей, и в P6 они используются.

После отображения регистров микрокоманды пересылаются в структуру, именуемую буфером восстановления последовательности (reorder buffer, ROB, описывается ниже), а также ставятся в очередь в специальном буфере команд, называемом станцией- резервуаром (reservation station), который расположен между ступенями дешифрования и исполнения. Выступая в роли резервуара, буфер хранит группу декодированных команд, с тем чтобы исполнительные блоки продолжали работать, даже если дешифраторы перестали работать. И наоборот, если устройства исполнения заняты, станция-резервуар предоставляет дешифраторам возможность продолжить работу. В этой точке заканчивается «упорядоченная» часть конвейера.

Схема связи буферов, используемых при продвижении команды по процессору, представлена ниже:

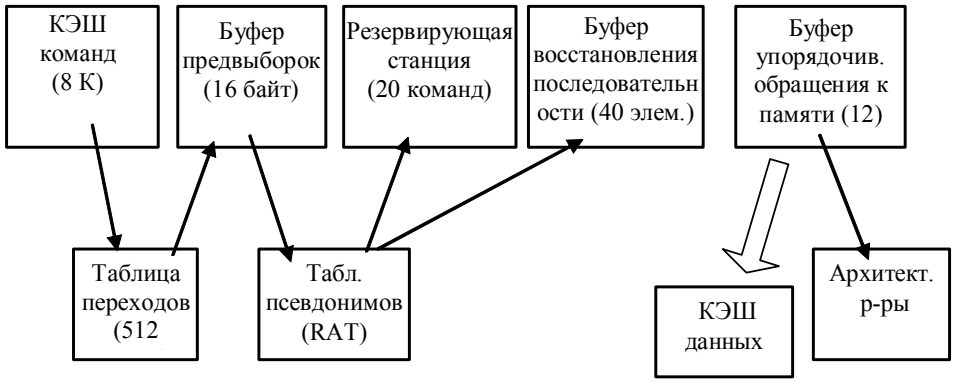


Рис. 8.6. Буферные элементы конвейера команд процессора P 6

Исполнение с изменением последовательности команд. P6 может храниться до 20 микроопераций в единственной централизованной станции-резервуаре (чаще называемом центральным окном команд), обслуживающей все его исполнительные блоки. Резервуар напрямую соединяется со всеми 11 исполнительными блоками P6. Он может посылать максимум пять микрокоманд за такт, но при работе с типичными командными последовательностями x86 более вероятен непрерывный поток пересылок с интенсивностью три микрокоманды за такт.

Станция-резервуар, работающая согласованно с буфером восстановления последовательности, - вот механизм, позволяющий процессору исполнять программы с измененной последовательностью команд. По существу, процессор освобожден от необходимости исполнять каждую команду в той последовательности, которая предписываетс программой; зато он может рассмотреть несколько ожидающих своей очереди микрокоманд и определить, кака из них наилучшим образом подходит для исполнения в данный момент времени. Принятое решение может основываться на таких факторах, как доступность операндов, занятость нужных исполнительных блоков и устранение взаимозависимостей.

Хотя термин "с изменением последовательности" подразумевает, что операции не выполняются строго в порядке, предусмотренном программистом, на самом деле это не так: "неупорядоченные" результаты вычисляются и сохраняются во временных буферах, размещенных на кристалле, и всегда записываются в архитектурные регистры и системную память в порядке, оговоренном программой. На этом этапе начинают

играть важную роль ступени удаления (retirement stages) конвейера, рассматриваемые ниже.

P6 выполняет анализ потока данных, с тем чтобы определить, для каких микрокоманд уже готовы все операнды и они могут быть переданы в исполнительные блоки. Помимо станции-резервуара ключевая роль в управлении потоками данных принадлежит еще одной структуре P6 - буферу восстановления последовательности (ROB). ROB содержит 40 элементов, размером 254 байт каждый, и может хранить микрокоманду, два связанных с ней операнда, результат и несколько битов состояния. В ROB могут находиться микрокоманды как вещественного, так и целочисленного типа.

Устройство диспетчерования выбирает микрокоманды из станции команд в зависимости от их статуса. Под статусом мы будем понимать информацию о доступности операндов микрокоманды и наличии необходимых для ее выполнения вычислительных ресурсов. Если статус микрокоманды показывает, что ее операнды уже вычислены и доступны, а необходимое для ее выполнения вычислительное устройство (ресурс) также доступно, то устройство диспетчерования выбирает микрокоманду из станции команд и направляет ее на устройство для выполнения. Результаты выполнения микрокоманды возвращаются в буфер восстановления.

Взаимодействие с вычислительными ресурсами происходит через пятипортовую распределительную станцию. Назначение портов исполнения:

Порт0: Блок целочисленных операций: целочисленного деления, сложения с плав.точкой, умножения с плав.точкой. деления с плав.точкой, сдвиговых операций

Порт1: Блок целочисленных операций, выполнения переходов

Порт2: Блок генерации адреса считывания

Порт3: Блок генерации адреса памяти

Порт4 Блок памяти данных

P6 может запускать на выполнение до 5 микрокоманд за такт, по одной на каждый порт. Средняя длительно поддерживаемая пропускная способность - 3 микрокоманды за такт.

Вывод. Команды, которые исполняются не в той последовательности, которая предписана программой, следует в конечном итоге расположить в должной последовательности - иначе процессор не всегда сможет получить правильные результаты. ROB сохраняет статус исполнения и результаты каждой микрокоманды. Микрокоманда выводится, и результаты заносятся в архитектурные регистры и память только после того, как станет известно, что предыдущие микрокоманды завершились.

Рассмотрим этот процесс. После того как команды были дешифрованы и регистры переименованы, микрокоманды помещаются в ROB - циклическую очередь FIFO - последовательно, в определяемом программой порядке. Это происходит тогда, когда эти же самые микрокоманды пересылаются в станцию-резервуар. Последовательный процесс упорядочения очень важен для восстановления программного порядка микрокоманд после исполнения с изменением последовательности. Это обеспечивает упорядоченное обновление архитектурных регистров и ячеек памяти после завершения исполнения.

Тот факт, что и ROB, и станция-резервуар получают одни и те же микрокоманды в одно и то же время из дешифраторов - важный элемент архитектуры P6. Пока ROB следит за программной последовательностью микрокоманд, станция-резервуар сохраняет их в буфере и определяет момент, когда конкретная микрокоманда будет

готова к пересылке в соответствующее устройство исполнения. (Микрокоманды готовы к пересылке, когда определены все их операнды.)

Результаты выполнения предыдущих микрокоманд можно использовать в качестве исходных операндов для последующих, поэтому все результаты из каждого устройства исполнения посылаются назад в станцию-резервуар. По сути, P6 применяет сложную схему межсоединений, подобную матричному переключателю, которая связывает все выходные порты исполнительных блоков со станцией-резервуаром. Это позволяет продвигать (forward или bypass) результаты в другой исполнительный блок, которому данный результат требуется в качестве входного операнда, без задержки, вызванной обновлением и повторным чтением регистров.

Результаты из полных блоков также направляются назад в ROB, который определяет, готова ли микрокоманда к удалению. В процессе удаления происходит запись результатов - обновление архитектурных регистров и сохранение данных в памяти. ROB обеспечивает вывод всех микрокоманд в указанном программой порядке; максимальная скорость вывода - три микрокоманды за такт, что примерно соответствует средней производительности дешифраторов.

Операции записи в память также откладываются до той поры, пока вызвавшая их микрокоманда не будет выведена. Для этого в P6 предусмотрен буфер упорядочени обращений к памяти (memory order buffer - MOB), в котором по командам, выдаваемым устройствами записи в память, сохраняется информация о данных и адресах. MOB пересылает данные в память, лишь когда ROB уведомит MOB о том, что микрокоманда, произведшая запись в память, удаляется.

Переходы. Микрокомандам перехода еще в упорядоченной части конвейера ставятся в соответствие адрес следующей команды и предполагаемый адрес перехода. После вычисления перехода реальная ситуация сравнивается с предсказанной. Если они совпадают, то проделанная, исходя из предположения об исходе перехода, работа оказывается полезной, так как соответствует реальному ходу программы, а микрокоманда перехода удаляется из пула команд.

Если же допущена ошибка (переход был предсказан, но не произошел, или было предсказано отсутствие перехода, а в действительности он состоялся), то устройство выполнения переходов изменяет статус всех микрокоманд, засланных в буфер восстановления последовательности команд после команды перехода, чтобы убрать их из буфера восстановления последовательности. Правильный адрес перехода направляется в буфер переходов, который перезапускает весь конвейер с нового адреса.

Устройство завершения также проверяет статус микрокоманд в буфере восстановления последовательности: оно ищет микрокоманды, которые уже выполнены и могут быть удалены. Именно при удалении микрокоманды результаты ее выполнения, хранящиеся в буфере восстановления последовательности, реально изменяют состояние вычислительной системы, например, происходит запись в регистры.

Предсказание переходов. P6 применяет статические и динамические методы для предсказания поведени. В P6 используется структура, называемая буфером адреса перехода (branch target buffer – BTB, в котором применен двухуровневый адаптивный исторический алгоритм, который не только регистрирует предысторию и предсказывает конкретные переходы, но может также предугадывать поведение групп команд переходов.

В большинстве высокопроизводительных процессоров, в том числе P6, возможности предсказания ветвлений усилены благодаря исполнению по предположению (speculative execution) - способности исполнять команды, следующие

за командой условного перехода, еще не зная того, правильно ли был предсказан исход ветвления. Процессор не должен обновлять архитектурные регистры или память до тех пор, пока исполненные по предположению команды перехода не будут однозначно разрешены. Если какой-либо из переходов был предсказан неправильно, то процессор должен иметь возможность корректно отменить все выполненные команды, следующие за точкой ветвления. Для этого ROB процессора P6 просто отбрасывает исполненные по предположению команды, прежде чем они будут удалены. P6, как и другие современные процессоры, допускает несколько уровней предположения, предсказывая и отслеживая более одного целевого потока команд.

P6 также содержит специальную логику для обработки особого типа перехода - пары команд вызова-возврата из подпрограмм CALL/RET. Поскольку подпрограмма может быть вызвана из многих различных точек в программе, то трудно предсказать, откуда процессор должен производить выборку команд, встретив команду RET. Механизм предсказания, называемый стеком возвратов (return stack), помогает ослабить влияние подобных ситуаций; адрес команды, следующей за командой вызова, представляющий собой правильный адрес возврата дл этого конкретного вызова подпрограммы, помещается в стек. Таким образом, логика предсказания ветвлений может сообщить процессору, что после возврата из подпрограммы должна быть произведена выборка надлежащей команды.

Работа с памятью. Есть два типа обращений к памяти: чтение из памяти в регистр и запись из регистра в память. При чтении из памяти должны быть заданы адрес памяти, размер блока считываемых данных и регистр-назначение. Команда чтения кодируется одной микрокомандой. При записи надо задать адрес памяти, размер блока записываемых данных и сами данные. Поэтому команда записи кодируется двумя микрокомандами: первая генерирует адрес, вторая готовит данные. Эти микрокоманды планируются независимо и могут выполняться параллельно; они могут переупорядочиваться в буфере записи.

Запись в память никогда не выполняется опережающим образом, так как нет эффективного способа организации отката в случае неверного предсказания. Разные команды записи никогда не переупорядочиваются друг относительно друга. Буфер записи инициирует запись, только когда сформированы и адрес, и данные, и нет ожидающих выполнения более ранних команд записи.

Была реализована архитектура подсистемы памяти, позволяющая командам чтения опережать команды записи и другие команды чтения. Буфер упорядочения памяти служит в качестве распределительной станции и буфера переупорядочивания. В нем хранятся отложенные команды чтения и записи, и он осуществляет их повторное диспетчерование, когда блокирующее условие (зависимость по данным или недоступность ресурсов) исчезает.

Конвейер. Суперконвейеризация делит ступени стандартного конвейера на более мелкие части; с увеличением числа ступеней каждая отдельная ступень выполняет меньшую работу и, следовательно, содержит меньше аппаратной логики. Уменьшение сложности логики уменьшает задержку распространения (propagation delay) - временной интервал между поступлением набора входных воздействий на входы схемы и появлением результирующих сигналов на ее выходах. Благодаря более коротким задержкам распространения становится возможным повышение частоты

Схема конвейера команд приведена ниже:

1. Блок упорядоченной обработки и декодирования команд	2 ступени – выравнивание и выборки 4 ступени – трансляции в	Блок выборки команд Планировщик Станция команд

in-order-front end	RISC-подобные "микрооперации" mu-ops 2 ступени - переименование регистров	Буфер восстановления Таблица псевдонимов регистров
2. Выполнение с изменением последовательности команд	2 ступени – передача в исполнительный блок 1 ступень – исполнение	11 испольнительных блоков
3. Запись результата	3 ступени восстановления последовательности	Буфер отложенной записи

Конвейер P6 - сложное устройство, насчитывающее 14 ступеней, разделенных на три блока. Входной блок упорядоченной обработки (in-order front end), отвечающий за декодирование и обработку команд, состоит из восьми ступеней. Ядро исполнения с изменением последовательности (out-of-order core), где собственно и происходит выполнение команд, имеет три ступени. И конвейер вывода команд из последовательности (in-order retirement) состоит из трех ступеней. Изящество архитектурного решения P6 состоит в том, что каждый из этих блоков работает достаточно самостоятельно.

Суперконвейерная архитектура имеет серьезный недостаток. Команды, которые вынуждают P6 очищать или приостанавливать свои конвейеры, а в их число входят неправильно предсказанные переходы и операции, подобные тем, что выполняют загрузку сегментных регистров, могут существенно снизить производительность. В процессе продвижения команды по конвейеру возможна приостановка конвейера на 7 ступени из-за заторов. Два вида заторов, наиболее часто встречающихся, приведены в таблице.

Виды затора	Механизм выполнения команд
1. Команды с частичными регистрами	Команда к полному р-ру ожидает на ступени 7 проходжение команд к частичному р-ру.
2. Команды ввода-вывода и загрузки сегментных регистров	• Команды проходят изолированно • Удаляются команды блока 1

Таким образом, основные особенности архитектуры:

- **Динамическое исполнение** - команды из станции выполняются не в порядке записи команд в программе, а по мере готовности данных. Команды выбираются на основании:
  - доступности операндов
  - занятости исполнительных блоков
  - устранения взаимозависимости
- **Спекулятивное обращение к памяти**. Результаты сохраняются во временных буферах ROB, MOB. Удаление происходит в предписанном программой порядке

со средней скоростью три команды/такт. Запись результатов в память делается только для удаленных команд.

- **Исполнение "по предположению" (Speculative Execution)** - выполнение команд за точкой ветвления, еще не зная результата ветвления. Два метода предсказания:
  - статический - всегда (не)выполнять определенные переходы
  - динамический (по поведению за предшествующий период)

Результативность предсказаний достигает 90%.

Архитектура шины

Системная шина Pentium Pro и Pentium-2 более эффективна для объединения процессоров по симметричной архитектуре, чем шины предыдущих процессоров, оптимизированные для обмена с памятью. Она позволяет без дополнительных схем объединять до четырех процессоров.

Сигналы системной шины объединяются в группы запросов (Request) и ответов (Response). Каждое устройство-агент, подключенное к этой шине (например, любой из процессоров), до инициализации запроса должно получить через механизм арбитража право на использование шины запроса. Запрос выходит за два смежных такта: в первом такте передается адрес, тип обращения (чтение-запись памяти или ввода/вывода) и тому подобная информация. Во втором такте передается уникальный идентификатор транзакции, длина запроса, разрешенные байты шины и т. п. Через три такта после запроса проверяется состояние ошибки (error status) для защиты от ошибок передачи или нарушений протокола. Любая обнаруженная ошибка вызывает повтор запроса, а вторая ошибка для того же запроса вызывает исключение контроля (machine check exception).

Шинные транзакции делятся на множество фаз, перекрывающих друг друга. На рисунке изображены две транзакции.

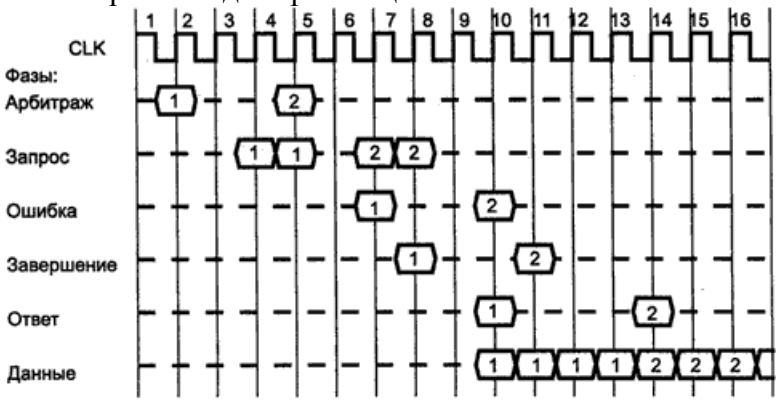


Рис. 2.7. Временная диаграмма шины P6.

На шине одновременно может присутствовать множество запросов и ответов, однако логический анализатор, "понимающий" протокол шины Pentium Pro, способен разложить их всех на соответствующие транзакции.

Pentium II

Процессоры Pentium II (1997) с тактовыми частотами 350 и 400 МГц обладают теми же возможностями повышения производительности, что и ранее выпущенные модели, включая архитектуру двойной независимой шины, технологию динамического исполнения, технологию MMX, а также компактную шину кэш-памяти второго уровня,

работающую на частоте, вдвое меньшей тактовой частоты процессора. Расширены режимы управления энергопотреблением:

В режим **Stop Grant** процессор входит по сигналу на входе **STPCLK#.** В этом состоянии он прекращает исполнение инструкций и не обслуживает прерывания, однако продолжает слежение за шиной данных, отслеживая кэш-попадания.

В состояние пониженного потребления **Auto HALT PowerDown** процессор переходит при исполнении инструкции HALT. В этом состоянии процессор реагирует на все прерывания и также продолжает слежение за шиной. Переход в режимы пониженного потребления разрешается битом Low-Power Enable.

В состоянии **Sleep** (спящий режим) процессор не тактирует свои внутренние узлы (кроме схемы умножителя частоты). Прерывания и циклы слежения не воспринимаются. Процессор может реагировать только на сигналы **SLP#, STPCLK# и RESET#.** По снятию сигнала процессор возвращается в состояние Stop Grant и возобновляет тактирование своего блока управления шиной и APIC.

В состояние «глубокого сна» **Deep Sleep** процессор переходит при остановке тактового сигнала на входе BCLK. В этом режиме процессор не выполняет никаких функций и его ток потребления определяется только токами утечки внутренних схем.

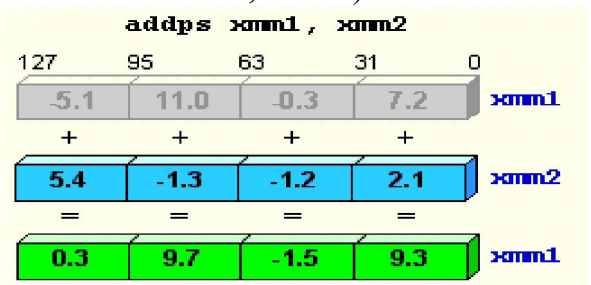
Питание процессоров. Для **питания** ядра процессора и вторичного кэша используются раздельные шины питания VCCP и VCCS соответственно. Они могут использовать различные номиналы питающих напряжений. Номинальная потребляемая мощность процессора процессора 200 МГц/512К номинальное потребление 33 Вт, максимальное 38 Вт. Допустимая температура корпуса 85°С.

Pentium III. Технология SSE

SIMD Single Instruction Multiply Data (одна инструкция - много данных).
SSE Streaming SIMD Extensions

В Pentium III реализовано 70 новых SIMD-инструкций, оперирующих со специальными 128-битными регистрами XMM0-XMM7. Каждый из этих регистров хранит четыре вещественных числа одинарной точности. Выполняя операцию над двумя регистрами, SSE фактически оперирует четырьмя парами чисел. То есть благодаря этому процессор может выполнять до 4-х операций одновременно

Суффиксы: ps – параллельная операция SIMD (maxps xmm1,xmm5)
ss – скалярная операция (операция над 0-31 битами) (addss xmm1,xmm5)



Пример операции перестановки: **shufps xmm1,xmm2,9Ch**

10(b2)-01(b1)-11(a3)-00(a0) in xmm1

SSE3 (2004 год в ядре процессора Pentium 4)

Набор SSE3 содержит 13 инструкций. Возможность горизонтальной работы с регистрами. Добавлены команды сложения и вычитания нескольких значений,

хранящихся в одном регистре. Существует также команда для преобразования значений с плавающей точкой в целые.

Инструкции SSE3

ADDSUBPD (Add Subtract Packed Double).
ADDSUBPS (Add Subtract Packed Single).
HADDPD (Horizontal Add Packed Double).
HADDPS (Horizontal Add Packed Single).
HSUBPD (Horizontal Subtract Packed Double).
HSUBPS (Horizontal Subtract Packed Single).
FISTTP — преобразование вещественного числа в целое с сохранением целочисленного значения и округлением в сторону нуля.

SSE4 (ноября 2008)

SSE4 состоит из 54 инструкций. Добавлены инструкции обработки строк 8/16 битных символов, вычисления CRC32 и др.

Инструкции SSE4.1

Ускорение видео ; Векторные примитивы
Вставки/извлечения ; Скалярное умножение векторов
Смешивания ; Проверки бит
Округления ; Чтение WC памяти

Инструкции SSE4.2

Обработка строк
Подсчет CRC32
Подсчет популяции единичных бит
Векторные примитивы

PHMINPOSUW xmm1, xmm2/m128 — *(Packed Horizontal Word Minimum)*
Input — { A0, A1,… A7 }
Output — { MinVal, MinPos, 0, 0… }
Поиск среди 16-ти битных беззнаковых полей A0…A7 такого, который имеет минимальное значение (и позицию с меньшим номером, если таких полей несколько). Возвращается 16-ти битное значение и его позиция.

AVX

Advanced Vector Extensions (AVX) — расширение системы команд x86 для микропроцессоров Intel и AMD, предложенное Intel в марте 2008.

Ширина векторных регистров SIMD увеличивается со 128 (XMM) до 256 бит (регистры YMM0 — YMM15). Существующие 128-битные SSE инструкции будут использовать младшую половину новых YMM регистров, не изменяя старшую часть. Для работы с YMM регистрами добавлены новые 256-битные AVX инструкции. В будущем возможно расширение векторных регистров SIMD до 512 или 1024 бит. Например, процессоры с архитектурой Larrabee уже имеют векторные регистры (ZMM) шириной в 512 бит, и используют для работы с ними SIMD команды с MVEX и VEX префиксами, но при этом они не поддерживают AVX.

AVX-512 (ZMM) register scheme as extension from the AVX (YMM) and SSE (XMM) registers

511 256	255 128	127 0
ZMM0	YMM0	XMM0
ZMM1	YMM1	XMM1
ZMM2	YMM2	XMM2
ZMM3	YMM3	XMM3
ZMM4	YMM4	XMM4
ZMM5	YMM5	XMM5
ZMM6	YMM6	XMM6
ZMM7	YMM7	XMM7
ZMM8	YMM8	XMM8
ZMM9	YMM9	XMM9
ZMM10	YMM10	XMM10
ZMM11	YMM11	XMM11
ZMM12	YMM12	XMM12
ZMM13	YMM13	XMM13
ZMM14	YMM14	XMM14
ZMM15	YMM15	XMM15
ZMM16	YMM16	XMM16
ZMM17	YMM17	XMM17
ZMM18	YMM18	XMM18
ZMM19	YMM19	XMM19
ZMM20	YMM20	XMM20
ZMM21	YMM21	XMM21
ZMM22	YMM22	XMM22
ZMM23	YMM23	XMM23
ZMM24	YMM24	XMM24
ZMM25	YMM25	XMM25
ZMM26	YMM26	XMM26
ZMM27	YMM27	XMM27
ZMM28	YMM28	XMM28
ZMM29	YMM29	XMM29
ZMM30	YMM30	XMM30
ZMM31	YMM31	XMM31

2.1.6. Процессор Pentium 4

В процессоре Pentium 4 (2000 год) сохраняется архитектура IA-32 (Intel Architecture – 32), характерная для всех 32-разрядных микропроцессоров Intel, начиная с i386. Поэтому пользователь будет иметь дело с хорошо знакомым набором регистров и способов адресации, работать с базовой системой команд и известными вариантами реализации прерываний и исключений. Однако внутренняя структура (микроархитектура) процессора Pentuim 4 значительно отличается от предшествующей микроархитектуры семейства P6, к которому относятся Pentium II, Pentium III . Кратко укажем эти отличия:

1. В Pentuim 4 используется гиперконвейерная технология (Hyper Pipelined Technology) выполнения команд, при которой число ступеней конвейера достигает 20 (в Pentium — 5 ступеней, в Pentium III — 10 ступеней). Таким образом одновременно в процессе выполнения может находиться до 20 простых команд, находящихся на разных стадиях (ступенях) реализации.

2. Advanced Dynamic Execution. Улучшенное предсказание переходов и исполнение команд с изменением порядка их следования (out of order execution).

Для выборки следующей инструкции для исполнения используется теперь окно величиной в 126 команд против 42 команд у процессора Pentium III. Буфер же, в котором сохраняются адреса выполненных переходов и на основании которого процессор предсказывает будущие переходы, теперь увеличен до 4 Кбайт, в то время как у Pentium III его размер составлял всего 512 байт.

3. Trace Cache на 12000 микроопераций. Специальный Кэш для кэширования декодированных инструкций.

Преимущества:

Первое Сохранение в Trace Cache микроопераций позволяет избежать повторного декодирования x86 инструкций при повторном выполнении того же участка программы или при неправильном предсказании переходов.

Второе Микрооперации сохраняются именно в том порядке, в каком они выполняются (на основании предсказания переходов).

4.Rapid Execute Engine. ALU процессора Pentium 4 работает на вдвое большей, чем сам процессор, частоте.

5. SSE2. Расширенный набор инструкций для обработки потоковых данных.

SSE2 представляя собой симбиоз MMX и SSE и позволяет работать с любыми типами данных, влезающими в 128-битные регистры

6. КЭШ

КЭШ первого уровня в Pentium 4 предназначен только для хранения данных. Его размер составляет 8 Кбайт. Также как и в Pentium III, L1 кэш Pentium 4 является write through и ассоциативным с 4 областями ассоциативности. При этом длина одной строки L1 кеша равна 64 байтам. Учитывая большую тактовую частоту Pentium 4, время реакции его L1 кеша составляет всего 1.4нс для 1.4 ГГц модели против 3нс у L1 кеша Pentium III 1 ГГц.

КЭШ второго уровня (Advanced Transfer Cache) объемом 256 Кбайт.

Также, как и в Pentium III, L2-кэш имеет широкую 256-битную шину

L2 кэш не является эксклюзивным, то есть он дублирует данные, находящиеся в L1 кэше

L2 кэш имеет восемь областей ассоциативности и строки длиной 128 байт. Однако, в отличие от Pentium III, каждая строка может быть изъята не целиком, а по 64-байтовым половинкам. Архитектура NetBurst поддерживает кэш третьего уровня размером до 4 Мбайт.

7. Одна из версий процессора содержит 42 млн. транзисторов, имеет площадь 217 кв.мм.

Работа процессора

Выборка и декодирование. Процессор (рис.) работает следующим образом: блок выборки пытается выбрать очередную команду из L2. В случае если ее там нет – производятся операции заполнения строки и в конечном итоге нужная строка оказывается в L2 и выбирается оттуда. Далее строка декодируется в микрооперации и строится «трасса исполнения» из последовательности микроопераций. Эта последовательность заносится в Trace Cache с сохранением порядка микроопераций, то есть все переходы и т.д. учитываются при построении трассы. Trace Cache при этом находится в режиме построения (build). После того как трасса построена, кэш переходит в режим «поставки» (delivery) микроопераций.

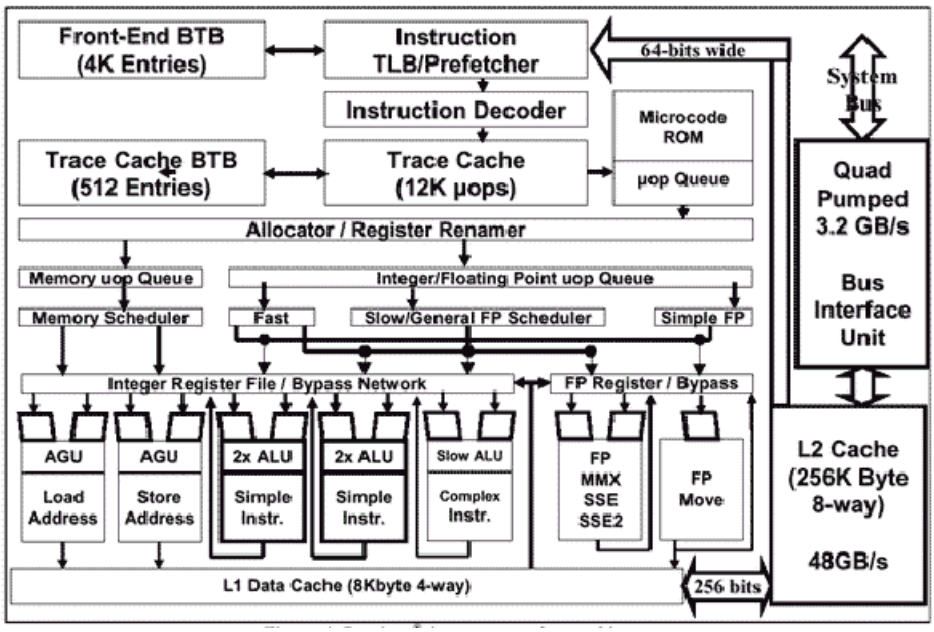


Рис. 8.7. Структурная схема процессора Pentium4

Кэширование не инструкций, а трасс и есть ключевой момент архитектуры. Если требуемые ядру инструкции есть в кэше, для них не требуется выполнять декодирование. Обеспечение правильного порядка возлагается на декодер. Таким образом, сделана попытка убрать ступень декодирования для тех команд, которые уже исполнялись.

Следует отметить, что Trace cache не хранит все микрооперации, которые нужны программе. Некоторые сложные команды, по-прежнему поступают на выполнение из памяти микрокода. Для этой цели служат MROM-векторы. Эти векторы помещаются в Trace Cache вместо длинных последовательностей микроопераций. Когда процессор извлекает такой вектор, он обращается к памяти микрокода (происходит событие Uop delivery changing from TC to MROM).

Кэш трасс тесно связан с блоком предсказания ветвления. Благодаря BTB, в кэш трасс не попадают микрооперации, которые никогда не будут исполняться. Переименование регистров и предсказание перехода происходит до построения трассы.

Pentium 4 поддерживает программные «подсказки» на переходы, для этого введены 2 новых префикса – 3Eh если переход будет и 2Eh если нет. Префиксы ставятся перед командами условных переходов.

Буфер BTB в P4 хранит 4K данных, то есть адреса 1024 команд перехода. В качестве адреса команды используется ее линейный адрес.

В процессоре применено также очень интересное решение, которое называется RAS (Return Address Stack). Суть его сводится к тому, что процессор кэширует вершину стека в своих внутренних регистрах. При интенсивной работе со стеком, например в случае частых вызовов процедур, если все изменяемые данные помещаются во внутренний стек процессора – обращение к вершине стека будет занимать примерно такое же время, как и обращение к кэш-памяти первого уровня, что существенно быстрее обращения в основную память. Для переименования регистров используется механизм похожий на P6 с RAT и ROB.

Диспетчеризация и исполнение. Ядро использует те же принципы неупорядоченного спекулятивного исполнения, что и ядро P6. Процесс обработки четко разделяется на Front End выполняющий выборку, декодирование и построение трассы исполнения в порядке программы и Back End, который включает ядро, неупорядоченно выполняющее микрооперации, и подсистемы ответственные за неупорядоченное исполнение.

Исполнительная часть ядра имеет 4 порта доступа, они показаны на рисунке.

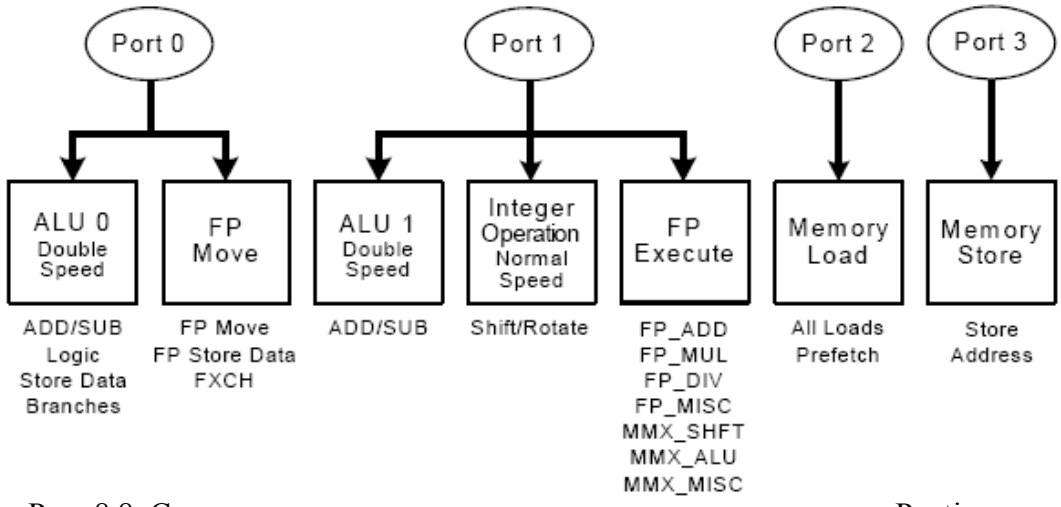


Рис. 8.8. Структурная схема исполнительного ядра процессора Pentium

Микрооперации из Trace cache поступают в эти порты, после чего - в очереди выполнения и выполняются ядром. Микрооперации могут поступать в порты еще до того, как выполнены все условия для их «штатного» выполнения, могут быть не готовы операнды микрооперации, либо она может находиться в неподтвержденной ветви перехода. При этом предполагается, что к тому времени, когда дело дойдет до исполнения, все эти условия будут выполнены, в случае осечки (например, промаха кэша, где должен быть операнд) микрооперация должна быть переиздана (reissue). Согласно интеловской терминологии, переиздание микрооперации называется replay.

Восстановление (retirement).Операция восстановления выполняет те же операции что и в P6 – извлечение результата операции и сохранение его в архитектурных регистрах, инкремент EIP на длину команды и т.д.

КЭШ трасс - Trace cache

Важнейшим элементом процессора Intel Pentium 4 является так называемый кэш трасс (Trace cache, T-кэш), который выполняет ту же функцию, что и кэш инструкций в классических микропроцессорах. Необходимость создания такого устройства была связана с тем, что разработчики процессора столкнулись с трудностями реализации обычного кэша инструкций, эффективно работающего на повышенных тактовых частотах. Эти трудности обусловлены сложностью декодирования и обработки машинных инструкций процессоров архитектуры x86 (x86-инструкций), имеющих переменную длину и неудобный формат. Уже в процессорах предыдущего поколения (Pentium Pro, Pentium II, Pentium III) было реализовано преобразование "нерегулярных" x86-инструкций в "регулярные" микрооперации (uOP's, МОПы), удобные для последующей обработки. Однако в кэше инструкции по-прежнему хранились в исходном виде. Сложность декодирования x86-инструкций усугублялась тем, что эти процессоры имеют суперскалярную архитектуру и способны исполнять до трёх инструкций за такт, и декодирование этих трёх инструкций должно производиться параллельно. В процессоре Pentium 4 был сделан следующий шаг - кэш разместили после блока декодирования инструкций, и теперь в нём хранятся микрооперации с регулярной структурой. Таким образом, критически важный этап выборки и окончательного декодирования инструкций может производиться в высоком темпе и с минимальными временными затратами. Новый кэш получил название "Trace-cache" (T-кэш).

Для начала нужно отметить, что рассматриваемый кэш является "кэшем трасс" (Trace-cache), а не "кэшем микроопераций" (uOP-cache). Этим определяется принципиальное отличие T-кэша от классического кэша инструкций - элементом хранения в кэше является не отдельная (микро)инструкция или выровненный блок из группы инструкций, а "трасса", т.е. непрерывная последовательность (микро)инструкций в соответствии с предполагаемым динамическим порядком их исполнения. Вследствие этого организация T-кэша и алгоритмы его функционирования принципиально отличаются от организации обычного кэша инструкций (который, в свою очередь, не отличается существенно от обычного кэша данных).

В том случае, когда микроинструкция, на которую происходит переход, отсутствует в T-кэше, начинается считывание x86-инструкций непосредственно из кэша 2-го уровня (L2-кэша) с их последующим декодированием и исполнением. Одновременно происходит формирование новой трассы микроинструкций и её размещение в T-кэше. Если встречается инструкция условного перехода, то трасса, начиная с этой инструкции, строится в соответствии с результатом работы блока предсказания перехода предекодера. Так как инструкции и данные хранятся в L2-кэше по физическим адресам, перед считыванием каждого блока из него происходит преобразование программного адреса в физический с помощью соответствующей таблицы трансляции адресов (Instruction Translation Look-aside Buffer, ITLB).

Предекодер обрабатывает входной поток со скоростью не более одной x86-инструкции за такт, в зависимости от сложности формата инструкции и наличия префиксов. Если x86-инструкция не может быть преобразована в последовательность, состоящую из небольшого числа микроопераций (МОПов) (до четырёх), она заменяется

на вызов "микрокодовой" подпрограммы, которая при работе будет генерировать МОПы и пересылать их на исполнение.

Как уже отмечалось выше, при появлении инструкции перехода построение трассы не прерывается, а продолжается в соответствии с предсказанным направлением этого перехода. Таким образом, в случае, если переход будет предсказан как "совершённый" (taken), непосредственно после инструкции (МОПа) перехода в трассе будет размещена инструкция (МОП), на которую происходит переход - причём оба этих МОПа могут оказаться в одном блоке и даже в одной "тройке", отсылаемой на исполнение. При предсказании перехода как "не совершённый" (not taken) формирование трассы продолжится в натуральном порядке. Однако в случае, если при первом исполнении инструкции перехода (после отсылки её на исполнение параллельно с формированием трассы) выяснится, что переход был предсказан неправильно, формирование трассы прекратится, и она будет оборвана после этого МОПа.

Также формирование трассы прекращается, если встречается инструкция безусловного косвенного перехода (indirect jump), вызова подпрограммы (call) или возврата из подпрограммы (return).

Может случиться, что правильно предсказанный "совершённый" переход является переходом по циклу. В таком случае появляется возможность "раскрутить" этот цикл и разместить в трассе несколько его итераций и, соответственно, повысить скорость выборки и исполнения инструкций за счёт снижения затрат на переход по циклу. С другой стороны, должен существовать механизм, ограничивающий раскрутку цикла разумными пределами. Согласно патентам, в этом механизме имеются ограничения по длине тела цикла и по числу оборотов раскрутки. Измерения показывают, что для коротких циклов число оборотов раскрутки равно по меньшей мере четырём.

Максимальная длина трассы составляет 64 блока. При достижении этого предела формирование трассы также прекращается.

По завершении формирования трассы происходит поиск в кэше другой трассы с требуемым стартовым программным адресом. Если трасса с нужным адресом не найдена, снова начинается считывание x86-инструкций из L2-кэша и формирование следующей трассы.

Если встречается инструкция условного перехода, то, в зависимости от результата работы предсказателя переходов, возможны следующие ситуации:

При правильном предсказании перехода:

1 - направление перехода совпадает с тем, которое было предсказано при построении трассы - в этом случае продолжается исполнение текущей трассы без перерыва и без потери тактов;

2 - направление перехода не совпадает с тем, которое было предсказано при построении трассы, и трасса по предсказанному адресу перехода найдена в кэше - в этом случае начинает исполняться найденная трасса;

3 - направление перехода не совпадает с тем, которое было предсказано при построении трассы, и трасса по предсказанному адресу не найдена - в этом случае начинается формирование новой трассы с одновременным исполнением записываемых в неё МОПов.

При неправильном предсказании перехода: сначала выполняются действия 1, 2 или 3 - в соответствии с предсказанным направлением. В тот момент, когда выяснится, что направление предсказано неверно, исполнение всех стартовавших инструкций (начиная с инструкции перехода) прекращается, результаты их работы аннулируются, и производится поиск трассы по другому (правильному) адресу. Далее начинается

исполнение найденной трассы либо формирование новой. Неправильно предсказанный переход при нахождении адресата в Т-кэше занимает около 20 тактов для процессора P4 и около 30 тактов для процессора P4E. Если адресата нет в Т-кэше, то требуется ещё около 20 тактов (по грубым оценкам).

Для инструкций безусловного косвенного перехода (indirect jump), вызова подпрограммы (call) и возврата из подпрограммы (return) возможны только случаи, подобные описанным в пунктах 2 и 3, так как такие инструкции "обрывают" трассу.

Перейдём теперь к рассмотрению работы механизма предсказания переходов. Этот механизм выполняет две основные функции - предсказание программного адреса инструкции, на которую производится переход, и предсказание направления ветвления (для инструкций условного перехода). Оба предсказания должны быть выполнены заблаговременно - по возможности даже раньше, чем начнётся обработка инструкции перехода - для того, чтобы поиск (при необходимости) новой трассы и считывание нового блока МОПов были произведены без потерь лишних тактов либо с минимальными потерями.

Предсказание адреса перехода производится по адресу исходной x86-инструкции перехода либо по позиции соответствующего МОПа с использованием таблиц адресов перехода (Branch Target Table, BTB). Необходимость такого предсказания вызвана тем, что этот адрес может быть извлечён из инструкции (МОПа) и вычислен только на финальной стадии декодирования, а для инструкций косвенного перехода - только на стадии выполнения. Для инструкций "прямого" перехода этот адрес (при условии его нахождения в BTB) является правильным и окончательным, а для инструкций косвенного перехода - гипотетическим, так как содержимое элемента данных с целевым адресом может меняться. Отдельным случаем является предсказание инструкций возврата из подпрограммы (return): так как подпрограмма может вызываться из различных мест с разными адресами возврата, для таких инструкций предусмотрен специальный аппаратный стек глубиной в 16 элементов.

Предсказание направления необходимо только для инструкций условного перехода. Рассмотрим по отдельности предсказание переходов при формировании новой трассы и при исполнении существующей трассы.

При формировании трассы предсказание целевого адреса и направления перехода необходимо для того, чтобы размещать МОПы в трассе в соответствии с предполагаемым динамическим порядком их последующего исполнения. Предсказание производится по адресу исходной x86-инструкции перехода. Этой цели служит так называемая динамическая таблица адресов переходов (Dynamic Branch Target Buffer, DBTB) и связанная с ней таблица истории переходов (Branch History Table, BHT). Таблица DBTB по своей структуре напоминает кэш-память и состоит из 4096 элементов, организованных в виде 1024 наборов по 4 элемента. Найденный элемент содержит предполагаемый программный адрес инструкции, на которую производится переход.

Несколько иначе производится предсказание переходов при исполнении трассы, считываемой из Т-кэша. В этом случае повышается роль скорости работы алгоритма предсказания, поэтому предсказание производится не по адресу x86-инструкции перехода, а по позиции соответствующего МОПа в Т-кэше. Так как минимальное время обработки каждого блока кэша составляет всего 2 такта, то предсказание адреса и направления перехода должно производиться с опережением: в момент обработки текущего блока (выборки и запуска МОПов на исполнение) проводится предсказание для МОПов перехода, находящихся в следующем блоке.

Для предсказания адреса перехода в первую очередь используется вспомогательная таблица адресов переходов Т-кэша (Trace-cache Branch Target Buffer, TBTB). Эта таблица содержит подмножество основной таблицы DBTB и имеет меньший размер (512 элементов у процессора P4 и 2048 элементов у процессора P4E). Направление перехода определяется с помощью той же таблицы истории переходов, которая используется для предсказания переходов при формировании трасс.

В предсказании переходов при исполнении трассы есть своя специфика. Строго говоря, необходимо предсказывать не направление перехода и его адрес (в случае, когда переход предсказан как "совершённый"), а "выход" (или "невыход") из текущей трассы и адрес, по которому должно быть передано управление (в случае, когда предсказан выход из трассы). Это связано с тем, что трасса построена в соответствии с динамическим порядком выполнения инструкции при первом прохождении алгоритма, и те переходы, которые были предсказаны в тот момент как "совершённые", уже "запаяны" в трассу. Поэтому для таких переходов, которые при очередном исполнении трассы снова предсказываются как "совершённые", выход из трассы производиться не должен, а для переходов, которые предсказываются уже как "не совершённые", должен быть произведён выход из трассы и передача управления по адресу x86-инструкции, следующей за инструкцией перехода. Для переходов, которые при построении трассы были предсказаны как "не совершённые", сохраняется натуральный порядок следования инструкций и привычное поведение при исполнении трассы.

Таким образом, назначение предсказателя переходов при исполнении трассы - принять решение "покинуть трассу" (Trace leave) или "не покидать трассу" (Not trace leave) и, для решения "покинуть трассу", выработать адрес перехода. Если инструкция перехода является последней в трассе, нужно только вычислить адрес, на который должно быть передано управление.

К числу ситуаций, требующих раннего определения направления и/или адреса перехода, помимо привычных инструкций условного перехода, безусловного косвенного перехода, вызова подпрограммы и возврата из подпрограммы, относится также и естественное завершение трассы. В свою очередь, для инструкции прямого безусловного перехода никакого предсказания не требуется, так как эта инструкция не может вызвать выхода из трассы.

Вытеснение блоков и реорганизация трасс. Ещё более сложным представляется вопрос организации вытеснения блоков и целых трасс из Т-кэша при затребовании блоков для новых трасс. Так как блок МОПов не может быть реконструирован отдельно от всей трассы, то и вытеснение по идее должно было бы производиться лишь целыми трассами - в противном случае повышается вероятность "разрушения" трасс в результате спонтанного вытеснения отдельных блоков.

Однако в нынешней реализации блоки вытесняются из Т-кэша всё же по отдельности, без какой-либо увязки со своим положением в трассе. При вытеснении блока, принадлежащего трассе, эта трасса "укорачивается" - предыдущий блок трассы помечается как "последний". Остальные блоки трассы становятся при этом недоступными. Однако особой беды в этом нет: так как вытеснение блоков происходит по механизму LRU (Least Recently Used, т.е. наименее используемый в последнее время), то "омертвевшие" блоки в хвосте трассы станут вскоре наиболее вероятными кандидатами на вытеснение, поскольку обращений к ним больше производиться не будет. Если же нужен кандидат на вытеснение в связи с заведением начального блока новой трассы, то в первую очередь будет вытеснен блок, являющийся также начальным блоком трассы и имеющий тот же мини-тэг, что и вновь заводимый (если такой блок обнаружится в данном наборе).

Также интересным является вопрос перестройки и динамической адаптации трасс. По мере накопления информации о поведении условных переходов в таблице истории переходов появляется возможность строить трассы более рационально. Возможность такой перестройки предоставляется мультизадачностью вычислительной системы: при полной смене контекста задачи вместе со сменой адресного пространства содержимое Т-кэша становится неактуальным, поскольку работа с кэшем ведётся в программных адресах. Таким образом, при обратной смене контекста содержимое рабочего множества программы в Т-кэше должно быть восстановлено "с нуля".

Это не относится к переключению двух потоков, исполняемых в рамках технологии Hyper-Threading, так как имеется механизм поддержки сосуществования таких потоков: у каждой трассы есть признак (тэг) номера потока, к которому она принадлежит.

Проведём грубую оценку потерь на полную смену содержимого Т-кэша. Предположим, что частота смены контекста составляет 100 раз в секунду, а время, необходимое для заполнения Т-кэша, равно 20-30 тысячам тактов при частоте процессора 2-3 GHz. В таком случае заполнение кэша займёт примерно 10 микросекунд из полного времени работы задачи между сменами контекста 10 миллисекунд. Таким образом, данная (предположительно завышенная) оценка показывает потери на уровне 0.1 % от общего времени выполнения.

SSE2

У микропроцессора Intel Pentium 4 появилась новая группа инструкций, получивших название SSE2. Цифра 2 в названии указывается для того, чтобы отличить новую группу от одноименной, уже поддерживаемой микропроцессором Pentium III. В состав SSE2 входят операции для работы с 64-х разрядными целыми и вещественными (представление с удвоенной точностью) числами.

Согласно документации Intel в группу SSE2 входят инструкции, выполняющие 144 новые операции. Если же вы подсчитаете количество новых имен инструкций, то их окажется значительно меньше. Расхождение объясняется тем, что одному имени инструкции может соответствовать несколько разных кодов операций. Типичный пример инструкции пересылки данных, у которых код операции зависит от направления пересылки - из регистра в память или из памяти в регистр.

Большинство новых инструкций двухадресные. Первый операнд является приемником (dest), а второй источником (src). Приемник, как правило, находится в 128-bit регистре xmm, источник может находиться как в регистре xmm, так и в оперативной памяти (ОЗУ). Исключением являются только инструкции пересылки, у которых приемник может располагаться в ОЗУ. Третий операнд, если он есть, является целым числом, размер которого не превышает одного байта.

При работе с вещественными числами возможно выполнение одной и той же операции над одной или двумя парами чисел. В первом случае имя операции оканчивается буквами SD (скалярные данные), а во втором - буквами PD (упакованные данные). В данном случае выражение "упакованные данные" означает, что два 64-bit числа расположены в одном 128-bit регистре или в ОЗУ подряд друг за другом. А выражение "скалярные данные" означает, что одно 64-bit число расположено в младшей половине 128-bit регистра. По утверждению специалистов Intel скалярные операции более рационально используют ресурсы процессора чем аналогичные операции, выполняемые FPU.

Приведем пример групповых арифметических операций над вещественными числами: нахождение максимума и минимума, извлечение квадратного корня.

Операнды (dest и src) содержат по два вещественных числа двойной точности (64-bit), соответственно в dest получаются два 64-bit результата. Введем условные обозначения:

- mmx - любой из восьми 64-х разрядных регистров, которые впервые появились у Pentium MMX. В текстах программ на ассемблере им соответствуют имена от mm0 до mm7.
- xmm - любой из восьми 128-ми разрядных регистров, которые впервые появились у Pentium III. В текстах программ на ассемблере им соответствую имена от xmm0 до xmm7.
- r32 - любой 32-х разрядный регистр общего назначения: EAX, EBX и так далее.
- m128, m64, m32, m8 - элемент памяти соответствующего размера в битах.
- imm8 - непосредственный способ адресации, число имеющее размер байта, например, константа сдвига.

Если в качестве операнда указано только имя регистра или только элемент памяти, то это означает, что операнд может находиться только в регистре или только в ОЗУ. Если же указано сочетание обозначений имени регистра и элемента памяти, разделенное наклонной скобкой, например, xmm/m128 то операнд может находиться либо в регистре, либо в ОЗУ.

SQRTPD xmm, xmm/m128 dest = sqrt(src)	- извлечение квадратного корня из двух вещественных чисел источника с записью результата в приемник
MAXPD xmm, xmm/m128 dest = max(dest,src)	- нахождение в каждой паре большего вещественного числа удвоенной точности
MINPD xmm, xmm/m128 dest = min(dest,src)	нахождение в каждой паре меньшего вещественного числа удвоенной точности

Более подробное описание команд смотри в документации.

HyperThreading.

HyperThreading – технология впервые примененная intel в процессорах Pentium 4. Если она включена (через BIOS) и поддерживается ОС, то последняя видит 2 «логических» процессора, которые поддерживаются одним физическим. Перечислим связанные с указанной технологией понятия.

Time-Slice Multithreading. Процессор переключается между программными потоками через фиксированные промежутки времени. Накладные расходы порой получаются довольно внушительными, особенно если какой-либо процесс находится в ожидании.

Switch-on-Event Multithreading. Переключение задач при возникновении длительных пауз, например "непопаданий в кэш" (cache misses), большое число которых характерно для серверных приложений. В этом случае процесс, ожидающий загрузки данных из сравнительно медленной памяти в кэш, приостанавливается, высвобождая ресурсы CPU для других процессов. Однако Switch-on-Event Multithreading, как и Time-Slice Multithreading, не всегда позволяет достичь оптимального использования ресурсов процессора, -- в частности из-за ошибок в предсказании ветвлений, зависимости инструкций и т. д.

Simultaneous Multithreading. В этом случае программные потоки выполняются на одном процессоре "одновременно", т. е. без переключения между ними. Ресурсы CPU распределяются динамически, по принципу "не используешь -- отдай другому".

Именно такой подход положен в основу технологии Intel Hyper-Threading, к рассмотрению которой мы и переходим.

Принцип действия Hyper-Threading основывается на том, что в каждый момент времени только часть ресурсов процессора используется при выполнении программного кода. Неиспользуемые ресурсы также можно загрузить работой -- например, задействовать для параллельного выполнения еще одного приложения (либо другого потока этого же приложения). В одном физическом процессоре формируются два логических процессора (LP -- Logical Processor), которые разделяют между собой вычислительные ресурсы CPU. Операционная система и приложения "видят" именно два CPU и могут распределять работу между ними, как и в случае полноценной двухпроцессорной системы.

Одна из целей реализации Hyper-Threading -- при наличии только одного активного потока позволить ему выполняться с тем же быстродействием, как и на обычном CPU. Для этого у процессора предусмотрены два основных режима работы: **Single-Task (ST) и Multi-Task (MT)**. В режиме ST активным является только один логический процессор, который безраздельно пользуется доступными ресурсами (режимы ST0 и ST1); другой LP остановлен командой HALT. При появлении второго программного потока бездействовавший логический процессор активируется (посредством прерывания), и физический CPU переводится в режим MT. Останов неиспользуемых LP командой HALT возложен на операционную систему, которая в итоге и отвечает за такое же быстрое выполнение одного потока, как и в случае без Hyper-Threading. Разные логические процессоры используют разные регистровые файлы, у каждого из них свой поток выполнения, но разница, по сравнению с двухпроцессорной системой, заключается в том, что микрооперации, декодированные из обеих ветвей исполнения, попадают в одни и те же порты исполнения и функциональным блокам приходится «работать за двоих».

Для каждого из двух LP хранится так называемый Architecture State (AS), что включает в себя состояние регистров различного типа -- общего назначения, управляющих, APIC и служебных. У каждого LP есть свои APIC (контроллер прерываний) и набор регистров, для корректной работы с которыми вводится понятие Register Alias Table (RAT), отслеживающей соответствие между восемью регистрами общего назначения IA-32 и 128 регистрами физического CPU (по одной RAT на каждый LP).

При работе двух потоков поддерживаются два соответствующих набора Next Instruction Pointers. Большая часть инструкций берется из Trace Cache (TC), где они хранятся в декодированном виде, и доступ к TC два активных LP получают поочередно, через такт. В то же время, когда активен только один LP, он получает монопольный доступ к TC без чередования по тактам. Аналогичным же образом происходит и доступ к Microcode ROM. Блоки ITLB (Instruction Translation Look-aside Buffer), задействующиеся при отсутствии необходимых инструкций в кэше команд, дублируются и доставляют команды каждый для своего потока. Блок декодирования инструкций IA-32 Instruction Decode является разделяемым и в случае, когда требуется декодирование инструкций для обоих потоков, обслуживает их поочередно (опять-таки через такт). Блоки Uop Queue и Allocator разделяются надвое, отводя по половине элементов для каждого LP. Schedulers числом 5 штук обрабатывают очереди декодированных команд (Uops) несмотря на принадлежность к LP0/LP1 и направляют команды на выполнение нужным Execution Units -- в зависимости от готовности к выполнению первых и доступности вторых. Кэши всех уровней являются полностью разделяемыми между двумя LP, однако для обеспечения целостности данных записи в

DTLB (Data Translation Look-aside Buffer) снабжаются дескрипторами в виде ID логических процессоров.

Таким образом, инструкции обоих логических CPU могут выполняться одновременно на ресурсах одного физического процессора, которые подразделяются на четыре класса:

- дублируемые (Duplicated);
- полностью разделяемые (Fully Shared);
- с дескрипторами элементов (Entry Tagged);
- динамически разделяемые (Partitioned) в зависимости от режима работы ST0/ST1 или MT.

2.1.7. Архитектура Core i7

На смену гонке за гигагерцами постепенно приходит погоня за числом процессорных ядер. Intel Core i7 980X (2010)- первый шестиядерный процессор для настольных компьютеров.

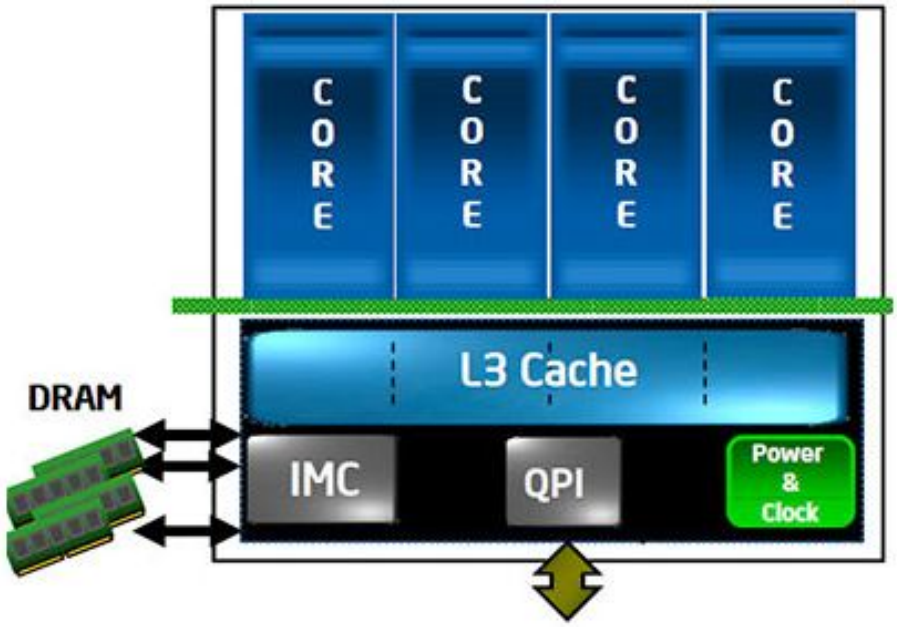
Однокристальное устройство: все ядра, контроллер памяти, контроллер PCI-E и кэш находятся на одном кристалле.

Поддержка Hyper-threading, с которым получается до 12 (в зависимости от модели CPU) виртуальных ядер.

Наборы команд - x86, x86-64, MMX, SSE, SSE2, SSE3,SSSE3, SSE4.1, SSE4.2, AES-NI.

Частота ЦП - 1.07 - 4.2 GHz.

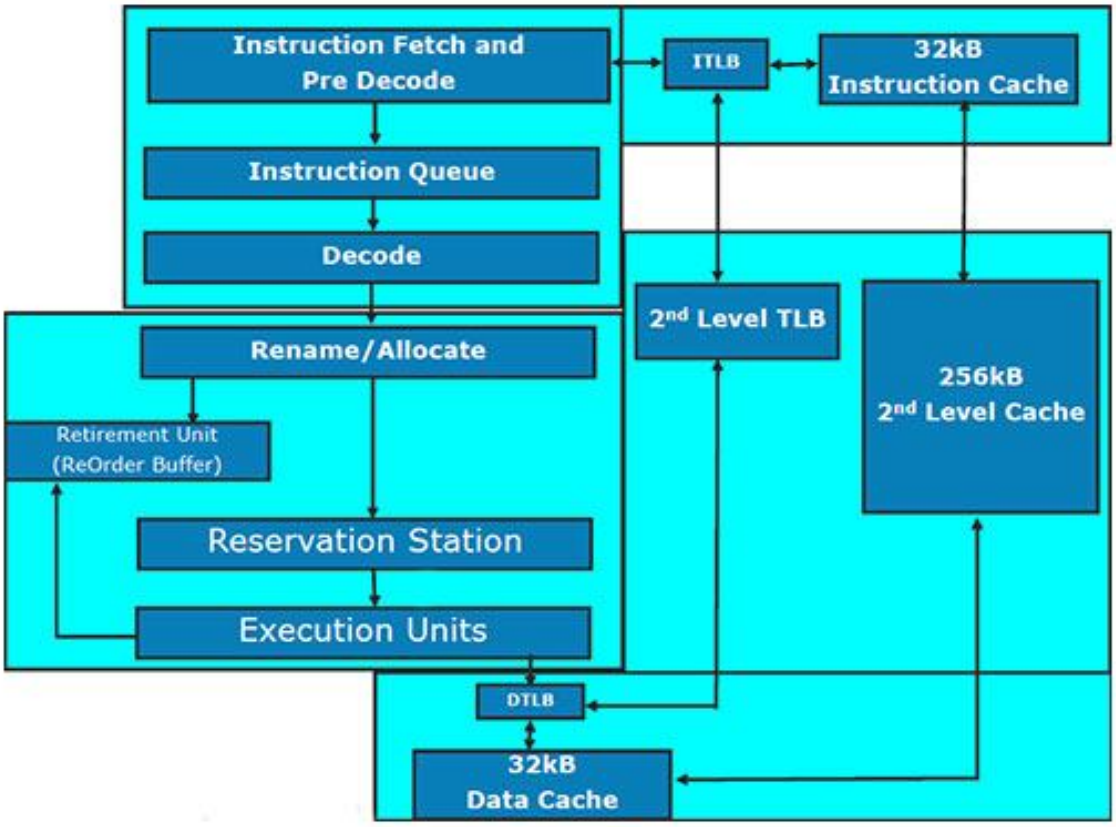
Технология производства 45—22 нм.



Блоки процессора

- разделяемый кэш 3-го уровня;
- контроллер памяти;
- контроллер шины QPI (QuickPath Interconnect);
- контроллер шины PCI Express;
- контроллер энергопотребления (PCU) и генератор частот;
- контроллер интегрированной графики.

Исполнительное ядро Core i7



http://www.ixbt.com/cpu/intel-ci7-theory.shtml

RISC процессоры

CISC – Complex Instruction Set Computer
RISC – Reduced Instruction Set Computer

Идеи RISC:

1. Команда выполняется за такт.
2. Отсутствует (ограничен) объем микрокода.
3. Упрощен набор команд и методов адресации.
4. Работа с памятью ограничена пересылками типа "регистр-память".
5. Большой размер регистровых файлов (больше 32).
6. Развитый конвейер команд. Он эффективно загружен за счет унификации форматов команд.
7. Сложные команды (функции аппаратуры) выполняются набором простых команд.

Пример уменьшения времени выполнения двух CISC команд RISC командами:

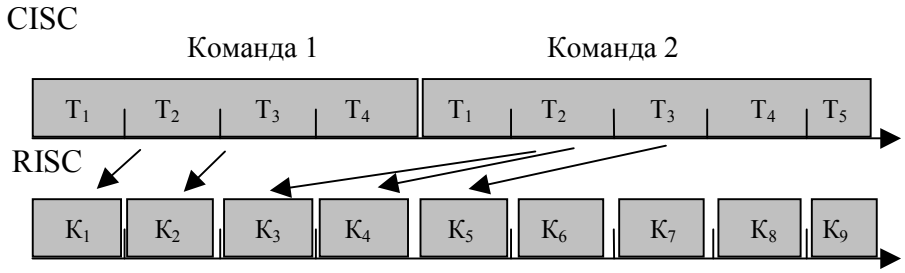


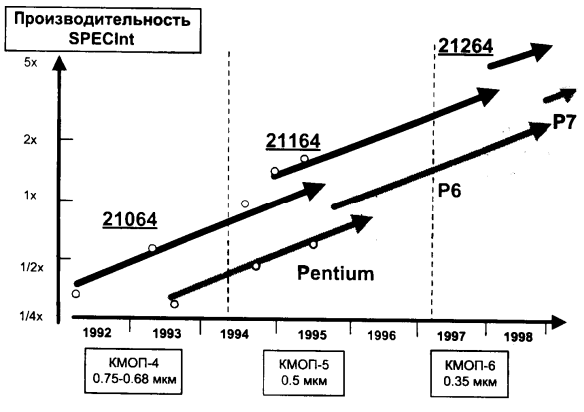
Рис. 8.9. Временная диаграмма исполнения команд процессоров CISC и RISC

2.2. Микропроцессоры с архитектурой Alpha

Микропроцессоры Alpha являются хорошей иллюстрацией концепции достижения высокой производительности за счет увеличения тактовой частоты при относительно простой логике функционирования. На протяжении многих лет эти процессоры являются самыми быстродействующими. В настоящее время семейство микропроцессоров с архитектурой Alpha представлено несколькими кристаллами, имеющими различные диапазоны производительности, работающие с разной тактовой частотой и рассеивающие разную мощность.

Микропроцессоры Alpha 2106x

Впервые представлен в 1992 году, а феврале 1993г. был выпущен первый из микропроцессоров - Alpha 21064 с тактовой частотой 200 Мц, (0,75 мкм КМОП 1,68 млн). 21064 стал первым процессором, занесенным в Книгу рекордов Гиннесса, - в 1992 г. он был назван самым быстрым в мире.



Он представляет собой RISC-процессор в однокристальном исполнении,

в состав которого входят устройства целочисленной и плавающей арифметики, а также кэш-память емкостью 16 Кб. Кристалл проектировался с учетом реализации передовых методов увеличения производительности, включая конвейерную организацию всех функциональных устройств, одновременную выдачу нескольких команд для выполнения, а также средства организации симметричной многопроцессорной обработки.

В кристалле имеются два регистровых файла по 32 64-битовых регистра: один для целых чисел, второй - для чисел с плавающей точкой. Самая мощная модель процессора 21064 работает на частоте 200 МГц. Микропроцессор Alpha 21064 имеет 64-разрядную суперскалярную RISC-архитектуру с двумя исполнительными конвейерами. На кристалле микропроцессоров Alpha 21066/21068 расположены контроллеры: ПДП, графический и шины PCI. Микропроцессор содержит на кристалле раздельные кэш-памяти команд и данных, каждый емкостью 16 Кбайт, по 32 регистра с плавающей и фиксированной точкой. Ширина внешних шин адреса и данных составляет 43 и 128 битов соответственно.

Микропроцессор Alpha 21164

Микропроцессор 21164, представленный в сентябре 1994 года, обеспечивает производительность 330 и 500 единиц, соответственно, по шкалам SPECint92 и SPECfp92 или около 1200 MIPS и выполняет до четырех инструкций за такт. На кристалле микропроцессора 21164 размещено около 9,3 миллиона транзисторов, большинство из которых образуют кэш. Кристалл построен на базе 0.5 микронной КМОП технологии компании DEC. Он собирается в 499-контактные корпуса PGA (при этом 205 контактов отводятся под разводку питания и земли) и рассеивает 50 Вт при питающем напряжении 3.3 В на частоте 300 МГц.

Ключевыми моментами для реализации высокой производительности является суперскалярный режим работы процессора, обеспечивающий выдачу для выполнения до четырех команд в каждом такте, высокопроизводительная неблокируемая подсистема памяти с быстродействующей кэш-памятью первого уровня, большая, размещенная на кристалле, кэш-память второго уровня и уменьшенная задержка выполнения операций во всех функциональных устройствах.

Микропроцессоры Alpha 21264 21364

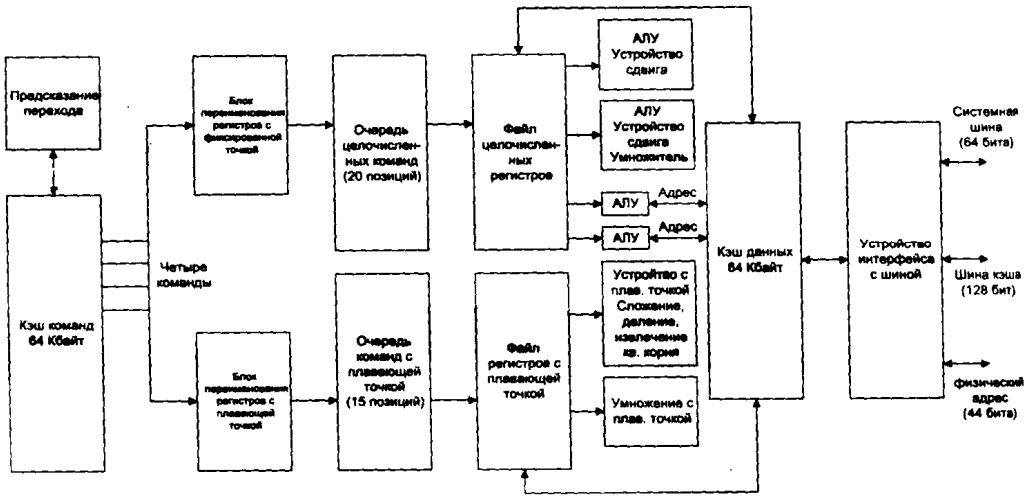


Рис. 2.11. Структурная схема процессора Alpha 21264

В декабре 1998 г. был объявлен новый микропроцессор шестого поколения Alpha – 21264 **(**500 МГц, 15,2 млн. транз., кристалл 310 мм^2**)**. Микропроцессор Alpha 21264 - суперскалярный 64-разрядный RISC-процессор с производительностью в 2GF/s. Обьем кэша L1 составляет 64 KB для данных и инструкций. Микропроцессор не содержит кэш-памяти 2-го уровня. Процессор содержит два блока операций с плавающей запятой, выполняющих сложение, умножение, деление, извлечение квадратного корня, и четыре целочисленных исполнительных устройства: два – общего назначения и два - адресных. Последнее наряду с простыми арифметическими и логическими операциями выполняют все команды загрузки и сохранения данных. Целочисленные АЛУ общего назначения выполняют арифметические и логические операции, сдвиги и переходы. Одно из целочисленных АЛУ выполняет также умножение, а другое - новый набор команд видео данных. Для динамического переименования доступны 41 из 80 целочисленных регистров и 41 из 72 регистров с плавающей точкой. Для динамического исполнения в микропроцессоре рассматриваются сразу 80 команд. После декодирования команды помещаются в одну из очередей: к устройствам с фиксированной или плавающей запятой. Команды, получившие все операнды, конкурируют за доступ к исполнительным устройствам. Внеочередное выполнение команд осуществляется как для очереди целочисленных команд, так и для очереди команд с плавающей запятой. Больший приоритет имеют команды, которые дольше находятся в очереди. 21264 способен выполнять до шести команд за такт (поддерживаемый уровень - четыре команды за такт). В это число включены также команды загрузки регистров и записи в память.

Механизм динамического исполнения команд:

- динамическое планирование с изменением последовательности команд,
- переименование регистров,
- спекулятивное выполнение команд.

Конвейер команд:

- выборка команды с учетом предсказания перехода;
- передача данных для команды в устройство переименования (отображения) регистров;
- выполнение переименования (отображения) регистров;
- выбор команды из очередей на выполнение;
- выполнение целочисленных команд или команд с плавающей точкой;
- запись результатов выполнения.

Микропроцессор 21364 использует ядро 21264, однако на его кристалле, содержащем 100 млн. транзисторов, интегрированы частично ассоциативный кэш второго уровня емкостью 1,5 Мбайт, контроллер памяти, поддерживающий работу с динамической памятью Direct Rambus, и сетевой интерфейс. Фактически 21364 представляет собой целую мини-систему с огромной полосой пропускания, высокоскоростными портами и собственной памятью. Такая конструкция обеспечит сокращение задержек и по крайней мере удвоит производительность исполнения приложений на симметричных многопроцессорных системах даже без повышения тактовой частоты. Благодаря встроенному сетевому интерфейсу упрощается объединение микропроцессоров в высокопроизводительные многопроцессорные системы. Сетевой интерфейс поддерживает 4 межпроцессорных соединения типа «точка-точка» со скоростью передачи данных 10 Гбайт/с каждый при задержке 15 нс. Сетевой интерфейс обеспечивает когерентность кэшей в многопроцессорной системе и реализует асинхронный обмен данными с адаптивной маршрутизацией.

2.3. Микропроцессоры с архитектурой PowerPC

Архитектура RISC-процессоров PowerPC (Performance Optimized With Enhanced RISC) разработана совместно тремя компаниями Apple, IBM и Motorola. Семейство Power в настоящее время состоит из следующих моделей: 603e, 750, G4, POWER7 и др.

PowerPC 601

PowerPC 601 (1991г.) представляет собой процессор среднего класса и предназначен для использования в настольных вычислительных системах малой и средней стоимости. Процессор 601 базировался на однокристальном процессоре IBM, который был разработан к моменту создания альянса трех ведущих фирм. Но по сравнению со своим предшественником, PowerPC 601 претерпел серьезные изменения в сторону повышения производительности и снижения стоимости. В Power 601 реализована суперскалярная обработка, позволяющая выдавать на выполнение в каждом такте 3 команды, возможно не в порядке их расположения в программном коде. На этапе загрузки команд (см. рис.) может происходить изменение последовательности: например команды FPU или BPU могут выполняться ранее команд IU.

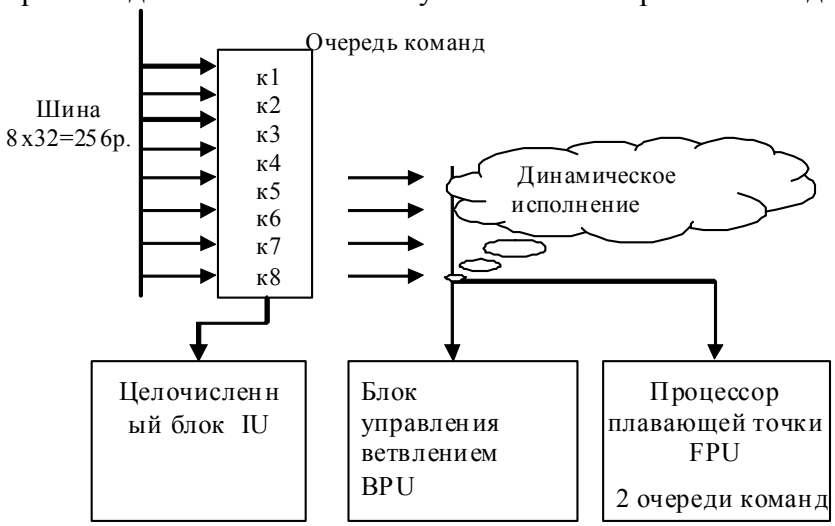


Рис.2.12. Схема загрузки исполнительных блоков

PowerPC 603

PowerPC 603 является первым микропроцессором в семействе PowerPC, который полностью поддерживает архитектуру PowerPC (рис.). Он включает пять функциональных устройств: устройство переходов, целочисленное устройство, устройство плавающей точки, устройство загрузки/записи и устройство системных регистров, а также две, расположенных на кристалле кэш-памяти для команд и данных, емкостью по 8 Кбайт. Поскольку PowerPC 603 - суперскалярный микропроцессор, он может выдавать в эти исполнительные устройства и завершать выполнение до трех команд в каждом такте. Для увеличения производительности PowerPC 603 допускает внеочередное выполнение команд. Кроме того, он обеспечивает программируемые режимы снижения потребляемой мощности, которые дают разработчикам систем гибкость реализации различных технологий управления питанием.

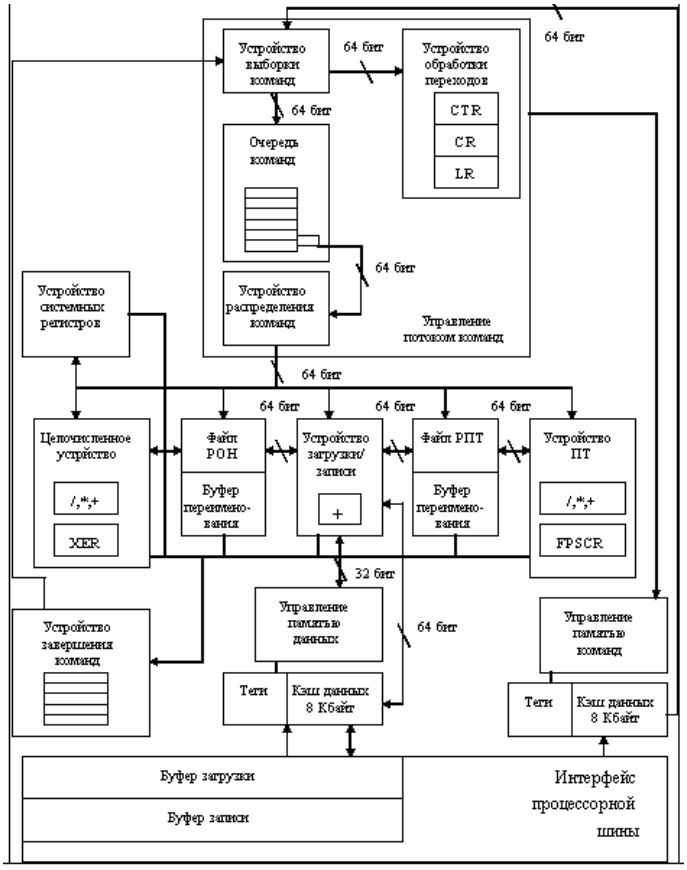


Рис. 2.13. Структурная схема процессора PowerPC 603

При обработке в процессоре команды распределяются по пяти исполнительным устройствам в заданном программой порядке. Если отсутствуют зависимости по операндам, выполнение происходит немедленно. Целочисленное устройство выполняет большинство команд за один такт. Устройство плавающей точки имеет конвейерную организацию и выполняет операции с плавающей точкой, как с одинарной, так и с двойной точностью. Команды условных переходов обрабатывается в устройстве переходов. Если условия перехода доступны, то решение о направлении перехода принимается немедленно, в противном случае выполнение последующих команд продолжается по предположению (спекулятивно). Команды, модифицирующие состояние регистров управления процессором, выполняются устройством системных регистров. Наконец, пересылки данных между кэш-памятью данных, с одной стороны, и регистрами общего назначения и регистрами плавающей точки, с другой стороны, обрабатываются устройством загрузки/записи.

В случае промаха при обращении к кэш-памяти, обращение к основной памяти осуществляется с помощью 64-битовой высокопроизводительной шины. После окончания выполнения команды в исполнительном устройстве ее результаты направляются в буфер завершения команд (completion buffer) и затем последовательно записываются в соответствующий регистровый файл по мере изъятия команд из буфера завершения. Для минимизации конфликтов по регистрам, в процессоре PowerPC 603 предусмотрены отдельные наборы из 32 целочисленных регистров общего назначения и 32 регистров плавающей точки.

Очередь команд и устройство распределения

Очередь команд (IQ) содержит шесть команд и может быть загружена двумя командами за такт. Устройство выборки пытается загрузить команды на все свободные

места в очереди. Все команды распределяются к соответствующим устройствам выполнения (IU1, FPU, LSU, SRU) из двух верхних позиций в очереди с максимальной скоростью две за такт. Устройство распределения проверяет зависимости регистров источника и приемника, определяет свободна ли место в очереди завершения команд, и распределяет последовательные команды по назначению.

Устройство обработки переходов

BPU получает команды перехода из устройства выборки и делает упреждающий поиск условных ветвей для их раннего предсказания, достигая попадания в большинстве случаев.

Команды безусловного перехода или с известным условием могут быть предсказаны сразу. Для переходов с неопределенными условиями используется статическое предсказание. Команды из предсказанной ветви выполняются, но не завершаются и не записывают результаты до подтверждения корректности перехода.

Когда переход сделан (или предсказан), команды из остальных ветвей удаляются и загружаются команды из нужной ветви. BPU содержит сумматор для вычисления адресов переходов и использует три регистра - регистр связи (LR), регистр-счетчик (CTR) и CR. BPU вычисляет точку возврата из процедуры и сохраняет результат в LR определенных команд перехода. Также в регистрах LR и CTR содержат адреса для некоторых команд обработки переходов. Из-за использования специальных регистров обработка команд переходов независима от выполнения целочисленных команд и команд с ПТ.

Устройство завершения команд

В точке распределения команд, порядок выполнения команд поддерживается назначение команде места в очереди завершения на 6 мест. Устройство завершения отслеживает команды от распределения через устройства выполнения и возвращает результаты в порядке выполнения команд в программе из 2 нижних мест в очереди выполнения.

Команда не может быть отправлена на выполнение, если нет места в очереди завершения. Команды перехода, не модифицирующие CTR и LR удаляются из потока команд и не занимают места в очереди завершения. Команды, модифицирующие CTR и LR занимают место в очереди, но не посылаются на выполнение.

Завершение команды состоит в записи результатов в регистры (GPR, FPR, LR и CTR). Завершенные команды удаляются из очереди завершения.

Устройства выполнения

1. Устройства выполнения целочисленных команд (IU) состоит из трех однотактовых подустройств - быстрый сумматор/компаратор, обработки логических операций и выполнения сдвигов и циклических сдвигов.

2. Устройство выполнения команд с плавающей точкой (FPU). FPU выполняет операции одинарной точности (32 разряда) за один проход, состоящий из трех тактов. FPU поддерживает все форматы с ПТ стандарта IEEE 754 (нормализованные, ненормализованные, NaN, ноль, бесконечность).

3. Устройство загрузки/записи (LSU). LSU выполняет все команды загрузки и сохранения и предоставляет интерфейс пересылки данных между GPR, FPR, и подсистем кэш/память.

Команды загрузки и записи выполняются в порядке программы; однако некоторые обращения в память могут происходить вне очереди команд. Команды синхронизации могут быть использованы для изменения порядка команд. Максимум одна операция загрузки из кэша вне очереди может быть выполнена за такт, с двухтактовой задержкой загрузки из кэша. Сохранение не может выполняться вне

очереди и операции сохранения находятся в очереди сохранения до разрешения на запись.

4. Устройство системных регистров (SRU). SRU выполняет различные команды системного уровня, такие как логические операции с регистром условия и команды работы с регистрами специального назначения. Команды, выполняемые SRU, сохраняются в нем и обрабатываются после выполнения всех предыдущих команд.

Устройство управления памятью (MMU)

MMU поддерживает до 4 Петабайт (252) виртуальной памяти и до 4 Гигабайт (232) физической памяти для команд и данных со страницами по 4 Кб и сегментами по 256 Мб. MMU контролирует привилегии доступа , разбивая память на блоки и страницы. Вообще, механизм преобразования адресов состоит в преобразовании эффективного адреса в промежуточный виртуальный исходя из сегментной информации и затем в физический по таблицам страниц. Дескрипторы сегментов, используемые для генерации промежуточного внутреннего адреса, хранятся как встроенные 32-разрядные сегментные регистры.

LSU и устройство управления потоком команд вычисляют эффективные адреса данных и команд. MMU преобразует эффективные адреса в физические для доступа к памяти.

Процессор поддерживает следующие режимы преобразования адресов (рис):

- Реальный (real addressing) - физический адрес совпадает с эффективным.
- Страничный (page address translation) - преобразует адреса страниц (4 Кб)
- Блочный (block address translation) - преобразует базовые адреса блоков (от 128 Кб до 256 Мб)

Процессор имеет 12 форматов команд и следующие способы адресации:

1. Регистровая
2. Косвенно-регистровая [reg]
3. Косвенно-регистровая с непосредственным смещением #const[reg]
4. Косвенно-рег. индексная [reg1] [reg2]

Среди форматов команд основными являются два: трех и четырех операндные.

Целочисленный - трехоперандный

```
add r0=r1+r2
cmpi r8=r2,r0
```

КОП	D	A	B	OE	disp/ imm/ extКОП	R_C

A+B > D

С плавающей запятой - четыхоперандный

```
fmadds fr0=fr1·fr10+fr2
```

КОП	FD	FA	FB	FC	X	R_C

FA·FC+B > D

OE – разрешение переполнения (overflow enable)

Rc – разрешение записи в р-р состояния (record bit)

PowerPC 620

PowerPC 620 (1995г. 7 млн. транзисторов 133 Мгц 64-разряда). В отличие от своих предшественников процессор PowerPC 620 полностью 64-битовый процессор. Подобно другим 64-битовым процессорам, PowerPC 620 содержит 64-битовые регистры общего назначения и плавающей точки и обеспечивает формирование 64-

битовых виртуальных адресов. При этом сохраняется совместимость с 32-битовым режимом работы, реализованным в других моделях семейства PowerPC.

В процессоре имеется кэш-память данных и команд общей емкостью 64 Кбайт, интерфейсные схемы управления кэш-памятью второго уровня, 128-битовая шина данных между процессором и основной памятью, а также логические схемы поддержания когерентного состояния памяти при организации многопроцессорной системы.

Основные особенности:

- RISC-ядро
- четырехконвейерная суперскалярная архитектура
- преддекодирование,
- буфер переупорядочивания на 16 элементов
- 6 независимых исполнительных блоков,
- 2 встроенных кэша первого уровня по 32Кбайт каждый
- переключение режимов адресации (Intel/Motorola)
- пространство ввода-вывода отражено на память

Для поддержания эффективной загрузки исполнительных блоков в процессоре применяется динамическое предсказание переходов совместно со спекулятивным выполнением кода на глубину до 4 предсказанных ветвлений. Шинный интерфейс включает унифицированную внутреннюю поддержку кэша 2-го уровня, не требует дополнительных тактов для управления логикой внешнего кэша. Кэш данных реализует режим сквозной и обратной записи и протокол MESI (Modified, Exclusive, Shared, Invalid), обеспечивающий синхронизацию кэшей в мультипроцессорных системах.

Перед тем как попасть во внутренний кэш, команды проходят через декодер. Декодированные команды находятся в кэше команд до их выборки блоком планирования/выполнения. Благодаря **предварительному декодированию** остальная логика декодирования объединяется с этапом планирования загрузки конвейеров микропроцессора, что позволяет сократить число этапов конвейера до 5 (выборка, декодирование/планирование: выполнение, завершение и запись).

Уникальной особенностью микропроцессоров PowerPC является так- же программное переключение режимов адресации (Intel/Motorola). Этот режим также определяется одним из битов MPR. Таким образом, рабочая станция на базе PowerPC 620 сможет выполнять код приложений разных ОС

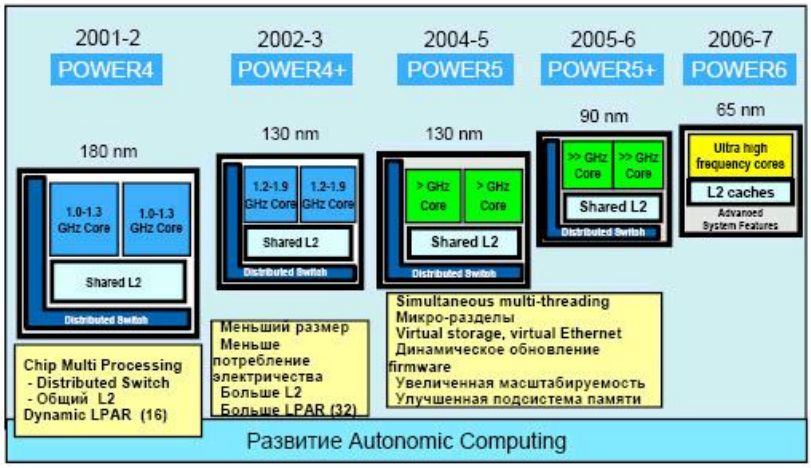
IBM POWER

(http://www.intuit.ru/department/hardware/adibm/2/)

В 1990 г. корпорация IBM начала выпуск первых систем RS/6000® на базе процессора POWER. POWER5 был выпущен в 2004 году Версия на 1,9 ГГц показала самые высокие результаты в тесте SPECfp для однопроцессорных систем среди всех коммерчески доступных процессоров.

120 регистров общего назначения (GPR) и 120 регистров с плавающей точкой (FPR). Каждое процессорное ядро имеет отдельный 64 КБ L1 кэш инструкций и 32 КБ L1 кэш данных. L1 кэш совместно используется двумя нитями на процессорном ядре. Оба процессорных ядра совместно используют 1.88 МБ L2 кэш. На процессорном чипе находится контроллер L3 кэша.Сам L3 кэш находится на отдельном чипе - Merged Logic DRAM (MLD) cache chip. L3 кэш - 36 МБ victim cache L2 кэша. L3 совместно

используется двумя процессорными ядрами чипа POWER5. L2 и L3 кэши совместно используются всеми аппаратными нитями на обоих процессорных нитях в чипе.



IBM POWER Simultaneous multi-threading

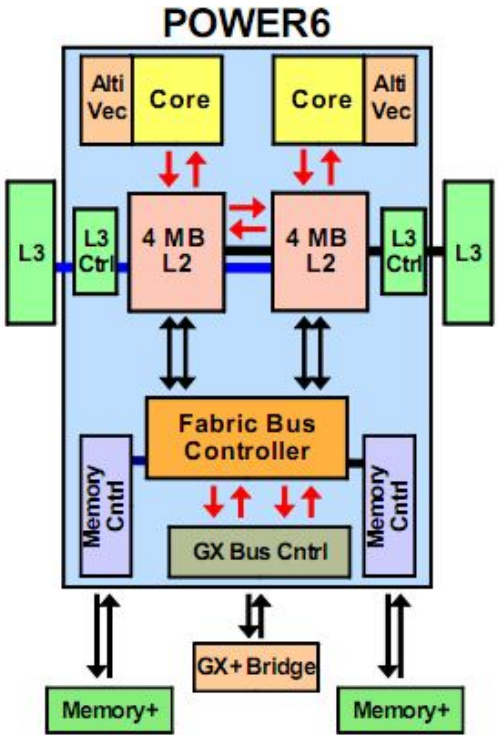
- •Каждый чип виден программному обеспечению как **4-way SMP**. (Каждое из двух ядер поддерживает 2 потока задач, итого 4 потока может исполняться одновременно.)•В режиме SMT процессор выбирает команды от больше чем одной нити. Выпускается в вариантах с 1 процессором и с 4 процессорами на плате.
- **Динамическая балансировка ресурсов**

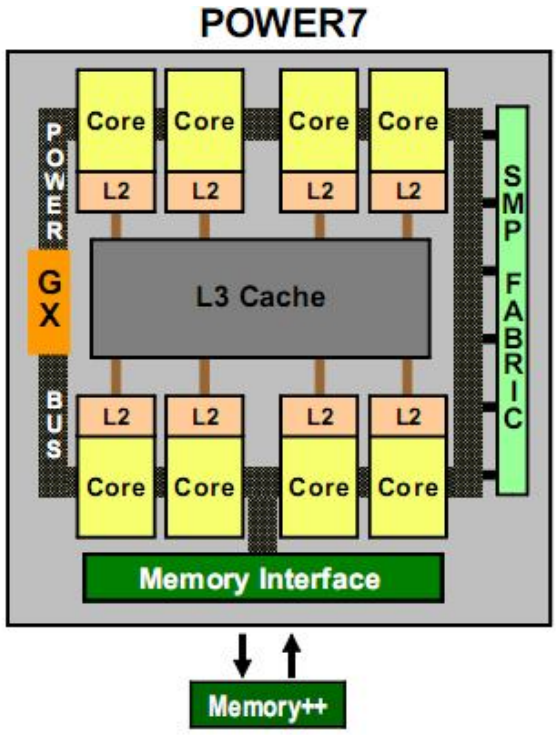
Цель динамической балансировки ресурсов состоит в том, чтобы гарантировать, что эти потоки, выполняющиеся на одном процессоре, равномерно загружают систему. Динамическая балансировка ресурсов контролирует ресурсы, чтобы определить, является ли один поток сильно загружающим их. Например, если один поток сталкивается с множественными промахами по L2 кэшу, зависимые команды могут простаивать в очередях, замедляя другой поток. Чтобы предотвращать это, динамическая балансировка ресурсов приостанавливает такой поток.

POWER6 выпущен 21 мая 2007 года. Привнёс в стандарт POWER инструкции VMX (параллельная обработка данных). Двухъядерный дизайн, тактовые частоты до 4,7 ГГц при нормах выпуска 65 нм. Содержит чрезвычайно развитую систему взаимодействия с другими такими же процессорами. Потребление энергии на уровне POWER5, тогда как производительность вдвое выше.

Power7 содержит восемь ядер, при этом каждое ядро обеспечивает обработку четырех потоков, что в совокупности дает 32 виртуальных ядра.

	POWER5	POWER5+	POWER6	POWER7
Технология	130nm	90nm	60nm	45nm
Размер	389 mm²	245 mm²	341 mm²	567 mm²
Транзисторы	276 M	276 M	790 M	1.2 B
Ядра	2	2	2	4 / 6 / 8
Частота	1.65 GHz	1.9 GHz	4+ GHz	3-4 GHz
L2 Cache	1.9MB Shared	1.9MB Shared	4MB / Core	256 KB / Core
L3 Cache	36MB	36MB	32MB	4MB / Core
Контр. памяти	1	1	2 / 1	2
Вирт. серверы	10 / Core	10 / Core	10 / Core	10 / Core





2.4. Микропроцессоры с архитектурой SPARC

Масштабируемая процессорная архитектура компании Sun Microsystems (SPARC - Scalable Processor Architecture) является наиболее широко распространенной RISC-архитектурой, отражающей доминирующее положение компании на рынке UNIX-рабочих станций и серверов. Процессоры с архитектурой SPARC лицензированы и изготавливаются по спецификациям Sun несколькими производителями, среди которых следует отметить компании Texas Instruments, Fujitsu, LSI Logic, Bipolar International Technology, Philips и Cypress Semiconductor. Эти компании осуществляют поставки процессоров SPARC не только самой Sun Microsystems, но и другим известным производителям вычислительных систем, например, Solbourne, Toshiba, Matsushita, Tatung и Cray Research.

В 1990 году Sun передала все права на архитектуру SPARC организации SPARC International, которая в настоящее время включает более 250 членов. Основными задачами этой организации являются лицензирование технологии SPARC для реализации, руководства и проверки совместимости со стандартами SPARC. Именно такая стратегия лицензирования позволила процессорам с архитектурой SPARC занять лидирующие позиции на рынке RISC-кристаллов

Первоначально архитектура SPARC была разработана с целью упрощения реализации 32-битового процессора. В последствии по мере улучшения технологии изготовления интегральных схем она постепенно развивалось и в настоящее время имеется 64-битовая версия этой архитектуры.

В отличие от большинства RISC архитектур SPARC использует регистровые окна, которые обеспечивают удобный механизм передачи параметров между программами и возврата результатов. Архитектура SPARC была первой коммерческой разработкой, реализующей механизмы отложенных переходов и аннулирования команд. Это давало компилятору большую свободу заполнения времени выполнения команд перехода командой, которая выполняется в случае выполнения условий перехода и игнорируется в случае, если условие перехода не выполняется.

Первый процессор SPARC был изготовлен компанией Fujitsu на основе вентильной матрицы, работающей на частоте 16.67 МГц. На основе этого процессора была разработана первая рабочая станция Sun-4 с производительностью 10 MIPS, объявленная осенью 1987 года (до этого времени компания Sun использовала в своих изделиях микропроцессоры Motorola 680X0). В марте 1988 года Fujitsu увеличила тактовую частоту до 25 МГц создав процессор с производительностью 15 MIPS.

Дальнейшее увеличение производительности процессоров с архитектурой SPARC было достигнуто за счет реализации в кристаллах принципов суперскалярной обработки компаниями Texas Instruments и Cypress. Процессор SuperSPARC компании Texas Instruments стал основой серии рабочих станций и серверов SPARCstation/SPARCserver 10/20. Имеется несколько версий этого процессора, позволяющего в зависимости от смеси команд обрабатывать до трех команд за один машинный такт, отличающихся тактовой частотой. Процессор SuperSPARC имеет сбалансированную производительность на операциях с фиксированной и плавающей точкой. Он имеет внутренний кэш емкостью 36 Кб (20 Кб - кэш команд и 16 Кб - кэш данных), раздельные конвейеры целочисленной и вещественной арифметики и при тактовой частоте 75 МГц обеспечивает производительность около 205 MIPS.

При создании своего нового процессора UltraSPARC-1 конструкторы из Sun решили добиться увеличения производительности процессора в тех направлениях, где это не противоречило экономическим соображениям. Чтобы сократить число

потенциальных проблем, было принято несколько принципиальных решений, которые определили основные характеристики UltraSPARC-1:

- Реализация на кристалле раздельной кэш-памяти команд и данных;
- Организация широкой выборки команд (128 бит);
- Создание эффективных средств динамического прогнозирования направления переходов;
- Реализация девятиступенчатого конвейера, обеспечивающего выдачу для выполнения до четырех команд в каждом такте;
- Оптимизация конвейерных операций обращения к памяти;
- Реализация команд обмена данными между памятью и регистрами плавающей точки, позволяющая не приостанавливать диспетчеризацию команд обработки;
- Реализация на кристалле устройства управления памятью (MMU);
- Расширение набора команд для поддержки графики и обработки изображений;
- Реализация новой архитектуры шины UPA.

Процессор **UltraSPARC-1** представляет собой производительный, высокоинтегрированный суперскалярный процессор, реализующий 64-битовую архитектуру SPARC-V9.B его состав входят: устройство предварительной выборки и диспетчеризации команд, целочисленное исполнительное устройство, устройство работы с вещественной арифметикой и модуль графики, устройства управления памятью, загрузки/записи и управления внешней кэш-памятью, модули управления интерфейсом памяти и кэш-памяти команд и данных (рис. 8.16)

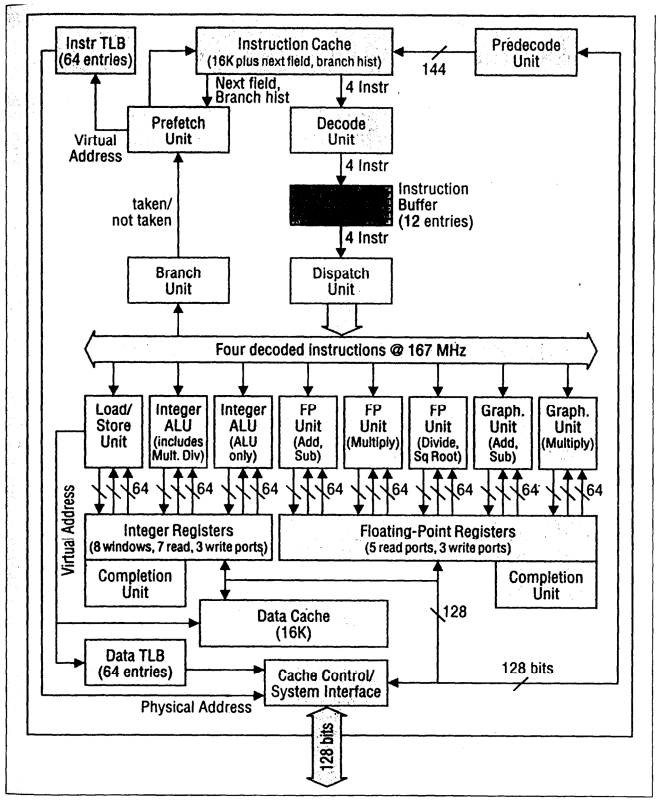


Рис. 2.16. Структурная схема процессора Sun Ultra SPARC-I (64 разряда)

Устройство предварительной выборки и диспетчеризации команд

Устройство предварительной выборки и диспетчеризации команд процессораUltraSPARC-1 обеспечивает выборку команд в буфер команд,

окончательную их дешифрацию, группировку и распределение для параллельного выполнения в конвейерных функциональных устройствах процессора. Буфер команд емкостью12 инструкций позволяет согласовать скорость работы памяти со скоростью обработки исполнительных устройств процессора. Команды могут быть предварительно выбраны из любого уровня иерархии памяти, например, из кэш-памяти команд(I-кэша), внешней кэш-памяти (E-кэша) или из основной памяти системы.

В процессоре реализована схема динамического прогнозирования направления ветвлений программы, основанная на двух битовой истории переходов и обеспечивающая ускоренную обработку команд условного перехода. Для реализации этой схемы с каждыми двумя командами в I-кэше связано специальное поле, хранящее двухбитовое значение прогноза. Таким образом, UltraSPARC-1 позволяет хранить информацию о направлении 2048 переходов, что на сегодняшний день превышает потребности многих современных прикладных программ. Поскольку направление перехода может меняться каждый раз, когда обрабатывается соответствующая команда, состояние двух бит прогноза должно каждый раз модифицироваться для отражения реального исхода перехода. Эта схема особенно эффективна при обработке циклов.

Кроме того, в процессоре UltraSPARC-1 с каждыми четырьмя командами в I-кэше связано специальное поле, указывающее на следующую строку кэш-памяти, которая должна выбираться вслед за данной. Использование этого поля позволяет осуществлять выборку командных строк в соответствии с выполняемыми переходами, что обеспечивает для программ с большим числом ветвлений практически туже самую пропускную способность команд, что и на линейном участке программы. Способность быстро выбрать команды по прогнозируемому целевому адресу команды перехода является очень важной для оптимизации производительности суперскалярного процессора и позволяет UltraSPARC-1 эффективно выполнять "по предположению"(speculative) достаточно хитроумные последовательности условных переходов.

Используемые в UltraSPARC-1 механизмы динамического прогнозирования направления и свертки переходов сравнительно просты в реализации и обеспечивают высокую производительность. По результатам контрольных испытаний UltraSPARC-1 88% переходов по условиям целочисленных операций и 94% переходов по условиям операций с плавающей точкой предсказываются успешно.

Кэш-память команд

Кэш-память команд (I-кэш) представляет собой двухканальную множественно-ассоциативную кэш-память емкостью 16 Кбайт. Она организована в виде 512 строк, содержащих по 32 байта данных. С каждой строкой связан соответствующий адресный тег. Команды, поступающие для записи в I-кэш проходят предварительное декодирование и записываются в кэш-память вместе с соответствующими признаками, облегчающими их последующую обработку. Окончательное декодирование команд происходит перед их записью в буфер команд.

Организация конвейера

В процессоре UltraSPARC-1 реализован девятиступенчатый конвейер. Это означает, что задержка (время от начала до конца выполнения) большинства команд составляет девять тактов. Однако в любой данный момент времени в процессе обработки могут одновременно находиться до девяти команд, обеспечивая во многих случаях завершение выполнения команд в каждом такте. В действительности эта скорость может быть ниже в связи с природой самих команд, промахами кэш-памяти или другими конфликтами по ресурсам. Первая ступень конвейера С выборка из кэш-памяти команд. На второй ступени команды декодируются и помещаются в буфер

команд. Третья ступень осуществляет группировку и распределение команд по функциональным исполнительным устройствам. В каждом такте на выполнение в исполнительные устройства процессора могут выдаваться по 4 команды: не более двух целочисленных команд или команд плавающей точки/графических команд, одной команды загрузки/записи и одной команды перехода. На следующей ступени происходит выполнение целочисленных команд или вычисляется виртуальный адрес для обращения к памяти, а также осуществляются окончательное декодирование команд плавающей точки (ПТ) и обращение к регистрам ПТ. На пятой ступени происходит обращение к кэш-памяти данных. Определяются попадания и промахи кэш-памяти и разрешаются переходы. При обнаружении промаха кэш-памяти, соответствующая команда загрузки поступает в буфер загрузки. С этого момента целочисленный конвейер ожидает завершения работы конвейеров плавающей точки/графики, которые начинают выполнение соответствующих команд. Затем производится анализ возникновения исключительных ситуаций. На последней ступени все результаты записываются в регистровые файлы и команды изымаются из обработки.

Целочисленное исполнительное устройство

Главной задачей при разработке целочисленного исполнительного устройства (IEU) является обеспечение максимальной производительности при поддержке программной совместимости с существующим системным и прикладным программным обеспечением. Целочисленное исполнительное устройство UltraSPARC-1 объединяет в себе несколько важных особенностей:

- 2 АЛУ для выполнения арифметических и логических операций, а также операций сдвига;
- Многократные целочисленные устройства умножения и деления;
- регистровый файл с восемью окнами и четырьмя наборами глобальных регистров;
- реализация цепей ускоренной пересылки результатов;
- реализация устройства завершения команд, которое обеспечивает минимальное количество цепей обхода (ускоренной пересылки данных) при построении девятиступенчатого конвейера.

Устройство загрузки/записи (LSU)

LSU отвечает за формирование виртуального адреса для всех команд загрузки и записи (включая атомарные операции), за доступ к кэш-памяти данных, а также за буферизацию команд загрузки в случае промаха D-кэша (в буфере загрузки) и буферизацию команд записи (в буфере записи). В каждом такте может выдаваться для выполнения одна команда загрузки и одна команда записи.

Устройство плавающей точки (FPU)

Конвейерное устройство плавающей точки построено в соответствии со спецификациями архитектуры SPARC-V9 и стандарта IEEE 754. Оно состоит из пяти отдельных функциональных устройств и обеспечивает выполнение операций с плавающей точкой и графических операций. Реализация раздельных исполнительных устройств позволяет UltraSPARC-1 выдавать и выполнять две вещественных операции в каждом такте. Операнды-источники и результаты операций хранятся в регистровом файле емкостью 32 регистра. Большинство команд полностью конвейеризованы имеют пропускную способность 1 такт, задержку в 3 такта и не зависят от точности операндов, имея одну и ту же задержку для одинарной и двойной точности. Команды деления и вычисления квадратного корня не конвейеризованы и без остановки процессора выполняются за 12/22 такта (одинарная/двойная точность). Команды, следующие за

командами деления/вычисления квадратного корня, могут выдаваться, выполняться и изыматься из обработки для фиксации результата в регистровом файле до момента завершения команд деления/вычисления квадратного корня. Процессор поддерживает модель точных прерываний путем синхронизации конвейера плавающей точки с целочисленным конвейером, а также с помощью средств прогнозирования исключительных ситуаций для операций с большим временем выполнения. FPU может работать с нормализованными и ненормализованными числами с одинарной (32 бит) и двойной точностью (64бит), а также поддерживает операции над числами с учетверенной точностью(128 бит).

Графическое устройство (GRU)

В процессоре UltraSPARC-1 реализован исчерпывающий набор графических команд, которые обеспечивают аппаратную поддержку высокоскоростной обработки плоских и трехмерных изображений, а также обработку видеоданных. GRU выполняет операции сложения, сравнения и логические операции над 16-битовыми и 32-битовымицелыми числами, а также операции умножения над 8-битовыми и 16-битовыми целыми. В GRU поддерживаются однотактные операции определения расстояния между пикселями, операции выравнивания данных, операции упаковки и слияния.

Устройство управления памятью (MMU)

Высокая суперскалярная производительность процессора поддерживается соответствующей скоростью поступления для обработки команд и данных. Обычно эта задача ложится на иерархию памяти системы. Устройство управления памятью процессора UltraSPARC-1 выполняет все операции обращения к памяти, реализуя необходимые средства поддержки виртуальной памяти. Виртуальное адресное пространство задачи определяется 64-битовым виртуальным адресом, однако процессор UltraSPARC-1 поддерживает только 44-битовое виртуальное адресное пространство. Соответствующее преобразование является функцией операционной системы.

В свою очередь, MMU обеспечивает отображение 44-битового виртуального адреса в 41-битовый физический адрес памяти. Это преобразование выполняется с помощью полностью ассоциативных 64-строчных буферов: iTLB C для команд и dTLB C для данных. Каждый из этих буферов по существу представляет собой полностью ассоциативную кэш-память дескрипторов страниц. В каждой строке TLB хранится информация о виртуальном адресе страницы, соответствующем физическом адресе страницы, а также о допустимом режиме доступа к странице и ее использовании. Процесс преобразования виртуального адреса в физический заканчивается сразу, если при поиске в кэш-памяти TLB происходит попадание(соответствующая строка находится в TLB). В противном случае замещение строки TLB осуществляется специальным аппаратно-программным механизмом. MMU поддерживает четыре размера страниц: 8Кбайт, 64Кбайт, 512Кбайт и 4Мбайт.

Как уже было отмечено, MMU реализует также механизмы защиты и контроля доступа к памяти. В результате выполняющийся процесс не может обращаться к адресному пространству других процессов, и, кроме того, гарантируется заданный режим доступа процесса к определенным областям памяти C на базе информации о допустимом режиме доступа к страницам памяти. Например, процесс не может модифицировать страницы памяти, доступ к которым разрешен только по чтению или которые зарезервированы для размещения системных программ.

Таким образом, основными новыми архитектурными решениями являются**:**

- **Концепция регистровых окон.** Используется 200 регистров, образующих 8 групп (окон) по 32 регистра с общими для двух соседних окон восемью регистрами.
- Предварительное декодирование. 9-ти этапный конвейер.
- Поддержка графических операций. Одна команда над 8 элементами.
- Внешняя шина 128 бит.

Ultra SPARC III является представителем последнего поколения микропроцессоров разработки компании Sun Microsystems. Существенной особенностью, отличающей Ultra SPARC III является наличие у него встроенного контроллера памяти. Память используется, в общем-то, тривиальная - SDRAM, но ее шина имеет ширину 144 разряда.

Особенностью работы процессора Ultra SPARC III является то, что в нем нет ставшего уже привычным внеочередного исполнения! Команды запускаются в том порядке, в котором они перечислены в программе. Однако архитектура процессора позволяет запустить на исполнение до 6 команд одновременно: четыре целочисленные (две arithmetic, logical или shift, одну load или store и одну на BR pipeline) и две команды работы с плавающей точкой, что намного больше, чем в x86-процессорах. Естественно, реальное количество параллельно исполняемых команд несколько меньше и составляет в среднем (на типичных приложениях) 3-4.

Интересной особенностью данного процессора является то, что результаты исполнения становятся доступны другим командам не после прохождения всего конвейера, а на следующей после получения результата стадии. Такая возможность возникает благодаря использованию дополнительного специального "рабочего" (т. е. невидимого для программиста) регистрового файла, с которым и производятся все манипуляции и откуда результаты переписываются (по завершении прохождения командой конвейера) в системный регистровый файл.

По состоянию на июнь 2011 самым быстрым суперкомпьютером в рейтинге TOP500 признан «K computer» компании Fujitsu он собран из 68 544 восьмиядерных процессоров SPARC64 VIIIfx и его мощность составляет 8,16 Пфлопс,

Процессоры с длинным командным словом

Архитектура VLIW представляет собой одну из реализаций концепции внутреннего параллелизма в микропроцессорах. Их быстродействие можно повысить двумя способами: увеличив либо тактовую частоту, либо количество операций, выполняемых за один такт. В первом случае требуется применение «быстрых» технологий.

Планирование порядка вычислений — довольно трудная задача, которую приходится решать при проектировании современного процессора. В суперскалярных архитектурах для распознавания зависимостей между машинными инструкциями применяется специальное довольно сложное аппаратное решение (например, в P6- и post-P6-архитектуре от Intel для этого используется буфер переупорядочивания инструкций — ReOrder Buffer, ROB). Однако размеры такого аппаратного планировщика при увеличении количества функциональных модулей обработки возрастают в геометрической прогрессии, что, в конце концов, может занять весь кристалл процессора. Поэтому суперскалярные проекты остановились на отметке 5-6 обрабатываемых за цикл инструкций. В качестве примера можно привести сверхсложный (и сверхдорогой) процессор Power4 от IBM, способный в теории выполнять до 8 команд за такт, но реально редко подымающийся выше 4--6. На самом же деле, текущие реализации VLIW тоже далеко не всегда могут похвастаться 100% заполнением пакетов — реальная загрузка около 6-7 команд в такте. Можно передать все планирование программному обеспечению, как это делается в конструкциях с VLIW. В значительной степени способность параллельного и (или) спекулятивного исполнения определяется и "полем зрения" системы оптимизации. Вполне очевидно, что у компилятора, обрабатывающего программу целиком и не стесненного временными рамками скорости исполнения кода, эти возможности несравнимо шире. «Умный» компилятор должен выискать в программе все инструкции, которые являются совершенно независимыми, собрать их вместе в очень длинные строки (длинные инструкции) и затем отправить на одновременное исполнение функциональными модулями,

VLIW-процессоры отличаются от суперскалярных тем, что код для них организован в виде последовательности очень длинных командных слов, каждое из которых содержит несколько команд (операций). Забота о корректном заполнении командных слов возлагается на компилятор (или программиста, пишущего на ассемблере). VLIW-процессоры в целом производительнее суперскалярных, поскольку не тратят время на динамический анализ зависимостей по данным и функциональным устройствам во время выполнения программы. Однако реальная эффективность выполнения программы целиком зависит от качества кода, сгенерированного компилятором.

2.5. Микропроцессоры с архитектурой Itanium

В августе 1999 г. впервые появились опытные образцы процессора, а осенью Intel представила Itanium как коммерческое наименование своего первого 64-разрядного процессора. Это процессор типа EPIC (Explicitly Parallel Instruction Computing).

Процессор Intel® Itanium® 2 является 64-битным. Это позволяет поднять производительность за счет обработки данных 64-разрядного формата. Кроме того, архитектура IA-64 позволяет преодолеть одну из главных трудностей современных 32-

разрядных процессоров — непосредственную адресацию только 4 Гб. Процессор Intel® Itanium® 2 может напрямую работать с 16 Тб оперативной памяти (64-битная виртуальная адресация, 50 бит физической адресации).

Инструкции, обрабатываемые шестью параллельными конвейерами глубиной 8 команд, непосредственно выполняются в 23 функциональных блоках. Исполняющие модули Intel® Itanium® 2:

- 6 целочисленных модулей;
- 2 модуля для вычислений с плавающей точкой;
- 2 модуля для вычислений с плавающей точкой двойной точности;
- 6 модулей для обработки мультимедийных команд;
- 4 модуля, управляющих загрузкой и выгрузкой данных;
- 3 модуля ветвлений.

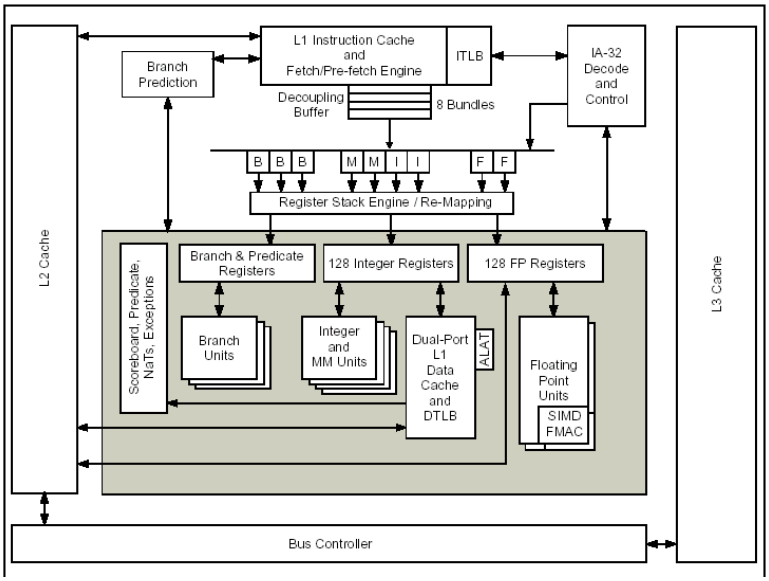


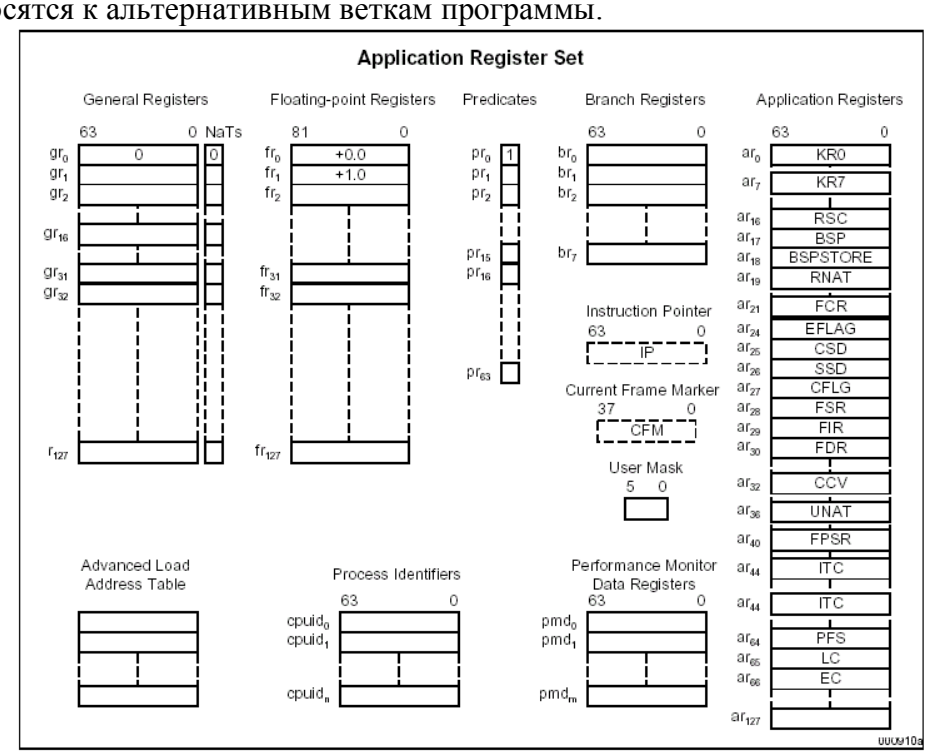
Рис. 8.17. Структурная схема процессора Itanium

Количество соответствующих модулей оптимально подобрано с учетом потребности в вычислительных ресурсах приложений, характерных для корпоративных СУБД, сложных инженерных расчетов и др. При этом команды практически не "задерживаются" в ожидании, пока освободится нужный исполняющий модуль. Ширина системной шины, по которой осуществляется взаимодействие нескольких процессоров друг с другом и доступ к памяти, увеличена до 128 бит, что также кардинально сказывается на скорости работы.

Число регистров, в которых размещаются данные для непосредственной обработки их процессором, свыше трехсот. Большое их количество позволяет избежать дефицита регистров при параллельной обработке многих команд. Кроме того, в процессоре Intel® Itanium® 2 реализованы такие механизмы повышения эффективности работы, как стек и переименование регистров.

В архитектуре Itanium насчитывается по 128 64-разрядных целочисленных регистров общего назначения и 80-разрядных регистров вещественной арифметики, а также 64 одноразрядных предикатных регистра. Все они доступны для программирования; кроме того, имеется множество недоступных внутренних служебных регистров, используемых самим процессором. 64 одноразрядных регистра используются для организации логики предсказания ветвления и выполнения команд в порядке, отличном от последовательного. Предикатные регистры - специальные поля,

служащие для организации одновременного исполнения обеих веток кода. При возникновении такой ситуации предикатные поля помечают, что эти команды относятся к альтернативным веткам программы.

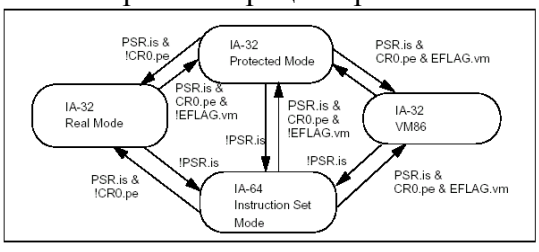


Название	Мнемоника	Размер	Количество
Целочисленные регистры (регистры общего назначения)	GR	64 бита	128
Вещественные регистры	FR	82 бита	128
Предикатные регистры	PR	1 бит	64
Регистры ветвлений	BR	64 бита	8
Прикладные регистры	AR	64 бита	128
Указатель команды	IP	64 бита	1

Рис. 2.18. Набор регистров процессора PowerPC 603

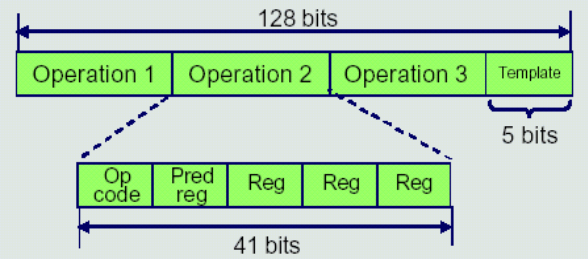
Кэш — быстродействующая внутренняя память процессора — используется для динамического хранения кода и данных, задействованных в вычислениях в текущий момент. Загрузка часто используемых данных и фрагментов кода в кэш позволяет существенно поднять производительность. Доступ к кэшу в несколько раз быстрее, чем к "медленной" оперативной памяти, и составляет 48 Гб/с. Объем кэш-памяти третьего уровня процессора Intel® Itanium® 2 достигает 6 Мб. Архитектура Itanium включает такие уникальные средства повышения надежности, как система расширенного самоконтроля EMCA (Enhanced Machine Check Architecture), обеспечивающая обнаружение, коррекцию и протоколирование ошибок, а также поддержку обработки кода ECC (Error Correcting Code) и контроля четности.

Режимы работы процессора:



Связка инструкций.

Формат командного слова (не команды, а именно слова из трех команд) таков: формируется пакет из трех команд общей длиной 128 бит и специального поля шаблона связки, определяющего внутренние зависимости команд в связке. Команды имеют фиксированную ширину 41 бит и строго подразделяются на 6 типов, что характерно для RISC-архитектуры. Шаблон определяет, какие из команд в связке могут быть исполнены параллельно, а какие последовательно. На исполнение может быть одновременно запущено две трехкомандные связки, таким образом, максимально возможный параллелизм исполнения составляет 6 команд за такт. Довольно небольшая (по современным меркам) глубина конвейера позволяет уменьшить "потери" в случае ошибочного предсказания ветвления, когда требуется перезагрузка конвейера При этом, напомним, весь доступный параллелизм определяется на этапе компиляции.



Итак, код, скомпилированный для процессора Intel® Itanium® 2, представляет собой команды, собранные в связки по три. Причем компилятор сам распределяет инструкции по связкам так, чтобы они, по возможности, были независимы и загружали бы разные исполняющие блоки ЦП, непосредственно осуществляющие вычисления. Это позволяет процессору, почти "не задумываясь", одновременно отправлять поступающие инструкции на выполнение независимыми модулями.

Работа стека регистров

Файл регистров GR отличается от FR и PR тем, что последние содержат фиксированные подмножества статических и вращаемых регистров, в то время как в файле GR вне подмножества статических регистров применяется стек регистров, и программной единице доступна лишь его часть - окно стека регистров. В отличие от статических регистров, стекируемое подмножество локально для любой программной единицы и может иметь размер от 0 до 96 регистров, начиная с GR32. Использование этого механизма связанных с сохранением/восстановлением большого числа регистров при вызовах подпрограмм и возвратах из них (однако статические регистры при необходимости все-таки приходится сохранять и восстанавливать, явно кодируя соответствующие команды). Автоматическое сохранение/восстановление стекируемого подмножества регистров осуществляет RSE, и в программе об этом заботиться не надо. В режиме IA-32 работа с этим стеком регистров, естественно, отключается.

38-разрядный регистр CFM сохраняет состояние "текущего" окна стека регистров. CFM содержит общий размер окна стека, число локальных регистров и (кратное 8) число вращаемых регистров в окне, а также 3 значения базы для переименования регистров - соответственно rrb.gr, rrb.fr и rrb.pr.

Итак, окно стека имеет две области переменного размера - локальную и выходную. Рассмотрим вызов процедур подробнее. При переходе типа "вызов процедуры" CFM вызывающей подпрограммы сохраняется в поле PFM (Previous Frame Marker) регистра PFS, и создается CFM вызываемой подпрограммы. Сразу после вызова размер локальной области вызываемой подпрограммы равен размеру выходной

области вызывающей и перекрывается с ней. При этом стекируемые регистры автоматически переименовываются таким образом, что первый регистр выходной области вызывающей подпрограммы становится регистром GR32 вызываемой. Перекрытие их выходных областей позволяет эффективно передавать параметры через регистры.

Вызываемая подпрограмма может изменить размеры своих локальной и выходной областей командой alloc; соответствующим образом будут изменены и поля в CFM. Команда alloc обычно используется вызываемой подпрограммой для того, чтобы распределить себе определенное число локальных регистров и заиметь выходную область для передачи параметров уже собственному "потомку".

При переходе типа "возврат из процедуры" CFM восстанавливается из PFM, а обратное переименование регистров восстанавливает состояние вызывающей подпрограммы. Если некоторые ее регистры были ранее "сброшены" RSE, то при возврате RSE приостановит процессор до тех пор, пока не будут восстановлены эти регистры.

Механизм условного выполнения команд предикатные команды

Еще одно нововведение Itanium состоит в использовании предикативных вычислений, которые намного сокращают количество условных переходов, так снижающих производительность. Каждая инструкция может содержать предикат, который показывает, выполнять эту команду или игнорировать. Предыдущие по ходу программы условные операторы могут устанавливать значения предикатов. За счет такой организации кода он становится "сквозным" и не содержит столь вредных условных переходов.

Примеры команд с предикатным полем:

```
(p1) add r1=r2,r3
(p2) ld8 r5=[r7]
(p3) chk.s r4,recovery
```

Примеры программ с предикатным полем в командах:

```
Пример1:                Пример2:
cmp.eq p1,p2=r1,r2 ;;   if (r4) { add r1=r2,r3
(p1)add r1=r2,r3        ld8 r6=[r5]}
                        cmp.ne p1,p0=r4,0 ;;// Set predicate reg
                        (p1)add r1=r2,r3
                        (p1)ld8 r6=[r5]
```

Связки команд могут содержать, в том числе и инструкции, относящиеся к разным ветвям. В этом случае оба варианта дальнейшего хода программы могут быть просчитаны параллельно. После таких спекулятивных вычислений нужный результат к моменту определения условия ветвления оказывается готовым.

Команды работы с предикатными регистрами устанавливают предикаты или по совпадению регистров или по совпадению одного бита. Команда cmp сравнивает два регистра GR (или регистр GR и литерал) на одно из 10 возможных условий (больше, меньше или равно и т.п.). Команда tbit тестирует заданный бит GR. Команда fcmp сравнивает два числа с плавающей запятой. Однако результатом сравнения является не единственный код условия, что типично для обычных процессоров. Логический результат сравнения (1 - истина, 0 - ложь) записывается обычно в пару предикатных регистров (во второй пишется отрицание первого).

```
cmp.eq  p1,p2 = r5,r6
tbit  p3,p4 = r6,5
```

Почти все команды выполнимы "под предикатом", т.е. могут выполняться или нет в зависимости от значения указанного в команде PR-регистра. Это позволяет во многих случаях избежать применения условных переходов, которые, как известно, отрицательно сказываются на производительности микропроцессоров. Вместо этого процессор с архитектурой Itanium, имеющий большое число ресурсов, может исполнить обе ветви программы – “жадно”.

Пример программы, показывающие уменьшение времени выполнения за счет использования предикатов с 5 циклов до 2.

Non-predicated	Predicated

if (r1)
r2 = r3 + r4;
else
r7 = r6 - r5;
2 cycles + (30% * 10 cycles) = 5 cycles
	cmp.ne p1,p2 = r1,0 ;;
	(p1) add r2 = r3,r4
	(p2) sub r7 = r6,r5
	2 cycles

Спекулятивное выполнение.

Рассмотрим теперь команды доступа в память. Прежде всего, это команды загрузки регистров и записи из них в оперативную память. Команда ld загружает в GR 1-, 2-, 4- и 8-байтные целочисленные величины; аналогично ldf загружает в FR числа с плавающей запятой размером 4, 8, 10 байт, а также пары 4-байтных чисел. Принципиальной является возможность кодирования указанных команд загрузки в специальных спекулятивных формах. Различают загрузку спекулятивную по управлению и спекулятивную по данным.

Спекулятивное по управлению выполнение означает возможность заранее выполнить команды, расположенные за командой условного перехода, до того, как будет известно, будет ли осуществляться этот условный переход на соответствующую ветвь программы. При наличии большого числа ресурсов процессора это позволяет заранее запускать на выполнение команды, которые начнут выполняться одновременно с уже начавшими выполняться другими командами. Однако позднее может выясниться, что эти спекулятивно выполненные команды оказались выполненными напрасно, так как переход на эту ветвь не произошел, и нужно произвести "откат".

Поскольку эти спекулятивно выполненные команды могут привести к прерыванию, в Itanium предусмотрен механизм, позволяющий зафиксировать, что возникло прерывание, но само прерывание "отложить" до тех пор, пока не будет затребован опрос его наличия. Признак отложенного прерывания записывается в регистр результата (затем его можно опросить специальной командой chk.s). В дальнейшем признак отложенного прерывания последовательно "распространяется" на регистры результатов спекулятивных команд, в регистрах исходных данных которых взведен признак отложенного прерывания.

Все команды можно разделить на спекулятивно выполнимые и спекулятивно невыполнимые. Последние могут вызывать прерывания, которые не могут быть отложены. Обычные вычислительные команды, имеющие GR или FR в качестве регистров результата, - спекулятивные. Если же команда изменяет другие типы регистров, она неспекулятивная.

Кроме обычных неспекулятивных команд (ld, ldf...) в Itanium имеются их спекулятивные модификации (ld.s, ldf.s...). В точке программы, где надо использовать

результат спекулятивного выполнения, следует применять спекулятивную команду chk.s, проверяющую признак отложенного прерывания. Если оно имелось, chk.s передаст управление по указанному в ней адресу, по которому программист должен расположить коды обработки ситуации. Поскольку стало ясно, что спекулятивное выполнение команды действительно понадобилось, можно закодировать копию спекулятивно выполненного фрагмента программы, но уже с неспекулятивными командами загрузки.

Другой тип спекулятивного выполнения может иметь место, когда вслед за записью в память идет команда загрузки регистра, и невозможно заранее определить, не будут ли перекрываться в памяти используемые этими командами данные. В Itanium имеются спекулятивные команды загрузки (ld.a, ldf.a...), которые называются "усовершенствованными" (advanced) командами загрузки. Аналогично взаимозависимости между командами по управлению, "расшиваемой" применением спекулятивных команд с "постфиксом" .s, продвинутые команды загрузки вместе с соответствующей командой проверки chk.a (аналог chk.s) позволяют исключить задержки выполнения при наличии взаимозависимости по данным.

Процессор имеет и другие “улучшенные команды”, обеспечивающие повышение быстродействия.

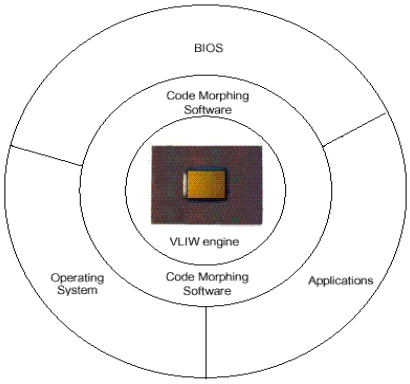
Таким образом, аппаратная поддержка механизма условного выполнения команд в Itanium заключается в следующем:

1. Предикатное поле в команде.
2. Предикатный регистровый файл.
3. Механизм исключения результатов команд со значением предикатного операнда "ложь".
4. Команды работы с предикатными регистрами.
5. Выделение множества спекулятивных команд.

2.6. Микропроцессоры для мобильных платформ с архитектурой Transmeta Crusoe

Low-power x86-Compatible Processors Implemented with Code Morphing™ Software - Crusoe™ processors (фирма Transmeta)

Crusoe (2000 год) — это первый в мире массовый микропроцессор с архитектурой VLIW. Он предназначен для мобильных платформ и имеет пониженное потребление.



Crusoe работает только в режиме эмуляции x86. При этом доступа программистов к внутренним VLIW-командам не предполагается: все программы (как и сама операционная система) работают поверх низкоуровневого программного обеспечения, которое Transmeta называет «морфинг кода» (code morphing) и которое ответственно за трансляцию x86-кодов в команды VLIW.

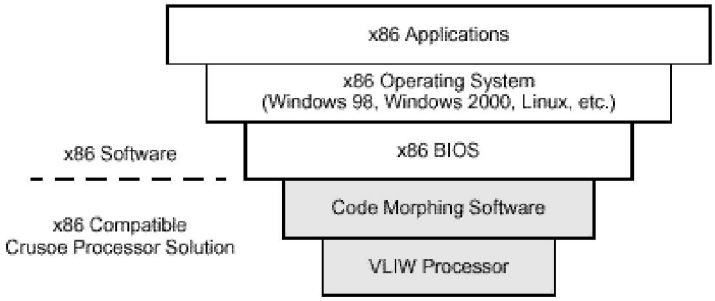


Рис. 8.19. Многоуровневая организация процессора

Процессор имеет аппаратное ядро (hardware engine), которое окружено программным обеспечением (см. рис.8.19). ПО, называемое Code Morphing, помогает аппаратной части микросхемы транслировать инструкции "на лету", запоминать их, чтобы при вторичном использовании не прибегать к относительно медленному процессу трансляции. Это ПО также управляет процессом "сшивания" команд в VLIW. Можно представить себе принцип работы Crusoe по схеме, где ядро Code Morphing как бы "обволакивает" физическое процессорное ядро, причем ПО и BIOS даже не видят процессора как такового, и все взаимодействие ведется посредством программной прослойки.

В отличее от других описанных ранее процессоров часть функций/узлов в Crusoe вынесена наружу, иными словами, выполняется программно, и к их числу относятся блок трансляций команд x86 во внутренний код процессора, предсказание переходов, работа с регистрами и инструкциями. Эквивалентная схема процессора с точки зрения прохождения команды приведена на рис. 8.20. Такой подход позволил Transmeta максимально упростить архитектуру кристалла, уменьшить число транзисторов в ядре, улучшить характеристики тепловыделения, энергопотребление, улучшить экономичность.

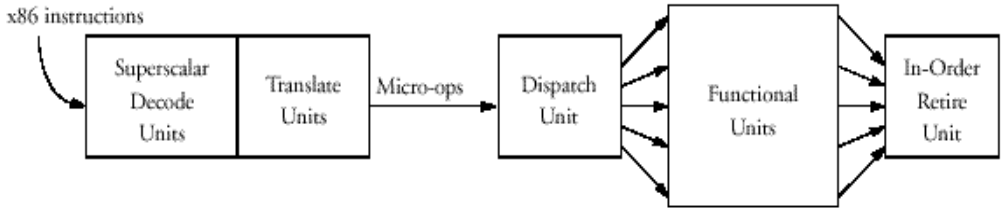


Рис. 8.20. Структурная схема процессора Crusoe

Ядро Crusoe состоит из пяти модулей четырех различных типов: два блока для операций с целыми числами, один для операций с числами с плавающей запятой, один - для операций с памятью и еще один - модуль переходов (рис.). Соответственно и каждая VLIW-инструкция ("молекула" в терминологии Transmeta) длиной 64 или 128 бит может состоять из четырех RISC-подобных операций этих типов ("атомов"). Все атомы выполняются параллельно, каждый соответствующим модулем. Молекулы идут друг за другом, в строгом соответствии с очередью, в отличие от большинства сегодняшних суперскалярных x86-процессоров, где используется механизм внеочередного выполнения команд (out-of-order). Это заметно упрощает внутреннюю структуру процессора, позволяя отказаться от некоторых громоздких функциональных модулей (например, декодера инструкций, коих в наборе x86 не так уж мало).

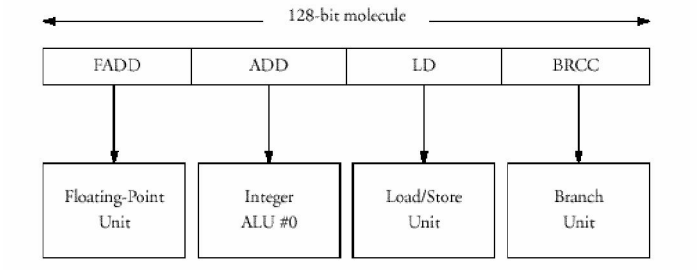


Рис. 8.21. Схема исполнительного ядра процессора Crusoe

На описанный выше уровень "молекулы", по возможности максимально плотно упакованные "атомами", попадают с уровня Code Morphing, где в них превращаются исходные инструкции (на данный момент речь идет только о x86, но в перспективе ничто не мешает сделать версию транслятора и для другого набора команд). Вся окружающая среда, с которой сталкивается процессор, начиная от BIOS и заканчивая ОС и приложениями, контактирует только с Code Morphing, не имея прямого доступа к самому ядру процессора. Такой подход очень удобен, учитывая, что даже у двух первых объявленных процессоров Transmeta это самое ядро - разное.

Consider again the example from the previous section, where the following x86 code:

A. addl %eax,(%esp)
B. addl %ebx,(%esp)
C. movl %esi,(%ebp)
D. subl %ecx,5

was translated into the following two molecules:

1. ld %r30,[%esp]; sub.c %ecx,%ecx,5
2. ld %esi,[%ebp]; add %eax,%eax,%r30; add %ebx,%ebx,%r30

Один из способов увеличения производительности такого нетрадиционного способа работы - очень логичная система кэширования. Каждая x86-инструкция, будучи оттранслирована один раз, сохраняется в специальной кэш-памяти, располагающейся в системной памяти. Таким образом, в следующий раз при выполнении этой инструкции этап трансляции можно пропустить, сразу достав из кэш-памяти необходимую цепочку молекул. Вдобавок, как обещает Transmeta, Code Morphing со временем еще и обучается: по мере выполнения программ оптимизирует их для более быстрого выполнения, обращает внимание на наиболее часто выполняемые участки кода, анализирует переходы в теле программы и т. д.

Линейка описываемых процессоров включает в частности следующие процессоры:

Crusoe™ Processor Model	TM5400	TM5600	TM5900
Process	.18µm	.18µm	0.13µm
On-chip L1 Cache	128KB	128	128
On-chip L2 Cache	256KB	512	512
Die Size	73mm^2	88mm^2	55mm^2

Все эти процессоры имеют общую структуры, отличающуюся для конкретных моделей только объемом КЭШ и комплектом интерфейсных элементов. Опишем более подробно процессор TM5600.

TM5600 снабжен кэшами данных и команд первого уровня емкостью по 64 Кбайт. Кроме большой емкости, следует указать на их высокую архитектурную

эффективность: кэш команд является 8-канальным наборно-ассоциативным, а кэш данных — даже 16-канальным наборно-ассоциативным. В дополнение к этому в процессор интегрирован кэш второго уровня емкостью 512 Кбайт с обратной записью, который является 4-канальным наборно-ассоциативным.

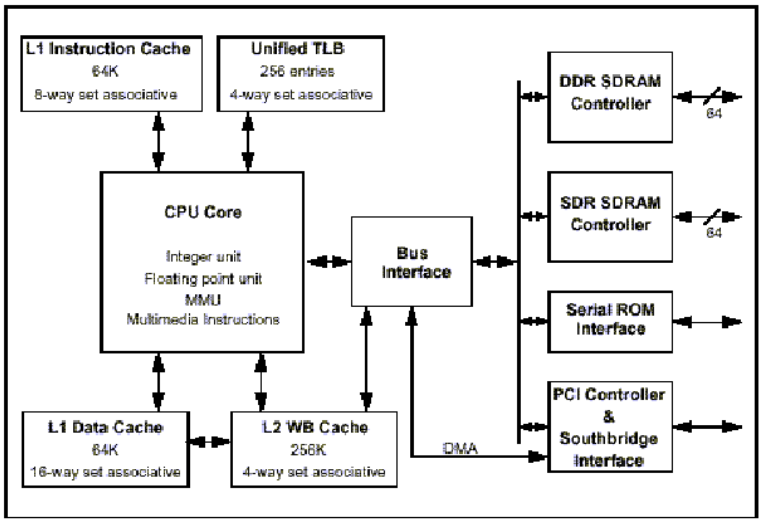


Рис. 8.22. Структурная схема процессора Crusoe.

Crusoe TM *Processor Model TM5600*

Особенность TM5600 — интеграция в микропроцессор контроллеров оперативной памяти. Их сразу два: для наиболее перспективной памяти DDR SDRAM (частоты 100/133 МГц, уровень напряжения 2,5 В) и для SDR SDRAM (частоты 66/100/133 МГц, напряжение 3,3 В). Первый контроллер обеспечивает максимальную пропускную способность и поддерживает до четырех банков, эквивалентных двум модулям DIMM; второй предполагает использование модулей SO-DIMM «шириной» 64 или 72 разряда.

Кроме контроллеров оперативной памяти в TM5600 интегрирован контроллер PCI-шины. Поддерживается стандарт PCI 2.1, однако при этом можно использовать только 32-разрядные PCI-платы с частотой 33 МГц с пониженным (3,3 В) напряжением. (Последнее объясняется стремлением к применению только аппаратуры с относительно низким энергопотреблением.) Общение с внешними устройствами на PCI-шине может происходить в DMA-режиме. Таким образом, в TM5600 интегрирована поддержка всех основных функций, характерных для современных северных мостов наборов микросхем.

TM5600 содержит 64 регистра общего назначения длиной 32 разряда и 32 регистра с плавающей запятой, имеющие длину 80 разрядов. Формат данных с плавающей запятой совпадает с форматом, используемым в x86.

Что касается функциональных исполнительных устройств, то TM5600, как и все другие модели Crusoe, включает два целочисленных АЛУ, устройство загрузки регистров/записи в память, устройство переходов и устройство с плавающей запятой (оно же мультимедийное). Всего за такт могут исполняться до четырех подкоманд VLIW. В Crusoe поддерживаются команды набора MMX, но нет поддержки Intel SSE или AMD 3DNow!.

Среди положительных особенностей архитектуры VLIW-ядра Crusoe следует указать на относительно короткие конвейеры: 7-стадийный целочисленный и 10-стадийный конвейер с плавающей запятой.

Принципиально важным отличием является то, что, по утверждениям представителей Transmeta, Crusoe практически не имеет потерь производительности из-за эмуляции кодов x86. Сначала процессор декодирует коды x86 в режиме интерпретации «байт за байтом». Но если код выполняется несколько раз, механизм морфинга кода транслирует его в оптимальную последовательность молекул, а результат трансляции кэшируется для повторного использования. Для этого в TM5400 и TM5600 предусмотрены специальные 8-килобайтные SRAM-кэши для команд и данных (в TM3120 кэш данных вдвое меньше).

Кроме того, TM5600 имеет интерфейс к флэш-памяти (технология NVRAM, емкость 1 Мбайт), в которой располагается программное обеспечение морфинга кода. В процессе начальной загрузки оно переписывается в оперативную память, занимая 8-16 Мбайт; эта часть памяти становится недоступной для программ.

TM5600/TM5400 (как, впрочем, и TM3120) обладают уникальными характеристиками энергопотребления. Основу низкого энергопотребления закладывает применение VLIW. Как мы уже видели выше, само ядро VLIW устроено относительно просто. Вместо сложной логики, обеспечивающей в современных суперскалярных микропроцессорах внеочередное спекулятивное выполнение команд, в Crusoe работает ПО. Упрощение аппаратуры позволило уменьшить и потребление энергии.

Следует упомянуть отличительную особенность TM5400/TM5600 — режим LongRun. В этом режиме процессор анализирует свою загрузку (за интервалы времени порядка 0,5 мкс) и в зависимости от нее может уменьшать частоту и напряжение ядра (в диапазоне 1,2–1,6 В), а тем самым и энергопотребление.

Отметим еще раз особенности процессора:

- Происходит динамическая трансляция команд X86 во внутренние VLIW команды. При повторном обращении к командам X86 трансляция или не происходит, или оптимизируется выходной код.
- При трансляции используются обратные связи.
- Выполнение с изменением последовательности out-of-order.
- Используется и предсказание (статическое и динамическое), и спекулятивное выполнение команд двух ветвей.
- Внутренние команды хранятся в КЭШ и ОЗУ.
- Регулирование потребления процессором в ходе выполнения программ.

VLIW processor and x86 Code Morphing TM software provide x86-compatible mobile platform solution. 100 mW in deep sleep. Processor core operates at 667MHz - 1GHz. Integrated northbridge core logic features facilitate compact system designs.

Глава 3. Мультипроцессорные системы.

Существует неписаное правило в информатике, которое гласит: "Как только отлажена и работает новая программа, возникает желание уменьшить время ее выполнения или увеличить объем обрабатываемых данных". Это требование на увеличение вычислительной мощности выставляется системными проектировщиками для повышения производительности вычислительных систем.

В основе архитектуры большинства современных ВМ лежит представление алгоритма решения задачи в виде программы последовательных вычислений. Базовые архитектурные идеи ВМ, ориентированной на последовательное исполнение команд программы, были сформулированы Джоном фон Нейманом. В условиях постоянно возрастающих требований к производительности вычислительной техники все очевидней становятся ограничения классической фон-неймановской архитектуры, обусловленные исчерпанием всех основных идей ускорения последовательного счета. Дальнейшее развитие вычислительной техники связано с переходом к параллельным вычислениям как в рамках одной ВМ, так и путем создания многопроцессорных систем и сетей, объединяющих большое количество отдельных процессоров или отдельных вычислительных машин. Для такого подхода вместо термина «вычислительная машина» более подходит термин «вычислительная система» (ВС). Отличительной особенностью вычислительных систем является наличие в них средств, реализующих параллельную обработку, за счет построения параллельных ветвей в вычислениях, что не предусматривалось классической структурой ВМ.

3.1. Уровни параллелизма. Гранулярность

Средства реализации параллелизма зависят от того, на каком уровне он должен обеспечиваться. Обычно различают следующие *уровни параллелизма:*

Уровень заданий. Несколько независимых заданий одновременно выполняются на разных процессорах, практически не взаимодействуя друг с другом. Этот уровень реализуется на ВС с множеством процессоров в многозадачном режиме.

Уровень программ. Части одной задачи выполняются на множестве процессоров. Данный уровень достигается на параллельных ВС.

Уровень команд. Команды выполняются параллельно или конвейерно. Во втором случае выполнение команды разделяется на фазы, а фазы нескольких последовательных команд выполняются параллельно. Уровень достижим на ВС с одним процессором.

Уровень битов (арифметический уровень). Биты слова обрабатываются один за другим, это называется *бит-последовательной операцией.* Если биты слова обрабатываются одновременно, говорят о *бит-параллельной операции.* Данный уровень реализуется в обычных и суперскалярных процессорах.

К понятию уровня параллелизма тесно примыкает понятие *гранулярности.*. Это мера отношения объема вычислений, выполненных в параллельной задаче, к объему коммуникаций (для обмена сообщениями). Степень гранулярности варьируется от мелкозернистой до крупнозернистой.

Крупнозернистый *параллелизм:* каждое параллельное вычисление достаточно независимо от остальных, причем требуется относительно редкий обмен информацией между отдельными вычислениями. Единицами распараллеливания являются большие и

независимые программы, включающие тысячи команд. Этот уровень параллелизма обеспечивается операционной системой.

Среднезернистый параллелизм: единицами распараллеливания являются вызываемые процедуры, включающие в себя сотни команд. Обычно организуется как программистом, так и компилятором.

Мелкозернист*ый параллелизм:* каждое параллельное вычисление достаточно мало и элементарно, составляется из десятков команд. Обычно распараллеливаемыми единицами являются элементы выражения или отдельные итерации цикла, имеющие небольшие зависимости по данным. Сам термин «мелкозернистый параллелизм» говорит о простоте и быстроте любого вычислительного действия. Характерная особенность мелкозернистого параллелизма заключается в приблизительном равенстве интенсивности вычислений и обмена данными. Этот уровень, параллелизма часто используется распараллеливающим (векторизирующим) компилятором.

Эффективное параллельное исполнение требует искусного баланса между степенью гранулярности программ и величиной коммуникационной задержки, возникающей между разными гранулами. В частности, в слабосвязанных программах предпочтительней крупнозернистое разбиение.

3.2. Закон Амдала

Приобретая для решения своей задачи параллельную вычислительную систему, пользователь рассчитывает на значительное повышение скорости вычислений: за счет распределения вычислительной нагрузки по множеству параллельно работающих процессоров. В идеальном случае система из n процессоров могла бы ускорить вычисления в n раз. В реальности достичь такого показателя по ряду причин не удается. Главная из этих причин заключается в невозможности полного распараллеливания ни одной из задач. Как правило, в каждой программе имеется фрагмент кода, который принципиально должен выполняться последовательно и только одним из процессоров. Это может быть часть программы, отвечающая за запуск задачи и распределение распараллеленного кода по процессорам, либо фрагмент программы, обеспечивающий операции ввода/вывода. Можно привести и другие примеры, но главное состоит в том, что о полном распараллеливании задачи говорить не приходится. Известные проблемы возникают и с той частью задачи, которая поддается распараллеливанию. Здесь идеальным был бы вариант, когда параллельные ветви программы постоянно загружали бы все процессоры системы, причем так, чтобы нагрузка на каждый процессор была одинакова. К сожалею эти оба этих условия на практике трудно реализуемы. Таким образом, ориентируясь, на параллельную ВС, необходимо четко сознавать, что добиться прямо пропорционального числу процессоров увеличения производительности не удастся, и, естественно, встает вопрос о том, на какое реальное ускорение можно рассчитывать.

Ответ на этот вопрос в какой-то мере дает закон Амдала.

Джин Амдал (Gene Amdahl) - один из разработчиков всемирно известной системы IBM 360, в своей работе, опубликованной в 1967 году, предложил формулу, отражающую зависимость ускорения вычислений, достигаемого на многопроцессорной ВС, от числа процессоров и соотношения между последовательной и параллельной частями программы. Показателем сокращения времени вычислений служит такая метрика, как *«ускорение».* Напомним, что ускорение S - это отношение времени T затрачиваемого на проведение вычислений на однопроцессорной ВС (в варианте наилучшего последовательного алгоритма), ко времени решения той же задачи на

параллельной системе (при использовании наилучшего параллельного алгоритма)
Пусть программа имеет последовательную (*f*) и потенциально распараллеливаемую части (1-*f*). Время выполнения программы – T. Коэффициент ускорения S определяется в виде: $S = \frac{1}{(1 - f + (f / p))}$

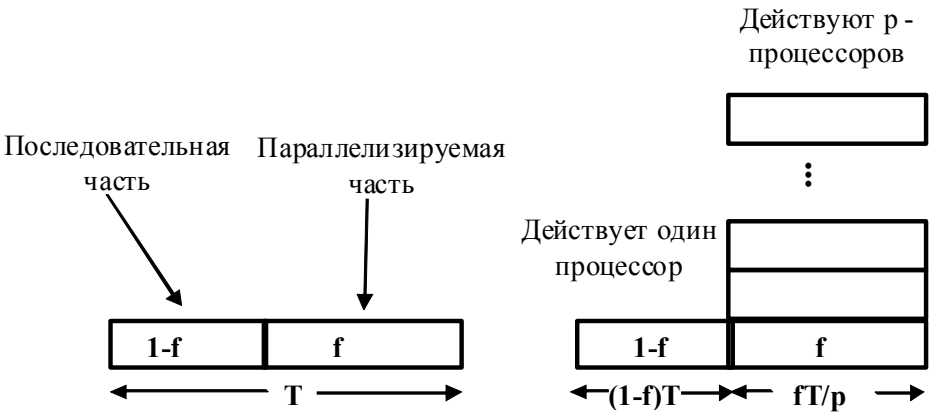


Рис. 3.1. Разбиения программы на части.

Ниже приведенные графики иллюстрируют ускорение в зависимости от количества параллельно работающих процессоров и процента распараллеливания кода.

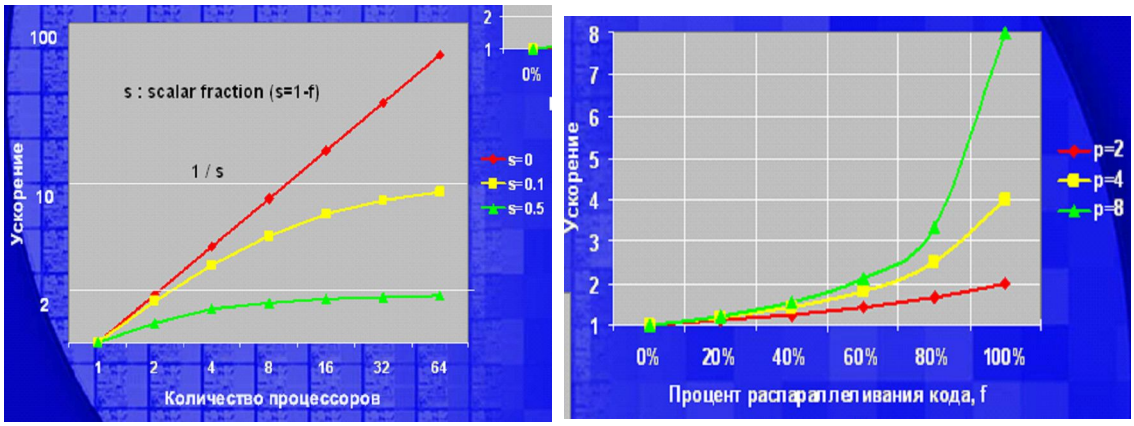


Рис. 3.2. Ускорение за счет параллельного выполнения.

Отметим, что существуют другие факторы, ограничивающие ускорение, к ним можно отнести :

1. Программные издержки. Даже если последовательные и параллельные алгоритмы выполняют одни и те же вычисления, параллельным алгоритмам присущи добавочные программные издержки - дополнительные индексные вычисления, неизбежно возникающие из-за декомпозиции данных и распределения их по процессорам; различные виды учетных операций, требуемые в параллельных алгоритмах, но отсутствующие в алгоритмах последовательных.

2. Издержки из-за дисбаланса загрузки процессоров. Между точками синхронизации каждый из процессоров должен быть загружен одинаковым объемом работы, иначе часть процессоров будет ожидать, пока остальные завершат свои j операции. Эта ситуация известна как *дисбаланс загрузки.* Таким образом, ускорение ограничивается наиболее медленным из процессоров.

3. Коммуникационные издержки. Если принять, что обмен информацией и вычисления могут перекрываться, то любые коммуникации между процессорами снижают ускорение. В плане коммуникационных затрат важен уровень гранулярности, определяющий объем вычислительной работы, выполняемой между коммуникационными фазами алгоритма. Для уменьшения коммуникационных издержек выгоднее, чтобы вычислительные гранулы были достаточно крупными и доля коммуникаций была меньше.

Существуют и другие работы, определяющие ускорение в ВС, в частности известен закон Густафсона, связывающий ускорение и число процессоров.

3.3. Информационные модели

В любой системе параллельной обработки процессоры, выполняющие разные части одной задачи, должны как-то обмениваться информацией. Частота обмена как мы уже сказали, зависит от гранулярности кусков кода. Существуют две основные схемы взаимодействия: с памятью совместного использования и с распределенной памятью/ передачей сообщений.

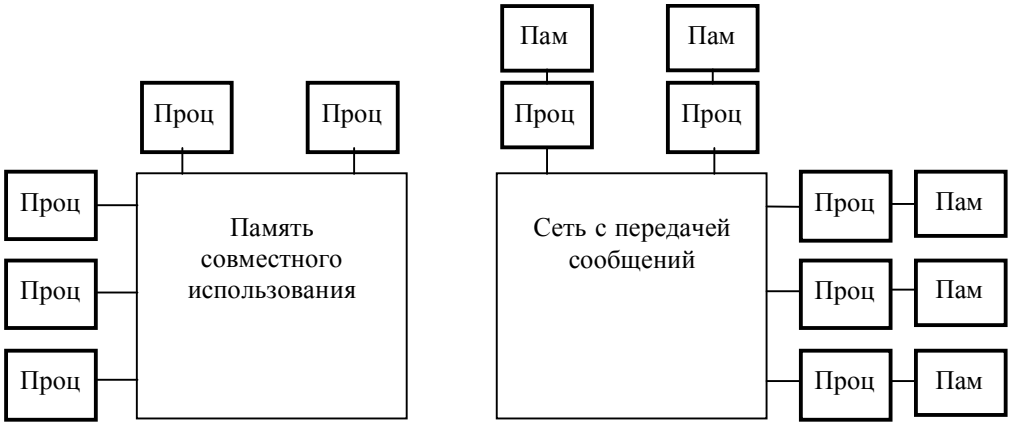


Рис. 9.3. Информационные модели работы с памятью.

Этим схемам соответствуют две альтернативных организации адресации памяти:

- Физически отдельные устройства памяти могут адресоваться как логически единое адресное пространство, что означает, что любой процессор может выполнять обращения к любым ячейкам памяти.
- Адресное пространство состоит из отдельных адресных пространств, которые логически не связаны и доступ, к которым не может быть осуществлен аппаратно другим процессором. Каждый модуль процессор-память представляет собой отдельный компьютер, поэтому такие системы называются многомашинными (multicomputers).

С каждой из этих организаций адресного пространства связан свой механизм обмена. Для машины с единым адресным пространством это адресное пространство может быть использовано для обмена данными посредством операций загрузки и записи. Поэтому эти машины и получили название машин с разделяемой (общей) памятью. Совместно используемая память может быть разделена на уровнях: аппаратном, операционной системы, программном. Для машин с множеством адресных пространств обмен данными должен использовать другой механизм: передачу сообщений между процессорами; поэтому эти машины часто называют машинами с

передачей сообщений.

Главным преимуществом систем с передачей сообщений является их хорошая масштабируемость: каждый процессор имеет доступ только к своей локальной памяти, в связи, с чем не возникает необходимости в потактовой синхронизации процессоров. Практически все рекорды по производительности на сегодняшний день устанавливаются на машинах именно такой архитектуры, состоящих из нескольких тысяч процессоров.

Отметим и недостатки систем с передачей сообщений:

1. Отсутствие общей памяти заметно снижает скорость и задержку межпроцессорного обмена, поскольку нет общей среды для хранения данных, предназначенных для обмена между процессорами. Требуется специальная техника программирования для реализации обмена сообщениями между процессорами.
2. Каждый процессор может использовать только ограниченный объем локального банка памяти.
3. Вследствие указанных архитектурных недостатков требуются значительные усилия для того, чтобы максимально использовать системные ресурсы. Именно этим определяется высокая цена программного обеспечения для систем с раздельной памятью.

3.4. Классы параллельных архитектур. Классификация Флинна.

Понятие архитектуры высокопроизводительной системы является достаточно широким, поскольку под архитектурой можно понимать и способ параллельной обработки данных, используемый в системе, и организацию памяти, и топологию связи между процессорами, и способ исполнения системой арифметических операций. Попытки систематизировать все множество архитектур начались в конце 60-х годов и непрерывно продолжаются по сей день.

Самой ранней и наиболее известной является классификация архитектур вычислительных систем, предложенная в 1966 году М.Флинном (Flynn). Классификация базируется на понятии потока, под которым понимается последовательность элементов, команд или данных, обрабатываемая процессором. На основе числа потоков команд и потоков данных Флинн выделяет четыре класса архитектур (см. расширенную классификацию Флинна рис. 3.4):

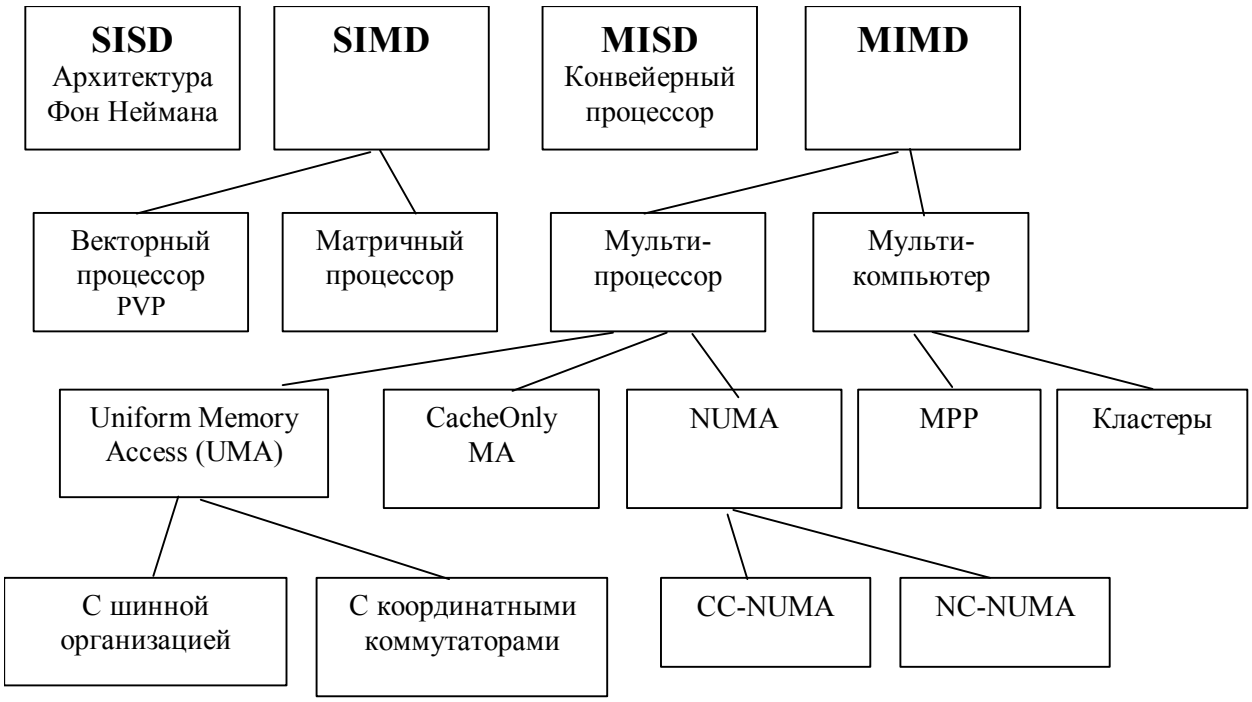
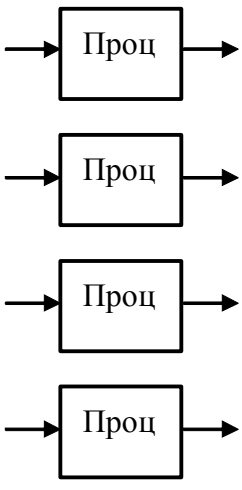


Рис. 3.4. Расширенная классификация Флинна.

SISD = Single Instruction Single Data
MISD = Multiple Instruction Single Data
SIMD = Single Instruction Multiple Data
MIMD = Multiple Instruction Multiple Data

SISD (single instruction stream / single data stream) - одиночный поток команд и одиночный поток данных. Вообще говоря, эта архитектура не имеет отношения к высокопроизводительным системам. К этому классу относятся, прежде всего, классические последовательные машины. В таких машинах есть только один поток команд, все команды обрабатываются последовательно друг за другом и каждая команда инициирует одну операцию с одним потоком данных. Не имеет значения тот факт, что для увеличения скорости обработки команд и скорости выполнения арифметических операций может применяться конвейерная обработка. В случае векторных систем векторный поток данных следует рассматривать как поток из одиночных неделимых векторов.

SIMD (single instruction stream / multiple data stream) - одиночный поток команд и множественный поток данных. Машины типа SIMD состоят из большого числа идентичных процессорных элементов, имеющих собственную память. Все процессорные элементы в такой машине выполняют одну и ту же программу. Очевидно, что такая машина, составленная из большого числа процессоров, может обеспечить очень высокую производительность только на тех задачах, при решении которых все процессоры могут делать одну и ту же работу. Модель вычислений для машины SIMD очень похожа на модель вычислений



SIMD – Single Instruction stream Multiple Data stream

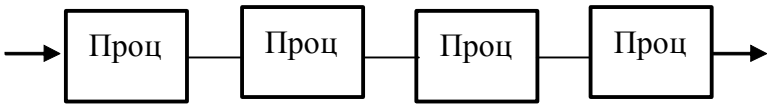
для векторного процессора: одиночная операция выполняется над большим блоком данных.

В отличие от ограниченного конвейерного функционирования векторного процессора, матричный процессор (синоним для большинства SIMD-машин) может быть значительно более гибким. Обрабатывающие элементы таких процессоров - это универсальные программируемые ЭВМ, так что задача, решаемая параллельно, может быть достаточно сложной и содержать ветвления. Модели вычислений на векторных и матричных ЭВМ настолько схожи, что эти ЭВМ часто обсуждаются как эквивалентные.

Организация матричного процессора, на первый взгляд, достаточно проста. Система имеет общее управляющее устройство, генерирующее поток команд и большое число процессорных элементов, работающих параллельно и обрабатывающих каждая свой поток данных. Таким образом, производительность системы оказывается равной сумме производительностей всех процессорных элементов. Однако на практике чтобы обеспечить достаточную эффективность системы при решении широкого круга задач, необходимо организовать связи между процессорными элементами с тем, чтобы наиболее полно загрузить их работой. Именно характер связей между процессорными элементами и определяет разные свойства системы.

SIMD процессоры иногда называют **потоковыми процессорами**. Потоковыми называют процессоры, в основе работы которых лежит принцип обработки многих данных с помощью одной команды. Может быть отдельный потоковый процессор (Single-streaming processor — SSP) и многопотоковый процессор (Multi-Streaming Processor – MSP).

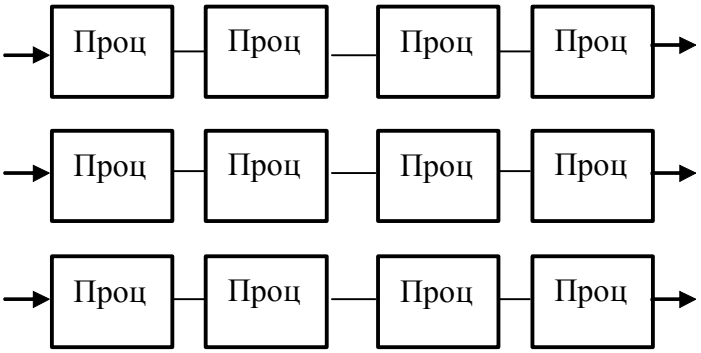
MISD (multiple instruction stream / single data stream) - множественный поток команд и одиночный поток данных. Определение подразумевает наличие в архитектуре многих процессоров, обрабатывающих один и тот же поток данных. Однако ни Флинн, ни другие специалисты в области архитектуры компьютеров до сих пор не смогли представить убедительный пример реально существующей вычислительной системы, построенной на данном принципе. Ряд исследователей относят конвейерные машины к данному классу, однако это не нашло окончательного признания в научном сообществе. В качестве аналог работы такой системы, по-видимому, можно рассматривать работу банка. С любого терминала можно подать команду и что-то сделать с имеющимся банком данных. Поскольку база данных одна, а команд много, то мы имеем дело с множественным потоком команд и одиночным потоком данных.



MISD - Multiple Instruction stream Single Data stream

MIMD (multiple instruction stream / multiple data stream) - множественный поток команд и множественный поток данных. Этот класс предполагает, что в вычислительной системе есть несколько устройств обработки команд, объединенных в единый комплекс и работающих каждое со своим потоком команд и данных. Таким образом, в системе такого рода можно наблюдать реальное распараллеливание. Класс MIMD чрезвычайно широк, поскольку включает в себя всевозможные мультипроцессорные системы: В настоящее время он является чрезвычайно заполненным и возникает потребность в классификации, более избирательно

систематизирующее архитектуры, которые попадают в один класс, но совершенно различны по числу процессоров, природе и топологии связи между ними, по способу организации памяти и, конечно же, по технологии программирования.



MIMD- Multiple Instruction stream Multiple Data stream

В нашем случае (предложено Э.Таненбаумом) системы MIMD разделяются на мультипроцессоры (память совместного использования) и мультикомпьютеры (машины с передачей сообщений). Мультипроцессоры делятся по способу реализации доступа к памяти: архитектуры с однородным доступом к памяти (UMA uniform memory access), с неоднородным доступом (NUMA nonuniform memory access) и архитектура доступа только к КЭШ памяти (COMA cache only memory access). В архитектуре UMA каждый процессор имеет одно и тоже время доступа к любому модулю разделяемой памяти. В NUMA нет такого свойства: обычно есть модуль памяти, который расположен близко к процессору, доступ к которому происходит гораздо быстрее, чем к другим модулям.

Мультикомпьютеры можно разделить на две категории: **MPP архитектура** (massive parallel processing) - массивно-параллельная архитектура и **кластерная** архитектура. MPP – это дорогостоящие суперкомпьютеры, которые состоят из большого числа процессоров, связанных высокоскоростной коммуникационной сетью. Обычно память в таких системах физически разделена. Под кластерной системой понимают набор рабочих станций (или даже персональных компьютеров) общего назначения, соединенных с помощью стандартных сетевых технологий (Fast/Gigabit Ethernet, Myrinet) на базе шинной архитектуры или коммутатора. Такие суперкомпьютерные системы являются самыми дешевыми, поскольку собираются на базе стандартных комплектующих элементов, процессоров, коммутаторов, дисков и внешних устройств. Гетерогенными (неоднородными) кластерами называется объединение в кластер компьютеров разной мощности или разной архитектуры. Архитектура кластерной системы (способ соединения процессоров друг с другом) в большей степени определяет ее производительность, чем тип используемых в ней процессоров.

SMP архитектура

SMP архитектура (symmetric multiprocessing) - симметричная многопроцессорная архитектура. Главной особенностью систем с архитектурой SMP является наличие общей физической памяти, разделяемой всеми процессорами. Все вычислительные устройства при обращении к памяти имеют равные права и одну и ту же адресацию для всех ячеек памяти. Поэтому SMP архитектура называется симметричной. Последнее обстоятельство позволяет очень эффективно обмениваться данными с другими вычислительными устройствами. SMP-система строится на основе высоко скоростной

системной шины, к слотам которой подключаются функциональные блоки трех типов: процессоры (ЦП), память и подсистема ввода/вывода (I/O).

Рис. 9.5. SMP архитектура

Наиболее известными SMP-системами являются машины серий SGI Power Chalenge, DEC Alpha Server, Cray T3D. Вся система работает под управлением единой ОС (обычно UNIX-подобной, но для Intel-платформ поддерживается Windows NT). ОС автоматически (в процессе работы) распределяет процессы по процессорам, но иногда возможна и явная привязка.

FRC (Functional Redundancy Checking) – функционально-избыточная система

В конфигурации **FRC** два процессора выступают как один логический: функционально-избыточная пара **master/checker**. Основной процессор (Master) работает в обычном однопроцессорном режиме. Проверочный процессор выполняет все те же операции «про себя», не управляя шиной, и сравнивает выходные сигналы основного (проверяемого) процессора с теми сигналами, которые он генерирует сам, выполняя те же операции без выхода на шину. В случае обнаружения расхождения вырабатывается сигнал ошибки IERR, который может обрабатываться как прерывание.

PVP архитектура

PVP (Parallel Vector Process) - параллельная архитектура с векторными процессорами.

Основным признаком PVP-систем является наличие специальных векторно-конвейерных процессоров, в которых предусмотрены команды однотипной обработки векторов независимых данных, эффективно выполняющиеся на конвейерных функциональных устройствах. Как правило, несколько таких процессоров (1-16) работают одновременно с общей памятью (аналогично SMP) в рамках многопроцессорных конфигураций.

Парадигма программирования на PVP системах предусматривает векторизацию циклов (для достижения разумной производительности одного процессора) и их распараллеливание (для одновременной загрузки нескольких процессоров одним приложением).

Основой построения архитектуры является суперкомпьютер класса SIMD Cray 1 (1976), который содержал векторные регистры, включающие 64 элемента с ПЗ. Типовой набор команд, которые выполнял векторное АЛУ включал операции:

Операция	Примеры
1. $A_i=f_1(B_i)$	f_1 - косинус, квадратный корень
2. Скаляр$=f_2(B_i)$	f_2 - сумма, минимум

3. $A_i=f_3(B_i, C_i)$ 4. $A_i=f_4$(скаляр, B_i)	f_3 - сложение, вычитание f_4 - умножение на константу

Ассоциативные процессоры.

Ассоциативный способ обработки данных позволяет преодолеть многие ограничения, присущие адресному доступу к памяти, за счет задания некоторого критерия отбора и проведения необходимых преобразований, только над теми данными, которые удовлетворяют этому критерию. Критерием отбора может быть совпадение с любым элементом данных, достаточным для выделения искомых данных из всех имеющихся. Поиск данных может происходить по фрагменту, имеющему большую или меньшую корреляцию с заданным элементом данных.

Ассоциативные системы относятся к классу: один поток команд – множество потоков данных (SIMD = Single Instruction Multiple Data). Эти системы включают большое число операционных устройств, способных одновременно по командам управляющего устройства вести обработку нескольких потоков данных. В ассоциативных вычислительных системах информация на обработку поступает от ассоциативных запоминающих устройств (АЗУ), характеризующихся тем, что информация в них выбирается не по определенному адресу, а по ее содержанию.

3.5. Мультипроцессоры UMA с шинной организацией.

Архитектура UMA с шинной организацией является самой простой архитектурой ВС, в которой можно использовать дешевые и в тоже время высокопроизводительные современные микропроцессоры персональных компьютеров. Начиная с 1980 года идея построения таких систем, подкрепленная широким распространением микропроцессоров, стимулировала многих разработчиков на создание небольших мультипроцессоров, в которых несколько процессоров разделяют одну физическую память, соединенную с ними с помощью разделяемой шины. Если шина занята, процессор ждет, когда она освободится. При наличии большого числа процессоров (более 16, 32) производительность системы будет полностью ограничиваться пропускной способностью шины, а большинство процессоров будут простаивать. Использование кэш памяти в каждом процессоре (см. рис.) частично решает эту проблему, так как существенно уменьшается число обращений к памяти.

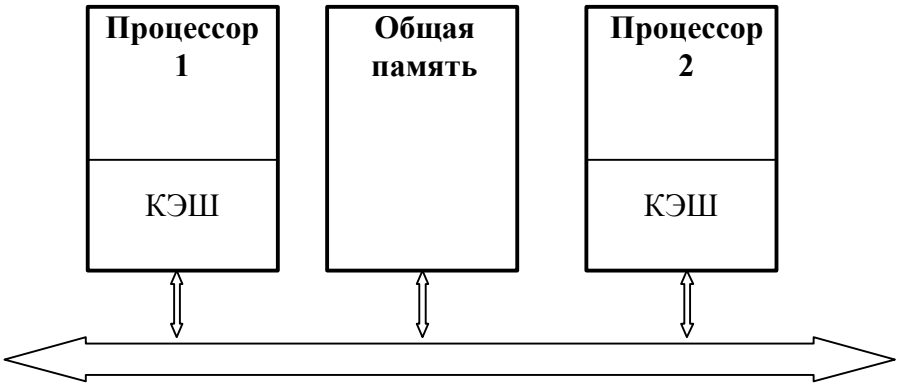


Рис. 3.6. Архитектура UMA с шинной организацией

Кэши могут содержать как разделяемые, так и частные данные. Частные данные - это данные, которые используются одним процессором, в то время как разделяемые данные используются многими процессорами, по существу обеспечивая обмен между ними. Когда кэшируется элемент частных данных, их значение переносится в кэш для

сокращения среднего времени доступа. Поскольку никакой другой процессор не использует эти данные, этот процесс идентичен процессу для однопроцессорной машины с кэш-памятью. Если кэшируются разделяемые данные, то разделяемое значение реплицируется и может содержаться в нескольких кэшах. Кроме сокращения задержки доступа и требуемой полосы пропускания такая репликация данных способствует также общему сокращению количества обменов. Однако кэширование разделяемых данных вызывает новую проблему: когерентность кэш-памяти.

Мультипроцессорная когерентность кэш-памяти

Проблема, о которой идет речь, возникает из-за того, что значение элемента данных в памяти, хранящееся в двух разных процессорах, доступно этим процессорам только через их индивидуальные кэши. Проблема когерентности памяти для мультипроцессоров и устройств ввода/вывода имеет много аспектов. Обычно в малых мультипроцессорах используется аппаратный механизм, называемый протоколом, позволяющий решить эту проблему.

Стратегия обеспечения когерентности кэш-памяти расценивается как смена состояний в конечном автомате. При таком подходе предполагается, что любой блок в локальной кэш-памяти может находиться в одном из фиксированных состояний. Обычно число таких состояний не превышает четырех, например MESI.

Неформально, проблема когерентности памяти состоит в необходимости гарантировать, что любое считывание элемента данных возвращает последнее по времени записанное в него значение. Это определение не совсем корректно, поскольку невозможно требовать, чтобы операция считывания мгновенно видела значение, записанное в этот элемент данных некоторым другим процессором. Если, например, операция записи на одном процессоре предшествует операции чтения той же ячейки на другом процессоре в пределах очень короткого интервала времени, то невозможно гарантировать, что чтение вернет записанное значение данных, поскольку в этот момент времени записываемые данные могут даже не покинуть процессор. Вопрос о том, когда точно записываемое значение должно быть доступно процессору, выполняющему чтение, определяется выбранной моделью согласованного (непротиворечивого) состояния памяти и связан с реализацией синхронизации параллельных вычислений. Поэтому с целью упрощения предположим, что мы требуем только, чтобы записанное операцией записи значение было доступно операции чтения, возникшей немного позже записи и что операции записи данного процессора всегда видны в порядке их выполнения.

Будем считать, что поддерживается строгая согласованность работы с общей памятью (согласованность памяти см. ниже):

1. Операция чтения ячейки памяти одним процессором, которая следует за операцией записи в ту же ячейку памяти другим процессором получит записанное значение, если операции чтения и записи достаточно отделены друг от друга по времени.

2. Операции записи в одну и ту же ячейку памяти выполняются строго последовательно (иногда говорят, что они сериализованы): это означает, что две подряд идущие операции записи в одну и ту же ячейку памяти будут наблюдаться другими процессорами именно в том порядке, в котором они появляются в программе процессора, выполняющего эти операции записи.

Протоколы поддержания когерентности данных.

1. **Сквозная запись**. Каждая запись в КЭШ сопровождается записью в общую память.

Это самый простой протокол, однако он недопустимо загружает шину на операции по записи данных.

Протоколы наблюдения

В протоколах наблюдения (snooping protocols или просто snooping) ответственность за поддержание когерентности всех кэшей многопроцессорной системы возлагается на контроллеры кэшей. В системах, где реализованы протоколы наблюдения, контроллер каждой локальной кэш-памяти содержит блок слежения за шиной, который следит за всеми транзакциями на общей шине и, в частности, контролирует все операции записи. Процессоры должны передавать на шину любые запросы на доступ к памяти, потенциально способные изменить состояние когерентности совместно используемых блоков данных. Локальный контроллер кэш-памяти каждого процессора затем определяет, присутствует ли в его кэш-памяти копия модифицируемого блока, и если это так, то какой блок аннулируется или обновляется.

Протоколы наблюдения характерны для мультипроцессорных систем на базе шины, поскольку общая шина достаточно просто обеспечивает как наблюдения, так и широковещательную передачу сообщений. Однако здесь необходимо принимать меры, чтобы повышенная нагрузка на шину, связанная с наблюдением и трансляцией сообщений, не «съела» преимуществ локальных КЭШей.

Широко используются следующие два протокола (их называют альтернативные протоколы):

2. **Протокол записи с аннулированием строк** (write ivalidate protocol)**.** Второй процессор (контроллер КЭШ памяти второго уровня) следит за тем, куда пишет первый и при КЭШ попадании аннулирует свои строки.

Это наиболее часто используемый протокол. Исключительное право доступа гарантирует, что во время выполнения записи не существует никаких других копий элемента данных, в которые можно писать или из которых можно читать: все другие кэшированные копии элемента данных аннулированы. Чтобы увидеть, как такой протокол обеспечивает когерентность, рассмотрим операцию записи, вслед за которой следует операция чтения другим процессором. Поскольку запись требует исключительного права доступа, любая копия, поддерживаемая читающим процессором должна быть аннулирована (в соответствии с названием протокола). Таким образом, когда возникает операция чтения, произойдет промах кэш-памяти, который вынуждает выполнить выборку новой копии данных. Для выполнения операции записи мы можем потребовать, чтобы процессор имел достоверную (valid) копию данных в своей кэш-памяти прежде, чем выполнять в нее запись. Таким образом, если оба процессора попытаются записать в один и тот же элемент данных одновременно, один из них выиграет состязание у второго (мы вскоре увидим, как принять решение, кто из них выиграет) и вызывает аннулирование его копии. Другой процессор для завершения своей операции записи должен сначала получить новую копию данных, которая теперь уже должна содержать обновленное значение.

3. **Протокол записи с обновлением строк** (write update protocol) **(с трансляцией)** (write broadcast protocol)**.** При КЭШ попадании контроллер КЭШ памяти второго процессора обновляет свои строки, параллельно с записью их в общую память. Обычно в этом протоколе для снижения требований к полосе пропускания полезно отслеживать, является ли слово в кэш-памяти разделяемым объектом, или нет, а именно, содержится ли оно в других кэшах. Если нет, то нет никакой необходимости обновлять другой кэш или транслировать в него обновленные данные.

Разница в производительности между протоколами записи с обновлением и с аннулированием определяется тремя характеристиками:

1. Несколько последовательных операций записи в одно и то же слово, не перемежающихся операциями чтения, требуют нескольких операций трансляции при использовании протокола записи с обновлением, но только одной начальной операции аннулирования при использовании протокола записи с аннулированием.

2. При наличии многословных блоков в кэш-памяти каждое слово, записываемое в блок кэша, требует трансляции при использовании протокола записи с обновлением, в то время, как только первая запись в любое слово блока нуждается в генерации операции аннулирования при использовании протокола записи с аннулированием. Протокол записи с аннулированием работает на уровне блоков кэш-памяти, в то время как протокол записи с обновлением должен работать на уровне отдельных слов (или байтов, если выполняется запись байта).

3. Задержка между записью слова в одном процессоре и чтением записанного значения другим процессором обычно меньше при использовании схемы записи с обновлением, поскольку записанные данные немедленно транслируются в процессор, выполняющий чтение (предполагается, что этот процессор имеет копию данных). Для сравнения, при использовании протокола записи с аннулированием в процессоре, выполняющим чтение, сначала произойдет аннулирование его копии, затем будет производиться чтение данных и его приостановка до тех пор, пока обновленная копия блока не станет доступной и не вернется в процессор.

Эти две схемы во многом похожи на схемы работы кэш-памяти со сквозной записью и с записью с обратным копированием. Также как и схема задержанной записи с обратным копированием требует меньшей полосы пропускания памяти, так как она использует преимущества операций над целым блоком, протокол записи с аннулированием обычно требует менее тяжелого трафика, чем протокол записи с обновлением, поскольку несколько записей в один и тот же блок кэш-памяти не требуют трансляции каждой записи. При сквозной записи память обновляется почти мгновенно после записи (возможно с некоторой задержкой в буфере записи). Подобным образом при использовании протокола записи с обновлением другие копии обновляются так быстро, насколько это возможно. Наиболее важное отличие в производительности протоколов записи с аннулированием и с обновлением связано с характеристиками прикладных программ и с выбором размера блока.

Имеются и другие менее распространенные протоколы, отметим два из них.

4. **Протокол однократной записи (Гудмена).** Это первый из упоминающихся в публикациях протоколов обеспечения когерентности кэш-памяти. Он относится к схемам наблюдения, действующим на принципе записи с аннулированием. Протокол. предполагает, что первая запись в любую строку кэш-памяти про изводится по схеме.' сквозной записи, при этом контроллеры других КЭШей объявляют свои копии измененного блока недействительными. С этого момента только процессор, проведший запись, обладает достоверной копией данных. Последующие операции записи в рассматриваемую строку выполняются в соответствии с протоколом обратной записи. Основной недостаток протокола в том, что он требует первоначальной записи в основную память, даже если эта строка не используется другими процессорами.

5. **Протокол Berkeley.** Права владения на КЭШ строку имеют общая память или процессоры. Если КЭШ строкой владеет процессор, то передача идет в обход памяти.

6. **MESI протокол. (рассмотрен ранее)**

7. **Протоколы на основе справочника (directory based).** Информация о состоянии блока физической памяти содержится только в одном месте, называемом справочником (физически справочник может быть распределен по узлам системы). Этот подход будет рассмотрен ниже в разд. 10.3.

Методы разделения шины

В настоящее время используются два типа шин, отличающиеся способом коммутации, получившие свои названия по аналогии со способами коммутации в сетях передачи данных:

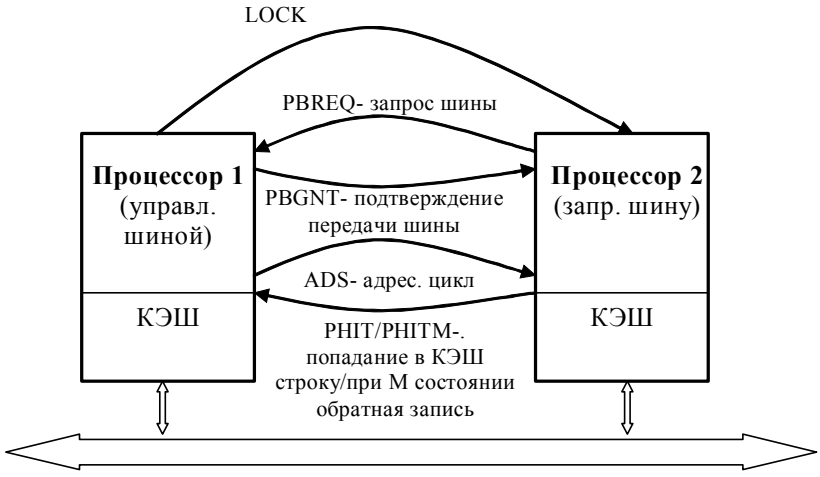
С коммутацией цепей (circuit-switched bus) – арбитраж/блокировка шины, нет расщепления транзакций.

С коммутацией пакетов (packet-switched bus) – расщепление транзакций (split transaction). (Шина соединения/разъединения (connect/disconnect) или конвейерная шина (pipelined bus)

Шина с коммутацией пакетов при наличии нескольких главных устройств шины обеспечивает значительно большую пропускную способность по сравнению с шиной с коммутацией цепей за счет разделения транзакции на две логические части: запроса шины и ответа. Такая методика получила название "расщепления" транзакций (split transaction). (В некоторых системах такая возможность называется шиной соединения/разъединения (connect/disconnect) или конвейерной шиной (pipelined bus). Транзакция чтения разбивается на транзакцию запроса чтения, которая содержит адрес, и транзакцию ответа памяти, которая содержит данные.

Шина с коммутацией цепей не делает расщепления транзакций, любая транзакция на ней есть неделимая операция. Главное устройство запрашивает шину, после арбитража помещает на нее адрес и блокирует шину до окончания обслуживания запроса. Большая часть этого времени обслуживания при этом тратится не на выполнение операций на шине (например, на задержку выборки из памяти). Таким образом, в шинах с коммутацией цепей это время просто теряется. Расщепленные транзакции делают шину доступной для других главных устройств пока память читает слово по запрошенному адресу. Это, правда, также означает, что ЦП должен бороться за шину для посылки данных, а память должна бороться за шину, чтобы вернуть данные. Таким образом, шина с расщеплением транзакций имеет более высокую пропускную способность, но обычно она имеет и большую задержку, чем шина, которая захватывается на все время выполнения транзакции. Транзакция называется расщепленной, поскольку произвольное количество других пакетов или транзакций могут использовать шину между запросом и ответом.

Аппаратная поддержка UMA SMP в процессорах Intel p5, p6



Pentium-Pro: **BREQ[3-0]** – линии шины запроса (занятости)
DID[7-0] – идентификатор отложенной транзакции

Рис. 3.7. Сигналы процессора Pentium, поддерживающие UMA SMP.

Процессоры Pentium находят наибольшее использование в мультипроцессорных системах UMA SMP. Рассмотрим элементы аппаратной поддержки.

1. Приватные сигналы:

PBREQ# запроса шины,

PBGNT# подтверждения передачи управления локальной шиной.

Эти сигналы обеспечивают коммутацию шины между процессорами. Сигналом PBREQ# второй процессор запрашивает шину у первого. Если первому шина не нужна, он отдает ее второму, сообщая об этом сигналом PBGNT#. Сигналы BREQ[3-0], DID[7-0] обеспечивают режим коммутации пакетов. Второй процессор при записи в общую память не запрашивает шину у первого процессора, а, анализируя сигналы BREQ[3-0], смотрит свободна шина или нет. Если шина свободна, то он выставляет запрос (транзакцию) к памяти или внешнему устройству, и отключается от шины. Затем второй процессор анализирует сигналы отложенной транзакции DID[7-0], и если запрошенные данные готовы, то читает их с шины. Производительность второго подхода намного выше.

2. Блокировки шины: LOCK# Ряд операций процессора не могут быть прерваны, в этом случае выставляется сигнал LOCK#. Блокировка шины происходит или по команде или автоматически:

- переключение задач при записи/ чтении TSS
- подтверждение прерывания
- перезапись дескрипторов
- перезапись элементов таблиц страниц

и в некоторых других случаях.

3. Локальные циклы слежения за когерентностью данных в КЭШ-ах.

PHIT#, PHITM# - попадание в КЭШ строку, при PHITM# передача управления для выполнения обратной записи из КЭШа.

В процессоре поддерживается протокол записи с аннулированием строк следующим образом. Если при анализе адреса по сигналу ADC# контроллер КЭШ второго процессора определит попадание в КЭШ строку, то он ответит или сигналом PHIT# при условии аннулирования не измененной КЭШ строки или сигналом PHITM# при условии попадания в КЭШ строку в M – состоянии. В этом случае в диаграмму работы первого процессора вклиниваются операции по обратной записи КЭШ строки второго процессора с последующей ее аннулированием.

4. Прерывания. В структуру процессоров введен расширенный программируемый контроллер прерывания APIC (Advanced Programmable Interruption Controller). Этот контроллер имеет внешние сигналы локальных прерываний LINT[1:0] и трехпроводную интерфейсную шину (PICD[1:0] и PICCLK), по которым оба процессора связываются с контроллером APIC системной платы. Запросы локальных прерываний обслуживаются только тем процессором, на выводы которого (LINTO, LINT1) поступают их сигналы.

5. Прямой доступ в память Сигналы системного арбитража (HOLD, HLDA, BOFF#) в двухпроцессорной системе действуют обычным образом, но воспринимаются и управляются поочередно текущим владельцем локальной шины.

Мультипроцессоры UMA с координатными коммутаторами

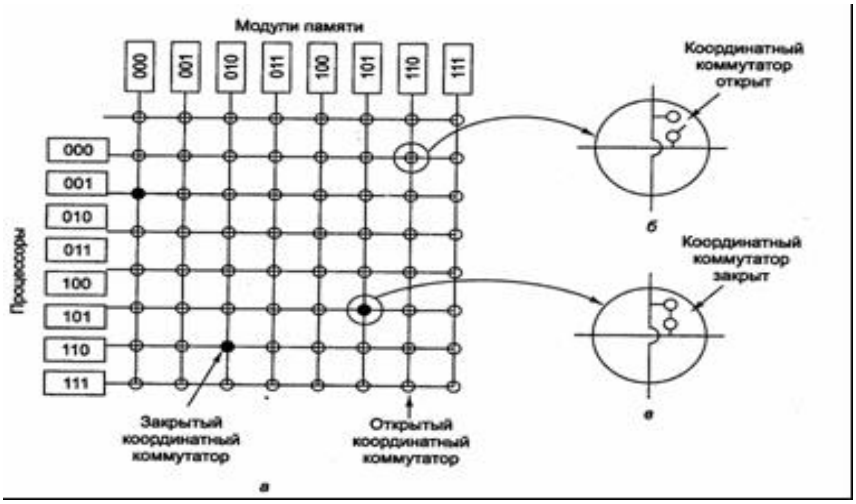


Рис. 3.8. Схема мультипроцессора 1 UMA с координатными коммутаторами.

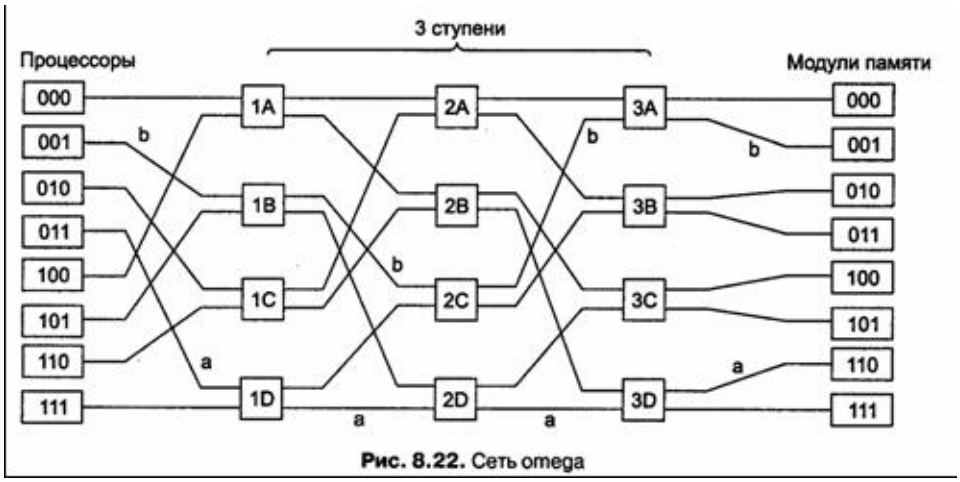


Рис. 3.9. Схема мультипроцессора 2 UMA с координатными коммутаторами.

3.6. Мультипроцессоры NUMA (nonuniform memory access)

Размер мультипроцессоров UMA с одной шиной обычно ограничен несколькими десятками процессоров, а мультипроцессоры UMA с координатными коммутаторами требуют дорогое аппаратное обеспечение. Этапом развития стало создание мультипроцессоров типа NUMA (nonuniform memory access) с неоднородным доступом к памяти. Как и UMA они обеспечивают единое адресное пространство.

Ключевыми характеристиками ВС типа NUMA являются:

1. Существует единое адресное пространство, видимое для всех процессоров.
2. Доступ к удаленной памяти происходит медленнее, чем доступ к ближней (локальной) памяти.
3. Доступ к удаленной памяти обычно производится с использованием специальных команд LOAD, STORE.

Различают две архитектуры NUMA: без КЭШ NC-NUMA (No Caching) и с согласованной КЭШ памятью CC-NUMA (Coherent Cache).

Рассмотрим схему машины (системы) NC-NUMA.

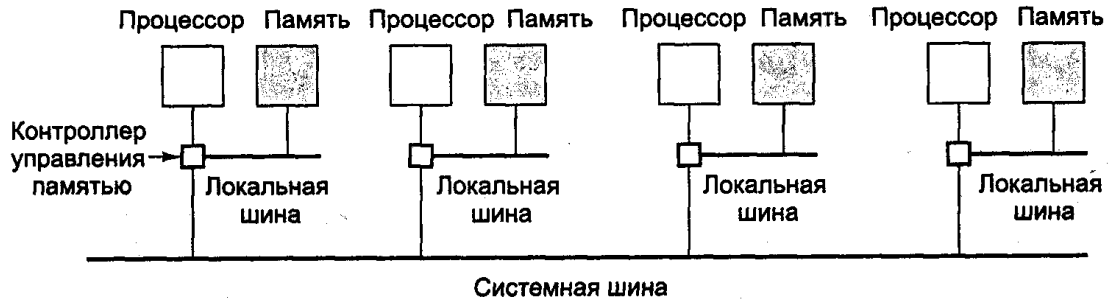


Рис. 3.10. Схема мультипроцессора NC-NUMA.

В системе используется, скажем, 4 процессора. Кроме процессоров в каждом узле находится память и контроллер управления памятью. Работу системы можно описать тремя положениями.

1. Запрос анализирует контроллер памяти. Он определяет находится ли нужное слово в локальной памяти, если нет, то запрос транслируется по системной шине.

2. Когерентность данных обусловлена отсутствием Кэш. Каждое слово находится только в одном месте. Нет опасности появления копий с устаревшими данными – здесь вообще нет копий.

3. Страничный сканер перемещает страницы для улучшения производительности. Он включается регулярно для оптимизации размещения страниц по узлам.

Мультипроцессоры CC-NUMA на основе каталога

Cache Coherent Non-Uniform Memory Access

Это самый популярный подход для построения больших мультипроцессоров. Он предполагает наличие КЭШей в каждом узле и каталога (справочника) для отслеживания местоположений КЭШ строк и их состояний. При обращении к КЭШ строке из каталога выявляется информация, где находится эта строка и изменялась она или нет. При чтении считывается информация из КЭШ ближайшего узла. При записи данных включаются протоколы обеспечения когерентности данных, например, с аннулированием. Поскольку обращение к каталогу происходит на каждой команде, то каталог должен находиться в быстрой памяти (часть ОЗУ).

Протоколы на основе справочника предполагают сбор и отслеживание информации о содержимом всех локальных КЭШей. Такие протоколы обычно реализуются помощью централизованного контроллера, физически представляющего собой часть контроллера основной памяти.

В настоящее время известны три способа реализации протоколов обеспечения когерентности КЭШ памяти на основе каталога.

1. Полный справочник. Централизованный справочник поддерживает информацию обо всех КЭШах и имеет большой объем.

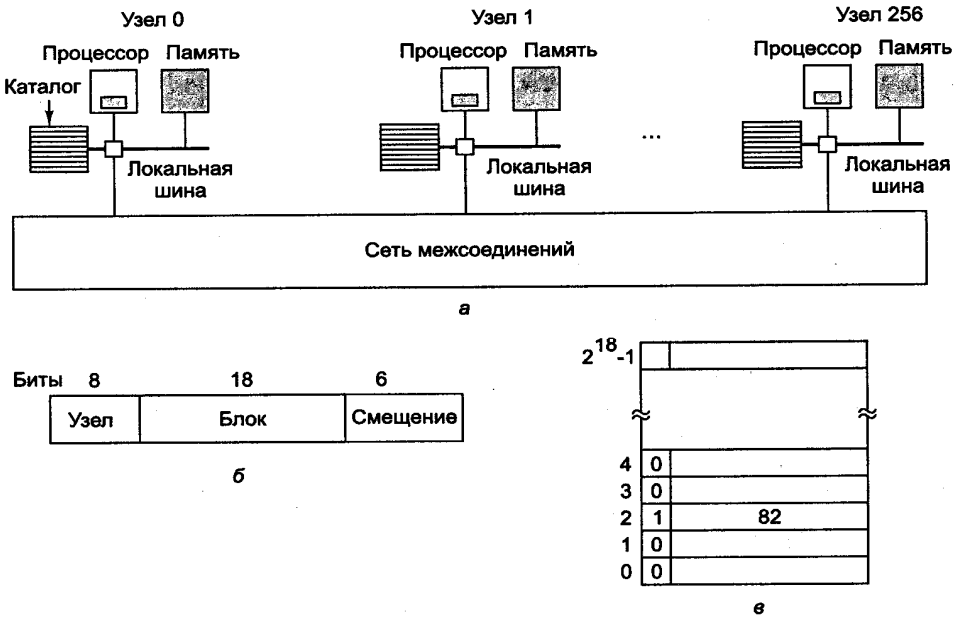
2. Ограниченный справочник. Копии строк могут находиться в ограниченном числе КЭШей, что уменьшает размер справочника.

3. Сцепленные справочники. В тег-ах имеется указатель на узел, в котором располагается еще одна копия данной кэш-строки.

В односвязном списке каждая запись справочника содержит указатель на копию строки в одном из локальных КЭШей. Копии одноименных строки разных кэшах

системы образуют однонаправленную цепочку. Для этого в их тегах предусмотрено специальное поле, куда заносится указатель на кэш-память, содержащую следующую копию цепочки. В тег последней копии цепочки помещается специальный символ-ограничитель. Сцепленный справочник допускает цепочки длиной в N, то есть поддерживает N копий ячейки. При создании еще одной копии цепочку нужно разрушить, а вместо нее сформировать новую.

Чтобы понять, что собой представляет мультипроцессор на основе каталога, рассмотрим систему из 256 узлов (см. рис.9.11), в которой каждый узел состоит из одного процессора и 16 Мбайт ОЗУ. Общий объем памяти составляет 2^{32} байтов. Она разделена 2^{26} строк кэш-памяти по 64 байта каждая. Память разделена по узлам, по 16 М в каждом узле. Узлы связаны через сеть. Сеть может быть в виде решетки, гиперкуба или другой топологии. Каждый узел содержит элементы каталога для 2^{18} 64-байтных строк кэш-памяти, составляя свою 2^{24}- байтную память. Мы предполагаем, что строка модет содержаться только в одной кэш-памяти.



Мультипроцессор на основе каталога, содержащий 256 узлов (*а*); разбиение 32-битного адреса памяти на поля (*б*); каталог в узле 36 (*в*)

Рис. 3.11. Схема мультипроцессора CC-NUMA.

Чтобы понять, как работает каталог, проследим путь команды LOAD из процессора 20, который обращается к кэшированной строке. Сначала процессор, выдавший команду, передает ее в блок управления памятью, который переводит ее в физический адрес, например 0x24000108. Блок управления памятью разделяет этот адрес на три части, как показано на рис.б. В десятичной системе счисления эти три части - узел 36, строка 4 и смещение 8. Блок управления памятью видит, что слово памяти, к которому производится обращение, находится в узле 36. Он посылает запрос через сеть в узел 36, где находится нужная строка 4, узнает есть ли эта строка в кэш-памяти, и если да, то где именно.

Когда запрос приходит в узел 36, он направляется в аппаратное обеспечение каталога. Каталог индексирует таблицу и извлекает элемент 4. Из рисунка видно, что эта строка отсутствует в кэш-памяти, поэтому аппаратное обеспечение вызывает строку 4 из локального ОЗУ, отправляет ее в узел 20 и обновляет элемент каталога 4, чтобы показать, что эта строка находится в кэш-памяти в узле 20.

Рассмотрим второй запрос из процессора 20 на этот раз о строке 2 из узла 36. Из рис видно, что эта строка находится в кэш-памяти в узле 82. В этот момент обновляется элемент 2 каталога, туда записывается 20. Затем строка из узла 82 передается в узел 20. В КЭШе узла 82 строка объявляется недействительной.

Очевидный недостатком этой разработки является то, что строка может быть, только в одном узле. Чтобы строки можно было кэшировать в нескольких узлах, потребуется расширить функции каталога. Одна из возможностей - предоставить в каталоге новые поля для хранения номеров узлов, в которых находятся копии Кэш строк. Вторая возможность - заменить номер узла битовым отображением, один бит на узел. Третья - использование связанного списка.

Впервые идею гибридной архитектуры NUMA предложил Стив Волох. Идею подхватил Сеймур Крей (Seymour R.Cray) и добавил новый элемент - когерентный кэш. Первый мультипроцессор CC-NUMA на основе каталога DASH был создан в Стенфордском университете как исследовательский проект. На рис.9.12 представлена упрощенная схема разработки. Она состоит из 16 узлов (кластеров), каждый из которых представляет SMP UMA мультипроцессор с шинной организацией. Каждый процессор отслеживает свою локальную шину. Глобальная согласованность построена с использованием каталога, который следит какие узлы в настоящий момент имеют копии КЭШ строк.

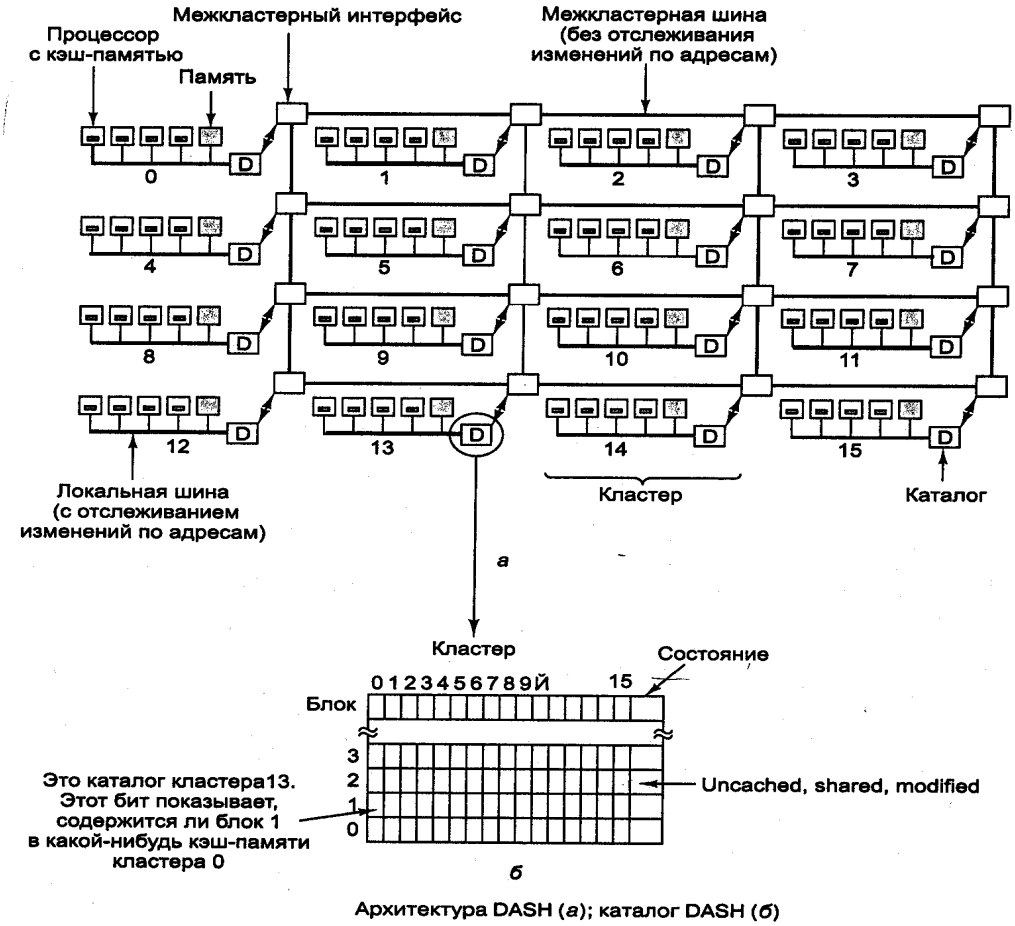


Рис. 3.12. Схема мультипроцессора CC-NUMA из 16 кластеров.

В списке Top500 за 2000г доля суперкомпьютеров с архитектурой NUMA и cc-NUMA составили 22% от общего количества высокопроизводительных систем. Наиболее известными системами архитектуры cc-NUMA являются: HP 9000 V-class в

SCA-конфигурациях, SGI Origin2000, Sun HPC 10000, IBM/Sequent NUMA-Q 2000, SNI RM600. Масштабируемость NUMA-систем ограничивается объемом адресного пространства, возможностями аппаратуры поддержки когерентности кэшей и возможностями операционной системы по управлению большим числом процессоров. На настоящий момент, максимальное число процессоров в cc-NUMA-системах составляет 512 (серия Origin2000).

Модели согласованности обращения к памяти

1. **Строгая согласованность –** возвращается значение последней записи**.**
2. **Согласованность по последовательности –** процессоры наблюдают одну и туже последовательность запросов (единый порядок записей).
3. **Процессорная согласованность –** процессоры наблюдают одинаковый порядок записи любого процессора и в любое слово памяти.
4. **Слабая согласованность –** операции синхронизации задают моменты синхронизации, в которые завершаются незавершенные операции.
5. **Свободная согласованность –** используются операции входа и выхода в критические области: acquire(приобрести) / release(освободить переменную синхронизации с окончанием всех ранее начатых записей).

3.7. MPP архитектура. Сеть процессоров.

MPP архитектура (massive parallel processing) - массивно-параллельная архитектура. Главная особенность такой архитектуры состоит в том, что память физически разделена. В этом случае система строится из отдельных модулей, содержащих процессор, локальный банк операционной памяти (ОП), сетевой адаптер, иногда - жесткие диски и/или другие устройства ввода/вывода. По сути, такие модули представляют собой полнофункциональные компьютеры. Доступ к банку ОП из данного модуля имеет только процессоры (ЦП) из этого же модуля. Модули соединяются специальными коммуникационными каналами. Пользователь может определить логический номер процессора, к которому он подключен, и организовать обмен сообщениями с другими процессорами. Используются два варианта работы операционной системы (ОС) на машинах MPP архитектуры. В одном, полноценная операционная система работает только на управляющей машине (front-end), на каждом отдельном модуле работает сильно урезанный вариант ОС, обеспечивающий работу только расположенной в нем ветви параллельного приложения. Во втором варианте, на каждом модуле работает полноценная UNIX-подобная ОС, устанавливаемая отдельно на каждом модуле.

Системами с раздельной памятью являются суперкомпьютеры IBM RS/6000 SP2, Intel PARAGON/ASCI Red, SGI/CRAY T3E, Hitachi SR8000, транспьютерные системы Parsytec. Машины последней серии CRAY T3E от SGI, основанные на базе процессоров Dec Alpha 21164 с пиковой производительностью 1200 Мфлопс/с (CRAY T3E-1200), способны масштабироваться до 2048 процессоров.

Рассмотрим простую MPP архитектуру - транспьютерную сеть, затем основы построения вычислительных систем и в качестве примера суперкомпьютер Cray.

Транспьютерная сеть

Транспьютер и его язык программирования Оккам созданы специально для использования в мультипроцессорных системах. Может быть соединено в сеть любое количество транспьютеров для построения мультипроцессорных систем. Производительность увеличивается с увеличением количества подключаемых процессоров. Они связаны через последовательные линии связи, для чего не требуется дополнительных схем. Система команд, элементы архитектуры транспьютера ориентированы на использование в сети.

В 1985 г. появился первый элемент этого семейства - 32- разрядный транспьютер T414, затем T800. Основные характеристики транспьютера:

1. ЦПУ с сокращенным набором команд (RISC).
2. 64-разрядный сопроцессор (FPU) плавающей арифметики с высокой пиковой производительностью, работающий параллельно с ЦПУ.
3. Внутрикристальное ОЗУ
4. 32-разрядная шина памяти
5. Четыре последовательных двунаправленных линии связи (Link), обеспечивающих взаимодействие транспьютера с внешним миром, работающих параллельно.
6. Таймер.
7. Системные управляющие сигналы Инициализация, Анализ, Ошибка, управляющие загрузкой и анализом состояния транспьютера, сигнализирующие об ошибках.
8. Интерфейс внешних событий (Event), обеспечивающий асинхронную связь внутреннего процесса и внешнего события.

Транспьютер IMS T800, содержит три регистра (A, B и C) для работы с целыми числами и адресами, которые формируют аппаратный стек. При записи в стек содержимое регистра B записывается в регистр C, содержимое регистра A - в B, и лишь после этого в регистр A заносится новое значение. При чтении из стека извлекается значение регистра A, после чего содержимое регистра B записывается в регистр A, а содержимое регистра C - в регистр B. Соответственно ППТ содержит трехрегистровый стек для хранения промежуточных результатов при работе с числами с плавающей точкой (вычислительный стек). Это регистры AP, BPи CP. Работа с этим стеком аналогична работе с регистрами A, B и C.

Наличие быстрой памяти на СБИС транспьютера позволило при проектировании процессора транспьютера обойтись небольшим количеством регистров; центральный процессор содержит шесть регистров, которые используются при выполнении последовательною процесса. Небольшое число регистров вместе с простым набором инструкций позволило создать простые (и быстрые) каналы данных и управляющую логику в процессоре. Этими регистрами являются:

- указатель на рабочее поле, который указывает на область памяти, где хранятся локальные переменные;
- счетчик команд, который указывает на следующую выполняемую инструкцию;
- регистр операнда, процессора, что позволяет один и тот же микрокод использовать для транспьютеров с различной разрядностью.

Каждая инструкция занимает один байт, который разделен на два четырехбитовых поля . Четыре старших бита этого байта являются кодом функции, а младшие четыре бита содержат операнд.

Непосредственные функции. Это представление позволяет иметь 16 функций, у которых значение операнда находится в пределах O - 15. Тринадцать из них используются для представления наиболее важных функций, выполняемых любым компьютером. Сюда входят:

load constant add constant
load local store local
load non-local store non-local
jump conditional jump
call load local pointer

Наиболее частыми операциями в программах являются загрузка небольших значений литералов, загрузка и сохранение одного или небольшого числа переменных. Инструкция load constant, является однобайтовой и позволяет загузить значение в диапазоне 0-15. Инструкции load local и store local обращаются по адресу памяти относительно указателя на рабочую область. Однобайтовой инструкцией можно обращаться к первым 16 ячейкам.

Префиксные функции_.. Для увеличения длины операндов инструкций используются еще два кода функций. Это следующие:

prefix negative prefix

При выполнении всех инструкций четыре бита данных записываются в четыре младших бита регистра операнда, который затем используется как операнд инструкции (рис.). После завершения всех инструкций, кроме префиксных, регистр операндов обнуляется, что необходимо для выполнения последующих инструкций.

При выполнении инструкции prefix четыре бита данных загружаются в регистр операнда, который затем сдвигается влево на четыре позиции. При выполнении инструкции negative prefix происходит то же самое, только до сдвига содержимое регистра операндов переводится в дополнительный код. При использовании

последовательности префиксных инструкций операнды могут быть расширены до любой длины вплоть до разрядности регистра операнда. С помощью одной инструкции prefix можно сформировать операнд в диапазоне от -256 - +255.

Косвенные функции. Последний код функции - operate - интерпретирует свой операнд как операцию над значениями, находящимися в вычислительном стеке. Это позволяет в однобайтовой инструкции закодировать до 16 таких операций. Однако инструкции prefix можно использовать для расширения размерности операнда инструкции operate так же, как и для любой другой шструкции. Следовательно, такое представление инструкций поизваляет иметь неограниченное число операций.

Кодирование косвенных функций выполнено таким образом, (что наиболее часто встречающиеся операции выполняются без использования инструкций prefix . Сюда входят арифметические, логические операции и операции сравнения.

Эффективность кодирования. Измерения показали, что 70 % выполняемых инструкций занимают один байт (т.е. не используется инструкция prefix). Для выполнения бальшинства из этих инструкций, таких как load constant и add, необходим только один цикл процессора.

Транспьютерная сеть содержит процессоры, коммутаторы линков и канальные адаптеры. На рис. 9.13 показана схема простой сети.

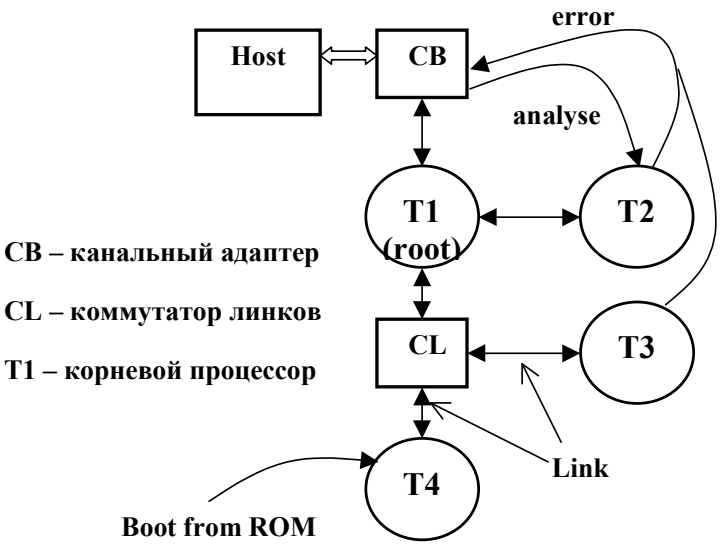


Рис. 3.13. Схема транспьютерной сети.

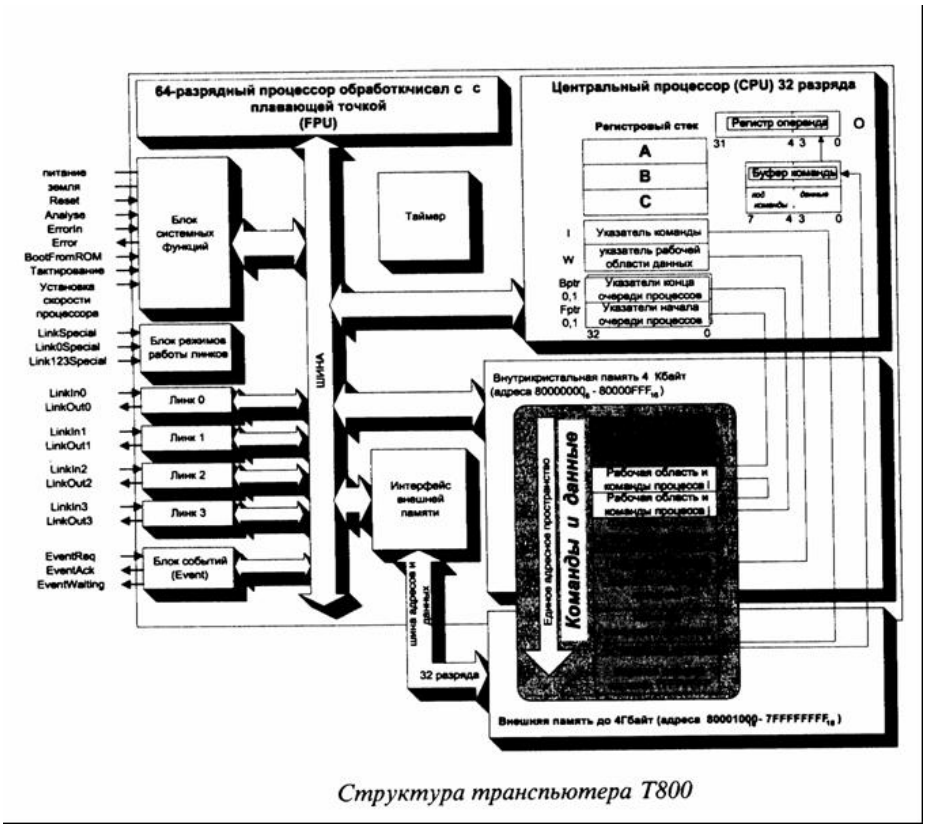
Загрузка прикладной программы в транспьютерную сеть происходит из главной ЭВМ (Host). Она соединена с корневым процессором через канальный адаптер. Транспьютеры в сети соединены друг с другом через линки (последовательные каналы) с использованием коммутаторов. Топология сети должна соответствовать топологии, заданной в описании конфигурации, иначе загрузка не работает нормально. Первичная загрузка процессора из ПЗУ происходит по команде *Boot from ROM.*

Платы фирмы INMOS кроме соединения линий связи обеспечивают функции системного контроля для управления состоянием транспьютерной сети. Управляющие линии состоят из трех сигналов:

reset (сброс) - сигнал передается из главной ЭВМ в сеть и осуществляет сброс всех транспьютеров в сети в режим загрузки.

analyse (анализ) - сигнал передается из главной ЭВМ в сеть и переводит все транспьютеры в сети в состояние контролируемого останова так, чтобы можно было анализировать состояние процессоров.

error (ошибка) - сигнал, приходящий от сети к главной ЭВМ, который показывает, что в одном из транспьютеров сети установлен флаг ошибки.



Структура транспьютера T800

Рис. 3.14. Схема транспьютера.

Основные понятия и характеристики сетей

В основе архитектуры любой многопроцессорной вычислительной системы лежит способность к обмену данными между компонентами этой ВС. Такими компонентами являются: процессоры, модули памяти, интерфейсы, каналы связи, коммутаторы. Организация внутренних коммуникаций вычислительной системы называется *топологией.*

Топологию сети межсоединений (СМС) определяет множество узлов *N,* объединенных множеством каналов С. Связь между узлами обычно реализуется по *двухточечной схеме* (point-to-point). Любые два узла, связанные каналом связи, называют *смежными* узлами или *соседями.* Часто пары узлов соединяют два канала - по одному в каждом направлении. Канал характеризуется шириной - числом сигнальных линий; частотой - скоростью передачи битов по каждой сигнальной линии; задержкой - временем пересылки бита из узла в узел. Для большинства каналов задержка находится в прямой зависимости от физической длины линии. Каналы связи могут быть симплексными (передача в одном направлении), полудуплексными (передача в обоих направлениях, но не одновременно), дуплексными (передача в обоих направлениях одновременно).

В зависимости от того, остается ли конфигурация взаимосвязей неизменной, по крайней мере пока выполняется определенное задание, различают сети со

статической и *динамической* топологиями. В статических сетях структура взаимосвязей фиксирована. В сетях с динамической топологией в процессе вычислений конфигурация взаимосвязей с помощью программных средств может быть оперативно изменена.

Узел в сети может быть терминальным, то есть источником или приемником данных, коммутатором, пересылающим информацию с входного порта на выходной, или совмещать обе роли. В сетях с *непосредственными связями* (direct networks) каждый узел одновременно является как терминальным узлом, так коммутатором, и сообщения пересылаются между терминальными узлами напрямую.

Тремя важнейшими атрибутами СМС являются:

стратегия синхронизации;

стратегия коммутации;

стратегия управления.

Две возможных стратегии синхронизации операций в сети - это *синхронная* и *асинхронная.* В синхронных СМС все действия жестко согласованы во времени, что обеспечивается за счет единого генератора тактовых импульсов, сигналы которого одновременно транслируются во все узлы. В асинхронных сетях единого генератора нет, а функции синхронизации распределены по всей системе. .

В зависимости от выбранной стратегии коммутации различают *сети* с *коммутацией соединений* и *сети* с *коммутацией пакетов.* Как в первом, так и во втором варианте информация пересылается в виде пакета. *Пакет* представляет собой группу битов, для обозначения которой применяют также термин *сообщение.*

В сетях с коммутацией соединений путем соответствующей установки коммутирующих элементов сети формируется тракт от узла-источника до узла-получателя, сохраняющийся, пока весь доставляемый пакет ни достигнет пункта назначения. Пересылка сообщений между определенной парой узлов производится всегда по одному и тому же маршруту.

Сети с коммутацией пакетов предполагают, что сообщение самостоятельно находит свой путь к месту назначения. В отличие от сетей с коммутацией соединений, маршрут от исходного пункта к пункту назначения каждый раз может быть иным. Пакет последовательно проходит через узлы сети. Очередной узел запоминает принятый пакет в своем буфере временного хранения, анализирует его и делает выводы, что с ним делать дальше. В зависимости от загруженности сети принимается решение о возможности немедленной пересылки пакета к следующему узлу и о дальнейшем маршруте следования пакета на пути к цели. Если все возможные тракты для перемещения пакета к очередному узлу заняты, в буфере узла формируется очередь пакетов, которая «рассасывается» по мере освобождения линий связи между узлами.

СМС можно также классифицировать по тому, как в них организовано управление. В некоторых сетях, особенно с переключением соединений, принято ц*ентрализованное* управление. Процессоры посылают запрос на обслуживание в единый контроллер сети, который производит арбитраж запросов с учетом заданных приоритетов и устанавливает нужный маршрут. К данному типу следует отнести сети с шинной топологией.

В схемах с *децентрализованным* управлением функции управления распределены по узлам сети.

Вариант с централизацией проще реализуется, но расширение сети в этом случае связано со значительными трудностями. Децентрализованные сети в плане подключения дополнительных узлов значительно гибче, однако, взаимодействие узлов

в таких сетях существенно сложнее.

Кардинальным вопросом при выборе топологии СМС является способ маршрутизации данных, то есть правило выбора очередного узла, которому пересылается сообщение. Основой маршрутизации служат адреса узлов. Каждому узлу в сети присваивается уникальный адрес. Исходя из этих адресов, а точнее, производится соединение узлов в статических топологиях или их коммутация в топологиях динамических.

Функции маршрутизации задают порядок выбора промежуточных узлов на пути от узла-источника к узлу-получателю. В некоторых топологиях используется единая для всей СМС функция маршрутизации, в других - многоступенчатых - при переходе от одной ступени к другой может применяться иная функция маршрутизации. Функция маршрутизации данных задает правило вычисления возможного адреса одного из смежных узлов по адресу второго узла. Сводится это к описанию алгоритма манипуляции битами адреса-источника для определения адреса-получателя.

Метрики сетевых соединений

Размер сети N (network size) - число узлов в сети.

Число связей I (number of links) - число каналов между узлами. В плане стоимости лучшей следует признать ту сеть, которая требует меньшего числа связей.

Диаметр D (network diameter) - минимальный путь между двумя наиболее удаленными узлами. С возрастанием диаметра сети увеличивается общее время прохождения сообщения, поэтому разработчики ВС стремятся по возможности обходиться меньшим диаметром.

Порядок узла d - число входов и выходов узла.

Пропускная способность (network bandwidth) - количество информации, которое может быть передано по сети в ед. времени.

Задержки (network latency) - время, требуемое для прохождения сообщения по сети.

Бисекция - срез сети, разделяющий ее примерно пополам.

Срез сети (cut of network) - это множество каналов, разрыв которых разделяет множество узлов сети на два непересекающихся набора узлов.

Ширина бисекции B (bisection width) - минимальное число каналов, разрываемых при всех возможных бисекциях сети. Ширина бисекции позволяет оценить число сообщений, которые могут быть переданы по сети одновременно, при условии, что это не вызовет конфликтов из-за попытки использования одних и тех же узлов или линий связи.

Полоса бисекции b (bisectioIl bandwidth) - это наименьшая полоса пропускания по всем возможным бисекциям сети. Она характеризует пропускную способность тех линий связи, кооторые разрываются при бисекции сети, и позволяет оценить наихудшую пропускную способность сети при попытке одномоментной передачи нескольких сообщений, если эти сообщения должны проходить из одной половины сети в другую.

Топологии сетей

1. Стандартная *кольцевая топология* представляет собой линейную цепочку, концы которой соединены между собой (рис. *)*. В зависимости от числа каналов между соседними узлами (один или два) различают *однонаправленные* и *двунаправленные*

кольца. Параметры кольцевой топологии: D= min(N/2); d=2; I=N; B=2. Кольцевая топология, по сравнению с линейной, традиционно была менее популярной, поскольку добавление или удаление узла требует демонтажа сети. Один из способов разрешения проблемы большого диаметра кольцевой сети добавление линий связи в виде хорд, соединяющих определенные узлы кольца. Подобная топология носит название *хордальной.*

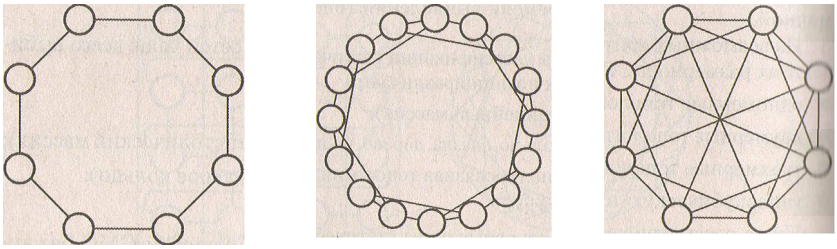


Рис. 3.15. Кольцевые топологии: стандартная; хордальная; с полной связью по хордам

2. Звездообразная

Звездообразная сеть объединяет множество узлов первого порядка посредством специализированного центрального узла - концентратора. Звездообразная организация узлов и соединений редко используется, но хорошо работает, когда информации идет от нескольких вторичных узлов, соединенных с одним центральным узлом, например при подключении терминалов. Параметры топологии: D=2; d=N-1; I=N-1; B=1. Общая пропускная сети обычно ограничивается быстродействием концентратора аналогично тому, как выступает шина в одношинной топологии.

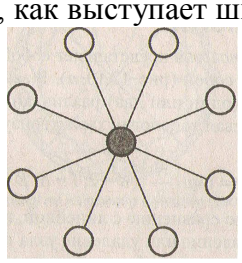
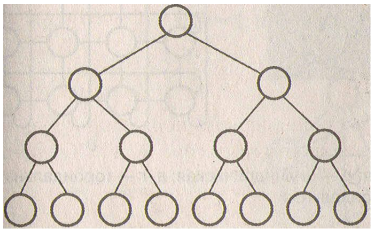


Рис. 3.16. Схема звездообразной сети.

Основное преимущество звездообразной схемы в том, что конструктивное исполнение узлов на концах сети может быть очень простым

3. Древовидные топологии

Еще одним вариантом структуры СМС является древовидная топология (рис.9.17). Сеть строится по схеме так называемого строго двоичного дерева, где каждый узел более высокого уровня связан с двумя узлами следующего по порядку более низкого уровня. Узел, находящийся на более высоком уровне принято называть родительским, а два подключенных к нему нижерасположенных узла дочерними. В свою очередь, каждый дочерний узел выступает в качестве родительского для двух узлов следующего более низкого уровня. Отмстим, что каждый узел связан только с двумя дочерними и одним родительским. Параметры топологии: d=3; I=N-1; B=1.



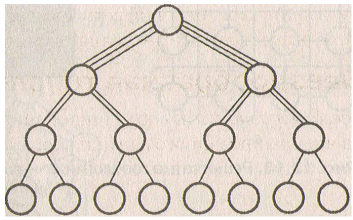


Рис. 3.17. Древовидная топология: а - стандартное дерево; *б* - «толстое" дерево

При больших объемах пересылок между несмежными узлами древовидная топология оказывается недостаточно эффективной, поскольку сообщения должны проходить через один или несколько промежуточных звеньев. Очевидно, что на более высоких уровнях сети вероятность затора из-за недостаточно высокой пропускной способности линий связи выше, Этот недостаток устраняют с помощью топологии, называемой *толстым деревом* "fat-tree" B=2 (рис. *б)*. Возможны и более сложные топологии. Архитектура "fat-tree" (hypertree) предложена Лейзерсоном (Charles E. Leiserson) в 1985 году. Процессоры локализованы в листьях дерева, в то время как внутренние узлы дерева скомпонованы во внутреннюю сеть. Поддеревья могут общаться между собой, не затрагивая более высоких уровней сети

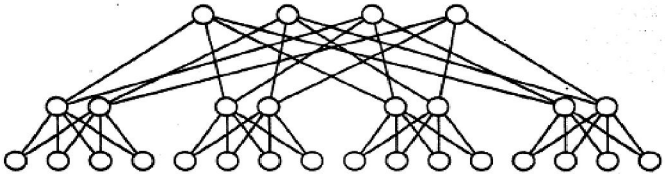
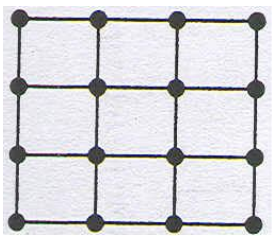


Рис. 3.18. Древовидная топология с поддеревьями

4. Решетчатые топологии.

Поскольку значительная часть научно-технических задач связана с обработкой массивов, вполне естественным представляется стремление учесть это в топологии ВС, ориентированных на подобные задачи. Такие топологии относят к *решетчатым* (mesh), а их конфигурация определяется видом и размерностью массива.



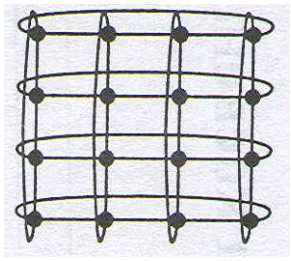


Рис. 3.19. Решетчатые топологии: плоская решетка и тороидальная топология

Для двумерных массивов данных наиболее подходит топология плоской прямоугольной матрицы (mxm) узлов, каждый из которых соединен с ближайшим соседом. (рис.3.19. а). Параметры топологии: D=2(m-1); d=4; I=2N-2m; B=m.Если провести операцию свертывания (wraparoulld) плоско и матрицы, соединив информационными трактами одноименные узлы левого и правого столбцов или одноименные узлы верхней и нижней строк плоской матрицы, то из плоской конструкции получаем топологию типа цилиндра. В топологии цилиндра каждый ряд (или столбец) матрицы представляет собой кольцо. Если одновременно произвести свертывание плоской матрицы в обоих направлениях, получим тороидальную топологию сети (рис. б).

5. Трехмерные топологии.

Теория же показывает, что если в системе максимальное расстояние между процессорами больше 4, то такая система не может работать эффективно. Поэтому, при соединении 16 процессоров друг с другом плоская схема является не эффективной. Для получения более компактной конфигурации необходимо решить задачу о нахождении фигуры, имеющей максимальный объем при минимальной площади поверхности. В трехмерном пространстве таким свойством обладает шар. Но, поскольку, нам необходимо построить узловую систему, то вместо шара приходится использовать куб (если число процессоров равно 8) или гиперкуб, если число процессоров больше 8.

Куб (рис. 3.20.а) - это простейшая трехмерная топология. На рис 3.20.б показан четырехмерный куб, полученный из двух трехмерных кубов, которые связаны между собой. N-мерный куб называется гиперкубом. Эта топология используется во многих компьютерах параллельного действия.

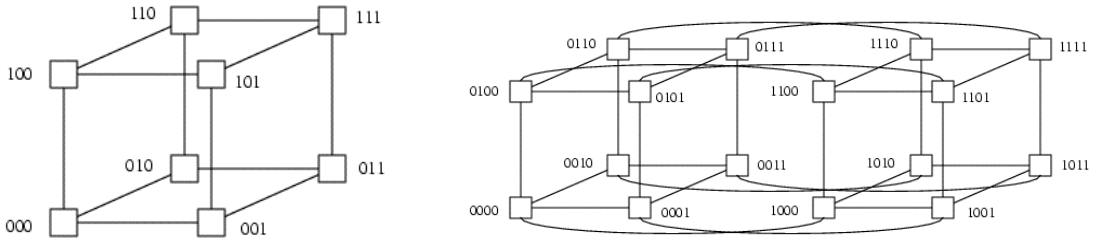


Рис. 3.20. Топологии куб (а) и трехмерный гиперкуб (б)

Топология	Диаметр	Порядок узла	Число связей	Ширина бисекции	Симме-тричност	Размер сети
Полносвязная	1	*N-1*	*N(N-1)/2*	$N^2/4$	Да	N узлов
Звезда	2	1	*N-1*	1	Нет	N узлов
Дерево	2(h-1)	3	*N-1*	1	Нет	h - высота $h=\log_2 N$
Кольцо	*N/2*	2	*N*	2	Да	N узлов
Двумерная решетка	2(m-1)	4	*2N-2m*	*sqr(N)*	Нет	mxm
Гиперкуб	n	n	*nN/2*	*N/2*	Да	N узлов$ $n=\log_2 N$ n - число измерений

Коммутаторы

Коммуникационные среды вычислительных систем (ВС) состоят из адаптеров вычислительных модулей (ВМ) и коммутаторов, обеспечивающих соединения между ними. Используются как простые коммутаторы, так и составные, компонуемые из набора простых. Простые коммутаторы могут соединять лишь малое число ВМ в силу физических ограничений, однако обеспечивают при этом минимальную задержку при установлении соединения. Составные коммутаторы, обычно строящиеся из простых в виде многокаскадных схем с помощью линий «точка-точка», преодолевают ограничение на малое количество соединений, однако увеличивают и задержки.

Простые коммутаторы.

Достоинства: простота управления и высокое быстродействие. Недостатки: малое количество входов и выходов. Типы простых коммутаторов:

- с временным разделением; используются в системах SMP Power Challenge от SGI
- с пространственным разделением, (Gigaplane) используются в семействе Sun Ultra Enterprise.

Простые коммутаторы с временным разделением.

Простые коммутаторы с временным разделением называются также шинами или шинными структурами. Все устройства подключаются к общей информационной магистрали, используемой для передачи информации между ними. Обычно шина является пассивным элементом, управление передачами осуществляется передающими и принимающими устройствами.

Для разрешения конфликтов, возникающих при одновременном запросе устройств на доступ к шине, используются различные приемы, в частности:

- назначение каждому устройству уникального приоритета (статического или динамического);
- использование очереди запросов FIFO;
- выделение фиксированных временных интервалов каждому устройству.

Простые коммутаторы с пространственным разделением

Простые коммутаторы с пространственным разделением позволяют одновременно соединять любой вход с любым одним выходом (ординарные) или несколькими выходами (неординарные). Такие коммутаторы представляют собой совокупность мультиплексоров, количество которых соответствует количеству выходов коммутатора, при этом каждый вход коммутатора должен быть заведен на все мультиплексоры.

Достоинства: возможность одновременного контакта со всеми устройствами; минимальная задержка. Недостатки: высокая сложность порядка n x m, где n – количество входов, m – количество выходов; сложность обеспечения надежности.

Составные коммутаторы

Простые коммутаторы имеют ограничения на число входов и выходов, а также могут требовать большого количества оборудования при увеличении этого числа (в случае пространственных коммутаторов). Поэтому для построения коммутаторов с большим количеством входов и выходов используют совокупность простых коммутаторов, объединенных с помощью линий "точка-точка". Составные коммутаторы имеют задержку, пропорциональную количеству простых коммутаторов, через которые проходит сигнал от входа до выхода, т.е. числу каскадов. Однако объем оборудования составного коммутатора меньше, чем простого с тем же количеством входов и выходов.

Коммуникационные процессоры

Коммуникационные процессоры – это микрочипы, представляющие собой нечто среднее между жесткими специализированными интегральными микросхемами и гибкими процессорами общего назначения. Коммуникационные процессоры программируются, как и привычные для нас ПК-процессоры, но построены с учетом сетевых задач, оптимизированы для сетевой работы и на их основе производители – как процессоров, так и оборудования – пишут программное обеспечение для специфических приложений. Коммуникационный процессор имеет собственную

память и оснащен высокоскоростными внешними каналами для соединения с другими процессорными узлами. Его присутствие позволяет в значительной мере освободить вычислительный процессор от нагрузки, связанной с передачей сообщений между процессорными узлами. Скоростной коммуникационный процессор с RISC-ядром позволяет управлять обменом данными по нескольким независимым каналам, поддерживать практически все распространенные протоколы обмена, гибко и эффективно распределять и обрабатывать последовательные потоки данных с временным разделением каналов.

Сама идея создания процессоров, предназначенных для оптимизации сетевой работы и при этом достаточно универсальных для программной модификации, родилась в связи с необходимостью устранить различия в подходах к созданию локальных сетей (различные подходы к архитектуре сети, классификации потоков и т.д.). Несомненно, истинной причиной бума сетевых процессоров стало ускорение темпов развития рынка. Когда рынок движется на «Internet-скорости», поставщики оборудования уже не могут тратить по два года на разработку специализированных микросхем для реализации конкретных сетевых функций. Эти два года (и вложенные деньги) будут потрачены зря, если рынок за это время уйдет в другом направлении. Выход один – разрабатывать процессоры, которые поставщики оборудования могут внедрить и выпустить в новом продукте в течение нескольких месяцев. Бум сетевых процессоров, окончательно оформившийся в середине 1999 г., не был кратким, и в последующие годы индустрия развивалась крайне бурно.

Суперкомпьютеры.

Современные суперкомпьютеры имеют различные архитектуры, основные элементы этих архитектур описаны ранее. В основном это мультипроцессоры архитектур MPP, SMP, CC-NUMA и векторные.

Основные производители современных высокопроизводительных компьютеров и комплектующих к ним: IBM, HP, Cray, Sun, SGI, Compaq, Fujitsu, NEC и др.

Список пяти самых быстродействующих систем из TOP500 на ноябрь 2013 г. приведен ниже.

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
5	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945

Rank - порядковый номер в списке Top500
Site - организация, в которой установлен компьютер
System/Vendor - название (тип) компьютера, указанное поставщиком/производитель
Processors - количество процессоров
Rmax - максимальная полученная производительность по LINPACK
Rpeak - теоретическая пиковая производительность

3.8. Кластеризация.

Развитие сетевых технологий привело к появлению недорогих, но эффективных коммуникационных решений. Это и предопределило появление кластерных вычислительных систем, фактически являющихся одним из направлений развития компьютеров с массовым параллелизмом. Классические суперкомпьютеры, использующие специализированные процессоры таких производителей как Cray или NEC (векторно-параллельные или массивно-параллельные), недешевы, поэтому и стоимость подобных систем несравнима со стоимостью систем, находящихся в массовом производстве.

С помощью кластеризации достигаются следующие преимущества:

- абсолютная масштабируемость системы – создание больших кластеров, каждый узел которых представляет мультипроцессор;
- наращиваемая масштабируемость – возможность добавления новых узлов;
- высокая надежность и отказоустойчивость системы;
- превосходное отношение "стоимость/производительность";
- совместимость программного обеспечения.

Масштабируемость представляет собой возможность наращивания числа и мощности процессоров, объемов оперативной и внешней памяти и других ресурсов вычислительной системы. Масштабируемость должна обеспечиваться архитектурой и конструкцией компьютера, а также соответствующими средствами программного обеспечения.

Так, например, возможность масштабирования кластера ограничена значением отношения скорости процессора к скорости связи, которое не должно быть слишком большим (реально это отношение для больших систем не может быть более 3-4, в противном случае не удается даже реализовать режим единого образа операционной системы). С другой стороны, последние 10 лет истории развития процессоров и коммуникаторов показывают, что разрыв в скорости между ними все увеличивается. В идеале добавление процессоров к системе должно приводить к линейному росту ее производительности. Однако это не всегда так. Потери производительности могут возникать, например, при недостаточной пропускной способности шин из-за возрастания трафика между процессорами и основной памятью, а также между памятью и устройствами ввода/вывода. В действительности реальное увеличение производительности трудно оценить заранее, поскольку оно в значительной степени зависит от динамики поведения прикладных задач.

Возможность масштабирования системы определяется не только архитектурой аппаратных средств, но зависит от свойств программного обеспечения. Масштабируемость программного обеспечения затрагивает все его уровни, от простых механизмов передачи сообщений до работы с такими сложными объектами как

мониторы транзакций и вся среда прикладной системы. В частности, программное обеспечение должно минимизировать трафик межпроцессорного обмена, который может препятствовать линейному росту производительности системы. Аппаратные средства (процессоры, шины и устройства ввода/вывода) являются только частью масштабируемой архитектуры, на которой программное обеспечение может обеспечить предсказуемый рост производительности. Важно понимать, что, например, простой переход на более мощный процессор может привести к перегрузке других компонентов системы. Это означает, что действительно масштабируемая система должна быть сбалансирована по всем параметрам.

Кластер представляет собой два или более компьютеров (часто называемых узлами), объединяемые при помощи сетевых технологий на базе шинной архитектуры или коммутатора и предстающие перед пользователями в качестве единого информационно-вычислительного ресурса. В качестве узлов кластера могут быть выбраны серверы, рабочие станции и даже обычные персональные компьютеры. Узел характеризуется тем, что на нем работает единственная копия операционной системы. Преимущество кластеризации для повышения работоспособности становится очевидным в случае сбоя какого-либо узла: при этом другой узел кластера может взять на себя нагрузку неисправного узла, и пользователи не заметят прерывания в доступе. Возможности масштабируемости кластеров позволяют многократно увеличивать производительность приложений для большего числа пользователей технологий (Fast/Gigabit Ethernet, Myrinet) на базе шинной архитектуры или коммутатора. Такие суперкомпьютерные системы являются самыми дешевыми, поскольку собираются на базе стандартных комплектующих элементов ("off the shelf"), процессоров, коммутаторов, дисков и внешних устройств.

Кластеризация может осуществляться на разных уровнях компьютерной системы, включая аппаратное обеспечение, операционные системы, программы-утилиты, системы управления и приложения. Чем больше уровней системы объединены кластерной технологией, тем выше надежность, масштабируемость и управляемость кластера.

Типы кластеров

Наиболее распространенными типами кластеров являются:

- системы высокой надежности;
- системы для высокопроизводительных вычислений;
- многопоточные системы.

Отметим, что границы между этими типами кластеров до некоторой степени размыты, и кластер может иметь такие свойства или функции, которые выходят за рамки перечисленных типов.

Кластеры для высокопроизводительных вычислений предназначены для параллельных расчетов. Эти кластеры обычно собраны из большого числа компьютеров. Разработка таких кластеров является сложным процессом, требующим на каждом шаге согласования таких вопросов как инсталляция, эксплуатация и одновременное управление большим числом компьютеров, технические требования параллельного и высокопроизводительного доступа к одному и тому же системному файлу (или файлам) и межпроцессорная связь между узлами, и координация работы в параллельном режиме. Эти проблемы проще всего решаются при обеспечении единого образа операционной системы для всего кластера. Однако реализовать подобную схему

удается далеко не всегда и обычно она применяется лишь для не слишком больших систем.

Многопоточные системы используются для обеспечения единого интерфейса к ряду ресурсов, которые могут со временем произвольно наращиваться (или сокращаться). Типичным примером может служить группа web-серверов.

В 1994 году Томас Стерлинг (Sterling) и Дон Беккер (Becker) создали 16-узловой кластер из процессоров Intel DX4, соединенных сетью 10 Мбит/с Ethernet с дублированием каналов. Они назвали его "Beowulf" по названию старинной эпической поэмы. Проект стал основой общего подхода к построению параллельных кластерных компьютеров, он описывает многопроцессорную архитектуру, которая может с успехом использоваться для параллельных вычислений. Beowulf-кластер, как правило, является системой, состоящей из одного серверного узла (который обычно называется головным), а также одного или нескольких подчиненных (вычислительных) узлов, соединенных посредством стандартной компьютерной сети. Система строится с использованием стандартных аппаратных компонентов, таких как ПК, запускаемые под Linux, стандартные сетевые адаптеры (например, Ethernet) и коммутаторы. Серверный узел контролирует весь кластер и обслуживает файлы, направляемые к клиентским узлам.

Архитектура кластерной системы (способ соединения процессоров друг с другом) в большей степени определяет ее производительность, чем тип используемых в ней процессоров. Критическим параметром, влияющим на величину производительности такой системы, является расстояние между процессорами. Так, соединив вместе 10 персональных компьютеров, мы получим систему для проведения высокопроизводительных вычислений. Проблема, однако, будет состоять в поиске наиболее эффективного способа соединения стандартных средств друг с другом, поскольку при увеличении производительности каждого процессора в 10 раз производительность системы в целом в 10 раз не увеличится.

Надежность

Важнейшей характеристикой вычислительных систем является надежность, т.е. работа системы без сбоев в определенных условиях в течение определенного времени. Повышение надежности основано на принципе предотвращения неисправностей путем снижения интенсивности отказов и сбоев за счет применения электронных схем и компонентов с высокой и сверхвысокой степенью интеграции, снижения уровня помех, облегченных режимов работы схем, обеспечения тепловых режимов их работы, а также за счет совершенствования методов сборки аппаратуры.

Понятие надежности включает не только аппаратные средства, но и программное обеспечение, которое используется, в частности, для анализа производительности систем и управления конфигурациями. Главной целью повышения надежности систем является целостность хранящихся в них данных. Единицей измерения надежности является среднее время наработки на отказ (MTBF - Mean Time Between Failure), иначе - среднее время безотказной работы.

Отказоустойчивость - это способность вычислительной системы продолжать действия, заданные программой, после возникновения неисправностей. Введение отказоустойчивости требует избыточного аппаратного и программного обеспечения. Направления, связанные с предотвращением неисправностей и с отказоустойчивостью, - основные для обеспечения надежности. Концепции параллельности и

отказоустойчивости вычислительных систем естественным образом связаны между собой, поскольку в обоих случаях требуются дополнительные функциональные компоненты. Поэтому на параллельных вычислительных системах достигается как наиболее высокая производительность, так и, во многих случаях, очень высокая надежность. Имеющиеся ресурсы избыточности в параллельных системах могут гибко использоваться как для повышения производительности, так и для повышения надежности. Структура многопроцессорных и многомашинных систем приспособлена к автоматической реконфигурации и обеспечивает возможность продолжения работы системы после возникновения неисправностей.

Некоторые понятия, связанные с надежностью:

- Высокая Готовность (High Availability).
- Эластичность к отказам (Fault Resiliency). Короткое время восстановления, которое позволяет системе быстро откатиться назад после обнаружения неисправности.
- Устойчивость к отказам (Fault Tolerance). Имеется избыточная аппаратура - горячий резерв
- Непрерывная готовность (Continuous Availability)..
- Устойчивость к стихийным бедствиям (Disaster Tolerance). Полное (зеркальное) дублирование системы вне основного местоположения, позволяющее принять на себя работу немедленно после отказа системы на основной площадке.

Для повышения надежности информационно-вычислительной системы идеальной схемой являются кластерные системы. Благодаря единому представлению, отдельные неисправные узлы или компоненты кластера могут быть без остановки работы и незаметно для пользователя заменены, что обеспечивает непрерывность и безотказную работу вычислительной системы даже в таких сложных приложениях как базы данных.

Основа надежности кластера - это некоторое избыточное количество отказоустойчивых серверов (узлов), в зависимости от конфигурации кластера и его задач.

Кластерная конфигурация узлов, коммуникационного оборудования и памяти может обеспечить зеркалирование данных, резервирование компонентов самоконтроля и предупреждения, а также совместное использование ресурсов для минимизации потерь при отказе отдельных компонентов.

Решение, обеспечивающее повышенную отказоустойчивость сервера, должно включать:

- компоненты с "горячей" заменой;
- диски, вентиляторы, внешние накопители, устройства PCI, источники питания;
- избыточные источники питания и вентиляторы;
- автоматический перезапуск и восстановление системы;
- память с коррекцией ошибок;
- функции проверки состояния системы;
- превентивное обнаружение и анализ неисправностей;
- средства удаленного администрирования системы.

Принцип быстрого проявления неисправности обычно реализуется с помощью двух методов - **самоконтроля и сравнения**. Средства самоконтроля предполагают, что

при выполнении некоторой операции модуль делает и некоторую дополнительную работу, позволяющую подтвердить правильность полученного состояния. Примерами этого метода являются коды обнаружения неисправности при хранении данных и передаче сообщений. Метод сравнения основывается на выполнении одной и той же операции двумя или большим числом модулей и сопоставлении результатов компаратором. В случае обнаружения несовпадения результатов работа приостанавливается.

Методы самоконтроля были основой построения отказоустойчивых систем в течение многих лет. Они требуют реализации дополнительных схем и времени разработки и, вероятно, будут доминировать в устройствах памяти и устройствах связи благодаря простоте и ясности логики. Однако для сложных устройств обработки данных экономические соображения, связанные с применением стандартных массовых компонентов, навязывают использование методов сравнения. Поскольку компараторы сравнительно просты, их применение дает некоторое увеличение логических схем при существенном сокращении времени разработки. Следует отметить, что в более ранних отказоустойчивых конструкциях 30% логических схем процессоров и 30% времени разработки уходило на реализацию средств самоконтроля. С этой точки зрения схемы сравнения добавляют лишь универсальные схемы с простой логикой. В результате сокращаются общие расходы на разработку и логику.

Еще одним средством построения отказоустойчивой архитектуры является принцип дублирования дуплексных модулей, который предполагает создание некоторой комбинации двух модулей ("супермодуля"), построенных на принципах быстрого проявления неисправности. Такой "супермодуль" продолжает работать, даже когда отказывает один из субмодулей.

На рис. 3.21. показана схема сервера, которая предусматривает и самотестирование и резервирование процессоров.

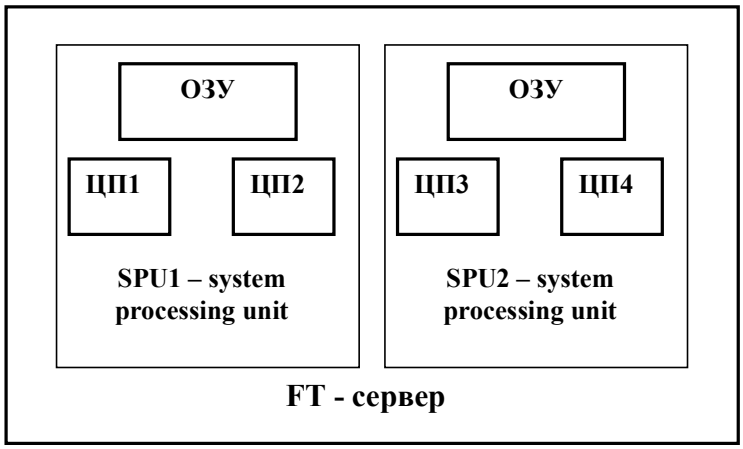


Рис. 3.21. Упрощенная схема сервера с резервированием

3.9. Нейронная сеть (нейропроцессоры)

1943 год Мозг - компьютер на базе нейронов.

Нейрокомпьютер – объединение искусственных **нейронов** в искусственную **нейронную сеть (ИНС)**. Это новая парадигма вычислительной математики, предлагающая принципиально новые высокоэффективные модели массового параллелизма в обработке и запоминании информации, способах анализа и

интерпретации результатов вычислений, построении формальных моделей и баз знаний, систем искусственного интеллекта.

Главная особенность нового подхода состоит в том, что при относительно простой модели самих «решающих элементов» – *нейронов* вся сложная деятельность «биологического компьютера», его функциональные возможности определяются не только и не столько сложностью отдельных нейронов, сколько связями и характером взаимодействия между ними. Совокупность идей и алгоритмических решений представления формальных моделей вычислителей, вытекающих из такого подхода, получило название *коннекционизма* (от английского слова “connection” – связь), а его лозунгом в несколько утрированной форме стала фраза: «свойства элементов – ничто, структура связей – все» .

Применение в задачах, в которых мозг работает гораздо эффективнее компьютера. К таким задачам относятся:

- распознавание образов, сцен, быстроменяющихся ситуаций,
- принятие решений на основе нечеткой и плохо структурированной информации,
- контекстно-зависимый анализ и интерпретация многомерных данных,
- нечеткие вычисления и заключения,

ряд других плохо формализуемых задач.

Идея работы:

Этап 1 Обучение на примерах и конструирование

Этап 2 Функционирование: предъявляется входной набор - сеть относит его к какому-то классу (переходит в одно из устойчивых состояний). Ассоциативная выборка.

Нейрон

Прототипом для создания нейрона послужил биологический нейрон головного мозга. Биологический нейрон имеет тело, совокупность отростков - дендридов, по которым в нейрон поступают входные сигналы, и отросток - аксон, передающий выходной сигнал нейрона другим клеткам. Точка соединения дендрида и аксона называется синапсом. Упрощенно функционирование нейрона можно представить следующим образом:

- нейрон получает от дендридов набор (вектор) входных сигналов;
- в теле нейрона оценивается суммарное значение входных сигналов. Однако входы нейрона неравнозначны. Каждый вход характеризуется некоторым весовым коэффициентом, определяющим важность поступающей по нему информации. Таким образом, нейрон не просто суммирует значения входных сигналов, а вычисляет скалярное произведение вектора входных сигналов и вектора весовых коэффициентов;
- нейрон формирует выходной сигнал, интенсивность которого зависит от значения вычисленного скалярного произведения. Если оно не превышает некоторого заданного порога, то выходной сигнал не формируется вовсе - нейрон "не срабатывает";
- выходной сигнал поступает на аксон и передается дендридам других нейронов.

Поведение искусственной нейронной сети зависит как от значения весовых параметров, так и от функции возбуждения нейронов. Известны три основных вида функции возбуждения: пороговая, линейная и сигмоидальная. Для пороговых элементов выход устанавливается на одном из двух уровней в зависимости от того,

больше или меньше суммарный сигнал на входе нейрона некоторого порогового значения. Для линейных элементов выходная активность пропорциональна суммарному взвешенному входу нейрона.

Для сигмоидальных элементов в зависимости от входного сигнала, выход варьируется непрерывно, но не линейно, по мере изменения входа. Сигмоидальные элементы имеют больше сходства с реальными нейронами, чем линейные или пороговые, но любой из этих типов можно рассматривать лишь как приближение.

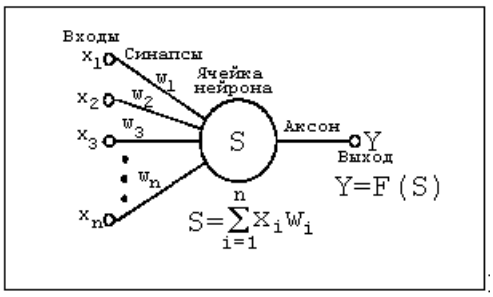


Рис. 3.22. Схема нейрона.

Уникальные возможности и разнообразие задач, решаемых мозгом, непрерывно открывают исследователям новые механизмы и модели его поведения. Поэтому существуют и постоянно добавляются различные модели нейронов, каждая из которых обладает своими свойствами, обуславливающими новые неисчерпаемые возможности построения нейросетевых парадигм.

Модель Маккалоха (1943). Теоретические основы нейроматематики были заложены в начале 40-х годов. У. Маккалох и его ученик У. Питтс сформулировали основные положения теории деятельности головного мозга. Ими были получены следующие результаты:

- разработана модель нейрона как простейшего процессорного элемента, выполняющего вычисление переходной функции от скалярного произведения вектора входных сигналов и вектора весовых коэффициентов;
- предложена конструкция сети таких элементов для выполнения логических и арифметических операций;
- сделано основополагающее предположение о том, что такая сеть способна обучаться, распознавать образы, обобщать полученную информацию.

Несмотря на то, что за прошедшие годы нейроматематика ушла далеко вперед, многие утверждения Маккалоха остаются актуальными и поныне. В частности, при большом разнообразии моделей нейронов принцип их действия, заложенный Маккалохом и Питтсом, остается неизменным. Недостатком данной модели является сама модель нейрона "пороговой" вид переходной функции. В формализме У. Маккалоха и У. Питтса нейроны имеют состояния 0, 1 и пороговую логику перехода из состояния в состояние. Каждый нейрон в сети определяет взвешенную сумму состояний всех других нейронов и сравнивает ее с порогом, чтобы определить свое собственное состояние.

Серьезное развитие нейрокибернетика получила в работах американского нейрофизиолога Френсиса Розенблата (Корнелльский университет). В 1958 году он предложил свою модель нейронной сети. Розенблат ввел в модель Маккалоха и Питтса способность связей к модификации, что сделало ее обучаемой. Эта модель была названа перцептроном.

Сеть

Несмотря на свою кажущуюся простоту, рассмотренные модели искусственных нейронов позволяют строить достаточно сложные и эффективные модели обработки данных, принципиально изменяющие возможности известных алгоритмов обработки. Новые качественные свойства нейронных моделей возникают при соединении отдельных нейронов в сети. Сложные сети в состоянии представлять комплексные задачи, которые не могут решаться отдельными нейронами.

Виды сетей:

слабосвязные - связаны соседи;

полносвязные - каждый с каждым;

многослойные - связаны слои с обратными связями

Многослойные линейные ИНС образуются каскадным соединением искусственных нейронов в слои. При этом выходы одного слоя автоматически являются входами для последующего слоя. Однако увеличение числа слоев в линейных ИНС не может привести к принципиальным изменениям их функциональных возможностей.

Другая классификация сетей включает категории:

1. Многослойные сети прямого распространения (Feedforward Multilayer Networks),
2. Рекуррентные сети (Feedback Recurrent Networks)
3. Ячеистые сети (Celliar Networks).

Структуры различных вариантов ИНС схематично изображены на рис. 3.23.

В многослойных сетях прямого распространения искусственные нейроны, часто называемые узлами или процессорными единицами, сгруппированы в форме слоев (рис.3.23.а). Это означает, что каждый нейрон может получать входной сигнал только из предыдущего слоя, но никак не с последующего. Компонентами таких слоев являются простейшие нейроны, не имеющие элементов запоминания и задержки, и следовательно, не обладающие инерционными свойствами. Выходные отклики в каждом слое, задающие текущее состояние сети, формируются только после предъявления на входе сети входного образа. Как следствие - в сетях прямого распространения отсутствует реальная динамика, а значит и проблема устойчивости, поскольку сеть практически мгновенно формирует нелинейное преобразование входного образа в реакцию на выходе сети.

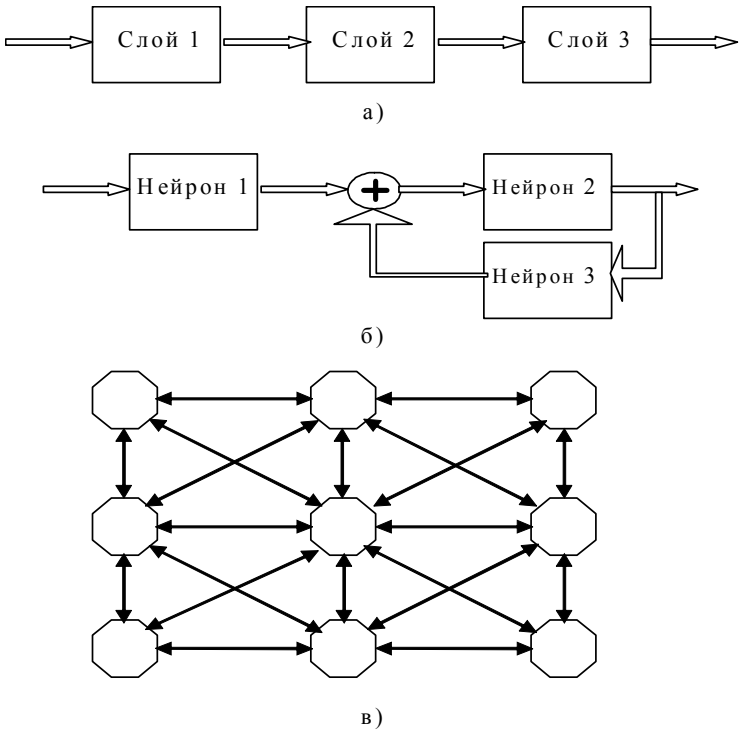


Рис. 3.23. Структуры нейронных сетей

Рекуррентные сети или сети с обратными связями – другой большой класс структур, отличительной особенностью которых является более сложная динамика, характеризующаяся наличием специальных режимов устойчивости и переходных (неустойчивых) состояний. В состав таких сетей входят специфические интегрирующие нейроны и элементы задержки, а также соединения обратных связей как внутри отдельных нейронов, так и групп нейронов. Обобщенная структура сети с обратными связями изображена на рис. 3.23.б. Проблема устойчивости – основная для таких систем. При описании поведения рекуррентных сетей приходится использовать системы нелинейных дифференциальных уравнений и уравнений в конечных разностях, что значительно усложняет их расчет. В сопоставлении с сетями прямого распространения, которые часто называют статическими, рекуррентные сети в полной мере обладают всеми свойствами нелинейных динамических систем.

Ячеистые сети имеют регулярную структуру, подобную структуре известного класса т.н. клеточных автоматов, представляющих регулярно распределенный набор взаимодействующих элементов-процессоров. В ячеистых НС отдельные нейроны взаимодействуют со своими ближайшими соседями посредством т.н. внутрислойных или латеральных связей. Ячейки – нейроны, не связанные вместе непосредственно, могут влиять друг на друга через другие нейроны в процессе распространения сигналов возбуждения по сети. Сеть обычно организована в виде регулярно распределенного двумерного массива или сетки, состоящей из ячеек четырехугольной, шестиугольной или или иной формы, зависящей от числа непосредственных связей со своими ближайшими соседями. В качестве примера на рис.в изображена сеть, состоящая из восьмиугольных ячеек, каждая из которых имеет связь с восемью ближайшими соседями. Благодаря такой схеме при возбуждении одного из нейронов происходит ансамблевое возбуждение участка сети, формируя пространственный (координатный) признак кодируемой информации.

В целом, искусственные нейронные сети могут иметь еще более сложные архитектуры, состоящие из разнородных нейронов или иерархически организованных

структур, каждая из которых привносит свои специфические возможности в свойства сети.

Обучение и конструирование

Цель обучения ИНС - добиться для некоторого множества входных воздействий или образов, предъявляемых сети, желаемой или близкой к желаемой реакции.

Достижение цели осуществляется путем последовательного предъявления сети т.н. ***обучающей выборки*** - набора входных образов с одновременной подстройкой весов в соответствии с определенным алгоритмом. Формально процесс обучения представляет итерационную процедуру подстройки весовых коэффициентов матрицы синаптических связей согласно:

$$w_{ij}[k+1] = w_{ij}[k] + \alpha \Delta w_{ij}[k]$$

где $w_{ij}[k]$ - значение веса от нейрона i к нейрону j до подстройки.
$w_{ij}[k+1]$ - значение веса от нейрона i к нейрону j после подстройки.
$\Delta w_{ij}[k]$ - величина приращения веса $w_{ij}[k]$, α - параметр, задающий скорость обучения.

Процесс обучения в общем виде состоит в сопоставлении реального вектора – отклика сети $\mathbf{f}=[f_1, f_2, \ldots, f_m]^T$ с ***целевым вектором*** эталонных откликов $\mathbf{f_{эт}} = [f_{эт\ 1}, f_{эт\ 2}, \ldots, f_{эт\ m}]^T$ с целью вычисления вектора невязки $\Delta\mathbf{f} = [\Delta f_1, \Delta f_2, \cdots, \Delta f_m]^T$ и последующей реализации по полученной невязке ***обучающего правила*** формирования градиентных векторов подстройки синаптических коэффициентов $\Delta\mathbf{w_j} = [\Delta w_{j1}, \Delta w_{j2}, \ldots, \Delta w_{jm}]^T$, (j=1,2,…, m) нейронной сети. (см. Шкодырев)

Способы обучения:

Способ обучения, реализуемый в процессе настройки сети, является ключевым признаком, определяющим принцип ее работы. Классическая теория выделяет две основные парадигмы, обуславливающие разделение сетей на различные классы:

- Обучение с учителем (Supervised Learning),
- Обучение без учителя (Unsupervised Learning).

Обучение с учителем – наиболее общая классическая парадигма обучения, ориентированная на достижение цели в задачах классификации, распознавания, верификации образов. Ее принципиальной особенностью является наличие в обучающей выборки т.н. ***обучающих примеров*** или ***обучающих пар***, когда каждому предъявляемому образу $\mathbf{x}_i$ ставится в соответствие требуемая реакция или ***целевой вектор*** $\mathbf{f}_{эт}^{(i)}$:

Формально приближение к целевой функции состоит в подстройке элементов $\Delta w_{ij}[k]$ весовой матрицы $\Delta\mathbf{W}[k]$.

Разновидностью обучения с учителем является т.н. ***обучение с подтверждением*** (Reinforcement Learning), при котором критерий качества или правильности поведения сети формируется по интегральной оценке серии опытов или испытаний, проводимых в процессе предъявления сети обучающих примеров. В отличие от предыдущей схемы алгоритм включает интегрирующий элемент ***накопления опыта***, формирующий интегральную ошибку по серии испытаний. В процессе такого обучения сеть получает некоторую оценку своих действий по результатам нескольких опытов, приводящих к определенному результату. По тому же образному сравнению, при обучении с подтверждением сеть становится подобна ребенку, оставленному отлучившейся на время мамой, и его попыткам самостоятельно изучить незнакомый ему предмет в виде сладкой конфеты или кислого лимона. Конечный результат такого обучения в форме

заключения «это вкусно» или «это невкусно» становится критерием отнесения образа к одному из классов.

Обучение без учителя – концепция самообучающихся и самоорганизующихся систем. Принципиальная особенность такого обучения – отсутствие эталонных реакции сети на предъявляемые примеры. В отсутствии учителя его функции берет на себя сама обучающая выборка, а критерием цели становится извлекаемая информация в виде скрытых закономерностей или некоторого смысла.. Упрощенная схема алгоритма обучения без учителя приведена на рис.1.5.3.

Самообучение по Кохонену - обучение ***на обучающих выборках,*** которое не нуждается в целевом векторе для сети и, следовательно, не требует сравнения с предопределенными эталонными ответами. Обучающая выборка – множество входных векторов, достаточно близких по своим свойствам, формирующим однотипные реакции сети.

В процессе обучения по Кохонену сеть сама выделяет скрытые статистические закономерности обучающего множества и группирует сходные вектора в кластеры. Предъявление на вход ИНС вектора из одного кластера будет давать общую реакцию, но до обучения невозможно предсказать какой выход будет сформирован данным классом входных векторов.

Схема обучения и конструирования:

1. Выбор начальной сети. Например, трехслойная, с числом нейронов внутреннего слоя равного полу сумме входов и выходов.

2. Обучение - подбираем веса входов нейронов выходного и внутреннего слоев так, чтобы для всех *j*

j - номер примера; Dj - выходное значение примера; Yj - выходное значение сети.

3. При неудаче меняем граф сети, используем обратные связи.

Пример – распознавание букв.

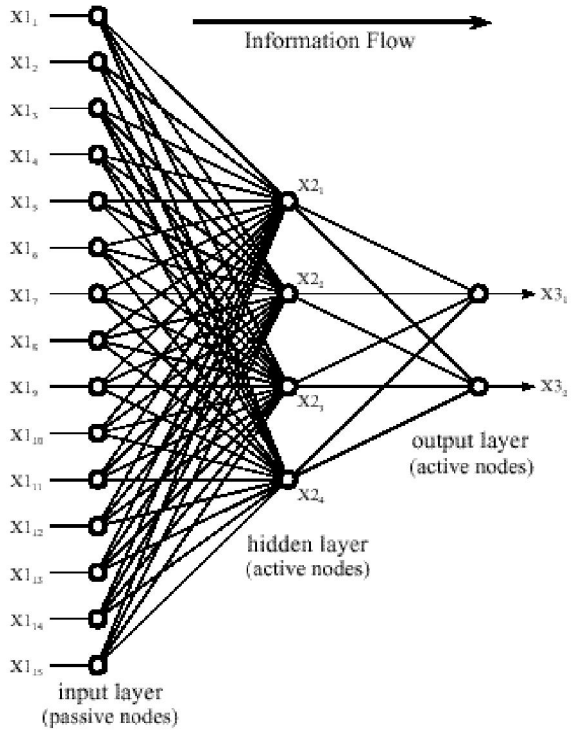


Рис. 3.24. Трехслойная сеть распознавания букв.

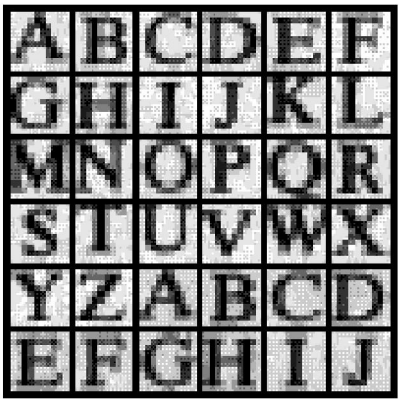


Рис. 9.25. Изображения букв.

Способы реализации нейронных сетей

Нейронные сети могут быть реализованы двумя путями: первый - это программная модель НС, второй - аппаратная. Основными коммерческими аппаратными изделиями на основе НС являются и, вероятно, в ближайшее время будут оставаться нейроБИС.

Среди разрабатываемых в настоящее время нейроБИС выделяются модели фирмы Adaptive Solutions (США) и Hitachi (Япония). НейроБИС фирмы Adaptive Solutions, вероятно, станет одной из самых быстродействующих: объявленная скорость обработки составляет 1,2 млрд. соединений/с. (НС содержит 64 нейрона и 262144 синапса). НейроБИС фирмы Hitachi позволяет реализовать НС, содержащую до 576 нейронов. Эти нейроБИС, несомненно, станут основой новых нейрокомпьютеров и специализированных многопроцессорных изделий.

Большинство сегодняшних нейрокомпьютеров представляют собой просто персональный компьютер или рабочую станцию, в состав которых входит дополнительная нейроплата. К их числу относятся, например, компьютеры серии FMR фирмы Fujitsu. Однако наибольший интерес представляют специализированные нейрокомпьютеры, непосредственно реализующие принципы НС.

Типичными представителями таких систем являются компьютеры семейства Mark фирмы TRW (первая реализация перцептрона, разработанная Розенблатом, называлась Mark I). Модель Mark III фирмы TRW представляют собой рабочую станцию, содержащую до 15 процессоров семейства Motorola 68000 с математическими сопроцессорами. Все процессоры объединены шиной VME. Архитектура системы, поддерживающая до 65 000 виртуальных процессорных элементов с более чем 1 млн. настраиваемых соединений, позволяет обрабатывать до 450 тыс. межсоединений/с. Mark IV - это однопроцессорный суперкомпьютер с конвейерной архитектурой. Он поддерживает до 236 тыс. виртуальных процессорных элементов, что позволяет обрабатывать до 5 млн. межсоединений/с.

Фирма Computer Recognitiion Systems (CRS) продает серию нейрокомпьютеров WIZARD/CRS 1000, предназначенных для обработки видеоизображений. Размер входной изображения 512 x 512 пиксел. Модель CRS 1000 уже нашла применение в промышленных системах автоматического контроля.

3.10. Процессоры с многозначной (нечеткой) логикой

Идея построения процессоров с нечеткой логикой (fuzzy logic) основывается на нечеткой математике. Математическая теория нечетких множеств, предложенная проф. Л.А. Заде, являясь предметом интенсивных исследований, открывает все более широкие возможности перед системными аналитиками. Основанные на этой теории различные компьютерные системы, в свою очередь, существенно расширяют область применения нечеткой логики.

Подходы нечеткой математики позволяют оперировать входными данными, непрерывно меняющимися во времени, и значениями, которые невозможно задать однозначно, такими, например, как результаты статистических опросов. В отличие от традиционной формальной логики, известной со времен Аристотеля и оперирующей точными и четкими понятиями типа истина и ложь, да и нет, ноль и единица, нечеткая логика имеет дело со значениями, лежащими в некотором (непрерывном или дискретном) диапазоне.

Функция принадлежности элементов к заданному множеству также представляет собой не жесткий порог «принадлежит – не принадлежит», а плавную сигмоиду, проходящую все значения от нуля до единицы. Теория нечеткой логики позволяет выполнять над такими величинами весь спектр логических операций – объединение, пересечение, отрицание и др

Согласно знаменитой теореме FAT (Fuzzy Approximation Theorem), доказанной Коско, любая математическая система может быть аппроксимирована системой, основанной на нечеткой логике. Свое второе рождение теория нечеткой логики пережила в начале восьмидесятых годов, когда сразу несколько групп исследователей (в основном в США и Японии) занялись созданием электронных систем различного применения, использующих нечеткие управляющие алгоритмы. Используя преимущества нечеткой логики, заключающиеся в простоте содержательного представления, можно упростить проблему, представить ее в более доступном виде и повысить производительность системы.

Задачи с помощью нечеткой логики решаются по следующему принципу:

1. численные данные (показания измерительных приборов, результаты анкетирования) фаззируются (переводятся в нечеткий формат);
2. обрабатываются по определенным правилам;
3. дефаззируются и в виде привычной информации подаются на выход.

Оказалось возможным создание нечеткого процессора, позволяющего выполнять различные нечеткие операции и приближенные рассуждения (нечеткий вывод) в соответствии с правилами логического вывода. В 1986 году в AT and T Bell Labs создавались процессоры с «прошитой» нечеткой логикой обработки информации. В начале 90-х компания Adaptive Logic из США выпустила кристалл, сделанный по аналогово-цифровой технологии. Он позволит сократить сроки конструирования многих встроенных систем управления реального времени, заменив собой традиционные схемы нечетких микроконтроллеров. Аппаратный процессор нечеткой логики второго поколения принимает аналоговые сигналы, переводит их в нечеткий формат, затем, применяя соответствующие правила, преобразует результаты в формат обычной логики и далее – в аналоговый сигнал. Все это осуществляется без внешних запоминающих устройств, преобразователей и какого бы то ни было программного обеспечения нечеткой логики. Этот микропроцессор относительно прост. Но так как

его основу составляет комбинированный цифровой/аналоговый кристалл, он функционирует на очень высоких скоростях (частота отсчетов входного сигнала – 10 кГц, а скорость расчета – 500 тыс. правил/с), что во многих случаях приводит к лучшим результатам в системах управления по сравнению с более сложными, но медлительными программами.

Многоядерность

Многоядерность – это мультипроцессорность в одном корпусе. Все, что было сказано ранее справедливо и в этом случае. Простейная двухядерная архитектура представляет собой UMA SMP архитектуру.

В целом выделяют три варианта многоядерных процессоров:

1. Совершенно независимые процессорные ядра, каждое со своей кэш-памятью, расположены на одном кристалле и просто используют общую системную шину.

2. Похожий вариант — когда несколько одинаковых ядер расположены на разных кристаллах, но объединены вместе в одном корпусе процессора (многочиповый процессор).

3. Наконец, ядра могут быть тесно переплетены между собой на одном кристалле и использовать некоторые общие ресурсы кристалла (скажем, кэш-память).

Многоядерность позволяет не только повысить производительность за счет параллельности, но и существенно снизить потребление на единицу скорости. В настоящее время увеличение производительности на 1% влечет за собой повышение потребляемой мощности на 3%. Это происходит из-за того, что при уменьшении размера транзисторов и их плотности на кристалле наряду с тактовой частотой увеличивается и ток утечки, что ведет к нагреву и неэффективному расходованию электроэнергии. Если плотность транзисторов будет расти нынешними темпами, то без усовершенствования управления питанием к 2015 г. микропроцессоры будут выделять десятки тысяч ватт тепла на квадратный сантиметр. Чтобы удовлетворять потребностям будущего, необходимо существенно сократить потребляемую мощность.

Процессоры Intel и другие будут состоять из десятков и даже сотен небольших ядер. Задачи, критичные по времени, будут работать на быстрых ядрах с большей потребляемой мощностью. В то время как остальные - на более медленных ядрах с пониженным энергопотреблением. Строятся архитектуры с интеллектуальным управлением питанием, которое сможет автоматически реконфигурировать процессор с учетом потребностей питания и рабочей нагрузки.

Часть 4. Дискретизация аналогового сигнала. Кодирование звука.

4.1. Дискретизация аналогового сигнала

Дискретизация — преобразование непрерывной функции в дискретную.

Частота дискретизации - частота взятия отсчетов непрерывного во времени сигнала при его дискретизации. Измеряется в герцах.

Чем выше частота дискретизации, тем более широкий спектр сигнала может быть представлен в дискретном сигнале.

Некоторые из используемых частот дискретизации звука:
8 000 Гц — телефон, достаточно для речи,;
22 050 Гц — радио;
44 100 Гц — используется в Audio CD;
48 000 Гц — DVD, DAT;
192 000 Гц — DVD-Audio (MLP 2.0);
2 822 400 Гц — SACD, процесс однобитной дельта-сигма модуляции, известный как DSD — Direct Stream Digital, совместно разработан компаниями Sony и Philips;

Табл.4.1. Сравнение звуковых форматов без сжатия

Название формата	Расширение файла	Квантование, бит	Частота дискретизации, кГц	Число каналов	Битрейт (на канал), Мбит/с	Степень сжатия/ упаковки	Назначение	Выпуск
AIFF	.aiff; .aif	8; 16; 24; 32	11,025; 22,05; 24; 32; 44,1; 48; 96; 192	1; 2; 3; 4; 6	до 6,144	1:1; без сжатия	хранение звуковых данных на ПК	1988, Apple
WAVE (WAV)	.wav	8; 16; 24; 32	11,025; 22,05; 24; 32; 44,1; 48; 96; 192	1; 2; 3; 4; 6	до 6,144	1:1; без сжатия	хранение звуковых данных на ПК	1991, Microsoft и IBM
DSD	.dff; .dsf	1	2822,4 (64f_s)[1]; 5644,8 (128f_s)	2, 5.1	2,8224; 5,6448	1:1; без сжатия, возможно сжатие DST	SACD	1998, Sony и Philips
Digital eXtreme Definition (DXD)	DXD	24; 32	352,8	2, 5.1	8,4672; 11,2896	1:1; без сжатия	профессиональное производство SACD	2004, Sony

Формат WAV. Импульсно-кодовая модуляция

Waveform Audio File Format (**WAVE**, **WAV**) — формат файла-контейнера для хранения записи оцифрованного аудиопотока. Этот контейнер, как правило, используется для хранения несжатого звука в импульсно-кодовой модуляции.

ИКМ (Pulse Code Modulation, PCM) - мгновенное значение аналогового сигнала измеряется аналого-цифровым преобразователем (АЦП) через равные промежутки времени.

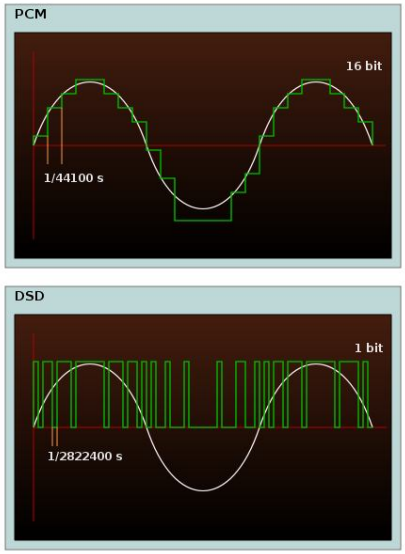


Рис. 4.1. Импульсно-кодовая и плотностно-импульсная модуляции

Формат DSD. Одноразрядное квантование

DSD (*Direct Stream Digital*) - однобитный аудиоформат, разработанный компаниями Sony и Philips, в котором используется кодирование *плотностно-импульсной модуляцией* (*Pulse Density Modulation* (PDM), разновидность сигма-дельта-модуляции) и применяется для хранения звукозаписей на оптическом носителе SACD (**Super Audio CD**).

Аналоговый звуковой сигнал конвертируется в цифровой с помощью дельта-сигма модуляции при частоте дискретизации 2 822,4 кГц (в 64 раза больше, чем у CD Audio), но с разрешением всего 1 бит, в отличие от используемых в формате CD 16 бит при частоте 44,1 кГц.

Такое преобразование, при котором отсчеты аналогового сигнала берутся с частотой, многократно превышающей верхнюю граничную частоту сигнала называется семплирование с передискретизацией (*oversampling*). Избыточное семплирование имеет большое значение при устранении шумов квантования. Мощность шума в частотном диапазоне, занимаемом полезным сигналом, уменьшается пропорционально повышению частоты дискретизации. Таким образом, отношение сигнал/шум увеличивается, когда частота дискретизации становится больше. Фазовая характеристика становится более схожей с высокочастотной характеристикой аналоговых систем.

Положительное изменение амплитуды будет представлено всеми «1». Отрицательное — всеми «0». Нулевая точка будет представлена сменой двоичного числа. Поскольку значение амплитуды аналогового сигнала в каждый момент представлено в виде плотности импульсов, этот метод иногда называют плотностно-импульсной модуляцией.

4.2. Ошибки дискретизации. Погрешности преобразования

Поскольку сигнал невозможно вводить в ЭВМ непрерывно, то требуется провести его дискретизацию. Сущность дискретизации заключается в том, что непрерывность во времени функции заменяется последовательностью коротких импульсов. Дискретизация вызывает потерю информации о сигнале, следовательно,

появляется погрешность. Определим какова эта погрешность и как на нее влияет период дискретизации Δt.

Период дискретизации (шаг дискретизации) должен быть таким, чтобы было возможно восстановление непрерывной функции по ее отсчетам с допустимой точностью.

При дискретизации аналогового сигнала (рис. 4.2) возникает ряд погрешностей, которые будут определять точность восстановления сигнала (качество прослушиваемого звука). Источниками погрешностей являются: механизм взятия отсчетов (погрешность дискретизации **Δд)** , квантование (оцифровка) значения отсчета, погрешности аналоговых преобразований (погрешность статическая **Δст)** и погрешности восстановления **Δвосст**.

Δ=Δст + Δд + Δвосст

Динамическая составляющая погрешности - доля погрешности, которая обусловлена неравенством нулю производных измеряемого сигнала.

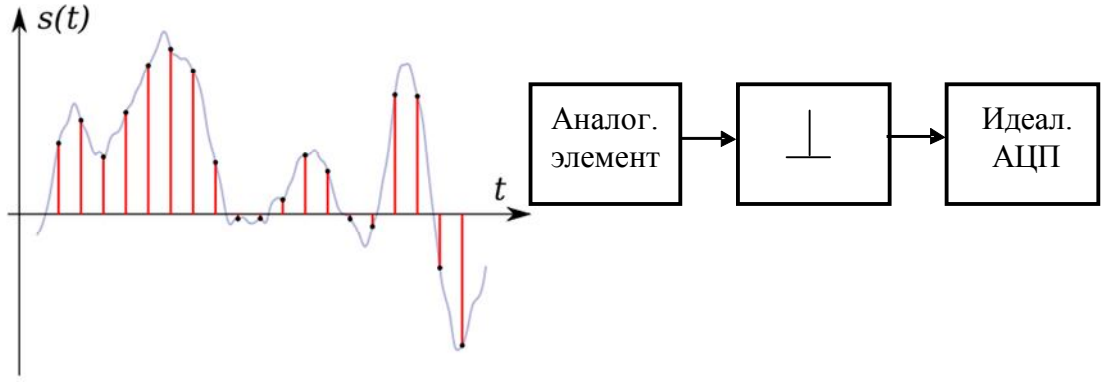


Рис. 4.2. Дискретизация аналогового сигнала.

Способы дискретизации:

1. Δt - const равномерная дискретизация
2. Δt - var неравномерная дискретизация

Δt -период дискретизации; *f*д=1/ Δt частота дискретизации.

Виды и формы представления погрешностей

Погрешность измерения - отклонение результатов измерения от истинного значения измеряемой величины.

Формы представления:

- Абсолютная $\Delta = X - X_u$

- Приведенная $\gamma = \dfrac{\Delta}{X_{max} - X_{min}}$

- Относительная $\delta = \dfrac{\Delta}{X}$

Аддитивная погрешность - погрешность средства измерения во всем диапазоне, ограниченная постоянным пределом.

Мультипликативная погрешность - погрешность, изменяющаяся пропорционально изменению входной величины.

Полная статическая погрешность имеет следующие составляющие:

$$\Delta_{\Sigma} = \Delta_c + \Delta^0 + \Delta_m + \Delta_{\bullet промахи} + \ldots$$

Методическая погрешность Δ_m - погрешность, определяемая методом измерения.

Систематическая погрешность [redacted] - не изменяющаяся со временем погрешность, которая может быть предсказана и благодаря этому почти полностью устранена введением соответствующих поправок. Выделяют два вида систематической .погрешности: постоянные и изменяющиеся. Опасность постоянных в том, что их трудно обнаружить. Единственный способ их обнаружения состоит в поверке (аттестации) приборов. Изменяющиеся погрешности (в процессе измерения неизменны) связаны с такими процессами как изменение температуры, напряжения питания и др. Можно выделить следующие составляющие систематической погрешности: погрешность усиления, квантования, нуля. Первая может вести себя как мультипликативная погрешность, а две оставшиеся как аддитивная погрешность. $\Delta_c = \Delta_y + \Delta_к + \Delta_н + \ldots$

Дрейф непредсказуемые погрешности медленно изменяющиеся во времени. Они могут быть скорректированы введением поправок лишь в данный момент времени, а далее вновь непредсказуемо возрастают.

Случайная погрешность Δ^0 - непредсказуемая (недостаточно изученная) ни по знаку ни по размеру погрешность. Она определяется совокупностью причин, трудно подлежащих анализу. Присутствие случайных погрешностей легко обнаружить при повторных измерениях в виде разброса получаемых результатов. Для обработки применяют статистические методы.

При наличии бесконечного множества факторов, имеющих разные законы распределения и воздействующих на измеряемую величину, считают, что измеряемая величина распределена нормально(теорема Ляпунова). Нормальный закон распределения (Гаусса) предполагает:

1. Погрешности принимают бесконечно разные значения.
2. Погрешности одинаковые, но разного знака встречаются одинаково часто (симметричность).
3. С ростом погрешности частота ее появления уменьшается.

Оценкой разброса результатов измерений вокруг среднего значения является среднеквадратическое отклонение σ или дисперсия. Поскольку бесконечное количество измерений невозможно, то встает вопрос насколько измеренное значение отличается от истинного. В этом случае используют правило 3-х σ :

$$\left|X_i - M(X)\right| \le 3\sigma \quad \text{с} \quad \text{p=99.7\%}$$

С вероятностью 99.7% измеренные значения попадают в интервал $\pm\ 3\sigma$

Всякому измерению должно предшествовать изучение измеряемой величины и ее систематической составляющей погрешности измерений с целью решения вопроса многократных измерений.

Если размах результатов предварительных наблюдений $|X_{max} - X_{min}| \le 0.8\sigma_c$ то доминирующая систематическая погрешность и измерение однократно.

Если $|X_{max} - X_{min}| > 8\sigma_c$ то суммарная погрешность определяется случайной составляющей и измерение должно быть многократным.

Если $|X_{max} - X_{min}| \approx 3\sigma_c$ то суммарная погрешность определяется суммой систематической и случайной составляющих и измерения многократны.

Приведем передаточную характеристику АЦП (рис. 4.3). Из-за технологического разброса параметров при изготовлении интегральных микросхем реальные АЦП не имеют идеальной передаточной характеристики. Отклонения от идеальной передаточной характеристики определяют статическую погрешность АЦП.

Закон распределения методической погрешности Δ_m считаем равномерным, т.е. вероятность попадания измеряемой величины в разные участки кванта одинакова. Среднеквадратическое отклонение для равномерного закона распределения: $\sigma_m = \frac{\Delta x}{2\sqrt{3}}$.

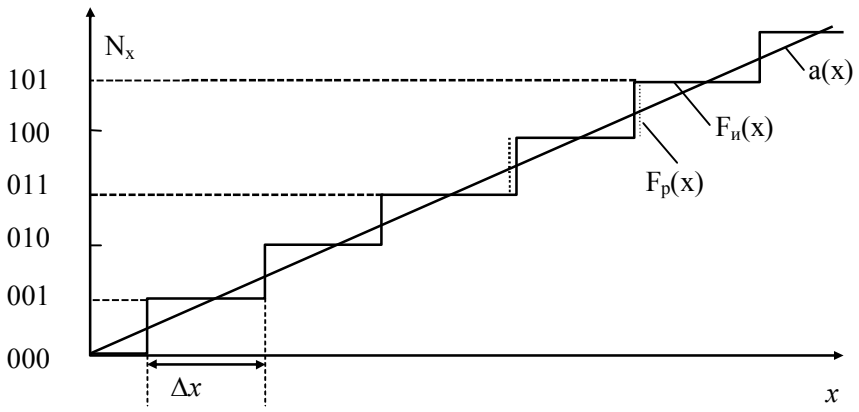


Рис. 4.3. Квантующая характеристика АЦП.
a(x)- идеальный анализатор; $F_и(x)$ - идеальное квантование (идеальный цифр.преобразователь); $F_р(x)$ - реальный цифровой преобразователь.

Перечислим основные типы погрешностей в АЦП и методы их измерения.

Аддитивная погрешность АЦП (Offset) может быть определена как смещение всей передаточной характеристики влево или вправо относительно оси входного напряжения.

Мультипликативная погрешность (погрешность полной шкалы) (Full-Scale) представляет собой разность между идеальной и реальной передаточными характеристиками в точке максимального выходного значения при условии нулевой аддитивной погрешности (смещение отсутствует). Это проявляется как изменение наклона передаточной функции.

Дифференциальная нелинейность (DNL). У идеальной передаточной характеристики АЦП ширина каждой "ступеньки" должна быть одинакова. Разница в длине горизонтальных отрезков этой кусочно-линейной функции из 2^n "ступеней" представляет собой дифференциальную нелинейность.

Интегральная нелинейность(INL). Интегральная нелинейность - это погрешность, которая вызывается отклонением линейной функции передаточной характеристики АЦП от прямой линии, как показано на рис. Обычно передаточная функция с интегральной нелинейностью аппроксимируется прямой линией по методу наименьших квадратов. Часто аппроксимирующей прямой просто соединяют наименьшее и наибольшее значения. Интегральную нелинейность определяют путем сравнения напряжений, при которых происходят кодовые переходы. Для идеального АЦП эти переходы будут происходить при значениях входного напряжения, точно кратных LSB. А для реального преобразователя такое условие может выполняться с погрешностью. Разность между "идеальными" уровнями напряжения, при которых происходит кодовый переход, и их реальными значениями выражается в единицах LSB и называется интегральной нелинейностью.

Динамическая погрешность.

Динамическая погрешность связана как с процессом измерения - дискретизации сигнала (погрешность дискретизации $\Delta_Д$), так и с процессом восстановления (погрешность восстановления $\Delta_{восст}$). Эти погрешности дополняют статическую составляющую полной погрешности измерения ($\Delta_{ст}$) $\Delta=\Delta_{ст} + \Delta_Д + \Delta_{восст}$

Источником погрешности дискретизации $\Delta_Д$ являются следующие причины:

1. Нестабильность временных интервалов между отсчетами.
2. Неточность взятия отсчетов.

Нестабильность временных интервалов Δt между отсчетами возникает из-за нестабильности устройств, синхронизирующих работу АЦП и нестабильности времени срабатывания ключей АЦП.

Неточность взятия отсчетов обязана своим появлением, как процессу дискретизации, так и процессу квантования и связана с конечным временем преобразования.

Неточности взятия отсчёта и погрешность из-за неопределённости в моменте отсчёта эквивалентны друг другу, т.е. одна из них может быть представлена через другую. Можно считать, что отсчёт делается абсолютно точно и имеет место погрешность в определении момента взятия отсчёта, или наоборот - момент отсчёта t, определён совершенно точно, а вся погрешность обусловлена неточностью взятия отсчёта из-за ненулевой скорости изменение входного сигнала. Практически имеют место оба явления, разделить их трудно, и динамическую погрешность характеризуют одним параметром - апертурным временем t_a.

Апертурное время - полное время, в течение которого имеет место неопределённость момента, к которому можно отнести значение отсчёта.

Погрешность восстановления.

Источниками погрешности восстановления являются:

1. Невыполнение теоремы отсчетов.
2. Использование приближенного метода восстановления.

Метод восстановления с использованием линейной интерполяции.

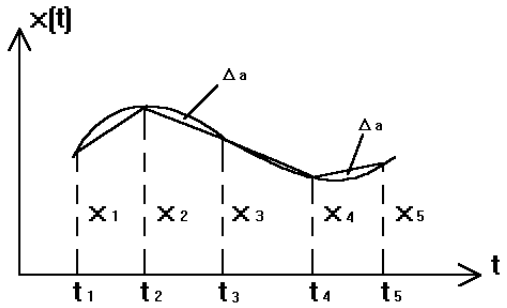


Рис. 4.4. Схема метода линейной интерполяции.

Простейшим и часто используемым видом локальной интерполяции является линейная интерполяция. Она состоит в том, что заданные точки x_1, x_2, … соединяются прямолинейными отрезками, и функция *x(t)* приближается к ломаной с вершинами в данных точках. Поскольку имеется n интервалов, то для каждого из них в качестве уравнения интерполяционного полинома используется уравнение прямой, проходящей через две точки.

Замечание: Аппроксимация - отыскание кривой, проходящей вблизи множество точек . Интерполяция - отыскание кривой, - через множество точек.

Δ_a - ошибка аппроксимации; $\Delta_a = \Delta_в$ - погрешность восстановления

По формуле Ньютона погрешность восстановления равна: $\Delta_в = |x(t)''|_{max}\, \Delta t^2/8$.

Определим, как часто надо брать отсчеты, чтобы получить относительную погрешность восстановления $\delta = 1\%$.

Пусть $\omega_в$ - наивысшая частота процесса. Зададим $x(t) = x_м sin(\omega_в t)$, тогда максимум второй производной : $|x(t)''|_{max} = x_м \omega_в^2$

$\Delta_в = x_м \omega_в^2\, \Delta t^2/8$; $\Delta t^2 = 8\Delta_д / x_м \omega_в^2 = 8\,\delta/\omega_в^2$, где $\omega_в = 2\pi f_в$

$f_к \geq 1/\Delta t = (\pi f_в)/(\sqrt{2}\sqrt{\delta})$ в результате отсюда частота квантования: $f_к \geq 22.2\, f_в/\sqrt{\delta\%}$

При $\delta\% = 1\%$ $\boldsymbol{f_к \geq 22.2 f_в.}$

Таким образом, для восстановления с точностью 1% необходимо 22 отсчета на периоде.

В этом методе существует задержка Δt, поскольку надо иметь минимум две точки для восстановления промежуточной. Если в системе такая задержка невозможна, то надо воспользоваться другим методом, например методом прямоугольников.

Если недостаточна точность восстановления, то можно использовать метод квадратичной интерполяции. В случае квадратичной интерполяции в качестве интерполяционной функции на отрезке (t_i, t_{i+1}) принимается квадратный трехчлен.

Метод прямоугольников.

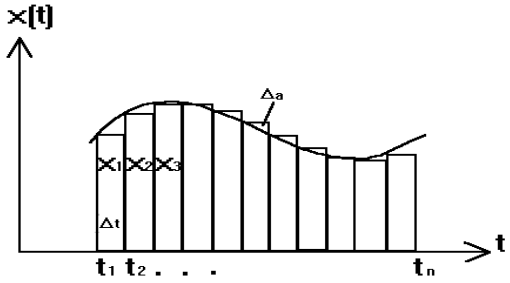


Рис. 4.5. Схема восстановления сигнала методом прямоугольников.

Погрешность восстановления соответствует погрешности вычисления интеграла методом прямоугольников и определяется выражением:

$$\Delta_в = \Delta_a = \left|\frac{dx}{dt}\right|_{\max} \cdot \Delta t$$

Пусть имеем синусоидальный сигнал: $x = x_M \sin\omega_в t$.Необходимо обеспечить для $x(t)$ наиболее точное восстановление:

$\left|\frac{dx}{dt}\right|_{\max} = x_M \omega_B$, где $\omega_в$ - высшая частота сигнала. Отсюда: $\Delta_в = x_M 2\pi f_B \Delta t$.

Выразим период квантования: $\Delta t = \frac{\Delta_в}{x_M \cdot 2\pi \cdot f_B}$ и частоту квантования:

$$f_k \geq \frac{1}{\Delta t} = \frac{2\pi \cdot f_B}{\delta} = 628\frac{f_B}{\delta\%}$$

Получим, что для восстановления с точностью 1% необходимо 628 отсчетов на периоде. Здесь отсутствует запаздывание, но частота квантовая fk много больше, чем при той же погрешности при линейном методе восстановления.

4.3. Теорема Котельникова. Эффекты дискретизации

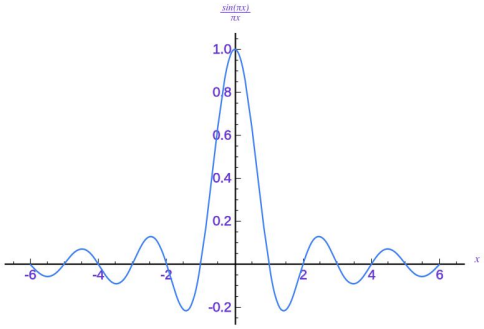
Величину периода дискретизации Δt, позволяющую восстановить x(t) без потери информации, определяет теорема Котельникова. Теорема была сформулирована В. А. Котельниковым в 1933 году в его работе «О пропускной способности эфира и проволоки в электросвязи» и является одной из основополагающих теорем в теории и технике измерений и цифровой связи. Теорема отсчётов была независимо сформулирована и другими учеными: Уиттекер**1915**, Найквист**1924**, Шеннон.

Теорема Котельникова. Заданную функцию времени $x(t)$, ограниченную по полосе частот от 0 до f_c можно полностью определить заданием её значений в дискретной последовательности точек, отстоящих друг от друга на $\Delta t = 1/2f_c$. Значения функции, промежуточные между отсчетами могут быть восстановлены с помощью ряда:

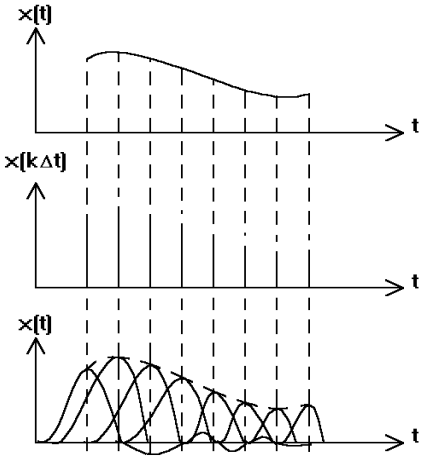
$$x(t) = \sum_{k=-\infty}^{\infty} x(k\Delta t)\frac{\sin(\omega_c(t - k\Delta t))}{\omega_c(t - k\Delta t)}$$

$k = 0, \pm1, \pm2 \ldots \pm\infty$; $\omega_C = 2\pi f_C = 2\pi/\Delta t_C$

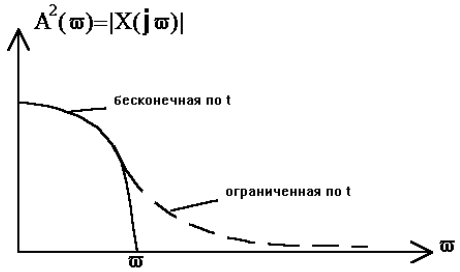
Функция отсчётов *sin z /z* имеет следующую форму :



Графическая схема восстановления функции представлена ниже. Восстановленная функция будет соответствовать исходной с любой наперёд заданной точностью.



Амплитудно-частотная характеристика сигнала $x(t)$ приведена на рис. Функция $x(j\omega)$ является спектром сигнала. $x(j\omega) = \int_{-\infty}^{\infty} x(t)e^{j\omega t}dt$



Неограниченные во времени функции имеют спектр ограниченный частотой среза ω_c. Реальные функции ограничены по времени и, следовательно, $\omega_c \to \infty$.

Условиям теоремы Котельникова полностью удовлетворить нельзя, поскольку надо знать $x(k\Delta t)$ при $-\infty < k < +\infty$, т.е. надо знать предысторию. Следовательно, восстановить можно лишь приближенно. Надо ввести искусственно ограничение на спектр ω_c, а, следовательно, появится погрешность:

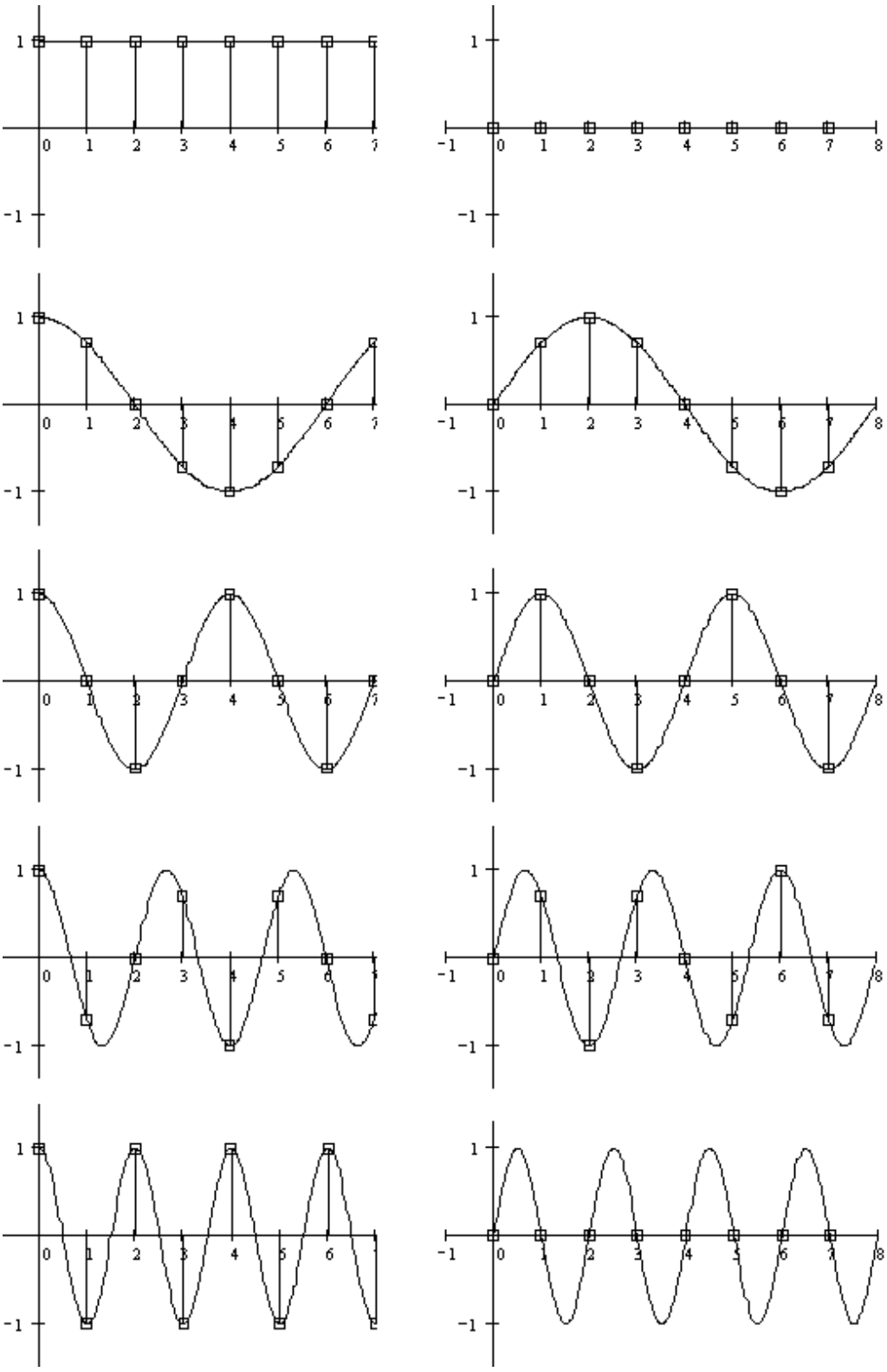
$$\varepsilon = \frac{E_0}{E_c} = \frac{\int_0^{\infty} A^2(\omega)d\omega - \int_0^{\omega_C} A^2(\omega)d\omega}{\int_0^{\infty} A^2(\omega)d\omega},$$

где Ео - энергия отброшенной части спектра; Ес – полная энергия сигнала.

Чем меньше ε, тем больше ω_c (т.е. ω_c сдвигается вправо). Таким образом, частота в теории Котельникова может быть выбрана из условия получения требуемой точности в процессе восстановления.

Функция, имеющая ограниченный спектр (**финитная по спектру)** может быть точно восстановлена рядом Котельникова.

Преобразование Фурье. Графическое представление



Непрерывное преобразование:

$$X(\nu) = \int_{-\infty}^{+\infty} x(t) \cdot e^{-2\pi i \nu t} dt \qquad x(t) = \int_{-\infty}^{+\infty} X(\nu) \cdot e^{2\pi i \nu t} d\nu$$

Дискретное преобразование:

- Базисные функции дискретного преобразования Фурье для сигнала длины N = 8.

- Имеем N/2 + 1 = 5 различных базисных частот.
- Имеем N+2 базисные функции, 2 из которых тождественно равны нулю.
- Обратное преобразование Фурье – вычисление суммы конечного ряда Фурье (сложить N штук N-точечных синусоид со своими коэффициентами).

$$A_k = \frac{1}{N}\sum_{i=0}^{N-1} x[i]\cos\frac{2\pi ki}{N} \qquad k = 0,\frac{N}{2}$$

$$B_k = \frac{2}{N}\sum_{i=0}^{N-1} x[i]\sin\frac{2\pi ki}{N} \qquad k = 0,\ldots,\frac{N}{2}$$

$$x[n] = \sum_{k=0}^{N/2} A_k \cos\frac{2\pi kn}{N} + \sum_{k=0}^{N/2} B_k \sin\frac{2\pi kn}{N}$$

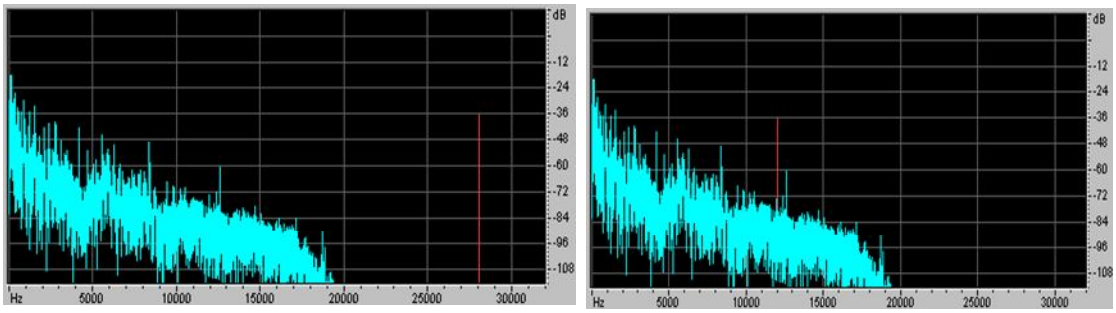
Алиасинг

Критерий Найквиста (теорема Котельникова) требует, чтобы частота дискретизации была, по крайней мере, вдвое больше полосы сигнала, в противном случае информация о сигнале будет потеряна. Если частота дискретизации меньше удвоенной полосы аналогового сигнала, возникает эффект, известный как наложение спектров (aliasing)

Пусть звук не содержит частот выше 20 кГц. Тогда, по теореме Котельникова, можно выбрать частоту дискретизации 40 кГц. Пусть в звуке появилась помеха с частотой 28 кГц. Условия теоремы Котельникова перестали выполняться.

Проведем дискретизацию с частотой 40 кГц, а затем – восстановим аналоговый сигнал sinc-интерполяцией.

Помеха отразилась от половины частоты дискретизации в нижнюю часть спектра и наложилась на звук. Помеха переместилась в слышимый диапазон. Алиасинг.

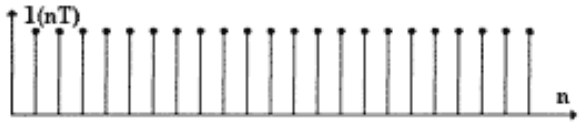


Размножение спектров.

Дискретный сигнал = произведение функции на решетчатую функцию. Решетчатая функция, представленная рядом Фурье(Решетчатые функции определены только в дискретные моменты времени):

$$s_д(t) = s(t) \cdot u(t)$$

$$u(t) = \sum_{m=-\infty}^{\infty} a_m e^{jm\frac{2\pi}{\tau}t}$$ τ - период дискретизации



$$S_д(\omega) = \int_{-\infty}^{\infty} s(t) \cdot e^{-j\omega \cdot t} dt = \int_{-\infty}^{\infty} \sum_{m=-\infty}^{\infty} s(t) a_m e^{-j\omega \cdot t} e^{jm\frac{2\pi}{\tau}t} dt = \sum_{m=-\infty}^{\infty} a_m S(\omega - m\frac{2\pi}{\tau})$$

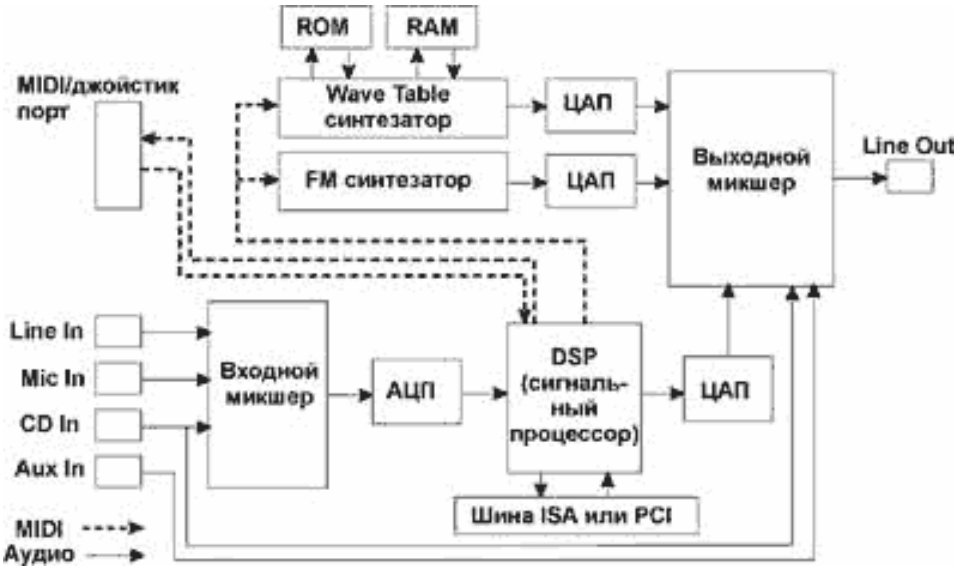
Размножение спектров

Коэффициенты a_m убывают медленно: их огибающей является спектральная плотность короткого импульса дискретизации

4.4. Организация Sound blaster

Звуковая карта или SB становится неотъемлемой частью современного мультимидийного компьютера. SB предназначен как для обработки и генерации звуковых сигналов, в ряде случаев может быть использован в качестве Data Acquisition Board. Качество работы SB очень сильно зависит от параметров самого SB и используемого программного обеспечения. Sound Blaster - одна из торговых марок Creative.

Структурная схема SB.



Основными элементами SB являются:

1. Микшер - смеситель аналоговых сигналов. Он объединяет сигналы с линейного и микрофонного входов, звукового входа накопителя CD-ROM, синтезаторов сигналов. Позволяет регулировать усиление по каждому из каналов.

2. Цифровой и/или аналоговый сигнальный процессоры. Обеспечивают сжатие/восстановление сигналов различными методами: А-методом, Мю-методом, MPEG; а также другие эффекты, связанные с обработкой сигналов

Сжатие звуковых сигналов существенно уменьшает объем звуковых файлов. Пример: Без сжатия - 10с звука(16разр., стерео) занимает 1.8МВ. Сжатие по стандарту True Speech уменьшает объем до 13кБ

3. АЦП Преобразователь позволяет получить цифровой код аналогового сигнала для ввода его в ЭВМ или цифровой сигнальный процессор.

4. ЦАП обеспечивает восстановление аналоговых сигналов.

Разрядность АЦП (ЦАП) определяет динамический диапозон работы бластера (Какое наименьшее изменение аналогого сигнала,которое может привести к изменению цифрового кода).

8р - 48дБ качество аналогого кассетного магнитофона
12р - 72дБ аналоговый катушечный магнитофон
16р - 96дБ проигрыватель компакт-дисков
Частота 44.1кГц при воспроизведении стериосигналов соответствует стандарту звуковых компакт-дисков.

Все сигналы с внешних аудиоустройств поступают на входной микшер звуковой платы. Входной микшер нужен для того, чтобы установить оптимальный уровень записи. После аналого-цифрового преобразования, данные поступают в сигнальный процессор. Этот процессор управляет обменом данными со всеми остальными устройствами компьютера, например через шину PCI.

На любой универсальной мультимедийной звуковой карте есть синтезатор. Практически на всех картах устанавливается не один, а два синтезатора: FM - для сохранения совместимости с Sound Blaster и Wave Table - для получения качественного звука. Именно эти синтезаторы показаны на рисунке.

Исторически так сложилось, что FM-синтезаторы звуковых плат звучат не очень хорошо. Как правило, на современные мультимедийные карты устанавливаются наборы микросхем (чипсеты) FM-синтезаторов производства Yamaha под названием OPL-2, OPL-3 или совместимые с ними. В музыкальных приложениях такие синтезаторы не применяются - они нужны исключительно для звукового сопровождения игр.

Мультимедийные Wave Table синтезаторы позволяют получить уже более качественный звук. На Рис. вы видите, что у Wave Table синтезатора есть не только постоянная память (ROM), но и оперативная (RAM). Оперативной памятью обладают семплеры, и используется она для загрузки любых звуковых файлов, которые проигрываются с разной высотой при нажатии клавиш на подключенной клавиатуре или поступлении команд от секвенсера. То есть Wave Table синтезатор, имеющий оперативную память помимо постоянной - это ни что иное, как комбинация синтезатора и семплера, которая может выполнять функции обоих устройств. Это означает, что вы можете использовать как образцы звучания, хранящиеся в постоянной памяти, так и загружать в оперативную память дополнительные библиотеки или создавать свои собственные звуки. Такая возможность расширяет творческие возможности компьютера.

Чтобы синтезаторы, установленные на звуковой карте можно было использовать в качестве музыкальных инструментов к MIDI/джойстик порту подключают либо MIDI-клавиатуру, либо автономный синтезатор, который может служить в качестве клавиатуры. Сигналы, поступающие с клавиатуры подаются в процессор, который направляет их либо через системную шину к центральному процессору, либо к синтезаторам звуковой карты. Путь MIDI-сигнала зависит от выполняющихся программ.

Каждый из синтезаторов, установленных на звуковой карте имеет свой собственный ЦАП. После преобразования сигналов в аналоговую форму, они поступают на выходной микшер звуковой карты. То есть вы можете устанавливать необходимый баланс синтезаторов, аудиотракта и аудиоустройства, подключенного к дополнительному входу. Такая возможность оказывается крайне полезной при окончательном микшировании композиций, записанных при помощи компьютера. Итоговый сигнал (микс) поступает на линейный выход (Line Out)., который также, как и входы находится на задней панели звуковой карты.

Программирование возможно на двух уровнях: программирование непосредственно регистров, использование функций/команд высокого уровня. В частности подсистема DirectSound определяет ряд функций, которые обеспечивают приложениям практически непосредственный доступ к аппаратуре звукового адаптера. DirectSound входит в семейство «мультимедийных» интерфейсов DirectX

Звуковые эффекты

Звуковые эффекты делаются при помощи изменения характеристик звуковой волны. Наиболее распространенные звуковые эффекты:

- вибрато - амплитудная или частотная модуляция сигнала с небольшой частотой (до 10 Гц). Амплитудное вибрато также носит название тремоло; на слух оно воспринимается, как замирание или дрожание звука, а частотное - как "завывание" или "плавание" звука (типичная неисправность механизма магнитофона).
- динамическая фильтрация (wah-wah — "вау-вау") – реализуется изменением частоты среза или полосы пропускания фильтра с небольшой частотой. На слух воспринимается, как вращение или заслонение/открывание источника звука — увеличение высокочастотных составляющих ассоциируется с источником, обращенным на слушателя, а их уменьшение — с отклонением от этого направления.
- фленжер (flange - кайма, гребень). Название происходит от способа реализации этого эффекта в аналоговых устройствах - при помощи так называемых гребенчатых фильтров. Заключается в добавлении к исходному сигналу его копий, сдвинутых во времени на небольшие величины (до 20 мс) с возможной частотной модуляцией копий или величин их временных сдвигов и обратной связью (суммарный сигнал снова копируется, сдвигается и т.п.). На слух это ощущается как "дробление", "размазывание" звука, возникновение биений - разностных частот, характерных для игры в унисон или хорового пения, отчего фленжеры с определенными параметрами применяются для получения хорового эффекта (chorus). Меняя параметры фленжера, можно в значительной степени изменять первоначальный тембр звука.
- реверберация (reverberation - повторение, отражение). Получается путем добавления к исходному сигналу затухающей серии его сдвинутых во времени копий. Это имитирует затухание звука в помещении, когда за счет многократных отражений от стен, потолка и прочих поверхностей звук приобретает полноту и гулкость, а после прекращения звучания источника затухает не сразу, а постепенно. При этом время между последовательными отзвуками (примерно до 50 мс) ассоциируется с величиной помещения, а их интенсивность – с его гулкостью. По сути, ревербератор представляет собой частный случай фленжера с увеличенной задержкой между отзвуками основного сигнала, однако особенности слухового восприятия качественно различают эти два вида обработки.

- эхо (echo). Реверберация с еще более увеличенным временем задержки - выше 50 мс. При этом слух перестает субъективно воспринимать отражения, как призвуки основного сигнала, и начинает воспринимать их как повторения. Эхо обычно реализуется так же, как и естественное - с затуханием повторяющихся копий.
- - дистошн (distortion - искажение) - намеренное искажение формы звука, что придает ему резкий, скрежещущий оттенок. Наибольшее применение получил в качестве гитарного эффекта (классическая гитара heavy metal). Получается переусилением исходного сигнала до появления ограничений в усилителе (среза верхушек импульсов) и даже его самовозбуждения. Благодаря этому исходный сигнал становится похож на прямоугольный, отчего в нем появляется большое количество новых частотных составляющих, резко расширяющих спектр. Этот эффект применяется в различных вариациях (fuzz, overdrive и т.п.), различающихся способом ограничения сигнала (обычное или сглаженное, весь спектр или полоса частот, весь амплитудный диапазон или его часть и т.п.), соотношением исходного и искаженного сигналов в выходном, частотными характеристиками усилителей (наличие/отсутствие фильтров на выходе).
- компрессия - сжатие динамического диапазона сигнала, когда слабые звуки усиливаются сильнее, а сильные - слабее. На слух воспринимается как уменьшение разницы между тихим и громким звучанием исходного сигнала. Используется для последующей обработки методами, чувствительными к изменению амплитуды сигнала. В звукозаписи используется для снижения относительного уровня шума и предотвращения перегрузок. В качестве гитарной приставки позволяет значительно (на десятки секунд) продлить звучание струны без затухания громкости.
- фейзер (phase - фаза) - смешивание исходного сигнала с его копиями, сдвинутыми по фазе. По сути дела, это частный случай фленжера, но с намного более простой аналоговой реализацией (цифровая реализация одинакова). Изменение фазовых сдвигов суммируемых сигналов приводит к подавлению отдельных гармоник или частотных областей, как в многополосном фильтре. На слух такой эффект напоминает качание головки в стереомагнитофоне - физические процессы в обоих случаях примерно одинаковы.

Методы синтезирования звука

1.**Прямое воспроизведение**, например проигрывание wav-файлов.

2. **FM - синтез**.

Цифровой FM - синтез (Frequency Modulation) или частотный синтез проводится с помощью набора специальных генераторов. Генераторы, называемые также операторами, используют фазовую и амплитудную модуляцию. Фаза - частото тона. Амплитуда (огибающая) -громкость. Для воспроизведения голоса одного инструмента достаточно двух операторов, один задает основной тон, а второй - обертон.

Современные FM - синтезаторы, построенные например на основе набор микросхем OPL-3 фирмы Yamaha позволяют воспроизвести 20 стереофонических голосов, а для синтеза каждого голоса используется до 4-х генераторов, связанных между собой.

3. Самплерный (sample - выборка).

В этом методе записывается реальное звучание (сампл), которое затем в нужный момент воспроизводится. Для получения звуков разной высоты воспроизведение ускоряется или замедляется; чтобы тембр звука не менялся слишком сильно, используется несколько записей звучания через определенные интервалы (обычно - через одну-две октавы). В ранних самплерных синтезаторах звуки в буквальном

смысле записывались на магнитофон, в современных применяется цифровая запись звука.

Метод позволяет получить сколь угодно точное подобие звучания реального инструмента, однако для этого требуются достаточно большие объемы памяти. С другой стороны, запись звучит естественно только при тех же параметрах, при которых она была сделана - при попытке, например, придать ей другую амплитудную огибающую естественность резко падает. Для уменьшения требуемого объема памяти применяется зацикливание сампла (looping). В этом случае записывается только короткое время звучания инструмента, затем в нем выделяется средняя фаза с установившимся (sustained) звуком, которая при воспроизведении повторяется до тех пор, пока включена нота (нажата клавиша), а после отпускания воспроизводится концевая фаза.

4. **WT- синтез**

Таблично-волновой (wave table). Разновидность самплерного метода, когда записывается не все звучание целиком, а его отдельные фазы - атака, начальное затухание, средняя фаза и концевое затухание, что позволяет резко снизить объем памяти, требуемый для хранения самплов. Эти фазы записываются на различных частотах и при различных условиях (мягкий или резкий удар по клавише рояля, различное положение губ и языка при игре на саксофоне и т.п.), в результате чего получается семейство звучаний одного инструмента. При воспроизведении эти фазы нужным образом составляются, что дает возможность при относительно небольшом объеме самплов получить достаточно широкий спектр различных звучаний инструмента, а главное - заметно усилить выразительность звучания, выбирая, например, в рависимости от силы удара по клавише синтезатора не только нужную амплитудную огибающую, как делает любой синтезатор, но и нужную фазу атаки.

Основная проблема этого метода - в сложности сопряжения различных фаз друг с другом, чтобы переходы не воспринимались на слух и звучание было цельным и непрерывным. Поэтому синтезаторы этого класса достаточно редки и дороги.

5. **MIDI** (Musical Instrument Digital Interface) - стандарт передачи звуковых команд.

Это протокол передачи команд по стандартному интерфейсу. MIDI-сообщение содержит не запись музыки, как таковой, а ссылки на ноты. Данный протокол поддерживают многие электронные инструменты. Часто понятие WT- синтез заменяют термином MIDI синтез.

4.5. Форматы файлов представления звука

4.5.1. Форматы контейнеров RIFF

Контейнер - это формат файла, определяющий распределение аудио, видео, а в некоторых случаях и текстовой информации внутри него.

Типом контейнера в большинстве случаев не выбирается тип кодирования (сжатия) информации внутри файла. А сам тип контейнера легко определяется по расширению файла.

RIFF (*Resource Interchange File Format*) — один из форматов файлов-контейнеров для хранения потоковых мультимедиа-данных (видео, аудио, возможно текст).

Наиболее известными форматами, использующими RIFF в качестве контейнера являются: AVI (видео), WAV (аудио), RMI (MIDI-треки).

Основной концепцией RIFF-формата является **chunk**, порция данных с заголовком и сигнатурой, указывающей на содержимое chunk'a.
Формат chunk'a:
FOURCC ckID сигнатура chunk'a
DWORD ckSize размер данных chunk'a
BYTE] ckData сами данные chunk'a

Если chunk содержит нечётное количество байт, то после него добавляется один байт. Таким образом chunk'и всегда выравнены на границу в 2 байта.

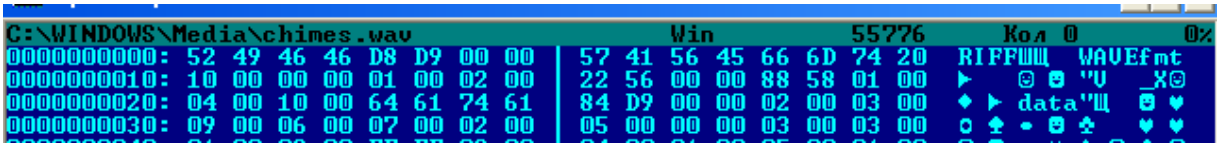
FOURCC (*Four Character Code*) — последовательность из четырёх символов, используемая для идентификации каких-либо данных.

4.5.2. Формат WAV файла.

WAV файл - это звуковой файл формата RIFF. WAV-файл состоит из трех блоков – двух заголовочных и одного блока звуковых данных. Первый блок имеет идентификатор "RIFF", за которым в 4-х байтах следует размер файла (без учета первых 8 байтов). В следующих 4-х байтах стоит идентификатор "WAVE", указывающий тип RIFF-файла. (первый заголовок 12 байт)

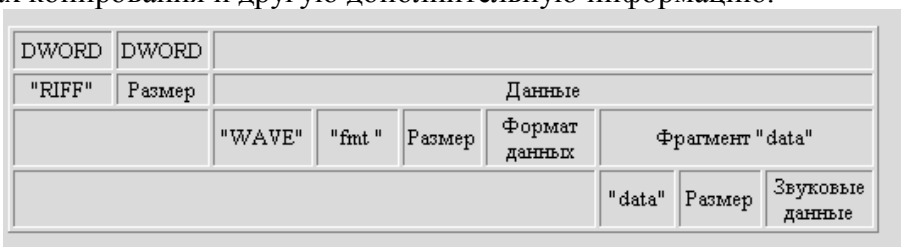
Следующий заголовочный блок содержит байты в следующем порядке:

4 байта – идентификатор "fmt_"
4 байта – число 16 + (размер данных блока)
2 байта – информация о кодеке (1 для PCM) – код сжатия
2 байта – число каналов (1 – моно, 2 – стерео)
4 байта – частота дискретизации (при 44100 Гц – AC44H)
4 байта – число байтов в секунду
2 байта – число байтов на один отсчет
2 байта – число битов на выборку B (если B не кратно 8, то выборки дополняются нулями до целого числа байтов).



Последний блок – данные отсчетов – начинается идентификатором "data", за которым расположены 4 байта – размер блока, а затем сами данные. В случае стереоотсчета данные для обоих каналов следуют друг за другом: сначала выборка для левого канала, затем выборка для правого, и т.д.

Файл может дополнительно содержать фрагменты других типов, поэтому не следует думать, что заголовок wav-файла имеет фиксированный формат. Например, в файле может присутствовать фрагмент "LIST" или "INFO", содержащий информацию о правах копирования и другую дополнительную информацию.



4.5.3. Формат контейнера AVI

Audio Video Interleave (сокращённо **AVI**; «чередование аудио и видео») — RIFF-медиаконтейнер, впервые использованный Microsoft в 1992 году в пакете Video for Windows.

Формат файлов с расширением AVI может содержать видео и аудио данные, сжатые с использованием разных комбинаций кодеков, что позволяет синхронно воспроизводить видео со звуком.

Обычно используется в сочетании с кодеками семейства MPEG4/DivX/Xvid и сжатым в mp3 звуком.

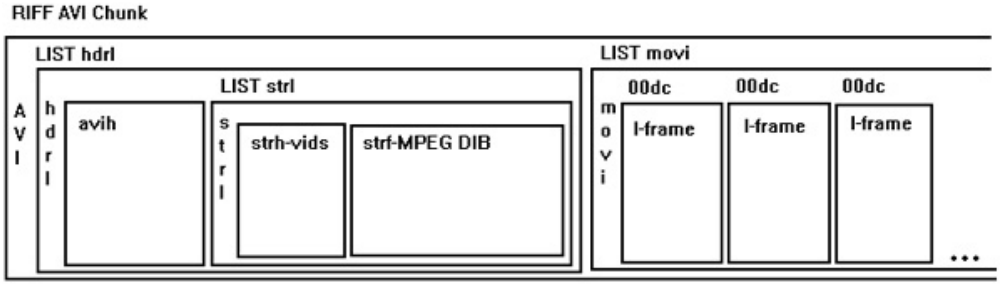
Все AVI файлы включают в себя два обязательных LIST chunk'а (фрагмент информации), которые определяют формат и данные потока. AVI файлы могут также включать индекс chunk. Этот дополнительный chunk определяет расположение видеоданных в файле. В соответствии с общей структурой RIFF-типа, AVI-файл должен иметь следующий вид:

RIFF 'AVI ' // четырехбуквенный идентификатор файла (в RIFF-формате)
LIST 'hdrl' // список заголовков блоков, определяющих форматы потоков
LIST 'movi' // блоки данных (потоков) AVI-файла
'idx1' // необязательный блок, определяющий размещение блоков данных внутри AVI-файла <AVI Index>

LIST hdrl

Типичная структура Chunk, содержащего заголовок AVI файла и заголовки потоков данных. :

LIST ('hrdl' 'avih' (<заголовок AVI файла>)
LIST ('strl' <заголовок потока 1>)
LIST ('strl' <заголовок потока 2>)
LIST('odml' <расширен. Загол. AVI файла>))



Основной заголовок файла

Файл начинается с основного заголовка. В AVI файлах, этот заголовок определяется chunk'ом с FOURCC 'avih'. Заголовок содержит **глобальную информацию для всего файла**, такую как число потоков в файле, ширина и высота видеопотока. Основной заголовок имеет следующую структуру:

typedef struct { DWORD dwMicroSecPerFrame; DWORD dwMaxBytesPerSec; DWORD dwReserved1; DWORD dwFlags; DWORD dwTotalFrames; DWORD dwInitialFrames; DWORD dwStreams; DWORD dwSuggestedBufferSize; DWORD dwWidth; DWORD dwHeight; DWORD dwReserved[4]; } MainAVIHeader;

dwMicroSecPerFrame — определяет количество микросекунд между кадрами. Это значение общее для всего файла.

dwMaxBytesPerSec — указывает примерную максимальную скорость передачи данных файла. Это значение указывает количество байт в секунду, которые система должна обрабатывать.

Список 'movi', в свою очередь, состоит из подблоков:

```
LIST 'movi'                          // блоки данных (потоков) AVI-файла
SubChunk | LIST 'rec '               // подблок | список записей
        '##wb' (размер блока 4 байта) (data) // звуковые данные (блок)
        '##dc' (размер блока 4 байта) (data) // видеоданные (блок)
        '##db' (размер блока 4 байта) (data) // видеоданные (блок)
```

Подблок данных организован в виде последовательности записей, каждая из которых состоит из одного кадра видео и соответствующего звукового сопровождения.

Первоначально ##dc-блок был предназначен для хранения сжатого изображения, а ##db-блок - для несжатого DIB (Device Independent Bitmap). Но фактически они оба могут содержать сжатые данные.

Вывод AVI файла с помощью OpenCV

```
#include “highgui.h”
int main() {
cvNamedWindow( “Example2”, CV_WINDOW_AUTOSIZE );
// Создаем окошко
CvCapture* capture = cvCreateFileCapture( “D:\\film.avi” );
// Открываем файл
IplImage* frame; // Здесь будет кадр
while(1) {
frame = cvQueryFrame( capture ); // Читаем кадр из файла
if( !frame ) break; // Если кадров больше нет - выходим
cvShowImage( “Example2”, frame ); // Выводим кадр
char c = cvWaitKey(33); // Ждем 33мс
if( c == 27 ) break; // Если нажали Esc - выходим    }
cvReleaseCapture( &capture ); // Закрываем файл
cvDestroyWindow( “Example2” ); // И окно
}
```

4.5.4. Контейнер MKV

Matroska (так по-английски пишется русское слово Матрёшка) — проект, нацеленный на создание **открытого** гибкого кросс-платформенного (включая аппаратные платформы) стандарта форматов мультимедийных контейнеров и набора инструментов и библиотек для работы с данными в этом формате. Возможности формата:

- трансляция по Интернету (протоколы HTTP и RTP);
- быстрая перемотка в файле;
- устойчивость к ошибкам;
- разбиение файла на главы (Chapters);

- переключаемые на лету субтитры (с возможностью добавлять свои шрифты прямо в контейнер);
- переключаемые звуковые дорожки;
- модульная расширяемость;

Это упаковка, которая может содержать **большое число потоков аудио, видео, субтитров** и других медиа-данных (картинки, шрифты и т.д.), позволяя пользователю хранить в одном файле **несколько вариантов фильма** и проигрывать его мультимедиа-проигрывателем в необходимой конфигурации.

Контейнер MPEG-4 (MP4)

Формат медиаконтейнера разработан группой MPEG. Предусматривает не только хранение аудио и видео, а ещё и **анимированного/интерактивного содержимого**.

Поддерживает аудио и видео кодеки из группы MPEG-4. : MPEG-1, MPEG-2, H.263, MPEG-4 ASP, H.264/MPEG-4 AVC. Поддерживаемые аудиокодеки: MPEG-1 Layers I, II, III (MP3), MPEG-2/4 (HE)-AAC, Vorbis, Apple Lossless.

Для проигрывания mp4 файлов необходимо установить в систему mp4-сплиттер.

Кодек

Кодек(codec) – общее понятие, которое включает в себя *encoder* и *decoder*. Примеры:

1. **MPEG-2**
2. **H.264**/**AVC** – более сложный стандарт сжатия видео, требующий большей вычислительной мощности от проигрывателя. Обеспечивает более высокую степень сжатия. По сравнению с MPEG-2 требует в среднем **в 2 раза меньше места** для сохранения видеоинформации при одинаковом качестве.
3. **VC-1**(**WMV9 Advanced Profile**, **WVC1**) – формат сжатия видеоданных созданный корпорацией Microsoft® на базе видео-кодека пакета Windows Media 9 Series. Так же как и **AVC**, в среднем в **2 раза более эффективен** по сравнению с MPEG-2.

4.5.5. Формат MP3

MP3 (MPEG Layer3) - формат цифрового кодирования звуковой информации **с потерями**, предназначен исключительно для кодирования звука. Имел предшественников в лице MP1 и MP2, отличатся высокой сложностью алгоритма, как следствие высокими требованиями к системным ресурсам. Лучший в своем классе по соотношению размер/ качество.

Bitrate - ширина потока. Количество бит, использующиеся для кодирования звукового потока. Измеряется в kbs, т.е. число килобит в секунду. CD-Audio имеет битрейт 1411,2 кбит/с

Помимо превалирующего режима **CBR** (Constant Bitrate — **постоянный битрейт**) (каждая секунда аудио кодируется одинаковым числом бит) существуют режимы **ABR** (Average Bitrate — **усреднённый битрейт**) и **VBR** (Variable Bitrate — **переменный битрейт**).

Квантование – процесс удаления частот, не воспринимаемых обычным человеческим слуховым аппаратом (ухом).

CD Audio - старейший формат цифрового звука, существует более 20 лет. Параметры 44Khz, 16-bit стерео.

Звуковая информация при кодировании разбивается на равные по продолжительности участки, которые называются **фреймами**. Все фреймы взаимно независимы. Каждый из этих фреймов кодируется отдельно со своими параметрами и имеет заголовок, в котором эти параметры описаны.

Сжатие фреймов

Для кодирования фреймов применяются математические алгоритмы сжатия. Качество при этом совершенно не страдает, но и размер уменьшается всего в четыре раза, bitrate ~ 320 кbs.

Выбрасываются звуки, которые считаются выходящими **за порог слышимости** человека (16kHz).

MP3 CODEC автоматически **выбрасывает** маломощные, неслышимые **частоты**.

Эффект маскирования. Слабый сигнал одного диапазона частот зачастую маскируется более мощным сигналом соседнего диапазона или мощным сигналом, предыдущего фрейма. Этот сильный сигнал вызывает временное понижение чувствительности уха к сигналу текущего фрейма «временного оглушения» bitrate 128 kbs (**сжатие 11:1**)

Режимы управления кодированием звуковых каналов MP3

1. **Стерео** — двухканальное кодирование, при котором каналы исходного стереосигнала кодируются независимо друг от друга, но распределение бит между каналами в общем битрейте может варьироваться в зависимости от сложности сигнала в каждом канале.
2. **Моно** — одноканальное кодирование. Различия между каналами полностью стёрты, так как два канала смешиваются в один, он кодируется и он же воспроизводится в обоих каналах стереосистемы.
3. **Двухканальное стерео** (*Dual Channel*) — два независимых канала, например звуковое сопровождение на разных языках. Битрейт делится на два канала. Например, если заданный битрейт 192 кбит/с, то для каждого канала он будет равен только 96 кбит/с.
4. **Объединённое стерео** (*Joint Stereo*, M/S Stereo) — (самый оптимальный способ двухканального кодирования). Например, в одном из режимов левый и правый каналы преобразуются в их сумму *(L+R)* и разность *(L−R)*. Для большинства звуковых файлов насыщенность канала с разностью *(L−R)* получается намного меньше канала с суммой *(L+R)*. Поэтому *объединённое стерео* позволяет либо сэкономить на битрейте канала разности *(L−R)*, либо улучшить ка́чество на том же битрейте, поскольку на канал суммы *(L+R)*отводится бо́льшая часть битрейта.

Этапы кодирования

1. Разбиение на фреймы.
2. Разложение в спектр.
3. Используются предыдущий и следующий фрейм.
4. Гармоники с меньшей амплитудой и гармоники, лежащие вблизи более интенсивных отсекаются.
5. Поскольку весь спектр актуален, высокочастотные гармоники не отсекаются, как в JPEG, а только выборочно удаляются, чтобы уменьшить поток информации за счёт разрежения спектра.
6. Математические методы сжатия.

Формат файла

1. Заголовок MP3 состоит из маркера, который служит для нахождения верного MP3-фрагмента. За ним следует бит, показывающий, что используется стандарт MPEG и два бита, показывающие использование layer 3; другими словами, это определяет **MPEG-1 Audio Layer 3** или MP3. Последующие значения могут варьироваться в зависимости от типа MP3-файла. Стандарт *ISO/IEC 11172-3* определяет диапазон значений для каждой секции заголовка, вместе с общей его спецификацией.
2. Блок данных MP3-файла содержит сжатую аудио информацию в виде частот и амплитуд.

Заголовок mp3-фрейма

Биты	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Пример, bin	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1	1	0	0	0	0	0	1	0	1	0	0	0	0
Пример, hex	F				F				F			B				7					0				5				0			
Описание	Маркер фрейма. Все биты должны быть установлены											Индекс версии MPEG (табл.2)		Индекс Layer (табл.3)		Бит защиты	Индекс битрейта (табл.4)				Индекс частоты (табл.5)		Бит смеще-ния		Индекс режима канала (табл.6)		Расширение режима (только для Joint stereo) (табл.7)		Копи-райт		Ори-гинал		Акцент (табл.8)	

Таблица 1

Битрейт

Значение	MPEG-1 Layer I	MPEG-1 Layer II	MPEG-1 Layer III	MPEG-2 Layer I и MPEG-2.5 Layer I	MPEG-2 Layer II, Layer II и MPEG-2.5 Layer II, Layer II
0 0 0 0	*не используется*				
0 0 0 1	32	32	32	32	8
0 0 1 0	64	48	40	48	16
0 0 1 1	96	56	48	56	24
0 1 0 0	128	64	56	64	32
0 1 0 1	160	80	64	80	40
0 1 1 0	192	96	80	96	48
0 1 1 1	224	112	96	112	56
1 0 0 0	256	128	112	128	64
1 0 0 1	288	160	128	144	80
1 0 1 0	320	192	160	160	96
1 0 1 1	352	224	192	176	112
1 1 0 0	384	256	224	192	128
1 1 0 1	416	320	256	224	144
1 1 1 0	448	384	320	256	160
1 1 1 1	*не используется*				

Таблица 4

MP3 – это MPEG-1 Layer III

В таблице хранятся значения битрейта в килобит/сек.

Заголовок mp3-фрейма

Биты	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Пример, bin	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1	1	0	0	0	0	0	1	0	1	0	0	0	0
Пример, hex	F				F				F			B				7					0				5				0			
Описание	Маркер фрейма. Все биты должны быть установлены											Индекс версии MPEG (табл.2)		Индекс Layer (табл.3)		Бит защиты	Индекс битрейта (табл.4)				Индекс частоты (табл.5)		Бит смеще-ния		Индекс режима канала (табл.6)		Расширение режима (только для Joint stereo) (табл.7)		Копи-райт		Ори-гинал		Акцент (табл.8)	

Таблица 1

[15] Бит защиты (Protection bit)

1 – нет защиты

0 – заголовок защищен 16-бит. CRC (следует за заголовком)

Частота дискретизации

Значение	MPEG-1	MPEG-2	MPEG-2.5
0 0	44100 Гц	22050 Гц	11025 Гц
0 1	48000 Гц	24000 Гц	12000 Гц
1 0	32000 Гц	16000 Гц	8000 Гц
1 1	*не используется*		

Таблица 5

Режим канала

Значение	Описание
0 0	Stereo
0 1	Joint stereo (Stereo)
1 0	Dual channel (2 моно канала)
1 1	Mono

Таблица 6

ID3 теги

ID3 (Identify a MP3) — формат метаданных, наиболее часто используемый в звуковых файлах в формате MP3. ID3 подпись **содержит данные о названии трека, альбома, имени исполнителя** и т. д., которые используются мультимедиапроигрывателями и другими программами, а также аппаратными проигрывателями, для отображения информации о файле и автоматического упорядочивания аудиоколлекции.

ID3v1 – имеет фиксированный размер в 128 байт, которые дописываются в конец mp3-файла. Там можно хранить: название трека, исполнитель, альбом, год, комментарий, номер трека (для версии 1.1) и жанр

Теги v2 имеют переменную длину и размещаются в начале файла, что позволяет поддерживать потоковое воспроизведение. (Формат ID3v2.4 позволяет так же хранить данные и в конце файла).
Данные ID3v2 состоят из заголовка и последующих фреймов ID3v2. Например, в версии ID3v2.3 существует более 70 типов фреймов.

Заголовок ID3v2

Байты	0	1	2	3	4	5	6	7	8	9
Пример, hex	49	44	33	03	00	00	00	00	07	76
Описание	Маркер 'ID3'			Версия	Суб-версия	Флаги	Длина ID3 в байтах			

4.5.6. Форматы Ogg Vorbis и WMA

Формат Ogg Vorbis. Файлы имеют расширение .ogg

Ogg Vorbis – это свободный, непатентованный стандарт сжатия аудиоданных с потерями.

OGG-файл – это **контейнер**, который может содержать аудиоданные различных типов. Так, часто в OGG-файлах используется кодек Vorbis – именно поэтому OGG Vorbis так называется. Для кодирования речи на низких битрейтах OGG может использовать кодек Speex, при кодирования без потерь в OGG-файлах могут храниться FLAC-данные.

Похож на MP3, однако используемые им алгоритмы отличаются. При сравнимом качестве Ogg-файлы занимают меньше места в сравнении с MP3.

Поддерживается практически всеми существующими операционными системами, теоретически, дает возможность на широкое распространение.

Ogg Vorbis имеет характеристики, которые приближают его к таким серьезным форматам, как *WMA* Pro или DTS. В частности, он поддерживает до 255 каналов.

Каждый канал может содержать аудиоданные с частотой дискретизации до 192 кГц и разрядностью до 32 бит.

Использует переменный битрейт, а максимум может доходить до 700 Кбит/с.

Ogg Vorbis имеет хорошо проработанную систему тегов, которая позволяет вставить в музыкальный файл практически любую дополнительную информацию.

Формат WMA. Файлы имеют расширение *.wma*

WMA, или Windows Media Audio – это закрытый стандарт, разработанный компанией Microsoft для хранения аудиоданных. На самом деле существует четыре различных стандарта, говоря о которых обычно употребляют наименование Windows Media Audio.

Наиболее часто под аббревиатурой *WMA* подразумевается кодек *WMA* Standard – изначально он разрабатывался как конкурент MP3. В результате немалых усилий со стороны Mirosoft *WMA* сегодня можно назвать одним из наиболее распространенных форматов после MP3.

Однако, у *WMA*, вследствие его закрытости, плохо с поддержкой не-Windows систем. Фактически, лишь пользователи Windows могут комфортно пользоваться *WMA*.

WMA способен создавать файлы меньшего размера, но практически такого же качества, как сопоставимого MP3.

4.6. Библиотеки программ для проигрывания звуковых файлов

1. Проигрывание Wave-файлов под MFC

PlaySound("name_of_file.wav", NULL, SND_FILENAME | SND_ASYNC);

wav-файл располагается там же, где и exe-файл программы

http://soundcoding.ru/WIN32/playsound_vc++.htm

2. **DirectSound** обеспечивает приложениям практически непосредственный доступ к аппаратуре звукового адаптера. DirectSound построен по объектно-ориентированному принципу в соответствии с моделью COM (Component Object Model - модель объектов-компонентов, или составных объектов) и **состоит из набора интерфейсов**. Каждый интерфейс отвечает за объект определенного типа -

устройство, буфер, службу уведомления и т.п. По сути, интерфейс представляет собой обычный набор управляющих функций, или методов, организованных в класс объектно-ориентированного языка.

http://www.codenet.ru/progr/directx/dxsound.php#002

3. **FMOD** - компания Firelight Technologies Pty www.FMOD.org.

Библиотека функций FMOD представляет собой реализацию API верхнего уровня, который включает широкий набор функций для работы со звуковыми файлами различных форматов, обработки звуковых данных и воспроизведения звука через аудиосистему

4.6.1. DirectSound

Программный интерфейс (API) в системе Windows для воспроизведения и записи звука. Входит в состав расширения DirectX.

Интерфейс DirectSound был разработан в середине 1990-х, главным образом для воспроизведения звуков.

Добавлены интерфейсы DirectSoundCapture, предназначенный для записи звука, и DirectSound3D, позволяющий работать с пространственными звуками.

DirectSound имеет объектно-ориентированную структуру, во многом похожую на COM (*Component Object Model*).

Интерфейс **сочетает в себе свойства как низкого уровня (приближённость непосредственно к аппаратуре), так и высокого (независимость от архитектуры конкретного устройства**).

Работая с DirectSound, программист описывает нужное ему количество источников звука, указывая свойства каждого из источников (вид звучания, его громкость, высота, пространственное положение, направление и скорость движения в DirectSound3D). Затем в любой момент любой источник может быть включён, при этом его звучание добавляется к звучанию остальных источников (звуки смешиваются). В любой момент могут быть изменены свойства источника, либо он может быть выключен.

DirectSound Объекты

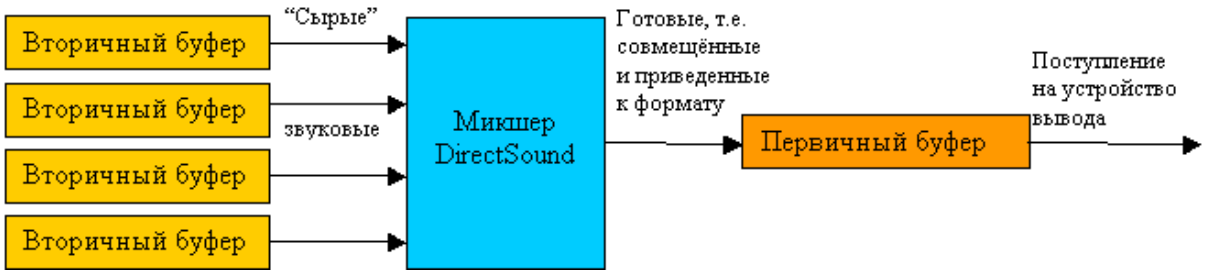
http://msdn.microsoft.com/ru-ru/library/ee416773(v=vs.85).aspx

Для воспроизведения в DirectSound используются устройства – источники звука, первичный буфер, вторичные буферы и микшер.

Вторичным буфером (secondary buffer) называется объект, содержащий звуковую информацию. В буфере может располагаться как вся информация, которую предполагается воспроизвести (в этом случае говорят, что он является **статическим** - static buffer), так и её часть (буфер является **потоковым** - streaming buffer). Информация в буфере должна быть записана в формате PCM (Pulse Code Modulation), это единственный формат, поддерживаемый DirectSound (для остальных форматов приходится использовать дополнительные средства, чтобы привести их к типу PCM).

Микшер DirectSound (DirectSound mixer) - механизм, отвечающий за совмещение информации, поступающей из проигрываемых вторичных буферов, в одно целое и отправке её в первичный буфер.

Первичный буфер (Primary buffer)- объект, содержащий готовую для передачи на устройство вывода звуковую информацию. Не может создаваться программистом и всегда присутствует в единственном экземпляре.



Вторичные буферы по мере воспроизведения микшируются и приводятся к формату первичного буфера.

DirectSound Объекты. Библиотека MSDN

Объект	Номер	Цель	Главный интерфейс
Устройство	Один в каждом приложении	Управляет устройством и создает звуковые буферы.	IDirectSound8
Вторичный буфер	Один для каждого звука	Управление статическими или потокового звука и воспроизводит его в первичный буфер.	IDirectSoundBuffer8, IDirectSound3DBuffer8, IDirectSoundNotify8
Первичный буфер	Один в каждом приложении	Миксы и играет звуки из вторичных буферов, и контролирует глобальные параметры 3D.	IDirectSoundBuffer, IDirectSound3DListener8
Эффект	Ноль или более для каждого вторичного буфера	Преобразование звука в вторичный буфер.	Интерфейс для конкретного эффекта, например, IDirectSoundFXChorus8

Общая схема взаимодействия

1. Приложение начинает работу с DirectSound, создавая **объект устройства** с интерфейсом IDirectSound - для воспроизведения звука или IDirectSoundCapture - для захвата (записи) звука. Объект устройства воспроизведения создается функцией **DirectSoundCreate**, объект устройства захвата - DirectSoundCaptureCreate.
2. Объекты вторичных звуковых буферов создаются при помощи метода **CreateSoundBuffer** - по одному для каждого источника звука; в этом же вызове задаются и форматы буферов.
3. Создав вторичный буфер, приложение должно заполнить его звуковыми данными. Процедуру занесения данных в буфер открывает метод **Lock**. Буфер заполняется данными, после чего вызывается метод **Unlock**, завершающий процедуру обновления данных.
4. Для запуска воспроизведения буфера вызывается метод **Play**, для остановки - **Stop**. Чтобы определить, какой фрагмент звучит в данный момент, используется метод GetCurrentPosition, для запуска звучания с определенного места - SetCurrentPosition.

Создание объекта (http://netlib.narod.ru/library/book0051/ch04_03.htm)

С объектом DirectSound ассоциируется звуковое устройство, используемое приложением, а также связываются все вторичные буферы, созданные с участием объекта DirectSound. Поэтому сразу после его удаления эти буферы уничтожаются.

Создаётся объект DirectSound вызовом функции DirectSoundCreate8, которой в качестве **первого параметра** передаётся указатель на идентификатор требуемого звукового устройства либо NULL **для использования текущего устройства**, либо одна из следующих констант:

DSDEVID_DefaultPlayback - используется аудиоустройство воспроизведения по умолчанию

DSDEVID_DefaultVoicePlayback - используется устройство для работы с голосом по умолчанию

Второй параметр функции - адрес указателя на интерфейс IDirectSound8, а **третий** - всегда NULL. DirectSoundCreate8 возвращает ошибку в случае, если нет звукового устройства.

```
LPDIRECTSOUND8 lpds;
HRESULT hr = DirectSoundCreate8(NULL, &lpds, NULL));
```

Создание вторичных буферов

Создание вторичных буферов выполняется методом **IDirectSound8::CreateSoundBuffer**, которому в качестве **первого параметра** передаётся структура DSBUFFERDESC, описывающая характеристики буфера, затем - **адрес указателя на интерфейс** IDirectSoundBuffer и всегда NULL в качестве третьего параметра.

```
// Create buffer.
 HRESULT CreateSoundBuffer(LPCDSBUFFERDESC pcDSBufferDesc,
LPDIRECTSOUNDBUFFER * ppDSBuffer, NULL)

typedef struct DSBUFFERDESC {
DWORD dwSize;
DWORD dwFlags;
DWORD dwBufferBytes;
DWORD dwReserved;
LPWAVEFORMATEX lpwfxFormat; GUID guid3DAlgorithm;
} DSBUFFERDESC;
```

Загрузка звука из файла

Прежде чем что-то проигрывать надо это что-то загрузить. Сам процесс загрузки подразумевает несколько действий. А так как мы работаем со статическими буферами (т.е. буферами, содержащими всё, что предполагается воспроизвести), то выполняется он следующим образом:

1. Заблокировать весь буфер, вызвав метод **IDirectSoundBuffer::Lock.**
2. Скопировать звуковые данные по полученному адресу.
3. Разблокировать буфер, вызвав метод **IDirectSoundBuffer::Unlock**

```
pSoundBuffer2->Lock(0,0,&pDst2,&dwSize2,0,0,DSBLOCK_ENTIREBUFFER);
memcpy(pDst2,pData,dwSize2);
pSoundBuffer->Unlock(pDst2,dwSize2,0,0);
```

Воспроизведение

После того, как буфер содержит необходимые звуковые данные, можно переходить к воспроизведению. Делается это методом **IDirectSoundBuffer::Play.**

Воспроизведение ведётся с **курсора воспроизведения**.

Если во время вызова IDirectSoundBuffer::Play звук уже проигрывается, то воспроизведение продолжается в соответствии с новыми параметрами.

Аналогично, если вызвать метод IDirectSoundBuffer::**SetCurrentPosition** во время проигрывания, оно тотчас же продолжится, но уже с новой позиции.

Воспроизведение останавливается методом IDirectSoundBuffer::**Stop**, прекращающим звучание и устанавливающим курсор воспроизведения на сэмпл, следующий за последним проигранным сэмплом.

4.6.2. Библиотека FMOD

Библиотека функций FMOD представляет собой реализацию API верхнего уровня, который включает широкий набор функций для работы со звуковыми файлами различных форматов, обработки звуковых данных и воспроизведения звука через аудиосистему компьютера.

Интерес представляют функции для работы с объемным (3D) звуком .

Поставляется в двух версиях — 3.75, и 4.x и поддерживает большинство форматов и платформ.

Поддерживаемые платформы:

Win32, Win64, Linux 32-bit, 64-bit, Mac OS X, iPhone, Android

(http://www.tiflocomp.ru/games/design/sound_games/fmod_rec.php)

Процесс записи звука состоит из следующих этапов:

1. Инициализация библиотеки FMOD;
2. Выделение памяти для создания "пустого" звукового образца;
3. Непосредственная запись звуковых данных с устройства записи;
4. Воспроизведение записанных данных;
5. Сохранение данных на диск.

FMOD 3. Простой пример

```
#include <conio.h>
#include "inc/fmod.h"
#pragma comment(lib, "fmodvc.lib")
FSOUND_SAMPLE* handle;
int main () {
FSOUND_Init (44100, 32, 0);
// Loads and decodes a static soundfile into memory.
handle=FSOUND_Sample_Load (0,"sample.mp3",0, 0, 0);
FSOUND_PlaySound (0,handle);
// wait until the users hits a key to end the app
while (!_kbhit()) { }
// Removes a sample from memory and makes its slot available again.
FSOUND_Sample_Free (handle);
FSOUND_Close(); }
```

Инициализация FMOD

По умолчанию FMOD самостоятельно определяет текущую звуковую систему и в качестве устройств ввода/вывода звука (**FSOUND_Init**)

Если ввод через другое устройство, то следует воспользоваться функцией :

int FSOUND_Record_SetDriver (int driver /* номер устройства */);

Функция в качестве параметра получает номер устройства в системе (значение 0 соответствует устройству по умолчанию), а возвращает TRUE если выбор устройства прошел без ошибок и FALSE в противном случае.

Получить количество установленных в системе устройств: **FSOUND_Record_GetNumDrivers ().** Эта функция не имеет параметров и возвращает число устройств, поддерживающих запись звука.

Узнать наименование этих устройств можно, вызвав функцию **FSOUND_Record_GetDriverName (),** которой в качестве параметра передается номер устройства, а возвращает она указатель на символьный массив, содержащий название устройства.

Пример вывода списка устройств в консольном приложении:

int num_dev = FSOUND_Record_GetNumDrivers (); char buf [256]; // промежуточный буфер для конвертации ANSI в OEM

for (int i=0; i< num_dev; i++)

{ CharToOem (FSOUND_Record_GetDriverName (i), buf); puts (buf); }

Запись звука в память

Создание звукового образца, которое можно условно назвать "резервированием места, куда будут записаны введенные звуковые данные", осуществляется функцией **FSOUND_Sample_Alloc ():**

FSOUND_SAMPLE * FSOUND_Sample_Alloc (int index, /* индекс звукового **пула** */ int length, /* размер в отсчетах (сэмплах) */ unsigned int mode, /* параметры звукового образца */ int deffreq, /* частота дискретизации в герцах */ int defvol, /* громкость (0 - 255) */ int defpan, /* соотношение каналов (правый - левый (0 - 255)*/ int defpri /* приоритет (0 - 255)*/);

В качестве индекса звукового пула обычно используются константы FSOUND_FREE или FSOUND_UNMANAGED. Первая означает, что управление пулом берет на себя FMOD, а вторая, что управление данным звуковым образцом ложится на программиста (в частности он должен самостоятельно освобождать память, отводимую под звуковой образец, для чего напрямую вызывать функцию **FSOUND_Sample_Free ()).**

После того как создан звуковой образец, его можно заполнять звуковыми данными **FSOUND_Record_StartSample ():**

int FSOUND_Record_StartSample (FSOUND_SAMPLE *samp, /* указатель на звуковой образец */ signed char loop /* режим цикл. записи */);

В качестве первого параметра функция получает указатель на уже созданный "пустой" звуковой образец. В качестве второго параметра передается значение TRUE, если нужно записывать звук в циклическом режиме (когда достигается конец буфера, то запись продолжается с его начала), или FALSE, когда циклический режим записи поддерживать не требуется. Функция возвращает TRUE, если процесс записи стартовал успешно, и FALSE в противном случае.

Остановить запись можно при помощи функции **FSOUND_Record_Stop().**

Получить номер текущего отсчета (сэмпла) в буфере звукового образца можно при помощи функции **FSOUND_Record_GetPosition ().** Функция возвращает номер

отсчета в буфере звукового образца, куда происходит запись звука. В случае неудачи функция возвращает значение -1 (минус один).

```
// инициализация FMOD
if (!FSOUND_Init (22050, 32, 0)) { // обработка ошибки ... }
// Параметры звукового образца: стерео, 16 бит, 22050 герц
// Длительность: не более 10 секунд
int rec_rate = 22050; // 22050 герц
int = rec_len = rec_rate * 10; // 10 секундный буфер
FSOUND_SAMPLE *samp1 = FSOUND_Sample_Alloc(FSOUND_UNMANAGED, rec_len, FSOUND_STEREO | FSOUND_16BITS , rec_rate, 255, 128, 255); if (!samp1) { // обработка ошибки ... };
// Начинаем запись
if (!FSOUND_Record_StartSample(samp1, FALSE)) { //обработка ошибки ... }
 //Крутимся в цикле, пока указатель текущей позиции  записи не достигнет конца буфера  или пока не будет нажата клавиша
do { Sleep(50); } while (FSOUND_Record_GetPosition() < rec_len && !kbhit());
 // Останавливаем запись
FSOUND_Record_Stop();
```

Прослушивание записанного звука

В результате выполнения вышеприведенного фрагмента кода получаем образец звука samp1, буфер которого заполнен записанными данными. Для того чтобы прослушать их, нужно воспользоваться обычными для FMOD действиями для прослушивания звука:

```
// Создаем звуковой канал и воспроизводим звук int channel;
 channel = FSOUND_PlaySound(FSOUND_FREE, samp1);
```

Сохранение записанных данных на диск

FMOD не содержит встроенных функций, поддерживающих запись на диск файлов в формате wav. Если вы желаете сохранить записанные звуковые данные в этом формате, то вам придется самостоятельно сформировать wav-заголовок файла и сохранить файл обычными средствами работы с файловой системой.

Если записанные звуковые данные предполагается использовать только внутри вашего приложения или игры, то можно сохранить звуковые данные в "чистом" виде (raw) ,без заголовков.

Для доступа к буферу звукового образца используются функции

FSOUND_Sample_Unlock ()

FSOUND_Sample_Lock ().

При помощи этих функций можно получить непосредственный доступ к звуковым отсчетам (сэмплам) и сохранить их на диске стандартными средствами для работы с файлами (функции fopen (), fwrite (), fclose ()).

Пример:

```
// Открываем файл для записи
FILE *f = fopen ("recdata.dat", "wb");
// Получаем доступ к звуковым данным в буфере звукового образца. Буфер имеет кольцевую структуру, поэтому задается двумя указателями.
void *ptr1, *ptr2; unsigned int len1, len2;
int length = rec_len * 4; // в байтах, сэмпл = 4 байта (стерео, 16 бит)
 FSOUND_Sample_Lock(samp1, 0, length, &ptr1, &ptr2, &len1, &len2);
```

```
 fwrite(ptr1, len1, 1, f);
if (len2) fwrite(ptr2, len2, 1, f);
 FSOUND_Sample_Unlock(samp1, ptr1, ptr2, len1, len2);
fclose(f);
```

Теперь записанный на диск файл можно воспроизвести, создав для него другой звуковой образец

Воспроизведение

```
// Загружаем файл в буфер звукового образца
FSOUND_SAMPLE *samp2 = FSOUND_Sample_Load (FSOUND_FREE,
"recdata.dat", FSOUND_LOADRAW|FSOUND_STEREO | FSOUND_16BITS, 0,0);
if (!samp2)
{ // Обработка ошибки ... }
// Устанавливаем параметры звука: частота, громкость, стереобаланс, приоритет
FSOUND_Sample_SetDefaults (samp2, rec_rate, 255, 128, 255);
// Воспроизводим звук
int channel2 = FSOUND_PlaySound(FSOUND_FREE, samp2);
```

Некоторые функции для работы со звуком

- FSOUND_GetVersion() - возвращает версию библиотеки FMOD, установленной на компьютере. Возвращаемое значение следует сравнить с константой FMOD_VERSION, которая хранит номер версии FMOD, для которой была скомпилирована программа.
- FSOUND_SetOutput () / FSOUND_GetOutput () - выбрать/получить базовую звуковую систему (Windows Multimedia, DirectSound, A3D и т.п.).
- FSOUND_SetDriver () / FSOUND_GetDriver () - выбрать/получить номер устройства вывода (звуковой карты).
- FSOUND_SetMixer () /FSOUND_GetMixer () - выбрать/получить тип цифрового микшера.
- FSOUND_Init() - инициализирует звуковую систему FMOD.
- FSOUND_Sample_Load () - загружает в память и декодирует звуковой файл (поддерживаются .wav, .mp2, .mp3, .ogg, .raw и др.).
- FSOUND_PlaySoundEx () - проигрывает звуковой файл, загруженный в память, через звуковой канал.
- FSOUND_SetPaused () - приостанавливает / возобновляет воспроизведение звука в канале.

FMOD 4 . www.FMOD.org. Version 4.44.11 Built on Mar 26, 2013

Пример 2

Нужны fmodex.dll и папки inc и lib

```
#include <fmod.hpp>
FMOD::System * system;
FMOD::System_Create(&system);
system->init(32 /*maximum number of channels*/, FMOD_INIT_NORMAL, 0);
FMOD::Sound * sound; // sound
FMOD::Channel * channel; // sound channel
system->system->createSound("mysound.mp3", FMOD_SOFTWARE |
FMOD_LOOP_OFF, 0, &sound); // creating sound
system->playSound(1 /*channel #1*/, sound, true /*start paused*/, &channel); // playing
sound (assigning it to a channel)
```

channel->setPaused(false); // actually play sound

4.7. 3d звук

Реализация пространственного звучания (3D звука) в том или ином виде, применительно к компьютерной технике, используется для придания естественности звуку в компьютерных играх или фильмах, для создания полного ощущения погружения в процесс игры или просмотра фильма.

Если произвести специальную обработку звукового потока с учетом максимального числа особенностей восприятия звука слуховым аппаратом, то, возможно, удастся имитировать пространственное звучание даже с использованием всего двух источников (колонок или наушников). Необходимо подчеркнуть, что любой алгоритм создания 3D звука реализовывается с помощью алгоритмов фильтрации (оперирующих с амплитудой и частотой звукового сигнала) той или иной сложности, которые определенным образом "обманывают" слуховой аппарат, "заставляя его считать", что то, что он слышит, расположено в трехмерном пространстве вокруг слушателя.

Можно обозначить три основных способа реализации пространственного звучания:

- расширение стерео базы (Stereo Expansion) - специальная обработка уже имеющегося стерео сигнала и, таким образом, расширение кажущегося звукового поля (имитация расширения расстояния между источниками);
- позиционирование звучания (Positional 3D Audio) - оперирование с множеством отдельных звуковых потоков и расположение каждого из них в пространстве вокруг слушателя;
- виртуальный (мнимый) окружающий звук (Virtual Surround Sound) - использование определенного числа звуковых потоков с целью воспроизведения истинного звучания с помощью ограниченного числа физических источников звука.

Модель пространства:

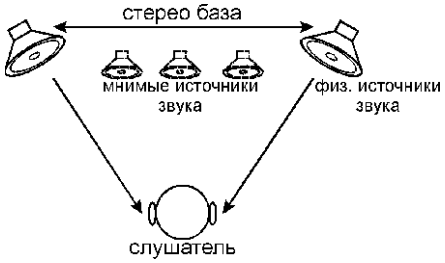
Используется левосторонняя декартова система координат, состоящая из трех ортогональных координатных осей. Ось X направлена вправо; ось Y направлена вверх; ось Z направлена вперед (то есть в монитор, если сидеть лицом к нему).

Расстояние измеряется в метрах. Координаты могут принимать как положительные, так и отрицательные значения.

Координатную триаду XYZ, характеризующую положение точки в пространстве, можно рассматривать как вектор, начало которого находится в начале отсчета, то есть в точке с координатами (0, 0, 0), и конечной точкой с координатами (X, Y, Z).

Кроме векторов положения, в FMOD используются векторы скорости, необходимые для вычисления доплеровского смещения в спектре звука движущихся источников. Вектор скорости задается тремя координатами своей конечной точки (начальной точкой вектора скорости считается точка (0, 0, 0)).

FMOD (DirectSound3D) рассчитывает статическую звуковую картину, которую программист может сделать динамичной для слушателя, меняя положение источников звука.



http://websound.ru/articles/technologies/3dsound.htm

Модель источника и слушателя

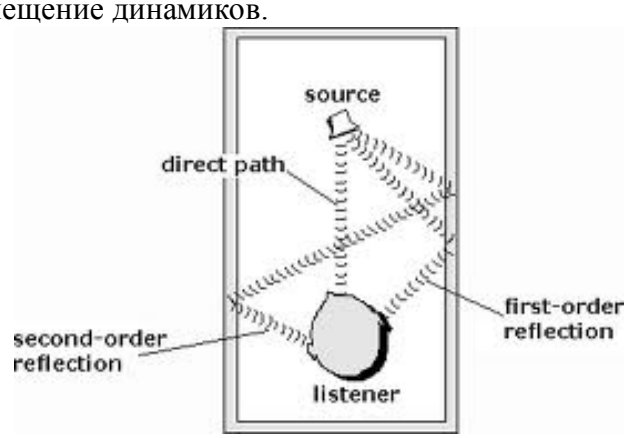
3D источник звука является каналом, который имеет положение и скорость. Когда канал 3D играет, его громкость, расположение колонок и шаг будут зависеть автоматически от положения слушателя.

Слушатель имеет положение, скорость как у источника звука, но он также имеет ориентацию.

Громкость определяется расстоянием между слушателем и источником.

Скорость перемещения источника относительно слушателя определяется эффектом Доплера.

Ориентации слушателя к источнику определяет панорамирование или размещение динамиков.

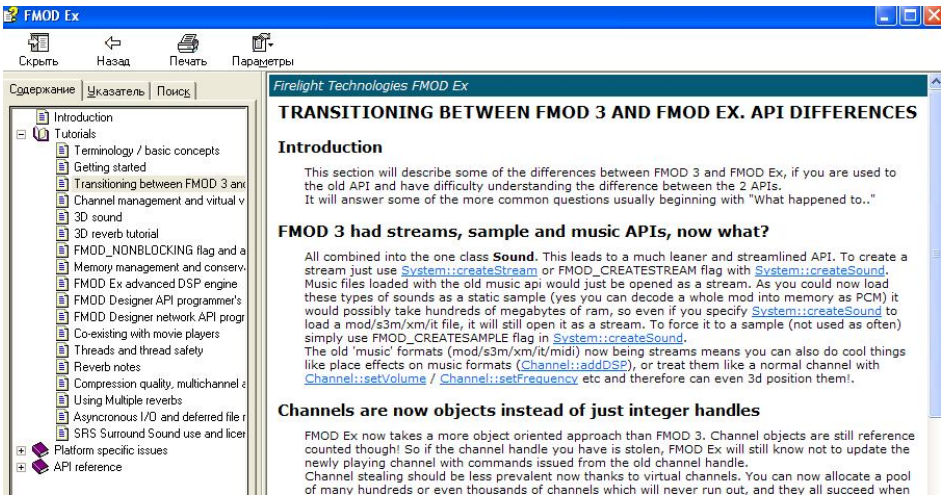


Некоторые функции для работы с 3d звуком

- FSOUND_3D_SetDistanceFactor () - позволяет установить единицы измерения длин, отличные от метров.
- FSOUND_3D_SetDopplerFactor () - позволяет установить доплеровское смещение. Базовое значение (1.0) соответствует скорости звука 340 м/с.
- FSOUND_3D_SetRolloffFactor () - позволяет установить уровень потерь энергии звуковой волны (затухания).
- FSOUND_3D_SetAttributes () / FSOUND_3D_GetAttributes () - установить/получить вектор положения и вектор скорости источника звука.
- FSOUND_3D_Listener_SetAttributes () / FSOUND_3D_Listener_GetAttributes () - установить/ получить вектор положения, вектор скорости и векторы ориентации слушателя.
- FSOUND_3D_SetMinMaxDistance () / FSOUND_3D_GetMinMaxDistance () установить / получить минимальное и максимальное расстояние слышимости источника звука. Минимальное расстояние от источника звука до слушателя - при уменьшении которого громкость звука больше не возрастает, а остается на том значении, которого она достигла на минимальном расстоянии. Устанавливая

разные минимальные расстояния, например, для самолета и шмеля, можно сделать их одинаково заметными на слух, несмотря на то, что гул мотора будет восприниматься как более мощный звук. Максимальным называется такое расстояние от источника звука до слушателя, начиная с которого громкость звука больше не уменьшается, а остается на уровне, который она достигла на максимальном расстоянии. Это означает, что как бы далеко не находился источник звука, он будет слышен.

Аналогичные функции есть и в библиотеке FMOD:



Часть 5. Фотоприемники и видеосистемы

5.1.Фотоприемники на приборах с зарядовой связью

В 1970 г. сотрудники фирмы Bell Laboratories У. Бойл и Дж. Смит в поисках электрического аналога схем на цилиндрических магнитных доменах предложили - и продемонстрировали экспериментально - принцип зарядовой связи. Самый первый прибор с зарядовой связью (ПЗС) представлял собой аналоговый регистр сдвига на 8 элементов, изготовленный по р-МОП технологии с молибденовыми затворами, а вскоре появились и двумерные матрицы. Очень быстро стало ясно, что присущее ПЗС свойство самосканирования (об этом чуть дальше) устраняет необходимость в регистрах сдвига, создававших столько проблем в предшествующих типах датчиков.

Дальнейший рывок в технологии и параметрах ПЗС был связан с появлением скрытого канала переноса и применением прозрачных электродов из поликристаллического кремния, что резко повысило чувствительность приборов. Уже в середине 70-х появились первые коммерческие матрицы производства фирм Fairchild, Bell и RCA в США и Philips в Европе, совместимые с ТВ стандартом (т. е. имеющие разрешение по вертикали 476 или 576 строк- соответственно для американского или европейского стандартов разложения, и по меньшей мере 350 элементов разложения по горизонтали). Вскоре в Японии было налажено массовое производство недорогих ПЗС приемлемого качества для бытовой электроники - и на смену кинокамерам в массовом порядке пришли видеокамеры. Сегодня массовое производство ПЗС линеек и матриц осуществляется многими фирмами: «Sony», «Texas Instruments», «Sharp», «Samsung», «Hitachi», «Toshiba», «Kodak» и др. В России ПЗС-матрицы выпускаются в ЦНИИ «Электрон», (Санкт-Петербург) и в НПО «Пульсар».

Появление и совершенствование ПЗС совпали по времени с существенным скачком в развитии цифровой электроники, и, прежде всего, с разработкой микропроцессоров. Единство кремниевой технологии обеспечило естественность построения телевизионных систем на ПЗС, в которых большое количество параметров регулируется с помощью микропроцессорных устройств. В последнее время развитие получили матричные фотоприёмники с координатной адресацией (КМОП фотоприемники).

Физические принципы работы ПЗС

На рис. 5.1 показана структура одного элемента линейного трехфазного ПЗС в режиме накопления. Структура состоит из слоя кремния р-типа (подложка), изолирующего слоя (диоксид кремния) и набора пластин - электродов.

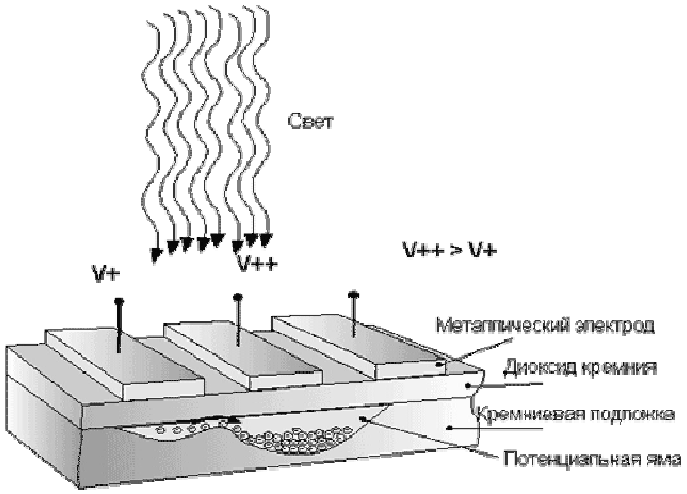


Рис. 5.1. Структура элемента ПЗС

Если подать небольшой положительный потенциал на один из электродов ячейки трехфазного ПЗС, а два других электрода оставить под нулевым потенциалом относительно подложки, то под положительно смещенным электродом образуется потенциальная яма. Когда кремнием поглощается фотон, то генерируется пара носителей заряда - электрон и дырка. Электростатическое поле в области элемента вытесняет "дырку" в глубь кремния, а электроны накапливаются в потенциальной яме под электродом, к которому подведен положительный потенциал. Здесь они могут храниться достаточно длительное время, поскольку дырки в обедненной области отсутствуют и электроны не рекомбинируют. Заряд, накопленный под одним электродом, в любой момент может быть перенесен под соседний электрод, если его потенциал будет увеличен, в то время как потенциал первого электрода будет уменьшен (рис. 5.1). Если все три электрода элемента ПЗС находятся под нулевым потенциалом, то накопление генерированных светом электронов не происходит.

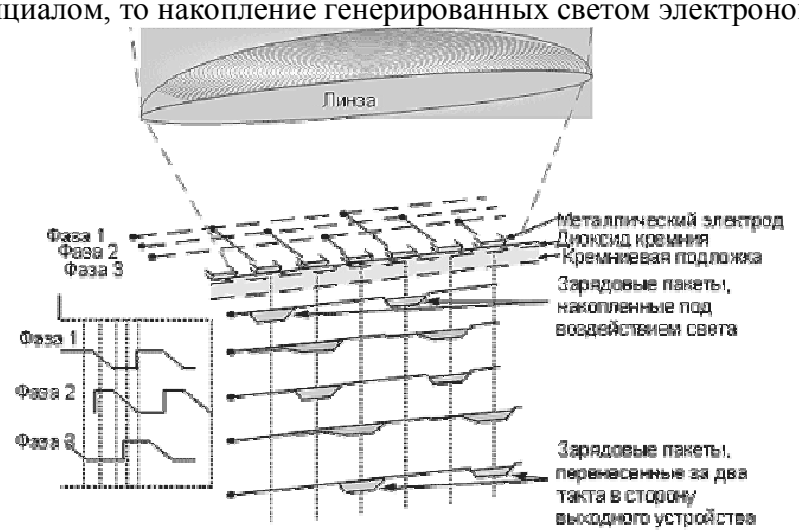


Рис. 5.2. Перенос зарядов в приборе с зарядовой связью

Накопление. Заряды в потенциальную яму могут попасть тремя способами:

1. Термогенерация - паразитическая генерация сигналов в подложке. С течением времени устанавливается равновесие зарядов, и яма полностью заполняется электронами. ПЗС - принципиально динамический прибор. В зависимости от типа прибора и температуры подложки, яма может заполнится за 1с или даже за несколько

десятков часов. Темновой ток в полупроводниковых фотоприёмниках экспоненциально зависит от температуры

2. Электрический ввод зарядовых пакетов. Для этого используется p-n переход.

3. Фотогенерация. Фотоны при попадании на ПЗС рождают пары e - дырка и не основные носители будут накапливаться в потенциальной яме. Накопленный заряд будет пропорционален количеству падающих фотонов, и следовательно освещенный. Около 1% фотонов рождают пару e - дырка.

Заряд, который может удерживаться в яме, определяется зарядовой емкостью ячейки (Q_{max}); $Q_{max}=C_0 S \mid \Delta U_{нов} \mid$, где C_0 - удельная ёмкость; S - площадь затвора; $\Delta U_{нов}$ - поверхност. потенциал. В потенциальную яму помещается 10^5 - 10^7 дырок или электронов.

Перенос в трехфазном ПЗС можно выполнить в одном из двух направлениях - влево или вправо, причем все зарядовые пакеты линейки элементов будут одновременно переноситься в одну и ту же сторону.

Двухмерный массив (матрицу) элементов получают с помощью стоп-каналов, разделяющих электродную структуру ПЗС на столбцы. Стоп- каналы - это узкие области, которые формируются специальными технологическими приемами в приповерхностной области и препятствуют растеканию заряда под соседние столбцы.

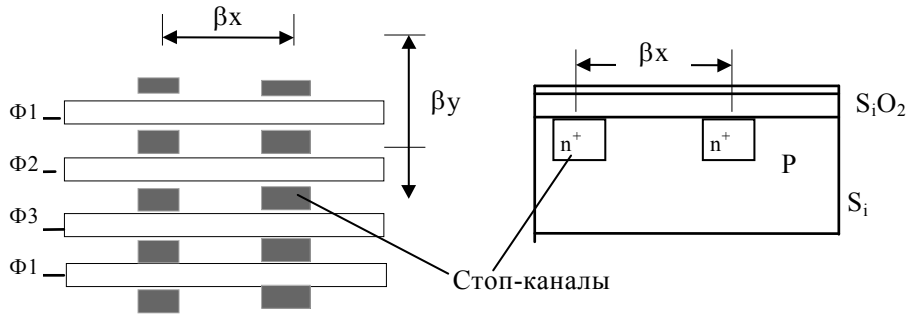


Рис. 5.3. Стоп каналы в ПЗС

Перенос. Накопленные в ПЗС - ячейках заряды необходимо считать, для чего перенести к выходному устройству. Способ последовательного переноса зарядов по ПЗС-ячейкам отличает эти фотоприёмники от других.

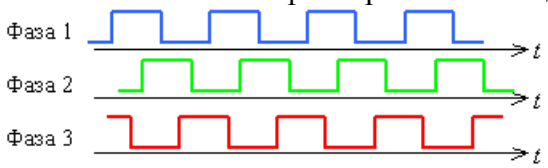


Рис. 5.4. Тактовые диаграммы управления трёхфазным регистром

Тактовые диаграммы работы ПЗС регистра показаны на рис. 5.4. Видно, что для его нормальной работы в каждый момент времени, по крайней мере, на одной тактовой шине должен присутствовать высокий потенциал, и по крайней мере на одной - низкий потенциал (потенциал барьера). При повышении потенциала на одной шине и понижении его на другой (предыдущей) происходит одновременная передача всех зарядовых пакетов под соседние затворы, и за полный цикл (один такт на каждой фазной шине) происходит передача (сдвиг) зарядовых пакетов на один элемент регистра. Любой заряд, накопленный от любого источника, перемещается из ямы в яму. Разделение индивидуальных зарядовых пакетов поддерживается в ходе этого процесса без смешивания зарядов между ямами.

Понятно, что на полную передачу заряда из одной ямы в другую требуется время, так что при высокой тактовой частоте (а для ТВ стандарта она составляет в регистре

считывания 7-13 МГц в зависимости от числа элементов по горизонтали) этого времени может и не хватить. Величина, показывающая, какая часть зарядового пакета передалась в следующий элемент ПЗС, называется эффективностью переноса η. Часто пользуются и связанной с ней величиной неэффективности ε = 1-η. Однако частотные ограничения - это ещё полбеды. Беда же в том, что для структуры ПЗС, обсуждавшейся до сих пор, все события происходят в очень тонкой (десятки ангстрем) области у границы раздела окисел-кремний. Сколь бы не была совершенной кристаллическая структура подложки, граница раздела - нарушение однородности кристалла, а из физики твёрдого тела известно, что всякое нарушение однородности кристаллической решётки приводит к возникновению разрешённых энергетических уровней в запрещённой зоне. Ясно, что такое нарушение, как граница раздела, даром не проходит, и образующихся при этом энергетических уровней столько, что они образуют квазинепрерывный спектр, а значит, среди них есть такие, которые способны захватывать электроны из зоны проводимости (ловушки), причём время, через которое захваченный электрон вернётся обратно в зону проводимости, зависит от энергии ловушки (и абсолютной температуры). И получается, что, пока над данной точкой границы раздела нет заряда (а это когда-нибудь да так), часть ловушек освобождается, эмитируя электрон обратно в зону проводимости, а когда придёт очередной зарядовый пакет - мгновенно заполняется, чтобы снова освободить захваченные электроны после того, как этот зарядовый пакет ушёл, так что освобождённые электроны попадают в другой, пришедший позднее, зарядовый пакет.

Более того, эмиссия электронов с ловушек обратно в зону проводимости, как всякий тепловой процесс, подвержена термодинамической флуктуации и привносит в распределение зарядов по ячейкам шум переноса. Кроме того, часть электронов, попавшая на глубокий уровень с длительным временем эмиссии, может вовсе не вернуться (это называется фиксированными потерями, и особенно заметно при переносе малых зарядовых пакетов). И наконец, через квазинепрерывный спектр ловушек происходит интенсивная генерация темнового тока (тепловой процесс спонтанного образования электронно-дырочных пар - к сожалению, процесс неизбежный при температуре, отличной от абсолютного нуля, а наличие уровней в запрещённой зоне резко повышает его вероятность).

Все эти неприятности, связанные с поверхностным каналом переноса, удалось полностью (или почти полностью) устранить инженерам фирмы Philips, в 1972 году предложившим ПЗС со скрытым каналом. Это решение, разом убивавшее несколько зайцев, оказалось настолько удачным, что с тех пор все ПЗС выпускаются только со скрытым каналом. От обычного он отличается тем, что в поверхностной области кремния создаётся тонкий (порядка 0,3 - 0,5 мкм) слой с проводимостью противоположного подложке типа и с концентрацией примеси такой, чтобы он мог полностью обедняться при подаче на него напряжения через соответствующий контакт.

Степень совершенства кристаллической решётки в современных материалах весьма высока, и ныне эффективность переноса в ПЗС со скрытым каналом (собственно, далее речь будет идти только о них) достигает в лучших приборах потрясающих величин 99,9999% (или $\varepsilon = 10^{-6}$) на перенос, т. е. после тысячи переносов искажения от неэффективности составляют 0,1%. Достигается это не только из-за крайне низкой плотности ловушек в объёме полупроводника, но и из-за того, что перенос происходит на некотором удалении от затворов, а значит, становятся заметными двумерные эффекты - электрическое поле одного затвора проникает под соседний, создавая тем самым дрейфовую составляющую переноса (тянущее поле), что вытягивает заряд гораздо быстрее, чем просто тепловая диффузия, так что частотные

ограничения эффективности в диапазоне частот, характерном для телевизионных матриц, практически незаметны. Отметим ещё одно отличие ПЗС со срытым каналом от ПЗС с поверхностным каналом: уровни управляющих напряжений для них биполярные,

Структуры ПЗС фотоприемников

Имеются линейные и матричные ПЗС фотоприемники. Рассмотрим матричные фотоприемники. Большинство типов ПЗС-матриц массового применения состоят из двух областей - области накопления и области хранения. Различают три топологии:

- с кадровым переносом (КП) - FT (Frame Transfer);
- со строчным переносом (СП) - IT (Interline Transfer);
- со строчно-кадровым переносом (СКП) - FIT (Frame-Interline Transfer).

Простейший вариант ПЗС матрицы КП изображён на рис. 11.5. В нём можно выделить два вертикальных регистра сдвига на ПЗС, образующие секцию накопления и секцию хранения с равным числом строк (каждая строка секции образована одной тройкой электродов), горизонтальный регистр сдвига и выходное устройство. Рассмотрим подробнее работу такой структуры.

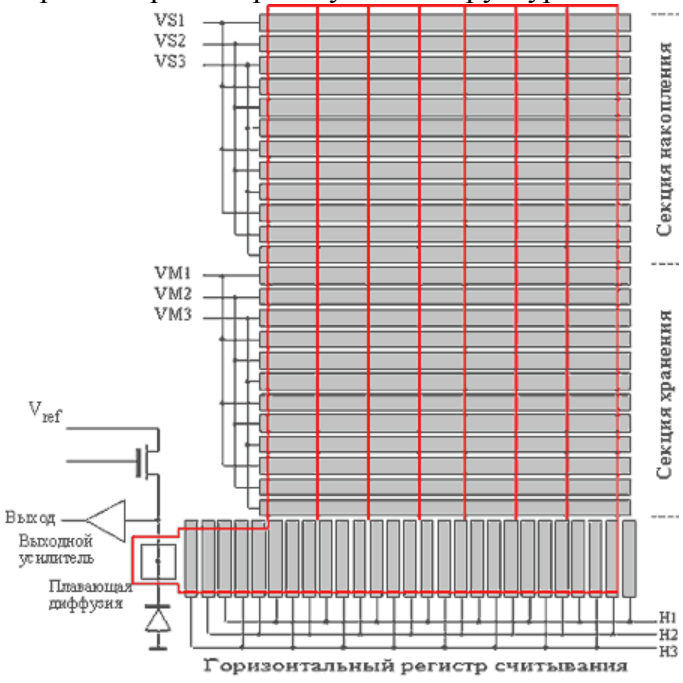


Рис. 5.5. ПЗС матрица с кадровым переносом

В течение времени накопления (прямого хода по кадру) секция накопления стоит, т. е. на неё подаются неизменные напряжения, формирующие потенциальные ямы только под одним электродом каждой тройки, скажем, под электродом первой фазы (VS1), причём потенциальные ямы образуются во всех элементах всех строк секции. По горизонтали отдельные ячейки накопления отделены стоп-каналами (выделены на рисунке красным цветом). Изображение, проецируемое на секцию накопления, вызывает фотогенерацию - образование электронно-дырочных пар. При этом фотогенерированные электроны остаются в потенциальной яме, дырки же, соответственно, уйдут в подложку или в вдоль поверхности в стоп-каналы. Таким образом, под действием света в ячейках накапливается зарядовый рельеф, т. е. в каждой ячейке собирается заряд, пропорциональный её освещённости и времени накопления.

По окончании прямого хода по кадру на обе секции подаются тактовые импульсы, вызывающие синхронный перенос заряда, при этом важно (и это показано на рисунке),

что обе секции образуют непрерывный регистр сдвига. После числа тактов, равного числу строк в каждой секции (напомним, что каждая строка образована тремя электродами), весь накопленный зарядовый рельеф целиком переместится в секцию памяти, закрытую от света, а секция накопления будет очищена от заряда. Этот перенос секции в секцию происходит достаточно быстро. Теперь, во время следующего цикла накопления секция накопления накапливает следующий кадр изображения, а из секции памяти заряды построчно передаются в горизонтальный регистр (каждый элемент регистра имеет зарядовую связь с соответствующим столбцом секции памяти, и за один раз передаётся одна строка), и затем выводятся в выходное устройство регистра, формируя видеосигнал.

При всей несомненной простоте, у матриц с рассмотренной организацией есть один существенный недостаток - сам кадровый перенос (КП). Тактовая частота, подаваемая на секции во время КП, составляет, как правило, несколько сот кГц (редко 1-2 МГц), что связано с большой ёмкостью фаз секций (до 10 000 пФ) и тем, что сами электроды имеют распределённые параметры (RC), и тактовые импульсы при их высокой частоте могут просто не дойти до середины электрода. А раз так, то КП занимает существенное время - доли мс. Если теперь учесть, что во время КП секция накопления остаётся освещённой, то яркие участки изображения успевают дать вклад в чужой зарядовый пакет даже за то короткое время, когда он проходит через них. Так на сигнале появляется смаз - вертикальный след от ярких участков изображения размером во весь кадр. Для борьбы с ним применяются разные ухищрения. Так, в малокадровых системах (прикладные системы с низкой кадровой частотой; яркий пример, опять же, - астрономия, где время накопления составляет порой часы) используется механический затвор, или же, если есть такая возможность, просто отключают источник света.

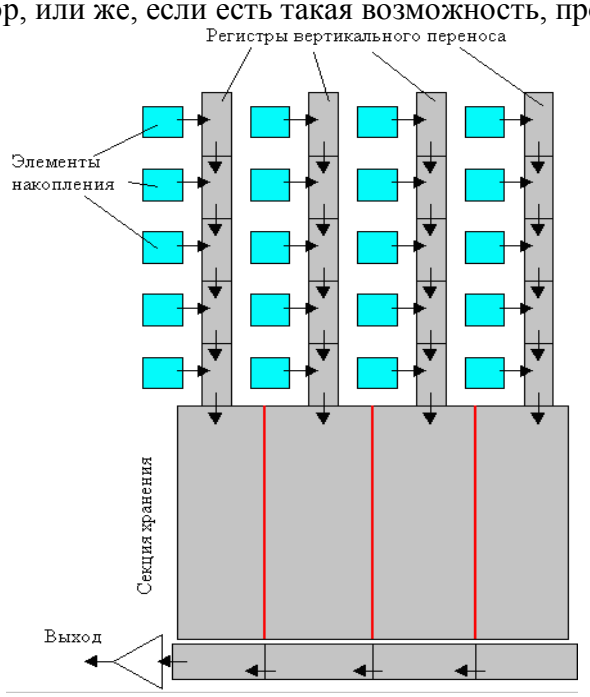


Рис. 5.6. ПЗС матрица со строчным переносом

Однако радикально проблема смаза решается в приборах со строчным переносом , завоевавших доминирующее положение на рынке бытовой видеотехники. Их организация изображена на рис. 5.6. В отличие от матриц с КП, функции накопления заряда и его переноса здесь разделены. Заряд из элементов накопления (это, как правило, фотодиоды - они тоже обладают ёмкостью и способны накапливать заряд!)

передаётся в закрытые от света ПЗС-регистры переноса, то есть секция переноса как бы вставлена в секцию накопления. Теперь перенос зарядового рельефа всего кадра происходит за один такт, и смаз, связанный с переносом, не возникает. Чтобы побороть ещё и искажения, возникающие из-за попадания в каналы переноса носителей, генерируемых в глубине подложки (если только не применяется фильтр ИК отсечки - а в видеокамерах он всегда применяется), к матрице с МП добавляется ещё одна секция памяти с соответствующим числом элементов (рис. 5.7). Смаз в такой матрице со строчно-кадровым переносом пренебрежимо мал.

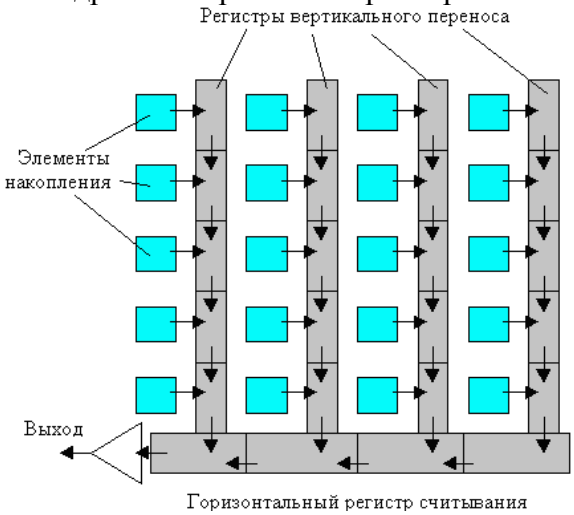


Рис. 5.7. ПЗС матрица со строчным-кадровым переносом

Параметры ПЗС

1. **Темновой ток**

Как уже упоминалось, темновой ток - это результат спонтанной генерации электронно-дырочных пар и есть явление неизбежное, однако бороться с ним можно. Дело в том, что теоретическая величина темнового тока для кремния (если брать в расчёт только прямую генерацию через запрещённую зону) крайне мала, и на самом деле темновой ток в ПЗС (как и обратные токи в других кремниевых приборах) определяется двустадийной генерацией через промежуточные энергетические уровни в запрещённой зоне. Понятно, что чем меньше концентрация этих уровней - а она определяется качеством исходного кремния, чистотой реактивов и степенью совершенства технологии - тем меньше темновой ток.

В настоящее время типовые значения темнового тока для лучших западных ПЗС составляют при комнатной температуре доли нА/см2, или несколько сотен (иногда тысяч) электронов на ячейку в секунду. И если для вещательного и бытового ТВ (время накопления 20 или 40 мс) такой темновой ток незаметен, то для научных применений, где регистрируются потоки в десяток фотонов на элемент, даже столь низкий темновой ток неприемлем. Действительно, время накопления в малокадровых системах, скажем, флуоресцентной микроскопии достигает минут, а в астрономии, когда нужно получить спектр звезды 20-й величины (совершенно типовое дело), - часов. В этом случае на помощь приходит охлаждение матриц. Как всякий термодинамический процесс, темновой ток сильно зависит от абсолютной температуры; принято считать, что при уменьшении температуры на каждые 7-8 градусов он уменьшается вдвое. Для глубокого охлаждения (в астрономических системах) используются азотные криостаты, где матрицы охлаждаются до -100^{o}C. Для более простых систем применяется термоэлектронное охлаждение с использованием батарей Пельтье, которые способны

обеспечить перепад в 70°С при подаче напряжения в 5-6 В, так что температура кристалла при комнатной наружной оказывается около -40°С, а темновой ток снижается до ~1 электрона на ячейку в секунду. Эти батареи столь компактны, что монтируются непосредственно в один корпус вместе с кристаллом ПЗС. Такие охлаждаемые приборы широко выпускаются как в США (например, фирмой SITe Technology или Hamamatsu Photonics) и в Европе (EEV, Великобритания), так и в России (фирма "Электрон-Оптроник", С.-Петербург).

2. **Неоднородность чувствительности**

Повторюсь, нет в мире совершенства. А потому ячейки ПЗС имеют неодинаковую чувствительность, т. е. даже при абсолютно однородной освещённости сигнал с них разный (иногда этот эффект называют геометрическим шумом). Величина этой неоднородности невелика и обычно не превышает 1-5% (для разных типов приборов), так что, скажем, в обычных ТВ камерах ею можно пренебречь. В научных системах, где требуется высокая фотометрическая точность, применяют довольно простой алгоритм коррекции неравномерности. Поскольку чувствительность каждого индивидуального элемента - фиксированная величина, то для её коррекции при некоторой равномерной освещённости запоминают сигналы со всех элементов прибора - и используют их как коэффициенты коррекции при всех последующих экспозициях. Предварительно, разумеется, проводят коррекцию темнового тока.

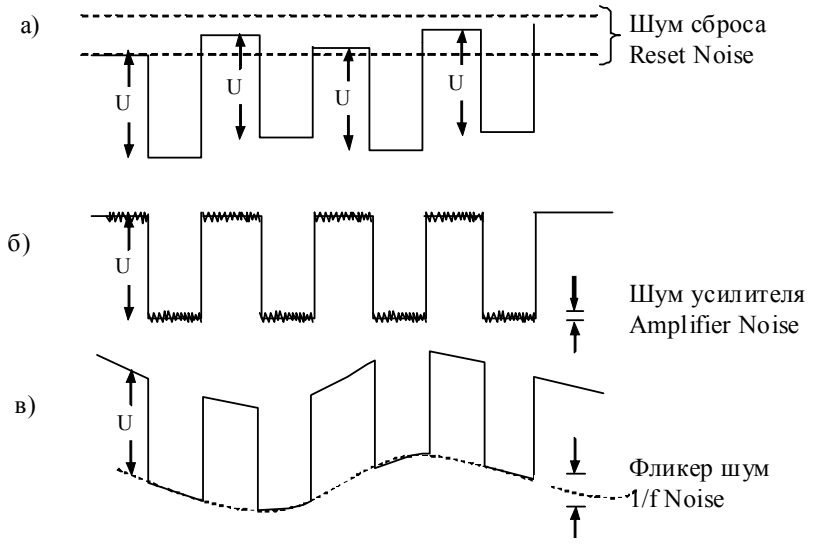
3. **Шумы**

Начнём с того, что шумит сам световой поток. То есть число фотоэлектронов, накопленное в ячейке, определено с точностью до квадратного корня из их числа (статистика Пуассона). Например, зарядовый пакет в 10000 электронов от кадра к кадру будет флуктуировать со среднеквадратическим отклонением в 100 электронов. Точно такой же статистике подвержен и темновой сигнал, и, следовательно, суммарный (световой + темновой). Это, однако, не снимает задачи снижения шумов собственно ПЗС, поскольку часто приходится работать с сигналами в десяток-другой фотонов на ячейку (к счастью, не в ТВ системах).

Для качественных приборов, где низки темновой ток и неэффективность переноса, доминирующим источником шума будет выходное устройство. Обратимся ещё раз к рис.5.5 и посмотрим на выходное устройство. Оно состоит из ёмкости считывания, как правило, диода, транзистора сброса Q1 и выходного усилителя (обычно это двухкаскадный истоковый повторитель с высоким входным импедансом). Работает такое выходное устройство так. Импульс сброса соединяет диод с источником опорного напряжения Vref., после чего транзистор сброса закрывается, и диод оказывается плавающим, т. е. его потенциал может изменяться при поступлении в него заряда - и он изменяется при следующем такте переноса заряда в регистре. Это изменение потенциала передаётся на выход прибора через усилитель. Так вот, фундаментальным свойством системы ключ - конденсатор (в случае ПЗС это транзистор Q1 и плавающая дифузия) является то, что каждый раз после размыкания ключа исходный потенциал считывающей ёмкости будет разным, причём среднеквадратическая величина этого шума (он называется установочным) равна $(kT/C)^{1/2}$, а эквивалентный шумовой заряд - $(kTC)^{1/2}$, где k - постоянная Больцмана, T - абсолютная температура, а C - ёмкость считывающего узла. При этом сам сигнал пропорционален 1/C. Стало быть, чем меньше ёмкость, на которой детектируется заряд, тем больше отношение сигнал/установочный шум для данного считывающего устройства.

Величина ёмкости считывания в современных ПЗС достигает 0,01-0,03 пФ, что соответствует установочному шуму примерно в 40-70 электронов. Для многих

применений такой уровень шума приемлем, однако существует метод, позволяющий практически полностью устранить его. Этот метод предложен М. Уайтом и другими из фирмы Westinghouse в 1974 и носит название двойной коррелированной выборки. Вдумаемся ещё раз в то, когда появляется установочный шум: после размыкания транзистора сброса (отмечу ещё раз, что "после" не значит "из-за"; причина установочного шума - в фундаментальных термодинамических законах), но до поступления заряда в плавающий диод. Поступление сигнального заряда вызывает только изменение потенциала плавающей диффузии, и если предварительно запомнить напряжение установочного шума, то потом его легко вычесть из результирующего сигнала и тем самым полностью его (шум) устранить. Метод двойной коррелированной выборки стал фактически стандартным методом предварительной обработки сигнала для всех малокадровых систем, работающих на сравнительно низких тактовых частотах, да и во многих ТВ камерах.



Остаётся только шум собственно выходного усилителя. Он имеет две компоненты: так называемый шум 1/f, присущий МОП-транзисторам, спектральная плотность которого, как следует из названия, растёт в области низких частот и сильно зависит от степени совершенства технологического процесса, и тепловой шум канала транзисторов, имеющий равномерный (белый) спектр и определяемый в основном геометрией транзистора. Шум 1/f во многом подавляется схемой двойной коррелированной выборки, которая служит фильтром верхних частот, причём степень подавления зависит от соотношения тактовой частоты и частоты излома спектральной характеристики плотности шума. Обычно конструкция выходного усилителя оптимизируется с точки зрения достижения минимального эквивалентного шумового заряда для данных условий применения, а полный эквивалентный шумовой заряд зависит ещё и от тактовой частоты работы ПЗС. Для современных приборов на частоте порядка 100 кГц типовым считается шум выходного усилителя 3-6 электронов (при охлаждении), а в лучших приборах достигается цифра 2 электрона. Поскольку заряд насыщения (максимальная величина зарядового пакета, передаваемого без искажений) составляет, как правило, 200-500 тыс. электронов, то динамический диапазон ПЗС достигает примерно 100-110 дБ; это примерно 18 или 19 бит. Кстати, динамический диапазон аудио-CD - всего лишь 16 бит. Впрочем, известны экспериментальные

конструкции усилителей с шумом.... 0,5 электрона. То есть электроны считаются поштучно.

Спектральная чувствительность

В зависимости от технологии изготовления, наличия поверхностных фильтров изменяются диапазоны спектральной чувствительности. Обычный трёхфазный ПЗС с поликремниевыми затворами имеет чувствительность в диапазоне 0,4 – 1,1 мкм, как показано на рис. 11.7 красной линией (нижняя характеристика).

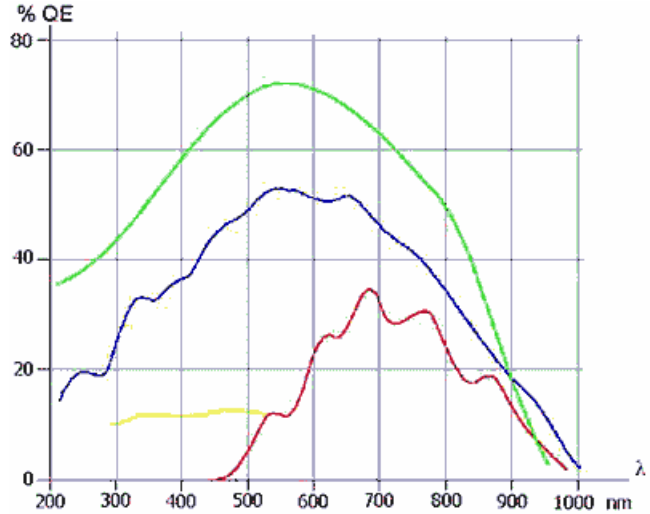


Рис. 5.7. Спектральные характеристики абсолютного квантового выхода:
обычного ПЗС (красный), ПЗС с люминофорным покрытием (желтый),
с освещением с обратной стороны подложки (зеленый)
и с виртуальной фазой (синий).

4. **Антиблюминг, или устойчивость к локальным пересветкам.**

Что же будет происходить в ячейке ПЗС, когда заряд в ней будет расти и расти? Когда заряд достигнет значения потенциала в канале по соседним электродом, заряд просто начнёт переливаться через этот незапертый участок канала в соседний элемент - причём в обе стороны. На изображении это проявляется в виде вертикального расплывания ярких деталей изображения. Это явление и называется оптической пересветкой (blooming), и если в системах регистрации слабых сигналов с ним ещё можно мириться (в силу невысокой вероятности с ним столкнуться и возможности изменить время накопления), то в камерах для ТВ оно совершенно недопустимо.

Бороться с блюмингом можно только разработкой специальной конструкции ячейки. Первый способ (горизонтальный антиблюминг) состоит в том, что вдоль каждого столбца фоточувствительных ячеек прокладывается узкая стоковая область, находящаяся под большим положительным потенциалом и отделённая от накапливающей сигнальный заряд потенциальной ямы некоторым барьером, потенциал канала в котором (иногда управляемый отдельным затвором) выше, чем в запертом канале, отделяющем ячейки друг от друга. В этом случае избыточный заряд будет переливаться в сток, и искажения сигнала в соседних элементах не возникает. Если используется специальный затвор управления антиблюмингом, то появляется возможность принудительной очистки заряда из накопительной ячейки даже без её переполнения, что есть не что иное, как **электронная регулировка экспозиции**.

Матричный ПЗС с разделением цветовых сигналов.

Для выделения информации о цвете наблюдаемых объектов на светочувствительную поверхность ПЗС наносят мозаику из кодирующих

светофильтров. Наибольшее распространение в настоящее время получил мозаичный фильтр из четырёх цветов: жёлтого (Ye = G+R), голубого (Cy = G+B), пурпурного (Mg = R+B) и зелёного (G). Пространственное расположение и спектральные характеристики элементов мозаики приведены на рис 5.8а и б соответственно. Такая комбинация цветов уступает по точности цветопередачи классической триаде «красный-зелёный-синий» (RGB), но обеспечивает лучшую чувствительность телекамеры.

Для обеспечения высокой чувствительности цветной ТВ-камеры обычно в ней используется режим накопления поля. В результате, из горизонтального регистр матрицы ПЗС для каждого элемента изображения попарно следуют отсчёты смеси цветов, например для нечётных строк: (Mg + Cy), (G + Ye), (Mg + Cy), (G + Ye) и т. д., и для чётных строк: (G + Cy), (Mg + Ye), (G + Cy), (Mg + Ye) и т.д. В дальнейшем выделяется яркостный и цветовой сигналы (см. описан. цветоразностный сигнал в части 1). Для получения яркостного сигнала для нечётных строк производится следующая операция:

Y = 1/2[(G+Ye) + (Mg+Cy)] = 1/2(2B + 3G + 2R).

Аналогичный алгоритм обработки, заключающийся в задержке во времени и попарном суммировании отсчётов, применяется и для чётных строк:

Y = 1/2[(G+Cy) + (Mg+Ye)] = 1/2(2B + 3G + 2R).

При получении цветоразностного сигнала для нечётных строк производится следующая операция:

(B — Y) = [(G + Ye) — (Mg + Cy)] = — [2B — G].

Для чётных строк алгоритм обработки также заключается в задержке и вычитании попарных отсчётов:

R — Y = [(Mg + Ye) — (G + Cy)] = [2R — G].

Приведённые выражения для чётных и нечётных строк матриц ПЗС показывают, что в видеосигнале каждой чётной строки матрицы содержится информация о цветах R и G, а в каждой нечётной — B и G. Поэтому при половинной частоте выборки можно отделить один цвет от другого. Эта операция производится в аналоговой форме с помощью отдельной схемы выборки-хранения либо в цифровой форме в видеопроцессоре. Из сигналов яркости и цветности затем получают композитный сигнал в системе PAL.

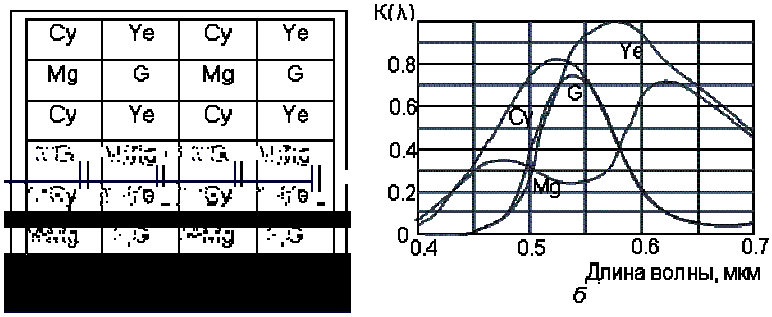
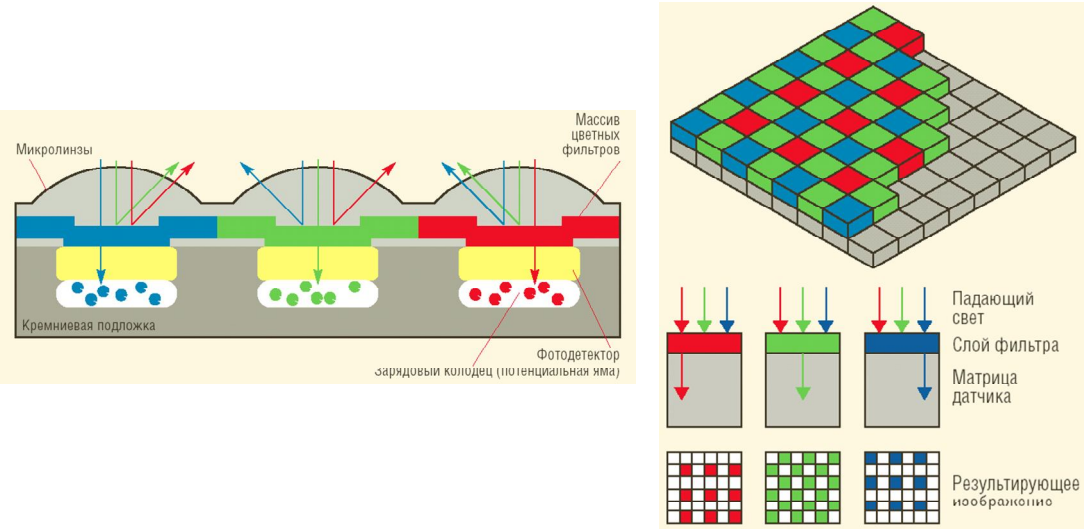


Рис. 5.8. Пространственное расположение и спектральные характеристики элементов мозаики

Массивы цветных фильтров Байера и схема расположения в матричном ПЗС.



5.2. КМОП-фотоприемники

Считывание электрического сигнала, накопленного под воздействием света, может быть осуществлено двухмерной координатной адресацией к элементам фотодиодных или фоторезистивных матриц (рис. 5.9).

Поочерёдное подключение каждого из элементов разложения осуществляется с помощью электронных ключей, выполненных по технологии комплементарных МОП-транзисторов (КМОП). Эти фотоприёмники имеют ряд достоинств по сравнению с ПЗС, хотя и уступают им по качеству изображения. Можно выделить такие свойства КМОП-фотоприёмников, как низкая мощность потребления, возможность считывания произвольного фрагмента изображения, низкая стоимость. Важным преимуществом КМОП-камер является возможность реализации функций накопления, управления считыванием, квантования и обработки видеосигнала на одном кристалле.

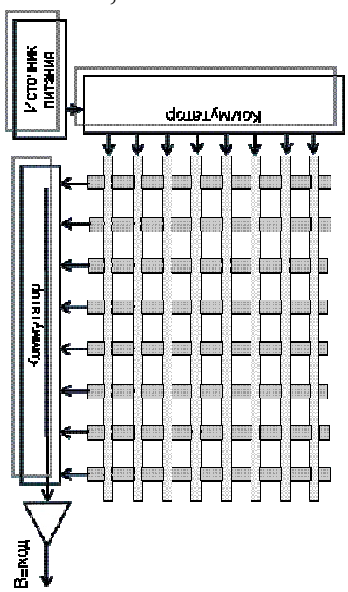


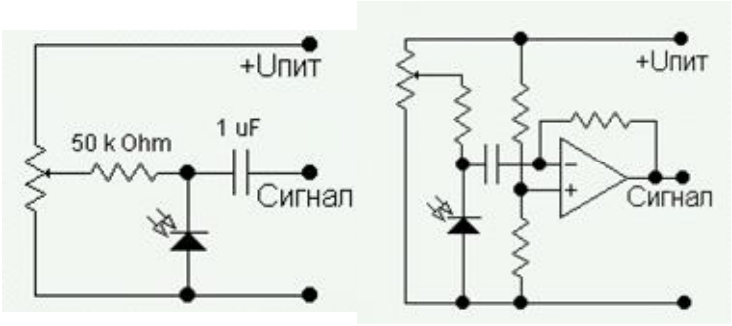
Рис. 5.9. Фотоприёмники с координатной адресацией

Фотодиод

Фотодиод – это полупроводниковый диод, обратный ток которого зависит от освещенности. Обычно в качестве фотодиода используют полупроводниковые диоды с

p-n переходом, который смещен в обратном направлении внешним источником питания.

При поглощении квантов света в p-n переходе или в прилегающих к нему областях образуются новые носители заряда. Неосновные носители заряда, возникшие в областях, прилегающих к p-n переходу на расстоянии, не превышающей диффузионной длины, диффундируют в p-n переход и проходят через него под действием электрического поля. То есть обратный ток при освещении возрастает. Величина, на которую возрастает обратный ток, называется фототоком.



Первое поколение КМОП-камер характеризовалось тем, что в них использовался единый усилитель на весь столбец фотодиодов. В камерах второго поколения к каждому фотодиоду был добавлен однотранзисторный буфер, а также введена схема двойной коррелированной выборки (ДКВ) на каждый столбец. Камеры такого типа называются камерами с активным элементом. Отличие третьего поколения КМОП-камер заключается в том, что для стабилизации коэффициента усиления усилителей каждого столбца, расположенных перед схемой ДКВ, используется обратная связь.

Активный пиксел

Active Pixel Sensors (APS) добавляет к каждому пикселу транзисторный усилитель для считывания, что даёт возможность преобразовывать заряд в напряжение прямо в пикселе. Это обеспечило также произвольный доступ к элементам фотодетектора наподобие реализованного в микросхемах ОЗУ.

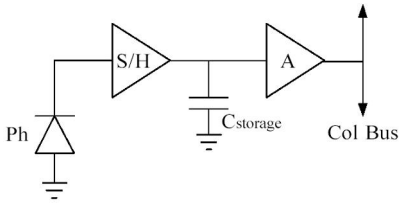


Схема элемента APS МОП фотоприемника

Телевизионный сигнал

ТВ сигнал схематично изображён на рисунке 5.10. Он содержит видеосигналы отдельных строк, разделённые интервалом обратного хода по строке (строчный гасящий интервал), необходимым для того, чтобы электронный луч как в кинескопе, так и в передающей камере (вспомним, что этот стандарт возник достаточно давно, в эпоху вакуумных приборов) успел вернуться к началу следующей сроки. Во время этого интервала подается и строчный синхроимпульс (он формируется не самим датчиком изображения, а замешивается в сигнал электронными схемами камеры). Уровень синхроимпульсов принят за 0, уровень черного в видеосигнале составляет 0,33 В, уровень гасящего - 0,3 В (30 мВ разницы образуют т. н. защитный интервал),

максимальный уровень видеосигнала (уровень белого) - 1,00 В. Когда переданы сигналы всех строк одного поля, начинается формирование кадрового гасящего интервала. Строчные синхроимпульсы в это время продолжают формироваться, чтобы не сбивать схемы строчной развёртки кинескопа (в реальности их частота на короткое время, равное 2,5 длительности строки, удваивается, а полярность инвертируется, чтобы обозначить кадровый синхроимпульс), а видеосигнал не формируется. Затем, по окончании кадрового гасящего, начинается прямой ход по кадру для следующего поля. По принятому, например, в Европе стандарту период строчной развёртки составляет 64 мкс, длительность прямого хода по строке - 52 мкс, длительность обратного хода по строке - 12 мкс, а длительность кадрового гасящего - 25 строк. При этом в каждом поле имеется 312,5 строки, из которых 287,5 - активные, т. е. имеющие видеосигнал (полстроки возникает из-за того, что полное число строк в кадре для чересстрочной развёртки нечётное - 625).

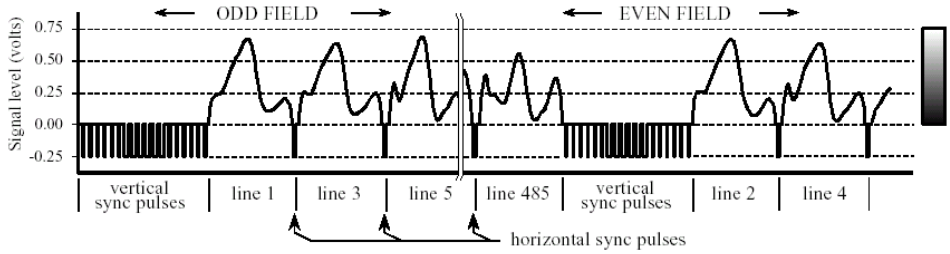


Рис. 5.10. ТВ сигнал

Тип системы	NTSC США Япония	PAL Западная Европа	SECAM Франция СССР
Горизонтальная частота развертки, кГц	15.374	15.625	15.625
Число строк в кадре	525	625	625
Число видимых (активных) строк в кадре	480	576	576
Вертикальная частота развертки, Гц	60	50	50
Тип модуляции цветовой поднесущей	Амплитудная	Амплитудная	Частотная
Полоса видеосигнала, МГц	4.2	5 для B/G, 5.5	для I, 6 для D/K

Y = 0.299R + 0.587G + 0.114B, U = R - Y, V = B - Y,
R = Y + U, B = Y + V, G = Y - 0.509U - 0.194V

5.3. Системы технического зрения

Системы технического зрения находят применение в разных областях, к которых требуется автоматическая обработка изображений и /или их регистрация в ЭВМ.

Примеры. 1. Оптико-телевизионная система для автоматического определения ориентации и пространственного положения головы и/или руки оператора в нашлемных системах целеуказания, в тренажерах для отработки управления, в транспортных роботах.

2. СТЗ для автоматического распознавания, определения координат, контроля внешнего вида объектов произвольной формы, может быть использована на сборочных операциях в машиностроении, в микроэлектронике, на конвейерах в промышленности, на операциях контроля деталей и надписей.

3. СТЗ для железнодорожной станции выполняет основные функции: непрерывный ввод кадров с заданной частотой с двух цифровых видеокамер; выделение и запись в файл кадров поезда; определение направления движения состава; определение числа цистерн в составе; локализация и распознавание номеров железнодорожных цистерн и др.

4. Системы охраны и наблюдения.

Типовая структура СТЗ

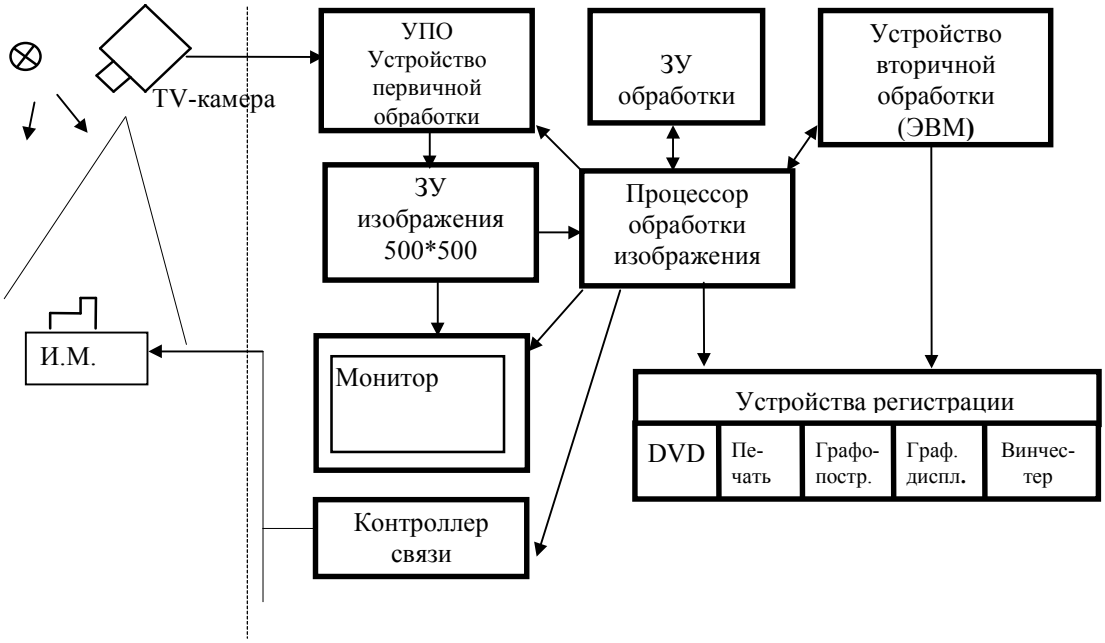


Рис. 5.11. Структура системы технического зрения

ИМ - исполнительный механизм

В СТЗ можно выделить два канала: канал регистрации изображения и канал обработки изображения.

Канал регистрации изображения включает TV-камеру, устройство первичной обработки, ЗУ и монитор. Изображение объекта через камеру и устройство первичной обработки записывается в ЗУ и одновременно отображается на TV-мониторе.

Устройство первичной обработки обеспечивает следующие функции: фрагментирование, фильтрацию, поиск максимумов (экстремумов), калибровку, вычитание темпового сигнала.

Канал обработки включает процессор обработки, буферные ЗУ, контроллер связи с исполнительным механизмом, средства вторичной обработки (ЭВМ) и средства регистрации. Процессор обычно осуществляет элементарное распознавание с целью выделения ключевых признаков, а также принимает решение на управление ИМ и передачу изображения на УВО и регистрацию. Второе ЗУ используется для вспомогательных целей, например, оно хранит изображение, которое надо распознать во входном сигнале.

Канал регистрации определяет точностные параметры СТЗ, а канал обработки определяет функциональные и временные параметры.

Точностные параметры:

1. Размерность изображения 512*512
2. Количество уровней яркости 8р - 256
3. Величина шума
   -временной 1 %
   -пространственный 5 %
4. Диапазон воспринимаемых освещенностей > 20 мк
5. Спектральный диапазон 0.4 - 1.1 мкм

Временные параметры:

1. Время анализа ситуаций > 1 сек
2. Время передачи изображения 10^6(пикселей)*10^{-6}(сек)=1с
3. Время реакции (управления ИМ) 100 мс по max
4. Набор операций по обработке

Точностные параметры в первую очередь определяются типом используемой камеры и фотоприемника в ней. Чем больше элементов разложения изображения (пикселов) и чем меньше шумов, тем выше качество/точность регистрируемого изображения.

5.4. Video Blaster. Web-камера

Video Blaster - является модулем системы обработки изображения, включает устройство предварительной обработки изображения и обычно видеопамять. Находит использование в мультимедийных приложениях. Возможности VB и следовательно качество канала регистрации изображения определяются прежде всего следующими параметрами: объемом ОЗУ, количеством разрядов для каждого цвета, используемым интерфейсом с ЭВМ (ПДП, шина PCI), сжатием изображения (MPEG) и программными средствами.

Основные функции VB:

1. Frame Grabbing - оцифровка и сохранение отдельного кадра в видеобуфере. Изображение на входе может поступить из разных источников и иметь разные форматы. Во-первых, возможны кодировки цвета YUV / RGB. Во-вторых, TV сигнал может поступить в системах PAL (Phase Alternation Line), NTSC (National Television System Commite), SECAM, HDTV (High Definition Television). HDTV - телевидение высокой четкости.

Frame Grabber - специализированный модуль для выполнения данной функции, который обычно используется в измерительных системах и может иметь несколько видеобуферов каждый размером на кадр.

2. Movie Grabbing - оцифровка и сохранение “живого видео”. Эта функция связана с возможностью записывать изображение в реальном масштабе времени на диск (например, в формате AVI- файла), а затем его воспроизводить. Важным элементом при этом является сжатие (компрессия) изображения. Параметрами, характеризующими эту функцию, являются: разрешение записываемых кадров, количество сохраняемых кадров в сек . количество цветов, совместимость с форматами AVI, Quick Time ..., продолжительность записи, объем выходного файла (коэффициент сжатия).

3. Live Video in Window - отображение живого изображения в окне без участия центрального процессора. Адаптер монитора переключается в режим “slave” (ведомый) и в качестве синхросигналов используются сигналы входного изображения. VB должен быть типа Overlay и иметь Feature Connector.

4. TV -тюнер. Данная функция обеспечивает выбор ТВ программ и их просмотр. Входным сигналом является сигнал с антенны.

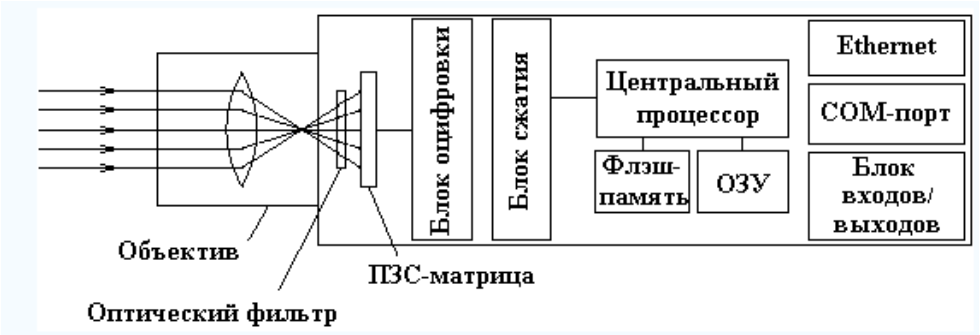
5. MPEG Decoding - воспроизведение изображений сжатых по алгоритму MPEG.
MPEG - разработан международным комитетом Motion Pictures Expert и принят в окончательной редакции ISO только в 1993г. Хотя MPEG-стандарт определяет правила кодирования и декодирования цифровых потоков как изображений, так и связанного с ними звука, в этом материале мы остановимся только на изображении. Стандарт JPEG отличается от MPEG тем, что проводит независимое сжатие каждого кадра изображения (формат MPEG описан во второй части).

Общее описание

VB сохраняет отдельные кадры изображения в собственной памяти (буфере) с последующей записью на диск либо выводить их непосредственно в „окно" на мониторе компьютера. Содержимое буфера платы обновляется с частотой смены кадров. Причем вывод видеоинформации происходит в режиме наложения (overlay). Для реализации окна на экране монитора с „живым" видео карта VB соединяется с графическим адаптером.

Web-камера

Переход от ПЗС-матрицы к КМОП-технологиям привел к тому, что подавляющее большинство современных производителей камер стали использовать именно КМОП-матрицы для производства дешевых камер. При этом стоит отметить, что основные характеристики камер, такие как формат кадра, цветность, частота кадров и т.п., практически не изменились.



Приведем основные характеристики типичные для современных Web-камер:
1. Максимальное разрешение от 640×480 пикселов.

2. USB-интерфейс подключения к компьютеру.

3. Частоту передачи кадров — до 30 fps.

4. Автоматическая адаптация к широкому диапазону освещенностей.

К основным недостаткам современных Web-камер можно отнести:

1. Низкое соотношение «сигнал/шум» и невысокое качество изображения, обусловленное дешевыми объективами и светочувствительными сенсорами.

2. Цветовую и световую изменчивость, вызванную работой внутреннего алгоритма адаптации автоматического уровня яркости и уровня баланса белого.

3. Зависимость между форматом кадра и скоростью передачи кадров (вызвана определенной пропускной способностью современных USB-портов).

Сканеры

Сканеры - устройства для ввода в компьютер двухмерных черно-белых или цветных, бипарных или многоградиционных изображений.

В 1857 году флорентийский аббат Джованни Казелли (Giovanni Caselli) изобрёл прибор для передачи изображения на расстояние, названный впоследствии пантелеграф. Передаваемая картинка наносилась на барабан токопроводящими чернилами и считывалась с помощью иглы.

В 1902 году, немецким физиком Артуром Корном (Arthur Korn) была запатентована технология фотоэлектрического сканирования, получившая впоследствии название телефакс. Передаваемое изображение закреплялось на прозрачном вращающемся барабане, луч света от лампы, перемещающейся вдоль оси барабана, проходил сквозь оригинал и через расположенные на оси барабана призму и объектив попадал на селеновый фотоприёмник. Эта технология до сих пор применяется в барабанных сканерах.

В дальнейшем, с развитием полупроводников, усовершенствовался фотоприёмник, был изобретён планшетный способ сканирования, но сам принцип оцифровки изображения остается почти неизменным. Такие сканеры появились в 1983-1985 г. Цветные появились в 1989 г. Наиболее широко сканеры используются в настольных издательских системах для обработки изображений и текстов.

Виды сканеров. В зависимости от способа сканирования объекта и самих объектов сканирования существуют следующие виды:

- Планшетные — наиболее распространённый вид сканеров, поскольку обеспечивает максимальное удобство для пользователя — высокое качество и приемлемую скорость сканирования. Представляет собой планшет, внутри которого под прозрачным стеклом расположен механизм сканирования.
- Ручные — в них отсутствует двигатель, следовательно, объект приходится сканировать пользователю вручную, единственным его плюсом является дешевизна и мобильность, при этом он имеет массу недостатков — низкое разрешение, малую скорость работы, узкая полоса сканирования, возможны перекосы изображения, поскольку пользователю будет трудно перемещать сканер с постоянной скоростью.
- Листопротяжные — лист бумаги вставляется в щель и протягивается по направляющим роликам внутри сканера мимо ламы. Имеет меньшие размеры, по сравнению с планшетным, однако может сканировать только отдельные листы, что ограничивает его применение в основном офисами компаний. Многие модели имеют устройство автоматической подачи, что позволяет быстро сканировать большое количество документов.

- Барабанные — применяются в полиграфии, имеют большое разрешение (около 10 тысяч точек на дюйм). Оригинал располагается на внутренней или внешней стенке прозрачного цилиндра (барабана).
- Слайд-сканеры — как ясно из названия, служат для сканирования плёночных слайдов, выпускаются как самостоятельные устройства, так и в виде дополнительных модулей к обычным сканерам.
- Сканеры штрих-кода — небольшие, компактные модели для сканирования штрих-кодов товара в магазинах.

Общая структура планшетных сканеров.

Рассмотрим принцип действия планшетных сканеров, как наиболее распространённых моделей. Сканируемый объект кладётся на стекло планшета сканируемой поверхностью вниз. Под стеклом располагается подвижная лампа, движение которой регулируется шаговым двигателем.

Принцип действия (рис. 5.13) Свет, отражённый от объекта, через систему зеркал попадает на чувствительную матрицу (CCD — Couple-Charged Device), далее на АЦП и передаётся в компьютер. За каждый шаг двигателя сканируется полоска объекта, которые потом объединяются программным обеспечением в общее изображение.

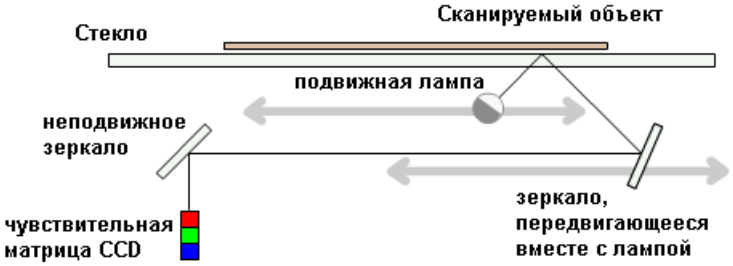


Рис. 5.13 Схема планшетного сканера

Характеристики аппаратуры:

- Оптическое разрешение (разрешение с интерполяцией) Разрешение измеряется в точках на дюйм (dots per inch — dpi). Указывается два значения например 600x1200 dpi, горизонтальное — определяется матрицей CCD, вертикальное — определяется количеством шагов двигателя на дюйм - до 3200*3200
- Способ сканирования - однопроходный или трехпроходный
- Разрядность глубины, цвет - 600 точек на дюйм в 24 разряда.
- Глубина цвета. Определяется качеством матрицы CCD и разрядностью АЦП. Измеряется количеством оттенков, которые устройство способно распознать. 24 бита соответствует 16 777 216 оттенков. Современные сканеры выпускают с глубиной цвета 24, 30, 36 бит. Несмотря на то, что графические адаптеры пока не могут работать с глубиной цвета больше 24 бит, такая избыточность позволяет сохранить больше оттенков при преобразованиях картинки в графических редакторах.
- Число элементов в ПЗС до 10000 элементов.

5.5. Библиотеки для работы с видеосистемами

Для работы с видеоадаптерами могут использоваться графические библиотеки Direct3D и OpenGL, которые хорошо документированы и общедоступны. Если установлен соответствующий драйвер, то они используют возможности акселератора

Пакет Direct3D разработан Microsoft и является одной из частей библиотеки DirectX, входящей в состав Windows

Библиотека OpenGL(Open Graphics Library — открытая графическая библиотека, графическое API) была создана в 1993 году фирмой Silicon Graphics для компьютеров совершенно другого класса и для иной операционной системы. OpenGL — спецификация, определяющая независимый от языка программирования платформонезависимый программный интерфейс для написания приложений, использующих двухмерную и трёхмерную компьютерную графику. Включает более 250 функций для рисования сложных трёхмерных сцен из простых примитивов. Используется при создании компьютерных игр, САПР, виртуальной реальности, визуализации в научных исследованиях.

Основы технологии OpenCV

http://docs.opencv.org/

OpenCV (*Open Source Computer Vision Library*, библиотека компьютерного зрения с открытым исходным кодом) — **библиотека алгоритмов компьютерного зрения**, обработки изображений и численных алгоритмов общего назначения с открытым кодом. Реализована на C/C++, также разрабатывается для Python, Java, Matlab и других языков. Может свободно использоваться в академических и коммерческих целях

OpenCV написана на языке высокого уровня (C/C++) и содержит алгоритмы для: интерпретации изображений, калибровки камеры по эталону, устранение оптических искажений, определение сходства, анализ перемещения объекта, определение формы объекта и слежение за объектом, 3D-реконструкция, сегментация объекта, распознавание жестов и т.д.

Фактически, OpenCV – это набор типов данных, функций и классов для обработки изображений алгоритмами компьютерного зрения.

Основные модули библиотеки:

В версии 2.2 вместо больших универсальных модулей библиотека OpenCV разделена на небольшие модули по функциональному использованию:

opencv_core — ядро: базовые структуры, вычисления (математические функции, DFT, DCT, ввод/вывод в XML и т.п.)

opencv_imgproc — обработка изображений (фильтры, преобразования ..)

opencv_highgui —загрузка/сохранение изображений и видео.

opencv_ml — методы и модели машинного обучения (SVM, деревья принятия решений и т. д.).

opencv_features2d — различные дескрипторы (SURF).

opencv_video — анализ движения и отслеживание объектов (оптический поток, шаблоны движения, устранение фона).

opencv_objdetect — детектирование объектов на изображении (вейвлеты Хаара, HOG и т. д.).

opencv_calib3d — калибровка камеры, поиск стерео-соответстсвия и элементы обработки трехмерных данных.**opencv_flann** — библиотека быстрого поиска ближайших соседей (FLANN).

opencv_contrib — сопутствующий код, еще не готовый для применения.

opencv_legacy — устаревший код, ради обратной совместимости.

opencv_gpu — ускорение некоторых функций OpenCV за счет CUDA .

Программа - Вывод картинки

```
#include <highgui.h>
int main()
{IplImage* img = cvLoadImage("D:\\foto.jpg"); // Загружаем изображение
cvNamedWindow("Example1", CV_WINDOW_AUTOSIZE); // Создаём окно
cvShowImage("Example1", img); // Выводим картинку в окно
cvWaitKey(0); // Ждём
cvReleaseImage(&img); // Освобождаем память из под картинки
cvDestroyWindow("Example1"); // Удаляем окно
return 0;
}
```

Программа - AVI Видео

```
#include "highgui.h"
int main()
{    cvNamedWindow( "Example2", CV_WINDOW_AUTOSIZE );
 // Создаем окошко
CvCapture* capture = cvCreateFileCapture( "D:\\film.avi" );
// Открываем файл
IplImage* frame; // Здесь будет кадр
while(1) {frame = cvQueryFrame( capture ); // Читаем кадр из файла
if( !frame ) break; // Если кадров больше нет - выходим
cvShowImage( "Example2", frame ); // Выводим кадр
char c = cvWaitKey(33); // Ждем 33мс
if( c == 27 ) break; // Если нажали Esc - выходим    }
cvReleaseCapture( &capture ); // Закрываем файл
cvDestroyWindow( "Example2" ); // И окно}
```

Часть 6. Средства вывода информации. Интерфейсы

6.1. Принтеры

Принтер является основным устройством вывода информации для получения «твердой копии». Классификация принтеров по способу печати:

- Строчные
- Последовательные
- Страничные

Принадлежность принтера к той или иной из перечисленных групп зависит от того, формирует он на бумаге символ за символом или сразу всю строку, а то и целую страницу,

Классификация принтеров по механическому принципу:

- Ударные (inpact)
- Безударные (non-inpact)

Классификация принтеров по используемой технологии печати:

- Матричные
- Струйные
- Лазерные
- LED-принтеры
- С термопереносом восковой мастики
- С термосублимацией
- С изменением фазы красителя

Матричные принтеры

Последовательные ударные матричные печатающие устройства (impact dot matrix) работают следующим образом: вертикальный ряд (или два ряда) игл (рис. 12.1), или молоточков, „вколачивает" краситель с ленты прямо в бумагу, формируя последовательно символ за символом. Более высокую производительность обеспечивают построчные (постраничные) матричные принтеры. Вместо маленьких точечно-матричных головок они используют длинные массивы с большим количеством игл, при этом достигается скорость печати порядка 1500 строк в минуту.

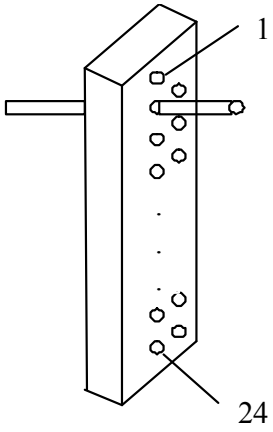


Рис. 6.1.Схема печатающей 24-х игольчатой головки

Принтеры поддерживают два режима печати: текстовый и графический. Переход в графический режим происходит по команде, в которой задано количество выводимых графических байтов. Выход происходит автоматически. В текстовом режиме изображение символов формирует сам принтер, использую ASCI код символа и таблицу сечений изображений символов. Таблица хранится в памяти принтера. Схема

формирования изображения символа из описания его отдельных сечений приведена на рис. 6.2.

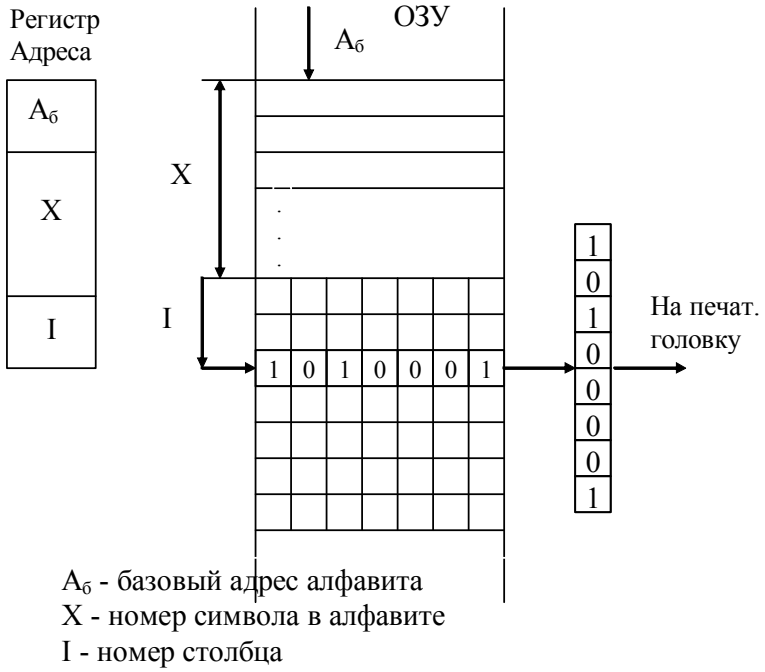


Рис. 6.2. Схема формирования изображения символа при печати.

Матричные принтеры долго служат и дешево обходятся в эксплуатации, но для вывода графической не подходят, так как имеют низкое качество и скорость печати. Работают обычно только с одним цветом. В настоящее время почти не употребляются.

Струйные принтеры

Струйные принтеры относятся к безударным печатающим устройствам. Носитель печатаемой информации не касается бумаги. Струйные чернильные принтеры относятся, как правило, к классу последовательных матричных безударных печатающих устройств. Последовательные, безударные, матричные, струйные, чернильные (liquid ink jet) принтеры, в свою очередь, подразделяются на устройства непрерывного (continuous drop, continuous jet) и дискретного (drop-on-demand) действия. Последние в своей работе опять же могут использовать либо „пузырьковую" технологию (bubble-jet или thermal ink-jet), либо пьезоэффект (piezo ink-jet). У чернильных устройств, как, впрочем, и у ударных матричных принтеров, печатающая головка движется только в горизонтальной плоскости, а бумага подается вертикально. Сопла (канальные отверстия) на печатающей головке, через которые разбрызгиваются чернила, соответствуют „ударным" иглам. Количество сопел у разных моделей принтеров, как правило, может варьироваться от 12 до 64. Поскольку размер каждого сопла существенно меньше диаметра иглы (тоньше человеческого волоса), а количество сопел может быть больше, то получаемое изображение должно быть в этом случае четче (если чернила не расплываются на бумаге). Максимальная разрешающая способность достигает значения 720 точек на дюйм. Например, принтер Hewlett-Packard PaintJet имеет 30 сопел для черного цвета и по 10 для бирюзового,

яркокрасного и желтого цветов. Чтобы иметь больше, чем 7 цветов, струйные принтеры используют прием, известный как подмешивание: печать смежных точек разными цветами, которые глаз воспринимает как блок одного цвета. Обобщенная схема построения принтера приведена на рис. 12.3.

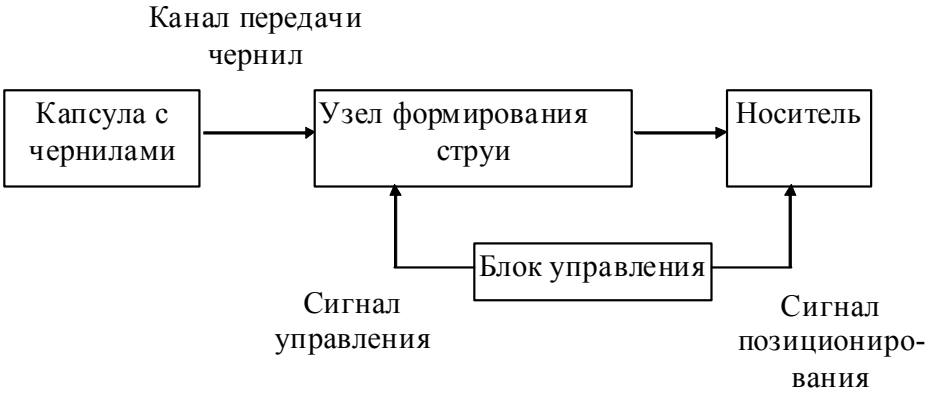


Рис. 6.3. Обобщенная схема построения струйного принтера.

Сами струйные принтеры относительно дешевы, но велика стоимость расходных материалов (чернил). Со временем у многих принтеров краски на бумаге выцветают.

Лазерные принтеры

В лазерных принтерах используется электрографический принцип создания изображения. Этот процесс включает в себя создание рельефа электростатического потенциала в слое полупроводника с последующей визуализацией полученного рельефа. Собственно визуализация осуществляется с помощью частиц сухого порошка - тонера, наносимого на бумагу. Наиболее важными частями лазерного принтера можно считать фотопроводящий цилиндр (печатающий барабан), полупроводниковый лазер и прецизионную оптико-механическую систему, перемещающую луч. К наиболее важным функциональным возможностям принтеров относятся такие, как поддержка технологии повышения разрешающей способности, наличие масштабируемых шрифтов (PostScript, TrueType), объем встроенной оперативной памяти и т.п. Отличаются более высоким быстродействием, чем струйный принтер.

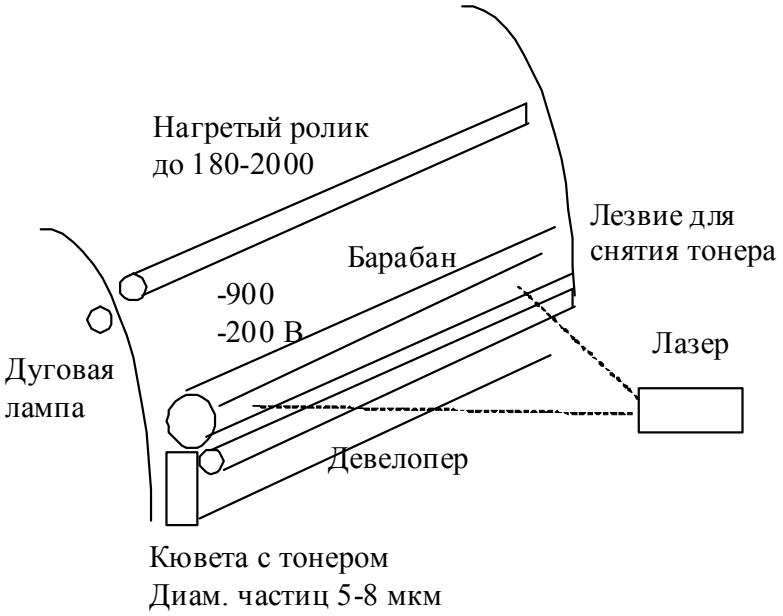


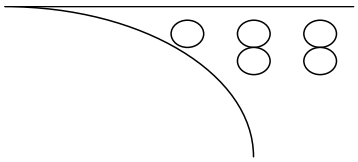
Рис. 6.4. Схема построения лазерного принтера.

Схема принтера приведена на рис. 12.4. На вращающемся барабане нанесен специальный фотопроводящий слой. Под действием лазерного луча в этом слое формируются заряженные области. Заряженные области притягивают порошок для электростатической печати. Барабан переносит порошок на бумагу. Нагретые ролики вплавляют его в страницу. Для цветной печати страницу изображения необходимо 4 раза пропустить через принтер. Достоинства: бесшумность, надежность, скорость, высокое разрешение до 2000 т/дюйм

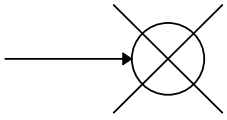
Методы повышения вертикального разрешения при использовании стандартного привода фирмы Canon, обеспечивающим разрешение 300*300 dpi. (Поворот барабана 1/300 дюйм).

1. Метод RET - Resolution Enhancement Technology - изменение диаметра напечатанных точек. Повышается разрешение в 1,5 раза. Диаметр точек модулируется:

1) на углах



2) на пересечениях

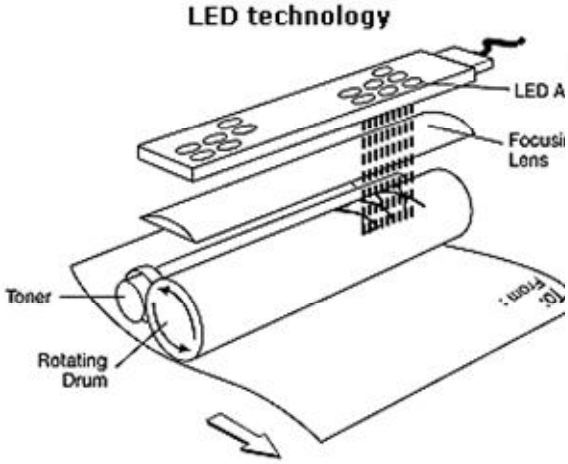


2. Метод TRE - Turbo Resolution Enhanced. По вертикали - каждая точка это столбик, управляя столбиком можно повысить разрешение до 1200 т/дюйм

LED-принтеры

Кроме лазерных принтеров существуют так называемые LED-принтеры (Light Emitting Diode), которые получили свое название из-за того, что полупроводниковый

лазер в них был заменен „гребенкой" мельчайших светодиодов. В этом случае не требуется сложная оптическая система вращающихся зеркал и линз.



Oki Electric Industry Co., Ltd.

Item	Specification
Effective printing width	216mm (A4 size)
Total number of dots	9984 dots
Resolution	1200 dpi
Printing speed	45PPM
Emission wavelength	760nm
Average light output	1.1μW(@1mA)
Variation in light output	±1%
Maximum current consumption	2A
Image formation system	Rod lens array with aperture angle of 20°

Table 1: Basic Specifications of the LED printer we developed

Рис. 6.5. Схема и спецификация LED-принтера.

С термопереносом восковой мастики

Принцип работы принтера с термопереносом (thermal wax transfer) состоит в том, что термопластичное красящее вещество, нанесенное на тонкую подложку, попадает на бумагу именно в том месте, где нагревательными элементами (аналогами сопел и игл) печатающей головки обеспечивается должная температура. Основными составными частями печатающей головки термопринтера являются несколько крошечных нагревательных элементов, которые расположены примерно так же, как расположены иглы в обычном матричном ударном принтере: друг над другом в два ряда. Как у ударных матричных и струйных принтеров, печатающая головка термопринтера позиционируется только в горизонтальном направлении, а подача бумаги осуществляется в вертикальном (последовательные принтеры). Поскольку между печатающей головкой и бумагой механический контакт отсутствует, термопринтеры относятся к классу безударных устройств. Такие принтеры работают на специальной бумаге. Со временем такие распечатки могут выцветать.

Принтеры с термосублимацией красителя

Принтеры с термосублимацией (dye sublimation) используют технологию, наиболее близкую к технологии термопереноса, только элементы печатающей головки нагреваются в данном случае уже до более высокой температуры. При сублимации переход вещества из твердого состояния в газообразное происходит минуя стадию жидкости. Таким образом, порция красителя сублимирует с подложки и осаждается на бумаге или ином носителе. Комбинацией цветов красителей можно подобрать практически любую цветовую палитру. Данная технология используется только для цветной печати, а реализующие ее устройства имеют очень хорошие технические характеристики и стоят довольно дорого. К их основным преимуществам относятся практически фотографическое качество получаемого изображения и широкая гамма оттенков цветов без использования растрирования.

Принтеры с изменением фазы красителя

Принцип работы устройств с изменением фазы красителя, или с твердым красителем (phase change ink-jet, или solid ink-jet), примерно следующий. Восковые стерженьки для каждого первичного цвета красителя постепенно расплавляются специальным нагревательным элементом и попадают в отдельные резервуары. Расплавленные красители подаются оттуда специальным насосом в печатающую головку, работающую обычно на основе пьезоэффекта. Капли воскообразного красителя на бумаге застывают практически мгновенно, но обеспечивают необходимое с ней сцепление. В отличие от обычной струйной технологии (liquid ink-jet) в данном случае не происходит ни просачивания, ни растекания, ни смешения красителей. Именно поэтому принтеры, использующие технологию с изменением фазы красителя, работают с любой бумагой. Качество цветов получается просто превосходным, к тому же допустима и двусторонняя печать.

Плоттеры

Плоттер является устройством вывода информации для получения «твердой копии». Используется для вывода преимущественно графической информации. Является векторным устройством и предпочтителен для вывода изображений, вырисовываемых линиями.

Первыми появились и традиционно широко используются перьевые плоттеры. Еще до отсутствия компьютеров существовал такой способ вывода графика выходного электрического сигнала из физических приборов. (Кто помнит древний прибор "Графопостроитель ЛКД" 70-80-х годов с чернильницей или просто со стержнем из шариковой ручки). Печатающим устройством в перьевых плоттерах являются вставленными во вращающийся барабан сменные фломастеры. Предшественниками для них были планшетные графопостроители. Основной конкурент для них - струйные плоттеры, использующие более современную технологию. Некоторые модели графопостроителей комплектуются или могут оснащаться насадками, дополняющими их функциями сканера.

Основные сферы применения плоттеров (и, соответственно, требующее векторного формата изображений): моделирование (САПР в машиностроении); автоматизированное проектирование (CAD); архитектурное проектирование; изготовление прототипов и линейных форм; дизайн упаковки; раскрой тканей.

Перьевые плоттеры условно можно разделить на три группы:

- плоттеры, использующие фрикционный прижим для перемещения бумаги в направлении одной оси и движения пера по другой;
- барабанные (или рулонные плоттеры), работающие примерно так же, как и фрикционные, но использующие для перемещения непрерывной перфорированной ленты бумаги специальный трактор (tractor feed);
- планшетные плоттеры, в которых бумага неподвижна, а перо перемещается по обеим осям.

6.2. Магнитные диски

1.Технология записи

Принцип цифровой магнитной записи аналогичен технологии аналоговой записи на магнитный носитель (магнитная запись идеально двоичная). Хотя сначала появилась аналоговая магнитная запись, а затем цифровая. Магнитная запись производится на поверхности магнито-чувствительного материала, обычно это двуокись железа, которая придает магнитным носителям их характерный ржаво-коричневый цвет. Магнитное покрытие довольно тонкое, чем оно тоньше, тем лучше его записывающие качества. Оно наносится на какую-либо основу, обычно на гибкий пластик, из которого делаются дискеты или ленты, или на алюминиевые пластины круглой формы для производства винчестеров.

Три вида покрытий магнитным слоем алюминиевого диска:

1. Оксидная паста. Диск вращается с высокой скоростью. Паста капается на центр и равномерно растекается. Толщина 70 мкм. Цвет коричневый или янтарный.

2. Анодированный носитель. Алюминиевые диски покрываются сплавом кобальта электрогальваническим способом. Толщина магнитного носителя 7 мкм. Анодированная пленка отличается высокой твердостью.

3. Напыление. Вакуумное осаждение магнитного покрытия. Толщина покрытия 5 мкм. Дополнительно напыляют защитный слой 2 мкм.

Тонкопленочные покрытия (2,3) имеют цвет зеркальный или хромированный. Они обеспечивают большую плотность записи, дают возможность приблизить считывающую головку к поверхности на расстояние до 10 мкм, повысить отношение сигнал/шум при операциях записи/чтения.

Магнитная поверхность рассматривается как массив расположенных на ней точек, каждая рассматривается, как отдельный бит, которому будет придан магнитный эквивалент 0 или 1. Так как расположение точек не определено заранее, то схема записи предполагает использование меток, которые помогают записывающему устройству найти позиции записи. Необходимость этих синхронизирующих меток является одной из причин, по которым диски должны быть "форматированы" перед использованием.

2. Организация быстрого доступа

Быстрый доступ к любой точке поверхности диска осуществляется на основе двух моментов.

Первый - это вращение диска. За малый момент времени диск может повернуться к нужному месту своей окружности. Скорость вращения диска 300 об/мин (винчестер - более 7200 об/мин). Это означает, что нужной точки на окружности можно достичь максимум за 1/5 сек. Второй - это движение магнитной записывающей головки с внешней стороны диска к центру. Для дискеты головка передвигается за 1/6 сек., а для жесткого диска за 1/25 сек и быстрее.

3. Разметка диска

Поверхность диска разделена на дорожки - окружности, начиная с внешней стороны диска, где записана первая дорожка.

Число дорожек зависит от типа диска и способа форматирования. Жесткие диски имеют от 300 дорожек. Дорожки обозначены номерами, начиная с нулевой на внешней стороне диска. Дорожки занимают небольшую площадь на поверхности диска. Расстояние между первой и последней дорожками для любых дискет составляет 2 см (3/4 дюйма) при 5,2 дюймовых дискетах. Дорожки разделены на секторы. Количество секторов определяется типом диска и его форматом. У DD-8 или 9 секторов, у HD-15 секторов, у жестких дисков 17... 26 ... секторов.

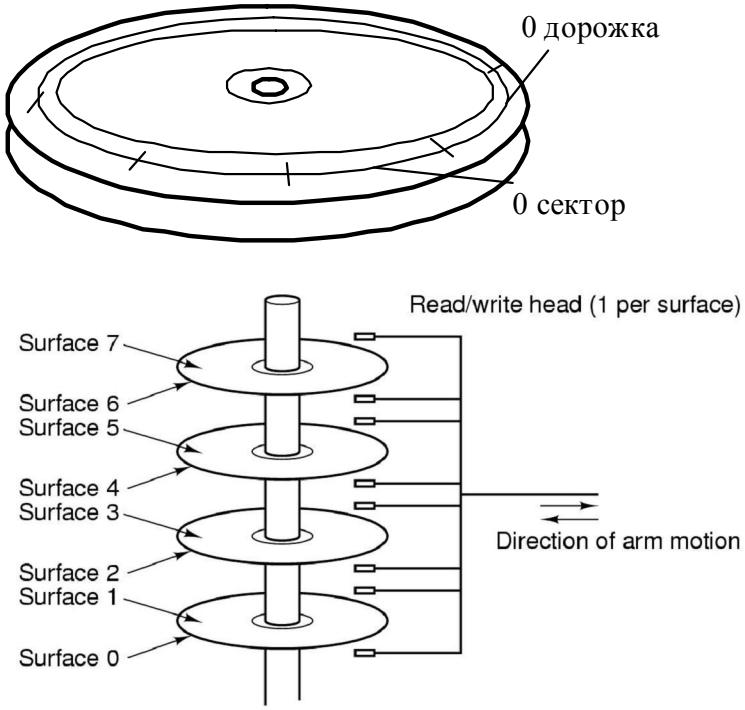


Рис. 6.6. Разметка диска и схема поверхностей дисковода.

Все сектора на одном диске имеют фиксированный размер. IBM PC могут работать с разными размерами секторов -от 128 до 1024 байт. Стандартом являются 512 байт. Операции по вводу-выводу на диск или с диска ведутся только с полными секторами, а для винчестеров - кластерами. Сектора обозначаются номерами начиная с единицы. Нулевой сектор отводится для целей идентификации, а не хранения информации.

Имеется еще одно измерение диска- это число его сторон. Информация может быть записана на обеих сторонах дискеты или только на одной ее стороне. Если дискета имеет две стороны, то винчестер может включать несколько дисков и иметь больше, чем две стороны. Стороны пронумерованы, начиная с нулевой для первой стороны.

Сочетание всех измерений дает объем памяти диска. Перемножив число сторон (S) на число дорожек (D) на одной стороне, на число секторов (T) на дорожке и количество байт в секторе (B) получим емкость диска (W): W=S*D*T*B.

4. Элементы дисковода

Основные компоненты дисковода: головки чтения/записи, привод головок, шпиндельный двигатель, монтажная плата.

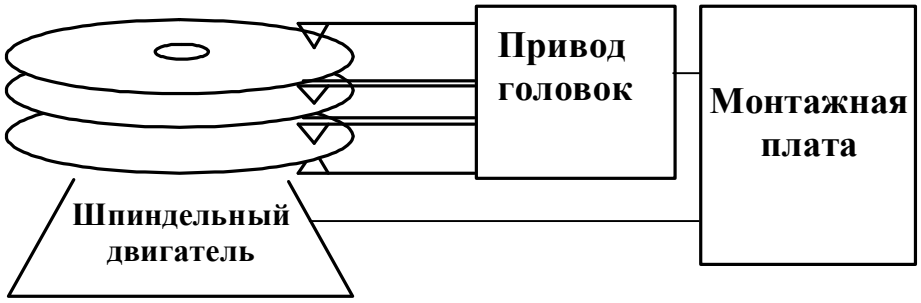


Рис. 6.7. Механическая схема дисковода

Типы головок

1. Композитные ферритовые головки. Каждая головка составная: головка записи находится между двумя стирающими головками. При записи пара стирающих головок урезает по ширине дорожку и между дорожками остаются размагниченные зоны. Сама головка - сердечник, заключенный в электромагнитную катушку.

2. Тонкопленочные головки. Это специальный полупроводниковый кристалл.

Типы приводов управления головками

1. Привод от шагового двигателя.

2. Соленоидный привод. Имеют большинство винчестеров. Привод имеет обратную связь и включает индексную головку, поверхность с серводанными, контроллер управления и электромагнитный механизм (пружина и электромагнит) для перемещения головок. Парковка автоматическая (при выключении питания пружина отводит головки в зону парковки). Сервоповерхность (DSS - Dedicated Servo Surface) может занимать одну из поверхностей дисковода, но чаще располагается между дорожками.

5. Физическое и логическое форматирование

Физическое форматирование состоит в создании секторов на диске с обозначением адреса каждого сектора, в установлении области сектора предназначенного для данных и заполнении ее фиктивными данными.

Логическое форматирование заключается в приспособлении диска к стандартам операционной системы.

6. Структура диска (DOS)

Диск делиться на две части системная область и область данных. Объем системной области: 0.1% на 100 Мб жестком диске.

Системная область включает: Область для программного запуска (1 сектор), Таблицу размещения файлов(FAT), Корневой каталог

БЛОК НАЧАЛЬНОЙ ЗАГРУЗКИ имеется на каждом диске. Он содержит программу для начальной загрузки DOS, объем программы такой, что она помещается в один 512 байтовый сектор.

ТАБЛИЦА РАЗМЕЩЕНИЯ ФАЙЛОВ (FAT) наиболее важная часть диска, которую нужно защищать очень тщательно. Содержит список кластеров. На диске хранятся две идентичные копии FAT, расположенные одна за другой

Кластер представляет собой группу последовательно расположенных секторов. Если DOS получает запрос на выделение дискового пространства, она выделяет сразу целый кластер. Размер кластера является компромиссом между двумя противоречивыми требованиями: с точки зрения экономии дискового пространства выгоден малый размер кластера, так как для маленького файла (размером даже в один байт) выделяется целый кластер, большая часть которого не используется. Но с другой стороны, при малом размере кластера их на диске получится очень много, и управление ими усложняется. FAT представляет собой "карту" дискового пространства - массив элементов, каждый из которых соответствует одному кластеру диска. Номер элемента соответствует номеру кластера.

Если в элементе таблицы признак "0",то кластер свободен. Если не "0",то кластер занят и это число определяет номер следующего кластера в файле или признак конца файла(FFF). Кластер содержит 1,2 или более сегментов. Обычно кластер содержит 2 сегмента. Первый используемый кластер имеет номер 2. Самый первый байт FAT содержит код для определения формата диска.

КОРНЕВОЙ КАТАЛОГ - основной встроенный каталог. Для каждого файла в каталоге записываются:8-значное имя файла,3-значное расширение размер файла(4 байта),дата и время создания (4 байта), номер начального кластера, атрибут файла(1 байт).

Атрибуты: подкаталоги имеют атрибут каталога, системные файлы - системный и скрытый атрибуты, доступ только для чтения, архивный атрибут. Корневой каталог имеет свой размер. Для 20 Мб винчестера корневой каталог имеет 32 сектора для 512 файлов.

В поле файлового имени есть два специальных кода которые используются в первом байте имени файла. Если этот байт "0" ,то элемент каталога не использовался. Если байт "E5",то этот элемент стерт. Он нам не виден ,но на диске существует.

7. Файловые системы

В широком смысле понятие "файловая система" включает:

- совокупность всех файлов на диске,
- наборы служебных структур данных, используемых для управления файлами, такие как, например, каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске,
- комплекс системных программных средств, реализующих управление файлами, в частности операции по созданию, уничтожению, чтению, записи, именованию файлов, установке атрибутов и уровней доступа, поиску и т.д.
- Различие между файловыми системами заключается, в основном, в способах распределения пространства между файлами на диске и организации на диске служебных областей.

• В большинстве операционных систем (Windows) реализуется механизм переключения файловых систем (File System Switch, FSS), позволяющий поддерживать различные типы ФС

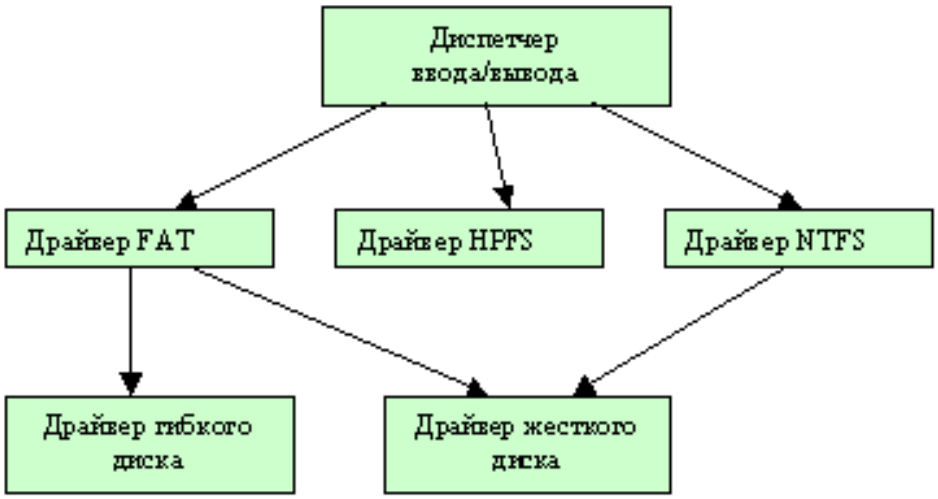


Рис. 6.8. Многоуровневая файловая система.

Файловая система представляет многоуровневую структуру (рис. 6.7), на верхнем уровне которой располагается так называемый переключатель файловых систем (в Windows, такой переключатель называется устанавливаемым диспетчером файловой системы - installable filesystem manager, IFS). Он обеспечивает интерфейс между приложением и конкретной файловой системой, к которой обращается приложение. Переключатель файловых систем преобразует запросы к файлам в формат, воспринимаемый следующим уровнем - уровнем драйверов файловых систем. Для выполнения своих функций драйверы файловых систем обращаются к драйверам конкретных устройств хранения информации.

8. NTFS (New Technology File System)

http://www.insidepro.com/kk/044/044r.shtml

NTFS - наиболее предпочтительная файловая система при работе с Windows NT, поскольку она была специально разработана для данной системы. В состав Windows NT входит утилита convert, осуществляющая конвертирование томов с FAT и HPFS в тома NTFS. В NTFS значительно расширены возможности по управлению доступом к отдельным файлам и каталогам, введено большое число атрибутов, реализована отказоустойчивость, средства динамического сжатия файлов, поддержка требований стандарта POSIX. NTFS позволяет использовать имена файлов длиной до 255 символов. NTFS обладает возможностью самостоятельного восстановления в случае сбоя ОС или оборудования, так что дисковый том остается доступным, а структура каталогов не нарушается.

Журналирование

NTFS - отказоустойчивая система, которая вполне может привести себя в корректное состояние при практически любых реальных сбоях.

Пример : Идет запись данных на диск. Вдруг отключается питание и система перезагружается. На какой фазе остановилась запись, где есть данные? На помощь приходит механизм системы - журнал транзакций.

Система, осознав свое желание писать на диск, пометила в метафайле $LogFile это свое состояние. При перезагрузке это файл изучается на предмет наличия незавершенных транзакций, которые были прерваны аварией и результат которых непредсказуем - все эти транзакции отменяются: место, в которое осуществлялась запись, помечается снова как свободное, индексы и элементы MFT приводятся в с состояние, в котором они были до сбоя, и система в целом остается стабильна.

Если ошибка произошла при записи в журнал? Тоже ничего страшного: транзакция либо еще и не начиналась (идет только попытка записать намерения её произвести), либо уже закончилась - то есть идет попытка записать, что транзакция на самом деле уже выполнена. В последнем случае при следующей загрузке система сама вполне разберется, что на самом деле всё и так записано корректно, и не обратит внимания на "незаконченную" транзакцию.

9. Интерфейсы винчестеров

1. ST-506/412 - Seagate Technologies 1980 г.

2. ESDI - Extended Storage Device Interface (Улучшенный интерфейс.) ESDI - высокоскоростной интерфейс кодер/декодер встроен в накопитель. 34 сектора на дорожку. Поддерживает чередование 1:1.

3. SCSI - Small Computer System Interface (Интерфейс малых компьютерных систем.) SCDI - это не дисковый интерфейс, а интерфейс с системного уровня.

Можно подключить до 8 контроллеров, связать в сеть, один из них главный - он в гнезде IBM и работает как канал передачи SCSI- шины и системы. SCSI не работает прямо с винчестером, винчестеру нужен контроллер (ST412 или ESDI), который подключен к SCSI интерфейсу. SCSI+ST412 - встроены в винчестер и не важно какой интерфейс винчестера.

Особенности:

1. Свойство отсоединения/присоединения (Disconnect/ Reconnect) - после посылки управляющей команды устройству, оно может отсоединится от шины и присоединиться обратно. Например, после команды данные считываются когда головка будет установлена на нужный цилиндр. За время отсоединения контроллер может дать команды другим устройствам.

2. Оптимизация очереди команд (Tagged Command Qveving) - для SCSI-2 В зависимости от положения головки происходит оптимизация очереди и запросы выполняются в порядке минимизации механических перемещений головок.

3. Имеются различные модификации, в том числе с последовательным каналом.

4. IDE - Integrated Drive Electronics (Интерфейс распределенной электроники). Позволяет подключать до 2х устройств (до 4-х IDE-2)

10. Схемы кодировки

MFM - Modified Frequency Modulation (Модифицированная частотная модуляция.)

RLL - Run Length Limited (Ограничения периода работы.)

MFM - Схема кодировки для преобразования цифровой информации в магнитные импульсы. Перемагничивание на диске равномерно отстает друг от друга по времени, так что начало одного бита может быть отделено от другого. Код имеет фиксированную длину.

RLL - обычно следуют цифры: RLL 2,7 где 2 - минимальное количество нулевых байт между двумя ненулевыми, а 7 -максимальное их количество. Менее регулярны интервалы между перемагничиванием, код не имеет фиксированной длинны. Вероятность сбоев выше. На 50% выше плотность и скорость передачи, чем MFM.

11. Структурная схема винчестера

Рассмотрим организацию винчестера и особенности его работы на примере винчестера с IDE интерфейсом. Основные элементы винчестера:

Сепаратор - выделяет из считанного сигнала синхроимпульсы и данные.

Микроконтроллер - по специальным меткам (Адресным маркерам), находящихся в формате магнитной записи, распознает поле идентификации и поле данных. В поле идентификации находятся номера головки, цилиндра, сектора. Сравнивает номера с заданными и проводит операцию считывания/записи.

Микропроцессор управления - распознает команды компьютера и управляет работой винчестера.

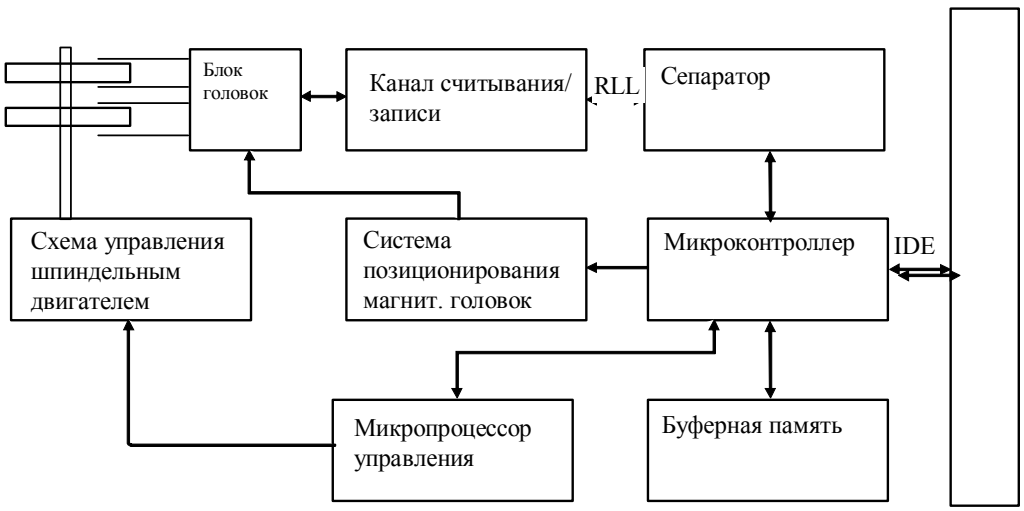


Рис. 12.9. Структурная схема дисковода.

Винчестер имеет запас по емкости. В избыточную емкость записывают сбойные сектора, рабочие программы, паспорт. Минимальный размер буфера должен иметь объем на полную дорожку (256 - 1024 Кбайт). Это поддерживает режим чередования секторов (Interleav фактор) 1:1.

Для повышения надежности считывания рекомендуется:

1) Использовать винчестеры с соленоидным приводом головок.

2) Встроенный контроллер.

3) Накопители с тонкопленочным носителем(анодированным, напыленным)

4) Кабели короткие, разъемы золоченные.

5) Формат низкого уровня сделайте при рабочей температуре и в рабочем положении.

6) Прокладывайте кабель вдали от источника шума(вентиляторов адаптера) системного блока.

Параметры и их значения.

1. Время успокоения вибрации головки при подходе ее к цилиндру - 1 мс.

2. Время поиска от дорожки к дорожке - время перемещения между соседними дорожками - 1-2 мс.

3. Производительность позиционера или среднее время доступа - 5 мс.

4. Скорость передачи данных. Определяется общей конструкцией и видом интерфейсов.

6.3. Оптические диски

CD-ROM. Накопители на компакт дисках.

CD-ROM - Compact Disc Read Only Memory. CD-ROM отличается от магнитного накопителя, как способом записи данных, так и форматом. Запись данных проводится путем выжигания лазерным лучом участков логической единицы, в результате чего образуются на поверхности маленькие углубления - питы (Pits). Питы располагаются на спиралевидной дорожке (рис. 6.10). Чтение проводится оптическим же методом, что в целом обеспечивает высокую долговечность дисков.

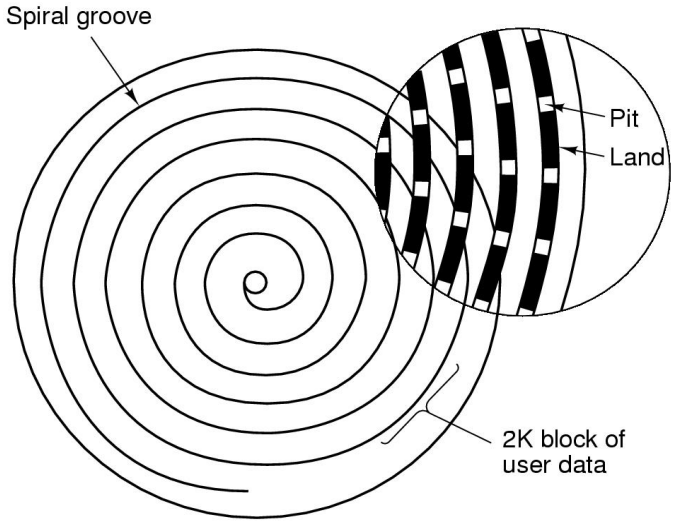


Рис. 6.10. Схема записи данных на CD-ROM диск.

Стандартная частота чтения аудиодорожки 44,1 кГц. Для движущихся картинок этой скорости недостаточно, имеются приводы с удвоенной, учетверенной скоростью вращения двигателя. Скорость вращения переключается.

Компакт-диск сделан из полимерного материала и покрыт слоем металла (обычно одним из сплавов алюминия). Поверх алюминия наносится еще один слой полимерного покрытия, защищающего металл (и, соответственно, записанные на диске данные) от

повреждений Другими словами, дорожка записи находится внутри диска. Все этикетки и надписи наносятся на верхнюю сторону диска, а считывание информации осуществляется с нижней поверхности. В процессе эксплуатации на нижней стороне компакт-диска могут возникать царапины, попадающие в область прохождения луча лазера. Однако на нижней поверхности луч еще не сфокусирован и имеет диаметр около 1мм, те намного превышает ширину царапины, поэтому практически не создается препятствий для прохождения луча и не возникают помехи воспроизведению.

Считывание информации с диска происходит за счет регистрации изменений интенсивности отраженного от алюминиевого слоя излучения маломощного лазера. Оно поступает на фотодатчик, величина электрического сигнала с которого зависит от того, отразился ли луч от гладкой поверхности или был рассеян на неоднородности. Углубления (питы), нанесенные на компакт-диск в процессе записи, и представляют из себя такие неоднородности. Более сильный сигнал с датчика соответствует ровной поверхности, более слабый- углублению. При движении дорожки записи под лучом формируется последовательность импульсных электрических сигналов, преобразуемых затем специализированным процессором либо в звук, либо в форму двоичных данных.

Глубина каждого штриха на диске равна 0.12 мкм, ширина- 0.6 мкм. Они расположены вдоль спиральной дорожки, расстояние между соседними витками которой составляет 1.6 мкм Длина питов и участков гладкой поверхности вдоль дорожки записи может изменяться в пределах от 0.9 до 3.3 мкм. Общая длина спирали составляет около 5 км

При записи данных на диск существует вероятность появления одной ошибки на 100000 бит, что недопустимо при цифровой записи. Поэтому используется корректирующий код, называемый кодом Рида-Соломона с перемежением. Кроме того, возможен случай, когда на дорожке записи достаточно долгое время отсутствуют питы, что делает невозможным бесконтактное слежение, осуществляемое по питам. Таким образом, необходимо ввести синхроимпульсы, которые становятся стандартными единицами измерения интервалов битов. Практически используется способ модуляции "восемь в четырнадцать".

Если на компакт-диске надо отыскать место, где записаны определенные данные, то его координаты предварительно считываются из оглавления диска, после чего считывающее устройство перемещается к нужному витку спирали и ждет появления определенной последовательности битов. Запись и воспроизведение данных с компакт-диска происходит с постоянной линейной скоростью перемещения дорожки относительно считывающего устройства. Это означает, что при считывании информации с внутренних витков спирали диск должен вращаться быстрее, а с внешних - медленнее. По этой причине все компакт-диски разбиваются на блоки (секторы), частота следования которых при записи и воспроизведении составляет 75 секторов в секунду.

На дисках, записанных в формате CD-DA, в каждом блоке содержится 2352 байта данных. На CD-ROM 304 из них используются для синхронизации, идентификации и записи кодов коррекции ошибок, а оставшиеся 2048 байта несут полезную информацию (рис. 12.10). Формат включает следующие поля: инициирующая дорожка, системная область, таблица содержимого тома (Volume Table of Contens - VTOC) и данные. Поскольку за секунду считывается 75 блоков, то стандартная скорость передачи данных для дисководов CD-ROM составляет 153600 байт/с = 150 Кбайт/с. При общем времени "звучания", равном 74 минутам, максимальная емкость CD-ROM составляет 650 Мб.

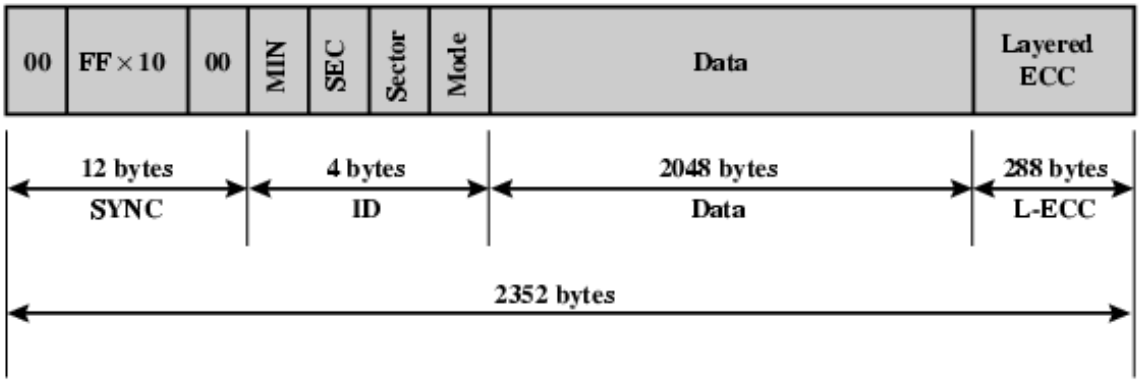


Рис. 6.11. Формат блока данных.

Устройство накопителя на CD-ROM.

Накопители на CD-ROM отличаются от проигрывателей музыкальных дисков, в основном, устройствами декодирования электрических сигналов. В звуковых проигрывателях записанные на компакт-диске цифровые данные преобразуются в аналоговые электрические сигналы, поступающие затем на стереоусилитель. При этом некоторые погрешности вполне терпимы- главное, чтобы они лежали за пределами чувствительности человеческого слуха. Но при считывании информации с CD-ROM это недопустимо- каждый бит должен быть воспроизведен совершенно достоверно. Поэтому в накопителях на CD-ROM используются весьма сложные алгоритмы поиска и исправления ошибок и довольно значительную часть всего объема CD-ROM занимают коды коррекции ошибок (к каждым 2048 полезным битам добавляется 288 контрольных), что позволяет снизить вероятность сбоя до приемлемой величины.

В самом упрощенном виде устройство накопителя на CD-ROM можно представить в следующем виде:

1. Серводвигатель по командам, поступающим со встроенного микропроцессора, смещает подвижную каретку с зеркалом к нужному витку спиральной дорожки записи. Поскольку дорожки смещены относительно центра, применяется специальное устройство для непрерывного слежения за дорожкой, называемое системой автотрекинга.

2. Луч от маломощного полупроводникового инфракрасного лазера проходит через разделительную призму и попадает на зеркало, направляющее его через фокусирующую линзу на поверхность диска. Биение поверхности диска создает трудности с фокусировкой луча, поэтому применяется система автофокусировки.

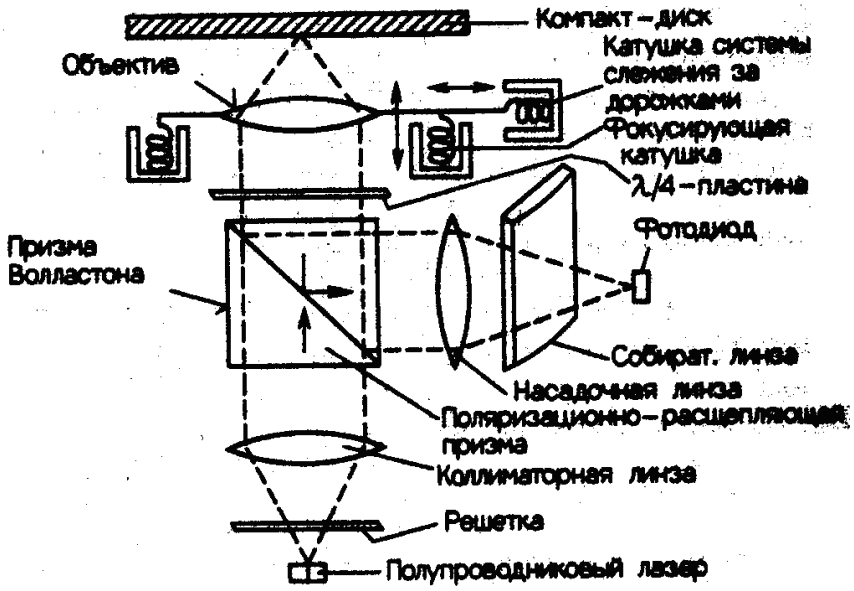


Рис. 6.12. Структурная схема оптической части CD дисковода.

3. Отраженный от диска луч снова фокусируется линзой, расположенной под диском, отражается от зеркала и возвращается на разделительную призму.

4 Разделительная призма перенаправляет отраженный луч на другую фокусирующую линзу, расположенную непосредственно перед фотодатчиком.

5 Фотодатчик преобразует падающее на него излучение в электрические сигналы.

6. Сигналы с фотодатчика декодируются встроенным микропроцессором и передаются в компьютер в виде данных

Для подключения накопителей на CD-ROM к компьютеру используют три разновидности интерфейсов: а именно: 1. SCSI/ASPI (Small Computer System Interface/Advanced SCSI Programming Interface); 2. IDE/ATAPI (Integrated Drive Electronics/AT Attachment Packet Interface); 3 специализированные нестандартные интерфейсы.

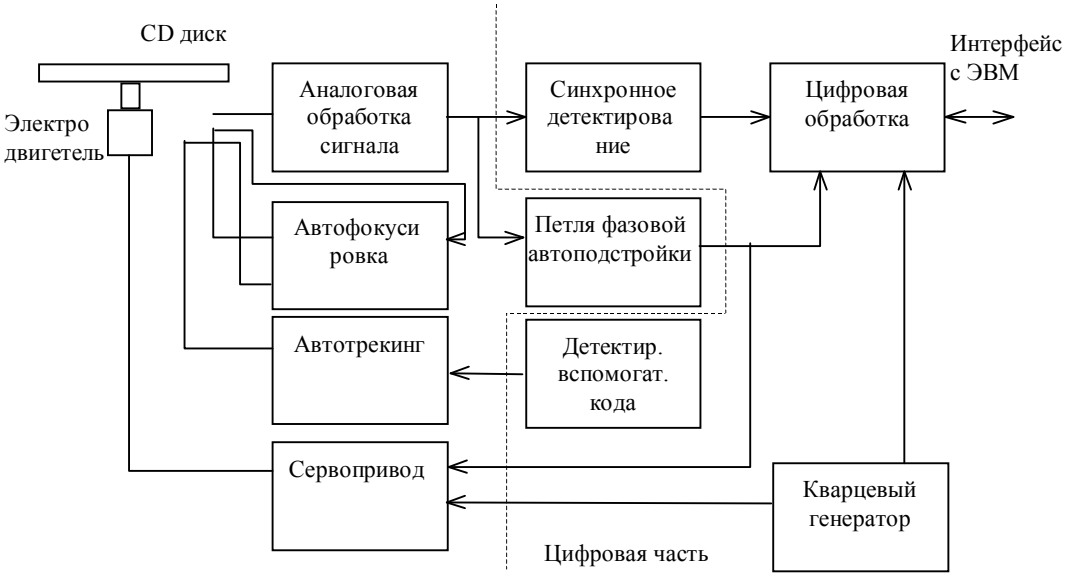


Рис. 6.13. Структурная схема CD дисковода.

DVD (Digital Versatile Disk) цифровой многоцелевой диск.

DVD диск одинаков по размерам с CD диском, однако вмещает до 17 Гб информации. Увеличение емкости диска и скорости чтения/записи информации достигнуто за счет двухкратного уменьшения размера элемента данных и расстояния между дорожками. Процесс чтения DVD дисков полностью идентичен процессу чтения CD-ROM. Однако для чтения и записи DVD дисков необходим лазер с другой длинной волны, лежащей не в инфракрасном, а красном диапазоне 650 и 635 нм вместо 780 нм у приводов CD-ROM, из-за чего и возникают проблемы при чтении компакт дисков на DVD-приводах.

Существующая спецификация предусматривает два типа совместимых между собой DVD дисков – DVD видеодиск и DVD диск для компьютерных программ. Емкость стандартного DVD диска составляет 4,7 Гб, что соответствует 133 мин высококачественного видеоизображения в формате MPEG-II со звуковым сопровождением на восьми языках в стандарте Dolby Digital (AC-3) и с субтитрами на 32 языках. Это предельная емкость однослойного и одностороннего диска. MPEG-II соответствует 500 линий на кадр. Dolby Digital (AC-3) является полностью цифровым форматом и предусматривает запись по шести раздельным каналам – передний левый и правый, задний левый и правый, фронтальный и басовый (алгоритм Dolby Surround 5.1). Частота дискретизации возросла с 44 до 48 или 98 кГц. Для воспроизведения DVD дисков нужен специальный DVD-привод, который отличается от CD дисководов в первую очередь типом лазера.

Структура DVD-дисков, принцип записи.

Основой записи и хранения данных на дисках DVD–RAM и DVD-RW является технология изменения фазового состояния вещества. При записи и считывании информации используется различие отражательной способности поверхности в

зависимости от того, находится ли она в кристаллическом или аморфном состоянии.

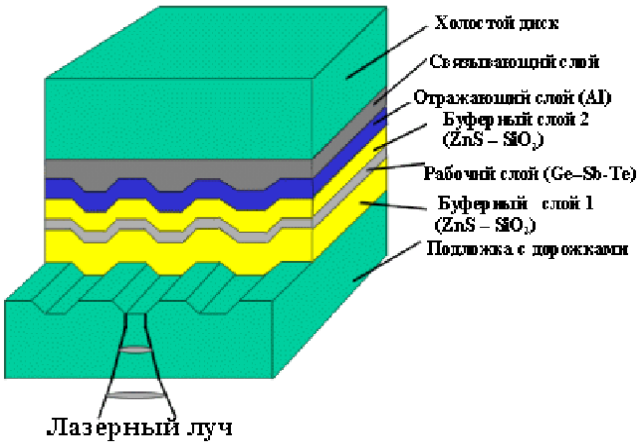


Рис. 6.14. Многослойная структура DVD диска.

При считывании информации с диска измеряется различие между темными аморфными и яркими прозрачными зонами. Эту технологию вполне можно назвать оптической - для чтения и записи достаточен всего лишь лазер. Послойная структура одной половины диска показана на рисунке 12.13.

Форматы DVD.

Это односторонние или двусторонние диски, с одним или двумя несущими информацию слоями на каждой стороне.

DVD-5 - односторонний диск с однослойной записью и емкостью 4,7 Гб. DVD состоит из 0,6 мм пленки, покрытой алюминием и наклеенной на чистую подложку. Технология напыления та же, что используется при изготовлении обычного CD. Алюминиевая пленка имеет толщину 55 нанометров, как и для аудио-CD и CD-ROM.

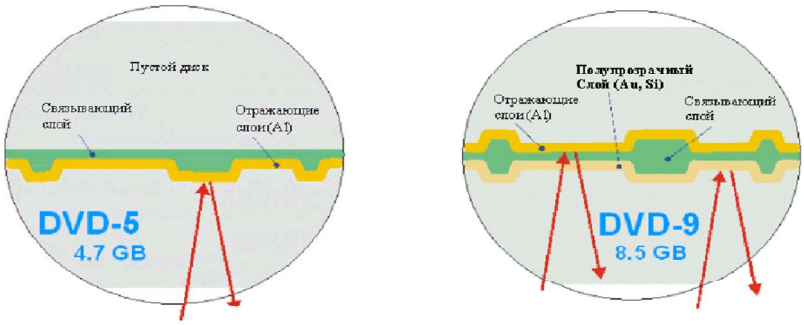


Рис. 6.15. Форматы DVD диска.

DVD-9 - это двухуровневый односторонний диск с емкостью 8,5 Гбайт. Для производства такого диска необходимо создать полупрозрачный слой, который отражает 18-30% лазерного излучения. Этого достаточно, чтобы можно было считывать информацию с верхнего слоя. И в то же время полупрозрачный слой будет пропускать достаточно излучения, чтобы сигнал от нижнего уровня с высокой отражательно способностью тоже читался. Информационные уровни разделяет высокооднородный клей (толщина клеевой прослойки составляет 40-70 микрон),

используемый для соединения двух половин диска. Это расстояние необходимо, чтобы различить сигнал, отраженный от одного и другого уровней. Структура DVD-9 показана на рисунке.

DVD-10 – однослойный двухсторонний диск с емкостью 9,4 Гб. В принципе это двойной DVD-5 без чистой подложки. Два диска, покрытых металлическими пленками, соединены вместе. Чтобы считывать информацию с двух сторон диска, используется один лазер.

Структура DVD-18 в принципе та же самая, как у DVD-9, но DVD-18 может читаться с обеих сторон.

6.4. Интерфейсы

Интерфейс - совокупность унифицированных аппаратурных, программных и конструктивных средств, необходимых для реализации взаимодействия различных функциональных элементов в системе при условиях, предписанных стандартом и направленных на обеспечение информационной, электрической и конструктивной совместимости.

Основным назначением интерфейса является унификация внутрисистемных и межсистемных связей и устройств сопряжения с целью эффективной реализации прогрессивных методов проектирования функциональных элементов вычислительных систем.

Основные функции интерфейса - обеспечение информационной, электрической и конструктивной совместимости между функциональными элементами системы.

Информационная совместимость - это согласованность взаимодействий функциональных элементов системы в соответствии с совокупностью логических условий, которые определяют структуру и состав унифицированного набора шин; процедур по реализации взаимодействия и последовательность их выполнения; способ кодирования и форматы данных, команд, адресной информации и информации состояния; временные соотношения между управляющими сигналами, ограничения на их форму и взаимодействие (перечисленные выше условия для большинства интерфейсов стандартизируются и лишь в отдельных случаях носят рекомендательный характер.

Электрическая совместимость - это согласованность статических и динамических параметров электрических сигналов в системе шин с учетом ограничений на пространственное размещение устройств интерфейса и техническую реализацию приемопередающих элементов.

Конструктивная совместимость - это согласованность конструктивных элементов интерфейса, предназначенных для обеспечения механического контакта электрических соединений и механической замены схемных элементов, блоков и устройств.

Функциональная организация определяет ряд основных функций интерфейса, которые необходимо реализовать для обеспечения информационной совместимости. К ним относятся: селекция информационного канала, синхронизация обмена

информацией, координация взаимодействия, буферное хранение информации, преобразование формы представления информации. Первые три функции возлагаются на канал управления, остальные на информационный канал.

Стык - место соединения устройств передачи данных (ГОСТ 23633- 79). Используется вместо понятия “интерфейс” для описания функций и средств сопряжения элементов.

Стандарты и рекомендации по стыку (например, С2) определяют характеристики: общие (скорость, последовательности передачи); функциональные и процедурные (категории цепей; правила их взаимодействия); электрические (параметры генераторов, приемников, нагрузок); механические (габариты, распределение контактов).

Протокол - строго заданная процедура или совокупность правил, определяющих способ выполнения заданных функций.

По способу передачи интерфейсы (стыки) разделяются на параллельные, последовательные и параллельно-последовательные. Этот признак характеризует размерность того элемента, который передается одновременно. Последовательный способ - бит, последовательно - параллельный - байт, чисто параллельный - одно машинное слово.

По принципу организации обмена информации во времени выделяются интерфейсы: с синхронной передачей (по меткам времени); с асинхронной передачей.

Последовательный интерфейс RS232.

Последовательные интерфейсы имеют наибольшее распространение, особенно с появлением USB, IrDA интерфейсов. Сначала рассмотрим интерфейс RS232, который используют при подключении к Com. Формат передаваемых данных показан на рис. 12.15. Данные начинают передаваться с младшего значащего разряда и завершаются передачей старшего разряда. Собственно данные (5, 6, 7 или 8 бит) сопровождаются стартовым битом, битом четности и одним или двумя стоповыми битами. Получив стартовый бит, приемник выбирает из линии биты данных через определенные интервалы времени. Очень важно, чтобы тактовые частоты приемника и передатчика были одинаковыми, допустимое расхождение - не более 10%. Скорость передачи по RS-232C может выбираться из ряда: 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 бит/с.

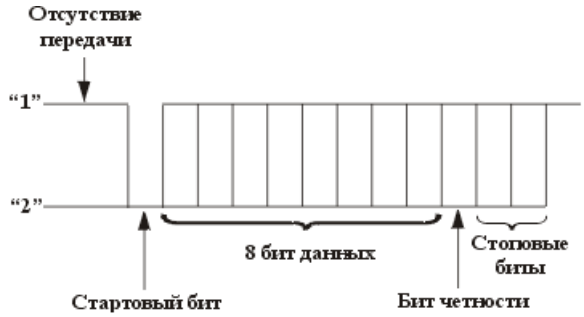


Рис. 6.16. Формат данных RS-232C.

Все сигналы RS-232C передаются специально выбранными уровнями, обеспечивающими высокую помехоустойчивость связи (рис. 12.16). Отметим, что данные передаются в инверсном коде (логической единице соответствует низкий уровень, логическому нулю - высокий уровень).

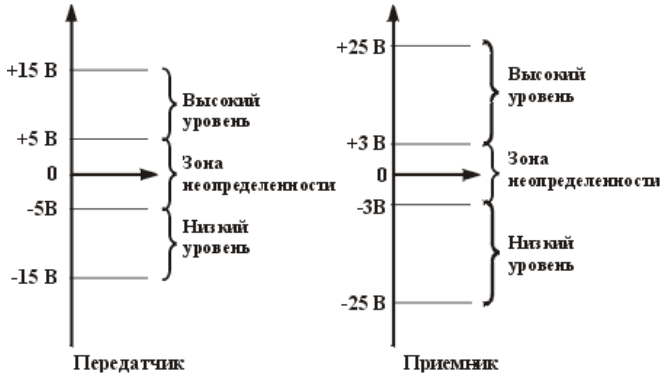


Рис. 6.17. Уровни сигналов в RS-232C.

Для подключения произвольного устройства к компьютеру через RS-232C обычно используют трех- или четырех-проводную линию связи (см. рис. 12.17), но можно задействовать и другие сигналы интерфейса. Линии дуплексной связи образуют два независимых контура.

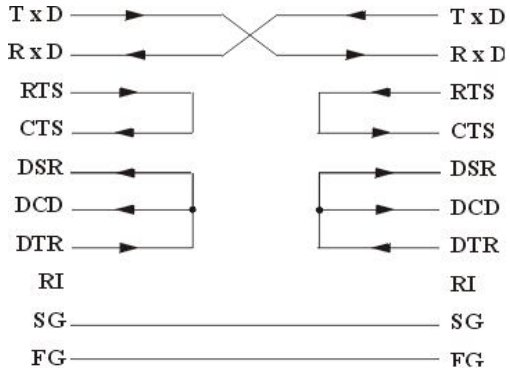


Рис. 6.17. Схема 4-проводной линии связи для RS-232C.

Табл. 6.1. Сигналы и номера контактов разъемов интерфейса RS-232C.

Наименование	Направление	Описание	Контакт (25-контактный разъем)	Контакт (9-контактный разъем)
DCD	IN	Carrie Detect (Определение несущей)	8	1
RXD	IN	Receive Data (Принимаемые данные)	3	2
TXD	OUT	Transmit Data (Передаваемые данные)	2	3
DTR	OUT	Data Terminal Ready (Готовность терминала)	20	4
GND	-	System Ground (Корпус системы)	7	5
DSR	IN	Data Set Ready (Готовность данных)	6	6
RTS	OUT	Request to Send (Запрос на отправку)	4	7
CTS	IN	Clear to Send (Готовность приема)	5	8
RI	IN	Ring Indicator (Индикатор)	22	9

Обмен по RS-232C осуществляется с помощью обращений по специально выделенным для этого портам COM1 (адреса 3F8h...3FFh, прерывание IRQ4), COM2 (адреса 2F8h...2FFh, прерывание IRQ3), COM3 (адреса 3F8h...3EFh, прерывание IRQ10),

COM4 (адреса 2E8h...2EFh, прерывание IRQ11). Форматы обращений по этим адресам можно найти в многочисленных описаниях. Перечислим только назначение портов.

3F8 регистр данных. Передатчик сохраняет байт в своем буфере данных (регистре) и в дальнейшем передает (принимает) его приемнику по интерфейсу. Если в регистре управления портом (3FB) седьмой бит выставлен в единицу, то регистр 3F8 выдает младший байт делителя частоты тактового генератора, старший байт в этом случае выводится в регистре 3F9. Получившееся слово характеризует скорость работы COM-порта.

3F9 Регистр управления прерываниями.

3FA Регистр идентификации прерывания определяет причину появления прерывания. Бит ноль, если он равен единице, сигнализирует об отсутствии прерываний, ожидающих обслуживание. В битах 1-2 зашифрованы следующие состояния: * 00 – прерывание генерируется при переполнении приемника, ошибке четности или формата данных, или при состоянии разрыв соединения, сбрасывается при чтении регистра состояния линии;

* 01 – данные приняты и доступны для чтения, сбрасывается после чтения из порта данных;

* 11 – устанавливается при изменении состояния входных линий CTS, RI, DCD и DSR.

3FB Регистр управления портом, позволяет задать параметры порта: размер передаваемого пакета данных, количество стоповых битов, тип контроля четности и т.д.

3FC Регистр управления устройством модуляции-демодуляции или просто модемом. Управляет состоянием линий DTR (бит 0) и RTS (бит 1), состоянием модемных резервных линий OUT1 (бит 2), OUT2 (бит 3) и запуском автодиагностики порта при замыкании выхода на вход (бит 4).

3FD Регистр состояния линии определяет в каком состоянии находится порт, и можно ли произвести нужное действие: * бит 0 – приемник получил данные, и их можно прочитать из регистра данных, при чтении из регистра 3F8 этот бит устанавливается в 0; * бит 1 - потеря данных, новый байт данных был получен, а старый не прочитан из регистра данных, и новый байт заместил в регистре данных старый байт;

* бит 2 – произошла ошибка четности;

* бит 3 – произошла ошибка синхронизации;

* бит 4 – обнаружен разрыв соединения;

* бит 5 – регистр переданных данных пуст, можно записать новый байт;

* бит 6 – данные переданы приемнику (то есть сдвиговый регистр, куда помещается байт из регистра данных, пуст);

* бит 7 – устройству не удалось связаться с компьютером в установленный срок.

3FE регистр состояния модема, имеет следующий формат:

* бит 0 – произошло изменение состояния линии CTS;
* бит 1 – произошло изменение состояния линии DSR;
* бит 2 – произошло изменение состояния линии IR;
* бит 3 – произошло изменение состояния линии DCD;
* бит 4 – состояние линии CTS;
* бит 5 – состояние линии DSR;
* бит 6 – состояние линии IR;
* бит 7 – состояние линии DCD;

Алгоритм работы с последовательным портом можно разбить на два этапа.

Инициализация порта.

- В регистре управления 3FB необходимо установить нужные параметры порта.
- В регистре 3F9 необходимо установить маски разрешения прерываний.
- Необходимо установить скорость передачи.

Передача и прием.

Надо постоянно проверять пятый бит регистра состояния порта 3FD, и если он становится равным единице, то можно писать байт в регистр данных 3F8. Перед тем как прочитать данные из регистра данных, необходимо убедиться, что там лежит новый байт. Делается это путем проверки нулевого бита регистра состояния порта 3FD. Если ты хочешь, чтобы твое устройство и управляющая программа работали безошибочно, то постоянно проверяй ошибки, то есть равны ли биты с 1 по 4 регистра состояния нулю. Если да, то все хорошо, если хоть один из них становится равным единице, принимай адекватные меры.

Интерфейс USB: описание и основы устройств сопряжения

Интерфейс USB (Universal Serial Bus - Универсальный Последовательный Интерфейс) предназначен для подключения периферийных устройств к персональному компьютеру. Позволяет производить обмен информацией с периферийными устройствами на трех скоростях (спецификация USB 2.0):

- Низкая скорость (Low Speed - LS) - 1,5 Мбит/с;
- Полная скорость (Full Speed - FS) - 12 Мбит/с;
- Высокая скорость (High Speed - HS) - 480 Мбит/с.

Для подключения периферийных устройств используется 4-жильный кабель: питание +5 В, сигнальные провода D+ и D-, общий провод.

Интерфейс USB соединяет между собой хост (host) и устройства. Хост находится внутри персонального компьютера и управляет работой всего интерфейса. Для того, чтобы к одному порту USB можно было подключать более одного устройства, применяются хабы (hub - устройство, обеспечивающее подключение к интерфейсу других устройств). Корневой хаб (root hub) находится внутри компьютера и подключен непосредственно к хосту. В интерфейсе USB используется специальный термин "функция" - это логически законченное устройств, выполняющее какую-либо

специфическую функцию. Топология интерфейса USB представляет собой набор из 7 уровней (tier): на первом уровне находится хост и корневой хаб, а на последнем - только функции. Устройство, в состав которого входит хаб и одна или несколько функций, называется составным (compaund device).

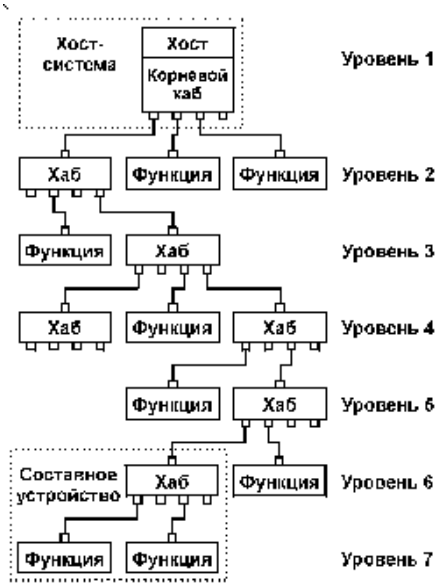


Рис. 6.18. Топология USB интерфейса.

Порт хаба или функции, подключаемый к хабу более высокого уровня, называется восходящим портом (upstream port), а порт хаба, подключаемый к хабу более низкого уровня или к функции называется нисходящим портом (downstream port).

Все передачи данных по интерфейсу инициируются хостом. Данные передаются в виде пакетов. В интерфейсе USB используется несколько разновидностей пакетов:

- пакет-признак (token paket) описывает тип и направление передачи данных, адрес устройства и порядковый номер конечной точки (КТ - адресуемая часть USB-устройства); пакет-признаки бывают нескольких типов: IN, OUT, SOF, SETUP;
- пакет с данными (data packet) содержит передаваемые данные;
- пакет согласования (handshake packet) предназначен для сообщения о результатах пересылки данных; пакеты согласования бывают нескольких типов: ACK, NAK, STALL.

Таким образом, каждая транзакция состоит из трех фаз: фаза передачи пакета-признака, фаза передачи данных и фаза согласования.

В интерфейсе USB используются несколько типов пересылок информации.

- Управляющая пересылка (control transfer) используется для конфигурации устройства, а также для других специфических для конкретного устройства целей.
- Потоковая пересылка (bulk transfer) используется для передачи относительно большого объема информации.
- Пересылка с прерыванием (iterrupt transfer) испольуется для передачи относительно небольшого объема информации, для которого важна своевременная его

пересылка. Имеет ограниченную длительность и повышенный приоритет относительно других типов пересылок.

- Изохронная пересылка (isochronous transfer) также называется потоковой пересылкой реального времени. Информация, передаваемая в такой пересылке, требует реального масштаба времени при ее создании, пересылке и приеме.

Потоковые пересылки характеризуются гарантированной безошибочной передачей данных между хостом и функцией посредством обнаружения ошибок при передаче и повторного запроса информации.

Когда хост становится готовым принимать данные от функции, он в фазе передачи пакета-признака посылает функции IN-пакет. В ответ на это функция в фазе передачи данных передает хосту пакет с данными или, если она не может сделать этого, передает NAK- или STALL-пакет. NAK-пакет сообщает о временной неготовности функции передавать данные, а STALL-пакет сообщает о необходимости вмешательства хоста. Если хост успешно получил данные, то он в фазе согласования посылает функции ACK-пакет. В противном случае транзакция завершается.

Когда хост становится готовым передавать данные, он посылает функции OUT-пакет, сопровождаемый пакетом с данными. Если функция успешно получила данные, он отсылает хосту ACK-пакет, в противном случае отсылается NAK- или STALL-пакет.

Пересылки с прерыванием могут содержать IN- или OUT-пересылки. При получении IN-пакета функция может вернуть пакет с данными, NAK-пакет или STALL-пакет. Если у функции нет информации, для которой требуется прерывание, то в фазе передачи данных функция возвращает NAK-пакет. Если работа КТ с прерыванием приостановлена, то функция возвращает STALL-пакет. При необходимости прерывания функция возвращает необходимую информацию в фазе передачи данных. Если хост успешно получил данные, то он посылает ACK-пакет. В противном случае согласующий пакет хостом не посылается.

В связи с тем, что в интерфейсе USB реализован сложный протокол обмена информацией, в устройстве сопряжения с интерфейсом USB необходим микропроцессорный блок, обеспечивающий поддержку протокола. Поэтому основным вариантом при разработке устройства сопряжения является применение микроконтроллера, который будет обеспечивать поддержку протокола обмена. В настоящее время все основные производители микроконтроллеров выпускают продукцию, имеющую в своем составе блок USB.

Табл. 6.2. Сигналы и номера контактов разъема USB.

Номер контакта	Назначение	Цвет провода
1	V BUS	Красный
2	D-	Белый
3	D+	Зеленый
4	GND	Черный
Оплетка	Экран	Оплетка

Интерфейс IrDA?

В 1979 году компания Hewlett-Packard объявила о начале продаж нового калькулятора, главной особенностью которого являлось наличие у него инфракрасного порта для вывода информации на печать. После этого в течение нескольких лет разработчиками электронного оборудования была предложена целая серия приборов и устройств, использующих для передачи информации открытый оптический канал в инфракрасном диапазоне. Однако, все эти устройства не могли получить широкого распространения вследствие своей несовместимости. Поэтому в 1993 году была основана Infrared Data Association (IrDA), международная некоммерческая организация, ставящая своей целью разработку единых стандартов, используемых для организации инфракрасных линий передачи информации.

Первым стандартом, принятым IrDA, был, так называемый, Serial Infrared standart (SIR). Данный стандарт позволял обеспечивать передачу информации со скоростью 115,2 kb/s. В 1994 году IrDA опубликовала спецификацию на общий стандарт, получивший название IrDA-standart, который включал в себя описание Serial Infrared Link (дословно - Последовательная Инфракрасная линия связи), Link Access Protocol (IrLAP) (Протокол доступа) и Link Management Protocol (IrLMP) (Протокол управления). Уже в 1995 году несколько лидеров на рынке электроники выпустили серию продуктов, использующих для передачи информации по открытому оптическому каналу IrDA-standart. И, наконец, в ноябре 1995 года Microsoft Corporation заявила о внесении программного обеспечения, обеспечивающего инфракрасную связь, использующую IrDA-standart, в стандартный пакет операционной системы Windows'95. В настоящее время IrDA-standart - самый распространенный стандарт для организации передачи информации по открытому инфракрасному каналу.

В общем виде схема организации IrDA - канала выглядит примерно так, как показано на рис. 12.12. Канал передачи данных состоит из двух основных элементов: микросхемы, обеспечивающей модуляцию и демодуляцию поступающего двоичного сигнала согласно определенного алгоритма, и инфракрасного (ИК-) приемно-передающего модуля.

Рис. 6.19. Типовая блок-схема организации IrDA-канала

Рассмотрим SIR-стандарт, обеспечивающий скорость передачи информации 115,2kb/s. В данном стандарте используется так называемая модуляция "3/16".

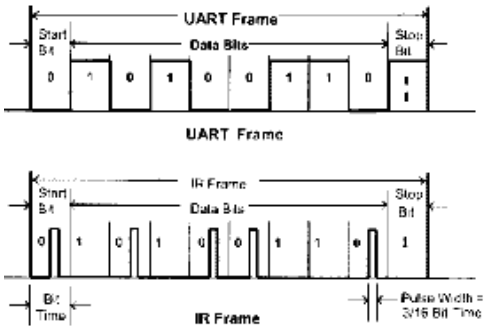


Рис. 6.20. Принцип модуляции "3/16", используемый в SIR-стандарте

Длительность импульса, подаваемого на приемно-передающий модуль равна 3/16 от длительности номинального бита данных. Кроме того, при SIR-модуляции используется инверсия бита данных. Эти преобразования обеспечиваются первым основным элементом схемы - модулирующей микросхемой. В зависимости от используемого интерфейса (шины данных) применяются различного рода чипы.

Стандартный параллельный интерфейс на PC

Основным назначением интерфейса Centronics (аналог-ИРПР-М) является подключение к компьютеру принтеров различных типов. Поэтому распределение контактов разъема, назначение сигналов, программные средства управления интерфейсом ориентированы именно на это использование. В то же время с помощью данного интерфейса можно подключать к компьютеру и другие внешние устройства, имеющие разъем Centronics, а также специально разработанные УС.

Назначение 36 контактов разъема Centronics приведено в таблице.

Таблица 6.3. Назначение контактов разъемов Centronics

Вывод	Наименование		Направление	Описание
1	/STROBE		Out	Strobe (Строб)
2	D0	Out	Data Bit 0	
3	D1	Out	Data Bit 1	
4	D2	Out	Data Bit 2	
5	D3	Out	Data Bit 3	
6	D4	Out	Data Bit 4	
7	D5	Out	Data Bit 5	
8	D6	Out	Data Bit 6	
9	D7	Out	Data Bit 7	
10	/ACK	In	Acknowledge (Подтверждение)	
11	BUSY	In	Busy (Занято)	
12	PE	In	Paper End (Конец бумаги)	
13	SEL	In	Select (Выбор)	
14	/AUTOFD		Out	Autofeed (Перевод строки)
15	/ERROR		In	Error (Ошибка)
16	/INIT	Out	Initialize (Инициализация)	
17	/SELIN		Out	Select In (Выбор)
18	GND	-	Signal Ground (Корпус)	
19	GND	-	Signal Ground (Корпус)	
20	GND	-	Signal Ground (Корпус)	

21 GND - Signal Ground (Корпус)
22 GND - Signal Ground (Корпус)
23 GND - Signal Ground (Корпус)
24 GND - Signal Ground (Корпус)
25 GND - Signal Ground (Корпус)

Сигналы Centronics имеют следующее назначение (тип выходных каскадов для всех сигналов - ТТЛ):

D0...D7 - 8-разрядная шина данных для передачи из компьютера в принтер. Логика сигналов положительная.

-STROBE - сигнал стробирования данных. Данные действительно как по переднему, так и по заднему фронту этого сигнала. Сигнал говорит приемнику (принтеру), что можно принимать данные.

-ACK - сигнал подтверждения принятия данных и готовности приемника (принтера) принять следующие данные. То есть здесь реализуется асинхронный обмен.

BUSY - сигнал занятости принтера обработкой полученных данных и неготовности принять следующие данные. Активен также при переходе принтера в состояние off-line или при ошибке, а также при отсутствии бумаги. Компьютер начинает новый цикл передачи только после снятия -ACK и после снятия BUSY.

-AUTO FD - сигнал автоматического перевода строки. Получив его, принтер переводит каретку на следующую строку.

Остальные сигналы не являются, вообще говоря, обязательными.

PE - сигнал конца бумаги. Получив его, компьютер переходит в режим ожидания. Если в принтер вставить лист бумаги, то сигнал снимается.

SLCT - сигнал готовности приемника. С его помощью принтер говорит о том, что он выбран и готов к работе. У многих принтеров имеет постоянно высокий уровень.

-SLCT IN - сигнал принтеру о том, что он выбран и последует передача данных.

-ERROR - сигнал ошибки принтера. Активен при внутренней ошибке, переходе принтера в состояние off-line или при отсутствии бумаги. Как видим, здесь многие сигналы дублируют друг друга.

-INIT - сигнал инициализации (сброса) принтера. Его длительность не менее 2,5 мкс. Происходит очистка буфера печати.

Временная диаграмма цикла передачи данных представлена на рисунке .

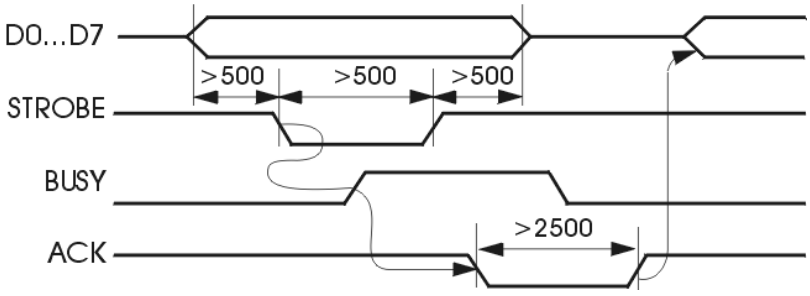


Рис.6.21. Временные диаграммы цикла передачи данных в Centronics
(все временные интервалы в наносекундах).

Перед началом цикла передачи данных компьютер должен убедиться, что сняты сигналы BUSY и -ACK. После этого выставляются данные, формируется строб, снимается строб, и снимаются данные. Принтер должен успеть принять данные с выбранным темпом. При получении строба принтер формирует сигнал BUSY, а после окончания обработки данных выставляет сигнал -ACK, снимает BUSY и снимает -ACK. Затем может начинаться новый цикл.

Все сигналы интерфейса Centronics передаются в уровнях ТТЛ и рассчитаны на подключение одного стандартного входа ТТЛ. Максимальная длина соединительного кабеля по стандарту - 1,8 м. Как видно из таблицы, в интерфейсе Centronics для подключения к компьютеру произвольных УС мы можем использовать 17 линий, назначение которых можно выбирать по своему усмотрению.

Формирование и прием сигналов интерфейса Centronics производится путем записи и чтения выделенных для него портов ввода/вывода. В компьютере может использоваться три порта Centronics, обозначаемых LPT1 (базовый адрес 378h), LPT2 (базовый адрес 278h) и LPT3 (базовый адрес 3BCh). При этом LPT3 используется в том случае, когда контроллер принтера находится на плате графического адаптера Hercules или EGA. Прерывания портов принтеров (IRQ5 для LPT2 и IRQ7 для LPT1) используются очень редко. Базовый адрес порта используется для передачи принтеру байта данных. Установленные на линиях данные можно считать из этого же порта.

Следующий адрес (базовый + 1) служит для чтения битов состояния принтера (бит 3 соответсвует сигналу -EEROR, бит 4 - сигналу PE, бит 6 - сигналу -ACK, бит 7 - сигналу BUSY). Последний используемый адрес (базовый + 2) предназначается для записи битов управления принтером (бит 0 соответствует сигналу -STROBE, бит 1 - сигналу -AUTO FD, бит 2 - сигналу -INIT, бит 3 - сигналу -SLCT IN и наконец бит 4, равный единице, разрешает прерывание от принтера).

Каждый параллельный адаптер обслуживается несколькими портами ввода/вывода. Перечислим порты ввода/вывода.

Порт 378H - Доступен для чтения и записи. Предназначен для вывода данных.

Порт 37AH - Доступен для чтения и записи. Предназначен для управления принтером

Порт 379H - Доступен для чтения. Предназначен для чтения битов состояния принтера.

Байты данных для вывода устанавливаются в порт 378H. Затем через 0,5 мс линия СТРОБ переводится с высокого в низкий уровень, после чего происходит запись информации во внешнее устройство. Входные сигналы считываются с порта 379H.

Режимы порта

SPP (Standart Paralell Port) — разработчик фирма Centronics, однонаправленный порт

EPP (Enhanced Parallel Port) — разработчики, компании Intel, Xircom и Zenith Data Systems — двунаправленный порт, со скоростью передачи данных до 2Мб/сек, осталась совместимость с SPP.

ECP (Extended Capabilities Port) — разработчики, компании Hewlett-Packard и Microsoft, совместим с SPP, в дополнение появились такие возможности, как наличие аппаратного сжатия данных, наличие буфера и возможность работы в режиме DMA.

Регистры EPP-порта

Имя регистра	Смещение	Режим	R/W	Описание
SPP Data Port	+0	SPP/EPP	W	Регистр данных стандартного порта
SPP Status Port	+1	SPP/EPP	R	Регистр состояния стандартного порта
SPP Control Port	+2	SPP/EPP	W	Регистр управления стандартного порта
EPP Address Port	+3	EPP	R/W	Регистр адреса EPP. Чтение или запись в него генерирует связанный цикл чтения или записи адреса EPP
EPP Data Port	+4	EPP	R/W	Регистр данных EPP. Чтение (запись) генерирует связанный цикл чтения (записи) данных EPP
Not Defined	+5...+7	EPP	N/A	В некоторых контроллерах могут использоваться для 16—32-битных операций ввода/вывода

Заключение

В учебном пособии рассмотрены современные процессоры и мультипроцессорные системы, а также периферийные устройства, много внимания уделено вопросам программирования периферийных устройств, создания встраиваемых приложений, использования системных ресурсов компьютера.

Компьютер представлен многослойным пирогом, в котором на нижнем уровне находятся вентильные схемы АЛУ и элементы памяти, на среднем – простые одноядерные процессоры со своими КЭШ, а на верхнем – современные вычислительные многоядерные системы, включающие все новые и новые элементы. Многие узлы компьютера программно не доступны, однако знания о них позволяет создавать более эффективные программно-аппаратные комплексы.

В учебном пособии значительное место занимают элементы, связанные с работой аудио- и видео - системами. Это объясняется широкой востребовательностью таких систем на предприятиях и в повседневной жизни. Большинство персональных компьютеров и ноутбуков оснащены звуковыми видеокамерами, которые стали неотъемлемой их частью.

Существенным дополнением к учебному пособию является пособие по лабораторному практикуму. В нем можно получить больше информации по программированию ЭВМ и периферийных устройств.

Библиографический список

1. Мелехин В. Ф., Павловский Е. Г. Вычислительные машины - М. Академия, 2013
2. Антонов А.С. Параллельное программирование с использованием технологии OpenMP: Учебное пособие.-М.: Изд-во МГУ, 2009. – 77 с. ISBN 978-5-211-05702-9
3. Mark D. Pesce Programming Microsoft Direct Show for Digital Video/Television Microsoft Press. 2003. ISBN: 0735618216
4. Горнаков С.Г. DirectX 9. Уроки программирования на C++ БХВ-Петербург, 2005.- 400 с. ISBN: 5-94157-482-7
5. Боресков А.В., Харламов А.А. Основы работы с технологией CUDA. ДМК Пресс, 2010.- 234с.
6. The Pentium ® Processor Family Developer's Manual: Pentium ® Processor. Intel Corp. Order Number 241428.
7. MPC603e & EC603e™ RISC Microprocessors User's Manual with Supplement for PowerPC 603™ Microprocessor. Motorola Inc.
8. Intel® IA-64 Architecture Software Developer Manual. Vol.1, Vol.2, Vol.3, Vol.4. Intel Corp.

Электронные ресурсы

9. Молодяков С.А. ЭВМ и периферийные устройства: учебное пособие. Ч.1. Основы организации ЭВМ; СПб., 2012.- <URL:http://dl.unilib.neva.ru/dl/2759.pdf> .
10. В. Шнитман Современные высокопроизводительные компьютеры http://citforum.amursu.ru/hardware/svk/contents.shtm
11. В.Шнитман Архитектура процессоров UltraSPARC http://citforum.amursu.ru/hardware/articles/art_3.shtml
12. Лазовский Л. ПЗС: прецизионный взгляд на мир http://www.digitalware.ru/static/dwscanners/DWRscanhand010608.asp
13. Павлова М. И. Визуализация, компьютерная графика и WEB-дизайн http://www.csa.ru/~zebra/
14. http://www.ferra.ru/online/processors
15. www.transmeta.com/crusoe
16. http://www.geek.com/procspec
17. http://www.compdoc.ru/comp/cpu/
18. http://www.parallel.ru/
19. http://osp.ru/
20. http://kazus.ru/articles/index.html
21. http://www.xoro.ru/docs/
22. http://www.xoro.ru/docs/dvd/

Молодяков Сергей Александрович,

АРХИТЕКТУРА ЭВМ
Часть 2.

Современные процессоры и мультипроцессорные системы. Периферийные устройства

Лицензия ЛР № 020593 от 07.08.97

Налоговая льгота – Общероссийский классификатор продукции
ОК 005-93, т. 2; 95 3005 – учебная литература

Подписано в печать _____2014. Формат 60×84/16. Печать цифровая
Усл. печ. л. ___. Уч.-изд. л. _____. Тираж . Заказ

Отпечатано с готового оригинал-макета, предоставленного авторами в цифровом типографском центре Издательства Политехнического университета:
195251, Санкт-Петербург, Политехническая ул., 29.
Тел. (812) 540-40-14
Тел./факс: (812) 927-57-76