МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский Нижегородский государственный
университет им. Н.И. Лобачевского»

П.Д. Басалин
А.Е. Тимофеев

СХЕМОТЕХНИКА И ОРГАНИЗАЦИЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Учебное пособие

Рекомендовано учебно-методической комиссией Института информационных технологий,
математики и механики для студентов ННГУ, обучающихся по направлению подготовки
09 03 03 – «Прикладная информатика»

Нижний Новгород
2022

УДК 621.3
ББК 32.971.321
Б27

Б27 Басалин П.Д., Тимофеев А.Е. Схемотехника и организация вычислительных систем: Учебное пособие. – Нижний Новгород: Нижегородский госуниверситет, 2022. – 123 с.

Рецензенты: д.т.н., профессор **В.П. Хранилов**
к.т.н., доцент **С.Н. Карпенко**

Учебное пособие посвящено рассмотрению базовых принципов организации, логических, схемотехнических и конструктивно-технологических основ современной вычислительной техники, расширяя это представление до глубины уровней аналоговой и цифровой схемотехники, микроархитектуры памяти, процессоров и интерфейсов вычислительных систем. При этом основное внимание уделено архитектуре, функциональным узлам и элементной базе современных цифровых вычислительных систем, развивающихся в направлении увеличения тактовых частот процессоров, конвейеризации выполнения инструкций, распараллеливания вычислительных структур на уровнях команд и алгоритмов, а также ряда других архитектурных усовершенствований, способствующих повышению эффективности вычислительного процесса.

Материал пособия не предполагает глубокого проникновения в области микроэлектроники, аналоговой и цифровой схемотехники, характерные для вузов технической направленности. Для его освоения необходимо знание основ электроники и электротехники (практически на уровне школьной программы) и алгебры логики.

Для студентов, изучающих вычислительные системы как средства технического обеспечения информационных систем различного назначения (САПР, АСУ, АСНИ, АОС и т.д.) и как объекты проектирования в САПР цифровой микроэлектронной аппаратуры.

УДК 621.3
ББК 32.971.321

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Развитие и внедрение в практику новых информационных технологий неразрывно связано с использованием в качестве основы для их реализации современной вычислительной техники, отличающейся широтой разнообразия и быстротой смены поколений. Данное учебное пособие преследует цель, дать общее представление о базовых принципах построения, логической организации и схемотехнических особенностях современных вычислительных систем, которое может послужить фундаментом (отправной точкой) для дальнейшего самообразования в области вычислительной техники.

Материал пособия включает шесть разделов.

В *первом разделе* вводятся исходные понятия, и рассматривается общее многоуровневое представление архитектуры вычислительной системы. Первые шесть уровней (всего уровней 10) представляют аппаратную составляющую архитектуры вычислительной системы, являющуюся предметом изучения в рамках данного пособия. Это базовый естественно-математический уровень, аналоговый уровень, уровень цифровой схемотехники, системотехнический уровень, микроархитектурный уровень и уровень машинных команд.

Второй раздел связан с рассмотрением на аналоговом уровне принципиальных схем базовых логических элементов и типичных для них выходов, определяющих специфику их применения в составе функциональных узлов цифровой аппаратуры.

Третий раздел учебного пособия посвящен описанию типовых узлов комбинационной и последовательностной логики, из которых складывается цифровая аппаратура. Среди комбинационных узлов дешифратор, мультиплексор, демультиплексор, компаратор, схема сдвига, схема контроля четности, сумматор, арифметико-логическое устройство. Последовательностная логика представлена схемами запоминающих элементов (конденсатор с ключевым транзистором, защелки, триггеры), регистрами различного назначения, регистровым файлом и двоичным счетчиком.

Четвертый раздел представляет многоуровневую архитектуру памяти цифровой вычислительной системы. После определения назначения, основных параметров и общей иерархии запоминающих устройств рассматриваются:

- базовые принципы организации адресной памяти и обобщенные ее структуры 2D, 3D и 2DM;

- микросхемы и модули динамической памяти, включая современные альтернативные архитектуры синхронной памяти DDR SDRAM и RDRAM;

- статическая память и ее применение для кэширования основной памяти вычислительной системы;

- энергонезависимая память.

Пятый раздел посвящен рассмотрению архитектур микропроцессоров, начиная с общих базовых принципов их организации (операционный автомат, управляющий автомат, микрооперации, микрокоманды, микропрограммы, жесткая логика, программируемая логика и т.п.) и продолжая до архитектуры уровня машинных команд, форматы которых (на определенном уровне детализации) приведены для сравнения архитектур RISK и CISK . Заканчивается раздел общим представлением микроархитектурного уровня процессора с рассмотрением технологий конвейеризации, прогнозирования ветвлений, суперскалярных вычислений, переименования регистров.

Шестой раздел учебного пособия связан с системотехническим уровнем общего представления архитектуры вычислительной системы. В нем излагаются основы магистрально-модульного принципа организации цифровых вычислительных систем. Дается описание функционального назначения и конкретных характеристик шинных интерфейсов, задействованных в современных вычислительных системах.

1. КОНЦЕПЦИЯ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

1.1. Исходные понятия и определения

Развитие и внедрение в практику новых информационных технологий рассматривается как комплексный системный подход к автоматизации процессов получения, хранения, обработки, передачи и использования информации. Это неизбежно приводит к более широкой трактовке многих понятий. Так под *вычислениями* уже подразумеваются *любые процедуры оперирования с информацией хотя бы в одном из перечисленных выше аспектов*. Распознать (вычислить) возникшую нештатную ситуацию, спланировать (вычислить) очередной шаг процесса проектирования сложного изделия, «вычислить» преступника – вот некоторые из примеров процедур, иллюстрирующих широту понятия вычислений.

В более широкой трактовке следует рассматривать и понятие *вычислительной системы* (ВС). Очевидно, что оно должно охватывать *любые взаимосвязанные аппаратные (технические), лингвистические и системные программные средства, обеспечивающие возможность организации вычислений*. Это могут быть простейшие системы передачи данных или сложнейшие вычислительные комплексы в виде взаимосвязанной совокупности одного или многих процессоров (возможно базирующихся на различных принципах организации вычислений), многоуровневой памяти, периферийных устройств, а также лингвистических и системных программных средств, обеспечивающих возможность реализации современных информационных технологий в различных предметных (проблемных) областях.

Данное определение настолько обще, что под него можно подвести и саму природу, рассматривая ее как самоорганизующуюся ВС, управляющую процессами, происходящими в реальном мире. Только при этом следует уточнить, что понимается под информацией, а также аппаратными, лингвистическими и системными программными средствами такой ВС.

Если вводить понятие информации посредством интенсионала, т.е. через категории более высокого уровня абстракции, то под ней можно понимать *универсальное свойство материи отражать события окружающего мира*. Конкретные проявления этого свойства и распространение их осуществляются посредством материальных полей в соответствии с объективными законами преобразования и сохранения энергии. Информационное поле абстрактно, ибо законы, аналогичные законам сохранения энергии (материи) в нем не

существуют. Появившись в одном месте, информация распространяется по окружающему миру, не обязательно исчезая в местах своего проявления.

Естественные ВС, созданные природой и обеспечивающие гармоничное взаимодействие ее составных частей, зачастую во многом определяют базовые принципы организации искусственных ВС, создаваемых человеком для решения своих информационных проблем.

Эффективность, надежность и гибкость любой искусственной ВС во многом определяются ее *архитектурой*, под которой будем понимать *совокупность базовых принципов и способов организации вычислений, отражающих суть протекающих в системе процессов и механизмов управления ими на различных уровнях детализации*.

Аналоговая ВС, оперируя с непрерывными физическими сигналами, соответствующими переменным реализуемых математических моделей, обладает колоссальным быстродействием в силу *существенного параллелизма* вычислений (все функциональные блоки аналогового процессора принимают одновременное участие в вычислительном процессе). Однако ей присущ ряд недостатков. Это и отсутствие универсальности, гибкости в плане переориентации системы на другую задачу (класс задач). Это проблемы обеспечения требуемой точности вычислений, которые могут возникать на определенных классах задач. Это низкая помехоустойчивость системы, функционирующей в условиях существенного влияния внешних факторов (температурных режимов, уровня радиации и т.п.).

Последовательный цифровой принцип архитектуры фон Неймана, базирующейся на формальном аппарате алгебры логики, отличается универсальностью. Он обеспечивает необходимую точность вычислений, высокую надежность и как пути повышения производительности использует:

- увеличение тактовых частот системы;
- конвейеризацию выполнения инструкций;
- распараллеливание вычислительных структур на уровнях микрокоманд, команд и алгоритмов,

а также ряд других архитектурных усовершенствований, способствующих повышению эффективности вычислительного процесса.

Универсальность цифрового процессора, возможность обеспечения практически любой точности, более высокая помехоустойчивость определяют то предпочтение, которое отдано в современной практике цифровым ВС.

Вместе с тем с точки зрения многих приложений заслуживает внимания успешно развивающийся альтернативный подход к организации вычислений,

базирующийся на применении *искусственных нейронных сетей* (ИНС) в качестве основы существенно параллельных ВС — *нейрокомпьютеров*. В идеальной (с точки зрения быстродействия) реализации это аналоговый подход с присущей ему высокой скоростью вычислений, но вместе с тем предоставляющий возможность достижения на определенных классах задач приемлемой точности путем:

- повышения мощности нейронной сети (увеличения количества слоев, числа нейронов в них);
- совершенствования ее архитектуры;
- применения эффективных процедур обучения;
- интеграции с элементами цифровой вычислительной техники, т.е. создания *гибридной* архитектуры, сочетающей в себе аналоговый и цифровой принципы организации вычислений.

Последнее позволяет также придать определенную универсальность, гибкость нейрокомпьютеру, хотя и уводит от существенного параллелизма.

Нейросетевая вычислительная парадигма обусловила появление нового типа представления задач. К традиционным перечисляющему представлению, представлению в пространстве состояний, иерархическому представлению и их комбинациям добавилось *представление задач в нейросетевом базисе*. Оно позволяет:

- выделить естественный для данного представления существенный параллелизм задачи;
- применить для ее формализации известные методы обучения искусственных нейронных сетей;
- определиться с архитектурой ВС, наиболее рациональной для решения рассматриваемого класса задач.

В настоящее время устойчиво наблюдается развитие цифровых ВС в направлении архитектур параллельного действия [1], [2]. Параллелизм, как способ повышения производительности ВС, вводится на разных уровнях организации вычислений. При рассмотрении концепции распараллеливания вычислительных процессов на уровне алгоритмов оперируют понятием *вычислительной структуры*, под которой понимается совокупность процедур конкретного содержания и определенного отношения их следования в вычислительном процессе. Как последовательно, так и параллельно выполняемые процедуры могут быть *примитивными* (в виде одной или нескольких последовательных команд) или *сложными* (состоящими из других процедур). Соответственно, возникает понятие *уровня детализации па-*

раллелизма — от *мелкозернистой* детализации (в базисе примитивных процедур) до *крупнозернистой* (в базисе сложных процедур).

Различные виды вычислительных структур предъявляют свои требования к архитектуре ВС, используемой для их реализации.

Последовательную (*линейную*) вычислительную структуру вполне устраивает классическая архитектура фон Неймана (архитектура SISD – Single Instruction stream Single Data stream − по классификации Флинна). Ее можно совершенствовать в направлении распараллеливания вычислительного процесса на уровне команд (Instruction-Level-Parallelism — ILP), т.е. построения эффективных механизмов конвейеризации выполнения инструкций и организации *суперскалярных* архитектур процессоров (с параллельными конвейерами или параллельными частями одного главного конвейера).

Последовательно-групповая вычислительная структура (процедуры в группах выполняются параллельно, а группы — последовательно) в зависимости от состава групп (однородный — из одинаковых процедур или неоднородный) может ориентироваться на различные типы архитектур ВС параллельного действия.

В случае одинаковых примитивных процедур внутри групп рационально применение архитектуры SIMD - Single Instruction stream Multiple Data stream. Она реализована для некоторых специальных типов векторных команд в обычных процессорах, в векторных (PVP — Parallel Vector Processor) и матричных (MPP — Massively Parallel Processor) процессорах.

В случае разных вычислительных процедур внутри групп целесообразно применение MIMD-архитектуры (Multiple Instruction stream Multiple Data stream), существующие разновидности которой делятся на два класса: *мультипроцессоры* и *мультикомпьютеры*.

Под *мультипроцессором* понимается параллельная ВС, в которой все процессоры используют единую (разделяемую) физическую память. Через нее они обмениваются информацией с помощью операторов записи и считывания.

Мультикомпьютер – это параллельная ВС с распределенной памятью, каждый процессор которой имеет свою, только ему доступную память. Взаимодействие процессоров в такой системе осуществляется путем передачи сообщений по сети межсоединений.

Программное обеспечение мультикомпьютера по сравнению с мультипроцессором имеет более сложную структуру, однако его аппаратная реализация обходится дешевле. Некоторый компромисс обеспечивается путем создания архитектур, которые наряду с распределенной памятью имеют и

совместно используемую память на определенном уровне организации ВС (в аппаратном обеспечении, на уровне операционной системы или на уровне системы программирования). При этом они обладают свойством открытости (расширяемости), т.е. допускают добавление новых процессоров без нарушения функционирования системы в целом.

Выбор той или иной архитектуры параллельной ВС во многом зависит от характера решаемых задач, конкретно от структуры реализуемых в ее среде алгоритмов. Найти компромиссное архитектурное решение для разных классов задач, обеспечив универсальность системы, представляется идеальным, но оказывается нереальным в среде современных дорогостоящих цифровых архитектур параллельного действия.

Все эти противоречия классических ВС параллельного действия повышают интерес к нейросетевым принципам организации вычислений, которые в определенном приближении можно рассматривать как MISD-архитектуру или Data Flow-машину, но в нейроисполнении.

1.2. Многоуровневое представление архитектуры вычислительной системы

Исходя из обобщенного понятия ВС, можно выделить десять уровней представления (детализации) ее архитектуры (рис. 1.1), с позиций которых система может рассматриваться конечным пользователем, прикладным программистом, системным программистом или специалистами по ее аппаратной части, включая теоретиков от естественных и точных наук, использующих законы, явления и эффекты окружающего мира для создания и исследования новых базовых принципов организации вычислений. С последними можно связать появление и внедрение в практику нейрокомпьютеров, эксперименты по созданию квантовых и молекулярных компьютеров.

Базовый естественно-математический уровень (рис. 1.1) представляет законы, принципы, явления и эффекты реального мира, а также математические модели, лежащие в основе создания и функционирования элементной базы аппаратной составляющей ВС.

Физические законы, принципы, явления и эффекты классической электроники [3] позволяют строить электронные аналоги математических моделей, определяющих вычислительный базис системы. Эти аналоги могут составить элементный базис классической аналоговой машины,

специализированной на определенный класс задач. Они могут служить основой для реализации искусственной нейронной сети [4], [5] (нейропроцессора) в аналоговом исполнении. При этом отправными для построения модели такого аналога являются биологические законы организации и функционирования человеческого мозга.

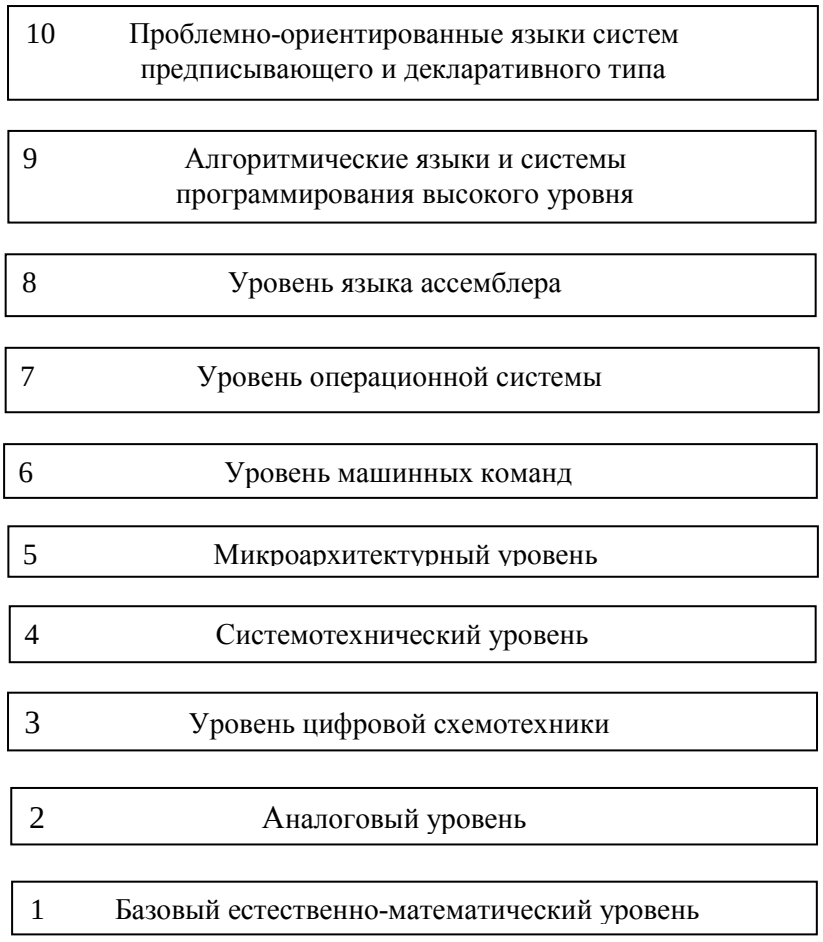


Рис.1.1. Уровни детализации общего представления архитектуры ВС

С использованием законов классической электроники и свойств полупроводниковых материалов [3] создается элементная база современной цифровой аппаратуры.

Законы квантовой электроники, описываемые математическим аппаратом квантовой механики, служат теоретической основой для построения *квантового компьютера*. Логическое ядро такого компьютера представляет собой квантовую систему, построенную на основе N атомов или элементарных частиц, имеющих два уровня состояния и называемых *кубитами* (от quantum bits). Линейность и обратимость квантовых законов поведения такой системы

позволяют использовать унитарные операторы для преобразования когерентной суперпозиции 2^N базовых состояний N-кубитового регистра в гильбертовом пространстве соответствующей размерности. Это так называемое *свойство квантового параллелизма* обеспечивает экспоненциальное ускорение вычислительного процесса, что играет важную роль при решении NP полных проблем.

Аналоговый уровень общего представления архитектуры ВС (рис. 1.1) связан с определением базовых аналоговых элементов, реализующих простейшие математические модели аналогового вычислительного базиса, и построением на их основе более сложных аналогов, выполняющих функционально законченные математические операции.

В рамках классической полупроводниковой электроники в качестве базовых аналоговых элементов определены диоды, транзисторы, емкости, сопротивления. Проектирование сложных электронных аналогов осуществляется методами аналоговой схемотехники с использованием законов Ома, Кирхгофа и известных моделей базовых аналоговых элементов.

На аналоговом уровне представления архитектуры цифровой или гибридной ВС рассматриваются вопросы построения аналоговых схем *логических элементов*, реализующих простейшие функции алгебры логики как математической основы цифровой аппаратуры. Удачные схемные решения, получаемые при этом, во многом определяют качество ВС по таким критериям, как быстродействие, надежность, потребляемая мощность, стоимость и другим.

Сложнее обстоит дело с построением аналога, функционирующего по законам квантовой электроники (квантового компьютера). Данное направление развития ВС в настоящее время находится скорее на стадии теоретических изысканий и попыток создания экспериментальных образцов из нескольких кубитов. Последние связаны с решением сложных вопросов выбора подходящей физической системы для реализации кубитов, определения способов селективного управления ими и измерения их состояния при извлечении результатов.

Уровень цифровой схемотехники (рис. 1.1) представляет иерархическую структуру цифровой аппаратуры [6] – [8], задействованной в ВС.

Нижний уровень в этой иерархии занимают *цифровые элементы*. Это наименьшие функциональные части, из которых складываются цифровые устройства при их функционально-логическом проектировании и конструктивно-технологическом исполнении. К цифровым элементам относятся:

- логические элементы;

- запоминающие элементы;
- буферные элементы.

Логические элементы реализуют простейшие функции или системы функций алгебры логики (И, ИЛИ, НЕ, И-НЕ, ИЛИ-НЕ и др.). Запоминающие элементы (конденсатор с ключевым транзистором, защелка, триггер) используются для хранения одного бита информации. Буферные элементы обеспечивают усиление сигналов на выходах цифровых устройств и согласованную (управляемую) передачу их в нагрузку.

Следует отдельно упомянуть категорию *вспомогательных элементов* цифровой аппаратуры, не выполняющих логические операции, а также функции хранения и передачи данных. К ним относятся элементы задержки, генераторы импульсных сигналов, элементы индикации и другие вспомогательные элементы.

По способу кодирования двоичных (булевых) переменных цифровые элементы классифицируются на:

- *потенциальные* (логические «0» и «1» ассоциируются с определенными, отличными друг от друга уровнями напряжения);
- *импульсные* (логические «0» и «1» ассоциируются, соответственно, с отсутствием и наличием импульсов напряжения или тока);
- *импульсно-потенциальные* (комбинируют в себе импульсный и потенциальный способы кодирования двоичных переменных).

Следующий уровень иерархии вслед за цифровыми элементами занимают *функциональные узлы*. Они выполняют микрооперации внутреннего (реализованного аппаратно) языка цифрового устройства.

Цифровые элементы и функциональные узлы являются основой для построения цифровых устройств более сложного функционального назначения. Это могут быть микропроцессоры (универсальные или специального назначения), запоминающие устройства, интерфейсные схемы и другая цифровая аппаратура ВС.

Системотехнический уровень представления архитектуры ВС (рис.1.1) определяет ее общую структуру, т.е. состав входящих в нее процессоров (или компьютеров), запоминающих устройств, периферийного оборудования и коммуникационных средств. Этот уровень отражает принципы построения и закономерности функционирования системы в целом с учетом влияния внешних факторов, технико-экономических и других показателей. На этом уровне в первом приближении распределяются функции между аппаратной и программной составляющими архитектуры ВС.

Процессоры, входящие в состав ВС, могут отличаться базовыми принципами организации вычислений, т. е. быть цифровыми, аналоговыми или нейропроцессорами в аналоговом, цифровом или гибридном исполнении.

Цифровую ВС с архитектурой SISD с аппаратной точки зрения следует рассматривать как техническую систему, предназначенную для выполнения вычислений на основе *алгоритмов*, т.е. предписаний, однозначно задающих процесс обработки исходной информации в виде последовательности элементарных дискретных шагов, приводящих за конечное число их применений к результату.

Можно выделить три важнейших свойства алгоритмов, оказавших существенное влияние на организацию цифрового компьютера:

- детерминированность вычислительных процессов, порождаемых алгоритмами;
- дискретность обрабатываемой информации;
- конечность и элементарность набора операций, реализующих алгоритмы.

Указанные свойства обусловили так называемый *принцип программного управления*, который был заложен в идеях английского математика Чарлза Беббиджа при разработке логической структуры универсальной цифровой вычислительной машины (*Аналитическая машина*, 1834-1848 г.г.). Практическое воплощение эти идеи нашли в начале 40-х годов прошлого века в машинах, построенных на основе электромеханических реле (К. Цузе – Германия, 1941 г.; Г. Эйкен – США, 1943 г.).

Принцип программного управления изначально предполагал кодирование исходных данных, промежуточных и конечных результатов в *двоичной форме*. При этом информация складывалась из элементов, называемых *словами*. Правила вычислений (*программа*) представлялись композицией операторов двух типов: *преобразователей* и *распознавателей*. *Преобразователи* обеспечивали собственно преобразование информации, т.е. выполнение действий над словами. *Распознаватели* управляли порядком выполнения операторов на основании анализа элементов информации. Операторы кодировались *управляющими словами (командами)*, определяющими наименование операции и слова участвующей в ней информации.

Идеи создания цифровых вычислительных машин на электронной основе (ЭВМ) прорабатывались с конца 30-х - начала 40-х годов прошлого века (Джон Винсент Атанасов, Сергей Алексеевич Лебедев, Джон Моушли). В 1945 г. под руководством Дж.Моушли и Дж.Эккерта была построена машина ENIAC –

Electronic Numerical Integrator and Calculator. Эта машина первого поколения содержала 18000 электронных ламп и производила 5000 операций сложения или 300 операций умножения в секунду (в 10^3 раз больше по сравнению с релейными вычислительными машинами). Она имела длину более 30 м, занимала объем 85 $м^3$, весила 30 т. Команды программы вводились вручную посредством коммутации нужным образом проводов. Это требовало значительного времени на подготовку вычислительного эксперимента. Содержание и порядок выполнения команд могли быть изменены только после выполнения всей программы.

В целях совершенствования процесса формирования программ работы Дж.Моушли и Дж.Эккерта были продолжены совместно с математиком Джоном фон Нейманом, который предложил ряд новых идей по организации ЭВМ. В плане совершенствования программного управления им была предложена *концепция хранимой в памяти программы*. При этом предполагалось:

- различать разнотипные слова информации (данные и команды) не по способам кодирования, а по способу использования;
- размещать слова информации в ячейках памяти и идентифицировать их последовательными номерами ячеек (*адресами слов*).

Однотипность кодирования данных и команд (двоичными кодами), расположение их в единой памяти предоставили возможность оперировать управляющими словами как данными и тем самым модифицировать исходную программу в ходе ее выполнения.

Дополненный идеями фон Неймана принцип программного управления определил архитектуру ЭВМ, во многих чертах сохранившуюся до настоящего времени. Ее называют *архитектурой фон Неймана*.

Первой ЭВМ, построенной в соответствии с архитектурой фон Неймана, была машина EDSAC – Electronic Delay Storage Automatic Calculator – электронный калькулятор с памятью на линиях задержки. Эту машину создал в 1949 году в Кембриджском университете Морис Уилкс.

Архитектура EDSAC предусматривала наличие в составе ЭВМ пяти функциональных устройств, обязательных для архитектуры фон Неймана. Это *арифметико-логическое устройство (АЛУ), запоминающее устройство (ЗУ), входное устройство, выходное устройство* и *устройство управления*.

В функции *АЛУ* входило выполнение команд, составляющих программу, с использованием предусмотренного в нем *набора базовых операций* (арифметических, логических, условного перехода и т.п.). Этот набор должен

был обладать *свойством полноты* (быть достаточным для описания любого алгоритма). При этом не исключалась некоторая избыточность, способствующая повышению эффективности обработки данных с учетом специфики конкретных реализуемых алгоритмов.

ЗУ (*память*) предназначалось для хранения программы, исходных и текущих данных. Место данных и команд в памяти определялось их *адресом* (адресуемая память) – номером ячейки ЗУ. Содержимое ячейки при чтении не менялось, при записи старое стиралось и заменялось на новое. В командах-преобразователях указывались не сами операнды, а их адреса. Это придавало программе универсальность формулы, в которую можно было подставлять различные данные.

Входное устройство предназначалось для ввода в ЭВМ информации, необходимой для решения задач: программ, исходных данных, управляющих символов, представленных в двоичных кодах.

Выходное устройство обеспечивало вывод получаемых результатов в понятной для пользователя форме, а также специальных управляющих символов.

Устройство управления выполняло функции организации:

- взаимодействия АЛУ с ЗУ (поочередной передачи команд программы в АЛУ, поиска в памяти нужных для выполнения очередной команды данных, записи получаемых результатов в память);
- приема информации от пользователя;
- выдачи информации пользователю.

Дойдя до пятого поколения ЭВМ, архитектура фон Неймана в общих чертах сохранилась и в современных микропроцессорных ВС. В организационном плане она вышла на качественно новый уровень, отличающийся широким спектром операционных блоков, тонкими технологиями управления вычислительным процессом, развитой многоуровневой организацией памяти, разнообразием периферийных устройств и современными принципами реализации, обеспечивающими открытость и определенную «интеллектуальность» ВС.

В отличие от свойственного первым ЭВМ принципа централизованного управления, в основе организации современных цифровых ВС лежит так называемый *магистрально-модульный* принцип. Суть его заключается в наличии набора магистралей (шин, Bus), к которым свободно подсоединяются функциональные модули, включающиеся в процесс обмена информацией по ВС в режиме разделения времени. Такой подход обеспечивает открытость

системы, т.е. возможность достаточно оперативно расширять ее функциональность и наращивать производительность путем изменения состава задействованных в ней модулей.

Цифровая однопроцессорная ВС в стандартной комплектации современного персонального компьютера имеет в своем составе центральный микропроцессор (одноядерный или многоядерный), многоуровневую память, а также стандартный набор периферийных устройств со своими адаптерами. Все это объединено в систему в соответствии с магистрально-модульным принципом ее организации посредством системной шины расширения и ряда локальных шин, выполняющих интерфейсные функции.

Многоуровневая память современного компьютера распределена по многим его компонентам. Самая быстрая и самая компактная *регистровая память*, используемая вычислительным трактом процессора для непосредственной интерпретации текущих операций и запоминания их промежуточных результатов, находится в его составе. На кристалле процессора размещены также один или два уровня внутренней *кэш-памяти* – сверхоперативной памяти небольшой емкости, доступной процессору и не адресуемой пользователем. При этом возможен третий уровень кэш-памяти, располагающийся близко к процессору на системной (материнской) плате.

Самым сбалансированным по быстродействию, информационной емкости и стоимости звеном в общей структуре памяти является *основная (оперативная) память*. Это адресная память с произвольным доступом, располагаемая на материнской плате. Она хранит исполняемую в текущий момент программу и связанные с ней данные, а также является средством оперативного обмена информацией (командами, данными) между процессором, внешней памятью и периферийными устройствами. Основная память или часть ее может кэшироваться, т.е. содержать копии блоков наиболее востребованной своей информации в более быстрой кэш-памяти.

Энергонезависимая память хранит записанную информацию при отсутствии питающего напряжения от сети, а *постоянная память*, как ее разновидность, не использует даже автономные источники питания. Энергонезависимая память располагается на материнской плате. Она предусмотрена специально для хранения неизменяемой или редко изменяемой информации системного плана, требующей защиты от несанкционированного изменения. Основной режим ее работы − считывание информации. Такой информацией прежде всего является BIOS (Basic Input-Output System – *базовая*

система ввода-вывода) – самый нижний уровень программного обеспечения компьютера, выполняющий следующие основные функции:

- инициализацию (приведение в исходное состояние) и тестирование аппаратных средств при включении питания компьютера;
- конфигурирование аппаратных средств и системных ресурсов;
- инициализацию загрузчика операционной системы;
- обслуживание прерываний от системных устройств;
- управление стандартными системными устройствами.

Специализированная память используется в адаптерах (контроллерах) периферийных устройств. К примеру, это буферы FIFO, LIFO, файловые и циклические запоминающие устройства видеопамяти.

Самый отдаленный от процессора и самый объемный уровень памяти – *внешняя память*. Она ориентирована на хранение больших объемов информации, реализуется на основе устройств с подвижными носителями информации (например, магнитные и оптические диски). Внешняя память принципиально отличается от внутренней (оперативной, энергонезависимой) памяти способом доступа процессора к ее содержимому. Ее устройства оперируют блоками информации, а не байтами и словами, как оперативная память.

Микроархитектурный уровень общего представления архитектуры ВС (рис.1.1) отражает логические и схемотехнические принципы интерпретации машинных команд с использованием технологий упреждающей выборки команд, конвейеризации, распараллеливания, кэширования, динамического предсказания ветвлений, переименования регистров и других приемов, способствующих повышению эффективности вычислительного процесса. Его организация зависит от выбора компромисса между критериями, основными из которых являются производительность и стоимость (аппаратная сложность) ВС.

Конвейеризация (*pipelining*) предполагает структурирование процесса обработки инструкции (команды) в виде последовательности этапов, каждый из которых связан с определенной функциональной ступенью конвейера. Такими ступенями могут быть: *предвыборка команды*, *декодирование*, *формирование адресов и выборка операндов*, *выполнение команды*, *запись результата*. Это естественный (классический) конвейер.

Конвейеризация, как способ повышения производительности процессора, хорошо работает с линейным кодом программы. Продвижение по ступеням

конвейера последовательно выбираемых команд одна за другой экономит общее время, затрачиваемое на выполнение программы.

Технология *предсказания переходов* (*прогнозирования ветвлений*), широко применяемая в современных процессорах, направлена на то, чтобы сохранить сформированный конвейер, не снижая его производительности. Поскольку декодирование и выполнение команды происходит на следующих ступенях конвейера после ее выборки, то до распознания команды перехода и определения нового направления выборки команд уже могут быть вызваны некоторые коды ненужных команд. Для исключения таких ситуаций может быть применена технология *отсрочки ветвления*, согласно которой компилятор заполняет одну или несколько позиций после команды перехода полезными с его точки зрения командами или пустыми командами, не производящими никаких действий. Это не нарушает правильности выполнения программы, но приводит к значительным потерям на холостых циклах. Поэтому в современных процессорах предусматриваются специальные средства предсказания переходов.

Различают технологии *динамического* и *статического* прогнозирования переходов. Первые реализуют прогнозы непосредственно в ходе выполнения программы и связаны с построением довольно сложных аппаратных средств. Вторые основную нагрузку возлагают на компиляторы, которые априори анализируют ситуации, связанные с условными переходами, и с помощью определенных битов кода (по сути дела новой команды) сообщают аппаратному обеспечению направление перехода. Иногда статическое прогнозирование выполняется компилятором на основании результатов прокрутки программы с фиксацией всех ее переходов. После этого компилятор вносит в программу все необходимые коррективы.

Процессор с единственным конвейером называется *скалярным* в отличие от *суперскалярного* процессора, имеющего два и более конвейеров, обрабатывающих инструкции параллельно. Основная идея *суперскалярной архитектуры* состоит в преобразовании исходной последовательной программы в как можно большее количество параллельных динамических вычислительных структур, одновременная реализация которых ускоряет выполнение программы. При этом речь идет о параллельности уровня команды.

Увеличение числа конвейеров путем дублирования полного (или почти полного) состава блоков требует создания громоздкого аппаратного обеспечения. Поэтому может быть использован более рациональный подход, реализующий идею одного (главного) конвейера с распараллеливанием только

самых сложных его этапов. Это могут быть этапы декодирования команд и непосредственного их выполнения функциональными блоками.

Ограничение возможности параллельного выполнения инструкций зачастую связано с использованием ими общих регистров. Обходить это ограничение позволяет применяемая в современных процессорах технология *переименования регистров* (*register renaming*). Такие процессоры фактически имеют большее количество физических регистров и предусматривают возможность использования для них параллельными инструкциями одинаковых логических имен.

Уровень машинных команд общего представления архитектуры ВС (рис.1.1) определяет набор команд (инструкций), выполняемых аппаратно или под управлением микропрограммы-интерпретатора операционными устройствами центрального процессора.

Система команд процессора – это уровень архитектуры, связывающий аппаратные средства компьютера с его программным обеспечением. Удачная система команд с одной стороны создает благоприятные условия для рациональной организации аппаратных средств, обеспечивающей необходимое их быстродействие и перспективы развития с учетом особенностей существующих и будущих технологий. С другой стороны, отличаясь регулярностью и полнотой, она облегчает построение эффективных компиляторов, переводящих программы с языков высокого уровня на язык машинных команд для последующей их интерпретации.

Для большинства архитектур процессоров характерны следующие основные типы команд.

Команды перемещения данных осуществляют копирование их из одного места в другое, преследуя цели:

- дублировать данные под новыми именами;
- обеспечить возможность быстрого доступа к данным;
- поддержать стековую организацию вычислительного процесса.

В зависимости от видов источника и пункта назначения данных возможны следующие категории команд: регистр - регистр, регистр - память, регистр - стек, стек - регистр, память - регистр, стек - память, память - стек, память - память (команды, связанные со стеком, выделены в отдельные категории в предположении, что стек реализован не программно, а аппаратно). Перемещаемые командой данные могут быть разной длины – от одного бита до всей памяти. Чаще всего передаются байт, несколько байтов, слово или несколько слов.

Команды, реализующие *бинарные операции*, выполняют действия над двумя операндами и фиксируют результат. К ним относятся арифметические команды сложения, вычитания, умножения и деления целых чисел и чисел с плавающей запятой (включая числа с удвоенной точностью), а также логические команды И, ИЛИ, И-НЕ, ИЛИ-НЕ и т.д.

Команды, реализующие *унарные операции*, производят результат действия над одним операндом. В их числе различные варианты команд сдвига и циклического сдвига, команда установки нуля (очистки), команда инкрементирования (увеличения значения операнда на единицу), команда аддитивной инверсии числа (получения его значения с противоположным знаком), команда логического отрицания (инверсии двоичного кода).

Команды сравнения и переходов в комплексе предназначены для изменения последовательности выполнения команд программы. Они подразделяются на *команды безусловного перехода*, *команды условного перехода* и *команды сравнения*.

Команды безусловного перехода определяют адрес следующей выполняемой команды вне зависимости от каких-либо условий. Конкретный вид команды связан с используемым в ней способом адресации команды, к которой осуществляется переход.

Команды условного перехода определяют адрес следующей команды, проверяя выполнение (или невыполнение) некоторого условия. Условие может быть связано с состоянием определенного бита (битов) регистра флагов, значением знакового бита числа, содержанием полей специальных регистров и других источников информации, используемых для формирования и проверки условий выполнения переходов. Различные варианты команд условного перехода отличаются способами формирования условий и адресации команд, к которым осуществляется переход.

Информация, используемая в условии перехода, может представлять собой признак результата обычной арифметической операции над целыми числами или числами с плавающей запятой, предшествующей команде условного перехода. Она может также вырабатываться как результат выполнения одной из *команд сравнения* двух операндов. В случае трехадресной машины операции сравнения и условного перехода могут быть объединены в одной команде. При двухадресной архитектуре команд команда сравнения, вырабатывающая результат сравнения, должна предварять команду условного перехода.

Команды обращения к процедурам реализуют передачу управления первой выполняемой команде вызываемой процедуры (подпрограммы) и сохраняют

адрес возврата в вызывающую программу после завершения выполнения подпрограммы. Механизмы помещения адреса возврата в фиксированную ячейку памяти или в определенное место подпрограммы (таким местом может быть первое слово процедуры, к которому осуществляется безусловный переход после ее выполнения; при этом выполнение подпрограммы начинается со второго слова) не срабатывают в случае рекурсии, т.е. вызова процедурой самой себя. Самым эффективным решением с этой точки зрения является размещение адреса возврата в стеке. Использование этой динамической структуры позволяет эффективно обрабатывать рекурсии практически любой глубины вложенности.

Команды управления циклом организуют выполнение выделенной группы команд заданное количество раз. При этом используется счетчик, увеличивающий или уменьшающий свое значение на определенную константу при каждом проходе цикла. Значения счетчика контролируются с целью отслеживания момента выхода из цикла в соответствии с заданным числом его выполнения. Конкретные варианты команд данного типа отличаются местом проверки окончания цикла (в его начале или конце), а также условиями повторения цикла.

В отдельные типы следует выделить *команды ввода-вывода, команды прерываний* и ряд команд системного уровня, связанных с управлением процессором.

С точки зрения определения степени влияния архитектурного уровня машинных команд на планирование внутренней структуры вычислительного процесса, реализуемой на микроархитектурном уровне, заслуживают внимания две альтернативных концепции.

Согласно первой концепции система команд не должна содержать каких-либо указаний по рациональному использованию функциональных блоков процессора. Планирование динамики вычислений должно осуществляться самим процессором в соответствии с технологиями его микроархитектурного уровня, реализованными гибким сочетанием аппаратных и компилятивных средств.

Вторая концепция, напротив, предполагает открытое управление на уровне машинных команд внутренними аппаратными ресурсами процессора с целью наиболее рационального их использования. Реализация такого подхода требует введения в коды инструкций дополнительных полей, которые в отличие от традиционных, указывающих на то, *что надо делать*, информируют о том, *как это надо делать*. Процессоры, следующие данной концепции, называют

процессорами с длинным командным словом (архитектура *VLIW – Very Long Instruction Word*). Очевидно, что при реализации таких процессоров основная часть нагрузки ложится скорее на компилятивные средства, нежели аппаратные, которые для выполнения многих функций могут оказаться слишком громоздкими.

Возможен компромиссный вариант, сочетающий положительные стороны обеих концепций.

Программное обеспечение (*software),* как составная часть ВС, активизирует ее аппаратные средства (*hardware)* на выполнение необходимых действий, связанных с решением задач. Оно складывается из двух составляющих: *системного программного обеспечения* и *прикладного программного обеспечения*.

Системное программное обеспечение содержит в своем составе компоненты, связанные с *уровнем операционной системы* (рис. 1.1). Этот уровень отличается от предыдущего уровня (уровня машинных команд) наличием дополнительных команд (*системных вызовов*), своей организации памяти, средств обеспечения мультипрограммного режима и других расширений, реализуемых специальной программой-интерпретатором, называемой *операционной системой*.

Операционная система обеспечивает взаимодействие пользователя с ВС и управление ее ресурсами в различных режимах ее функционирования.

Шестой и седьмой уровни изначально планируются как инструментальная среда для создания системного программного обеспечения (трансляторов, оболочек операционных систем и других расширений, поддерживающих языки более высоких уровней). В отличие от них средства восьмого и в основном девятого и десятого уровней ориентируются на прикладных программистов и конечных пользователей ВС.

Рассмотренное многоуровневое представление (рис. 1.1) является обобщающим для различных архитектур ВС и их составных компонентов. Его уровни приобретают определенное содержание и смысл при рассмотрении конкретных проектов ВС. Так, например, для аналоговых ВС имеют смысл только первый, второй и четвертый уровни. Цифровым или гибридным ВС, в которых наряду с аналоговой так или иначе присутствует цифровая составляющая, свойственны все упомянутые уровни представления.

2. АНАЛОГОВЫЙ УРОВЕНЬ ЦИФРОВОЙ ВС

2.1. Аналоговые схемы логических элементов

Как было отмечено выше, на аналоговом уровне представления архитектуры цифровой или гибридной ВС должны рассматриваться вопросы построения аналоговых схем логических элементов, реализующих простейшие функции алгебры логики − математической основы цифровой аппаратуры.

Схемотехнической основой логического элемента является принципиальная схема, построенная из аналоговых элементов. Простейший пример такой схемы − электронный ключ [6], выполняющий функцию логической инверсии (отрицания) – представлен на рис. 2.1. Рядом со схемой (справа) приведено условное графическое обозначение логического элемента, используемое для представления его в логических схемах, рассматриваемых на уровне цифровой схемотехники (кружок на выходе говорит о его инвертирующем характере).

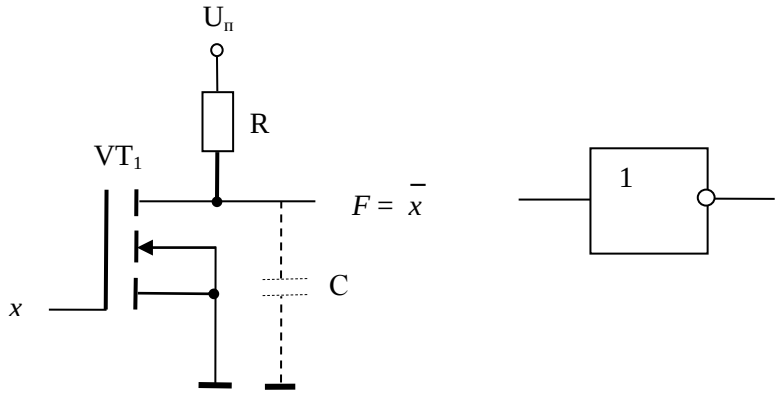


Рис. 2.1. Простейшая принципиальная схема инвертора и его условное обозначение

Электронный ключ характеризуется двумя состояниями: разомкнутым и замкнутым. Его схема на рис. 2.1 построена на основе МДП-транзистора (с индуцированным каналом) VT_1 , подключенного к источнику питания $U_п$ через резистивную нагрузку R.

Если на вход *x* подано низкое напряжение, соответствующее логическому «0», то транзистор закрыт (ключ разомкнут), и нагрузочная емкость C

(изображена пунктиром) заряжается до напряжения источника питания, т.е. на выходе *F* присутствует высокое напряжение, соответствующее логической «1».

При высоком напряжении на входе транзистор открыт (ключ замкнут), и нагрузочная емкость разряжается на «землю» через открытый транзистор, т.е. напряжение на выходе *F* – низкое, соответствующее логическому «0».

Время переключения такой схемы из состояния «1» на выходе в состояние «0» определяется временем разряда емкости С на «землю» через небольшое сопротивление открытого транзистора. Переключение из «0» в «1» определяется временем заряда емкости С до напряжения источника питания $U_п$ через достаточно большое сопротивление R. Это время можно уменьшить, используя вместо R динамическую нагрузку в виде транзистора VT_2, работающего в открытом режиме (рис. 2.2). В этой ситуации при переключении из «0» в «1» емкость С будет заряжаться через небольшое сопротивление открытого транзистора VT_2.

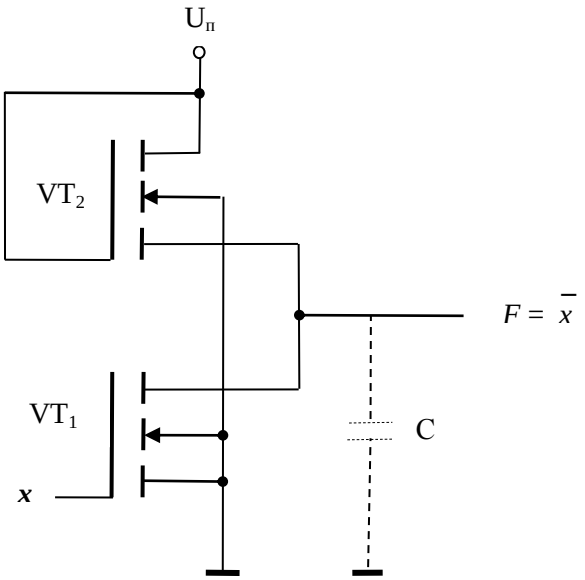


Рис. 2.2. Принципиальная схема инвертора с динамической нагрузкой

Путем соединения определенным образом электронных ключей можно строить принципиальные схемы других логических элементов (И, ИЛИ, И-НЕ и т.д.). В качестве примеров на рисунках 2.3, 2.4 приведены принципиальные схемы и условные графические обозначения логических элементов И-НЕ, ИЛИ-НЕ на МДП-транзисторах.

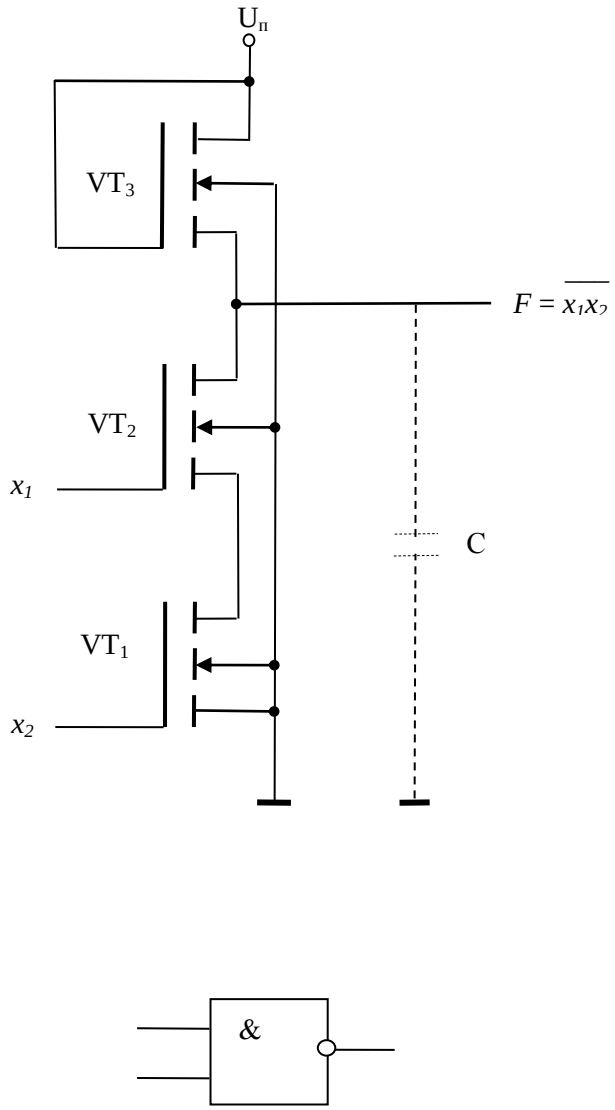


Рис. 2.3. Принципиальная схема логического элемента И-НЕ и его условное обозначение

Если выход элемента И-НЕ подать на инвертор, то получается логический элемент И. Его условное обозначение совпадает с обозначением элемента И-НЕ, только кружок на его выходе отсутствует. Аналогично из элемента ИЛИ-НЕ можно получить элемент ИЛИ.

Логические элементы могут иметь более двух входов и служить базисом для построения более сложных функциональных узлов и устройств цифровой аппаратуры.

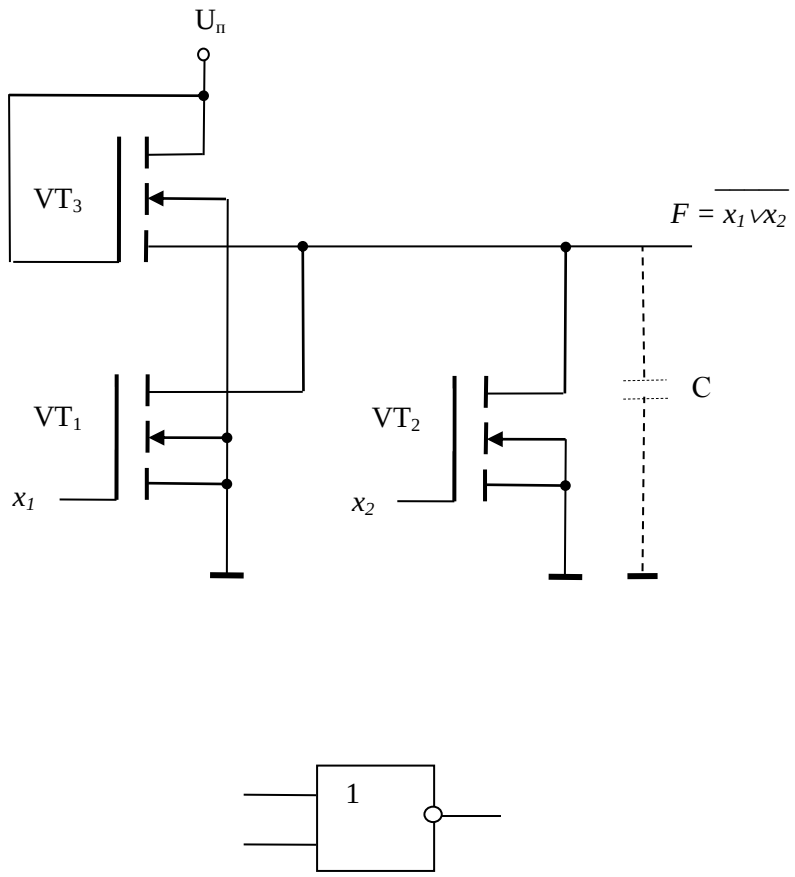


Рис. 2.4. Принципиальная схема логического элемента ИЛИ-НЕ и его условное обозначение

Надо заметить, что реализуемые на практике принципиальные схемы логических элементов могут выглядеть несколько сложнее рассмотренных, ибо при их проектировании помимо выполняемой логической функции могут приниматься во внимание и другие характеристики логического элемента: нагрузочная способность, коэффициент объединения по входу, помехоустойчивость, среднее время задержки, предельная рабочая частота, потребляемая мощность.

2.2. Выходы логических элементов

В зависимости от условий работы в более сложных структурах (логических цепях, запоминающих устройствах, магистрально-модульных микропроцессорных системах) логические элементы могут иметь выходы четырех типов [6]:

- *логические*;

- *с открытым коллектором (стоком)*;
- *с открытым эмиттером (истоком)*;
- *с третьим состоянием*.

На *логическом* выходе могут формироваться два уровня напряжения: низкое, соответствующее логическому «0», и высокое, соответствующее логической «1» (в случае использования позитивной логики). С целью увеличения быстродействия элемента его выходное сопротивление стремятся уменьшить, чтобы обеспечить достаточно большие токи перезаряда емкостной нагрузки. Это достигается применением на выходе элемента двухтактного каскада, представленного на рисунке 2.5, где прямоугольником с пунктирными границами обозначена вся микросхема *n*-входового логического элемента, реализующего функцию *F,* и выделен только двухтактный каскад.

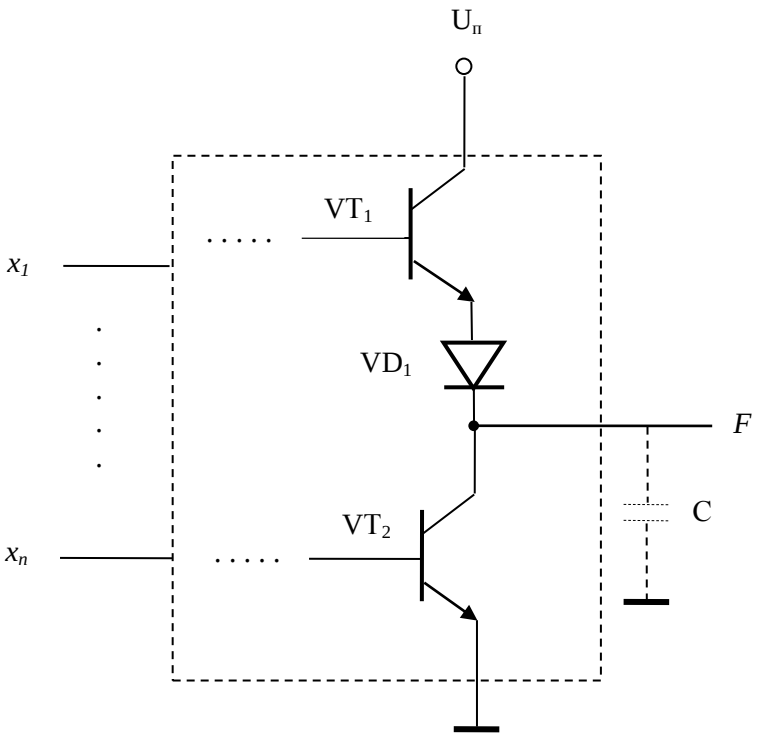


Рис. 2.5. Логический выход

Малое выходное сопротивление такой схемы при любом направлении переключения обеспечивается двумя работающими противофазно (один открыт, другой закрыт) транзисторами VT_1 и VT_2. При высоком напряжении на базе VT_1 через небольшое сопротивление открытого транзистора и диода VD_1 происходит заряд нагрузочной емкости С от источника питания $U_п$ до уровня «1» (транзистор VT_2 в это время закрыт). В противофазной ситуации

нагрузочная емкость C через открытый транзистор VT_2 разряжается на «землю» (диод VD_1 в это время обеспечивает надежное запирание транзистора VT_1**).**

Как недостаток логических выходов следует отметить невозможность их параллельного соединения (выхода на общую линию). Во-первых, это приводит к возникновению логической неопределенности. Во-вторых, возможно появление большого, опасного для электрических элементов выходных цепей, уравнительного тока. Это может иметь место, когда параллельные выходы находятся в противоположных состояниях.

Выход *с открытым коллектором* заканчивается одиночным транзистором VT_1, коллектор которого не соединен ни с одним элементом внутри схемы (рис. 2.6). Открытое состояние транзистора соответствует логическому нулю на выходе *F*, запертое – логической единице. Высокое напряжение при запертом транзисторе обеспечивается подключением к выходной линии через внешний резистор R источника питания $U_п$ (изображено пунктиром).

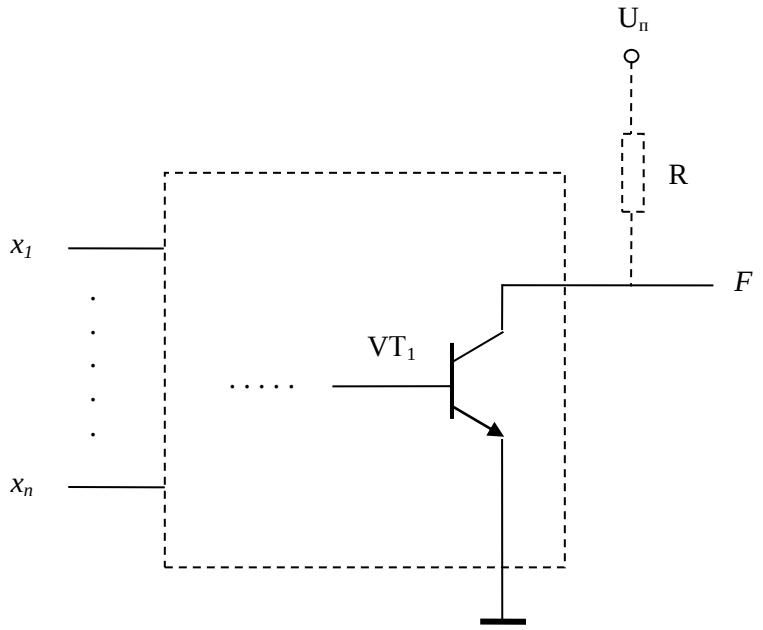


Рис. 2.6. Выход с открытым коллектором

Выходы с открытым коллектором могут быть соединены параллельно и обеспечены высоким напряжением через общую внешнюю цепочку R – $U_п$, как показано на рис. 2.7 для элементов И-НЕ (о выходе с открытым коллектором n-p-n-транзистора говорит символ «◊» на обозначении элемента).

Такая схема соединения может работать в двух режимах. Режим поочередной работы логических элементов на общую линию (один элемент активен, остальные заперты) обеспечивается подачей единичных сигналов на все входы элементов кроме активного. При разрешении одновременной активности всех элементов (второй режим) схема реализует так называемую *операцию монтажного И:*

$$F = \overline{\overline{x_1 x_2} \vee \overline{x_3 x_4} \vee \ldots\ldots \vee \overline{x_{n-1} x_n}} = x_1 x_2 x_3 x_4 \ldots\ldots x_{n-1} x_n.$$

Элементы с открытым коллектором при работе в магистрально-модульных структурах (первый режим) защищены от повреждений при ошибочном одновременном подключении к шине нескольких элементов. Такую защиту обеспечивает сопротивление R, которое берется для этого достаточно большим.

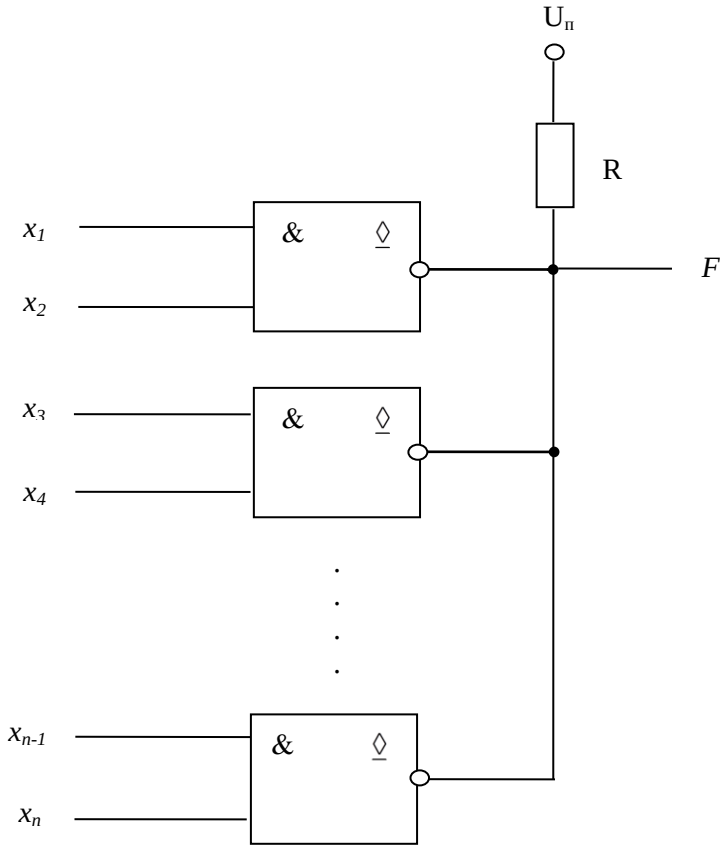


Рис. 2.7. Параллельное соединение выходов с открытым коллектором

Недостаток элементов с открытым коллектором состоит в большом времени задержки при переключении из «0» в «1» из-за малого тока заряда нагрузочной емкости через сопротивление R. Выход *с открытым эмиттером*

типичен для элементов *эмиттерно-связанной логики* (ЭСЛ) [3]. Эти элементы обычно имеют два противофазных выхода, заканчивающиеся одиночными транзисторами, коллекторы которых заземлены, а открытые (не соединенные ни с одним элементом внутри схемы) эмиттеры выводятся на внешние резисторы с подключенным к ним отрицательным питанием. На одном из выходов (прямом) реализуется логическая функция ИЛИ, на другом (инверсном) – функция ИЛИ-НЕ.

На рис. 2.8 приведена схема ЭСЛ, известная под названием «эмиттерный дот». В ней попарно параллельно соединены прямые и инверсные выходы двух логических элементов ЭСЛ (для наглядности одиночные транзисторы выходов с открытым эмиттером вынесены за пределы условных обозначений логических элементов, поэтому в них отсутствует символ « $\bar{\Diamond}$ », помечающий выход с открытым эмиттером).

Соединение прямых выходов дает объединение всех входных переменных в единую дизъюнкцию (расширение по ИЛИ).

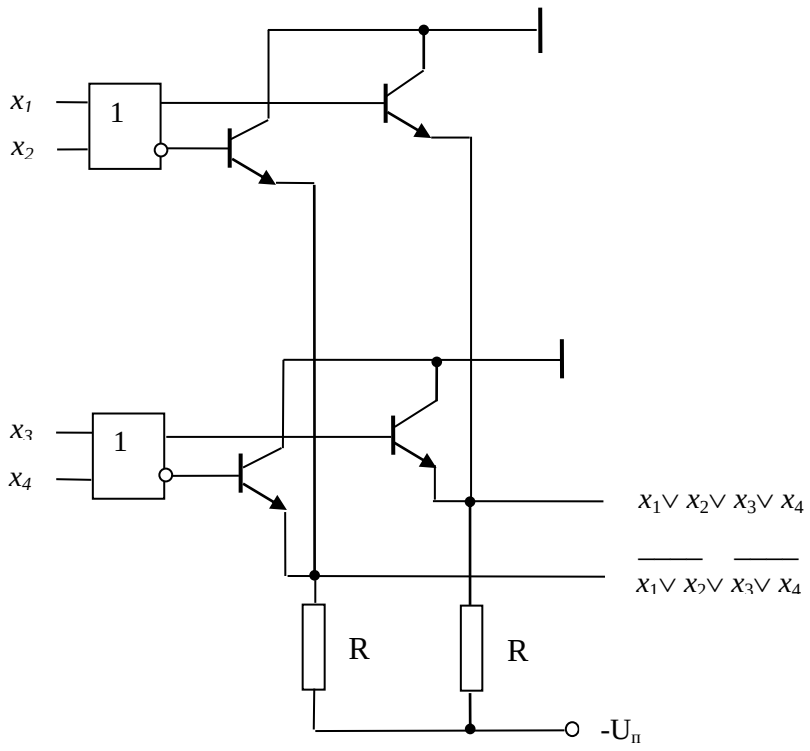


Рис. 2.8. Схема «эмиттерный дот»

Соединение инверсных выходов приводит к функции

$$F = \overline{x_1 \vee x_2} \vee \overline{x_3 \vee x_4} = \overline{x_1}\,\overline{x_2} \vee \overline{x_3}\,\overline{x_4},$$

т.е. операции И-ИЛИ применительно к инверсиям входных переменных.

Если соединить прямой выход 1-го элемента с инверсным выходом 2-го элемента, то получается логическая функция

$$F = x_1 \vee x_2 \vee \overline{x_3 \vee x_4} = x_1 \vee x_2 \vee \overline{x_3}\,\overline{x_4}\,.$$

Элементы, имеющие выход с открытым эмиттером, для работы на магистраль не используются.

Выход *с третьим состоянием* помимо состояний «0» и «1» может иметь третье состояние «отключено». Основой его является тот же двухтактный каскад, что и у логического выхода (рис. 2.5). В третье состояние, состояние запертых обоих транзисторов, когда ток в выходной цепи пренебрежимо мал, элемент переходит под управлением специального сигнала *OE* (*Output Enable*). Если *OE* = 1, то элемент выполняет свою обычную логическую функцию. При *OE* = 0 он переходит в состояние «отключено».

Буферные элементы с третьим состоянием используются в цифровой аппаратуре для управляемой передачи сигналов по общим линиям связи (магистралям, шинам). Их можно соединять параллельно, но с обязательным соблюдением условия, что в каждый конкретный момент времени активным может быть только один из них. При этом они сохраняют такие положительные качества элементов с логическим выходом, как быстродействие и нагрузочная способность.

Различают *инвертирующие* и *неинвертирующие* буферы, управляемые *H-активными* (высокоактивными) и *L-активными* (низкоактивными) сигналами *OE*. Их условные обозначения представлены на рис. 2.9 (символом «∇» помечен выход с третьим состоянием).

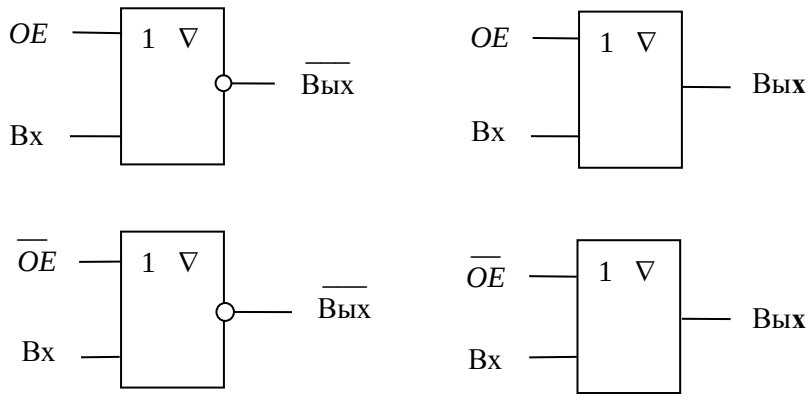


Рис. 2.9. Возможные сочетания разрешающих и выходных сигналов буферных элементов

3. УРОВЕНЬ ЦИФРОВОЙ СХЕМОТЕХНИКИ

3.1. Функциональные узлы комбинационной логики

Функциональные узлы цифровой аппаратуры [6], [8], [9] делятся на *комбинационные* и *последовательностные*. *Комбинационные* узлы (комбинационные цепи) – это устройства, выходные сигналы которых зависят только от текущих значений входных сигналов. В отличие от них, *последовательностные* узлы содержат элементы памяти. Их реакция определяется не только вектором входных сигналов, но и внутренним состоянием, зафиксированным в памяти. Поэтому их называют *автоматами с памятью*.

Рассмотрим типовые комбинационные узлы, наиболее часто выступающие в качестве базисных при построении более сложных устройств и систем цифровой аппаратуры.

Дешифратор (*Decoder*, DC) осуществляет преобразование *n*-элементного параллельного кода в код «1 из *m*», у которого только в одной позиции находится единица, все остальные позиции – нулевые. Количество выходов *m* так называемого *полного дешифратора* должно равняться числу всевозможных *n*-разрядных кодовых комбинаций на входе, т.е. $m = 2^n$. В качестве примера на рис. 3.1 *а* приведена логическая схема двухвходового дешифратора, ниже (рис. 3.1 *б*) представлено его условное обозначение.

Информационные входы дешифратора принято обозначать их двоичными весами. *EN* (*ENable*) – вход разрешения работы дешифратора. На выходах дешифратора формируются логические функции в виде системы конъюнкций, которая в случае *n* информационных входов имеет вид:

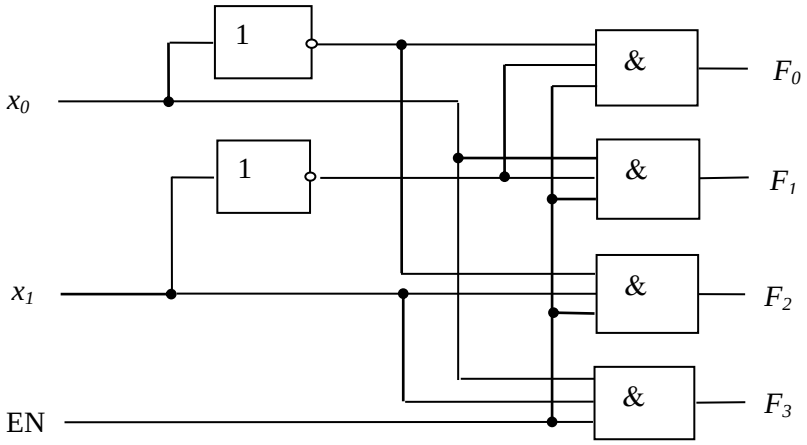
$$F_0 = \bar{x}_{n-1}\bar{x}_{n-2}\ldots\bar{x}_1\bar{x}_0 EN$$

$$F_1 = \bar{x}_{n-1}\bar{x}_{n-2}\ldots\bar{x}_1 x_0 EN$$

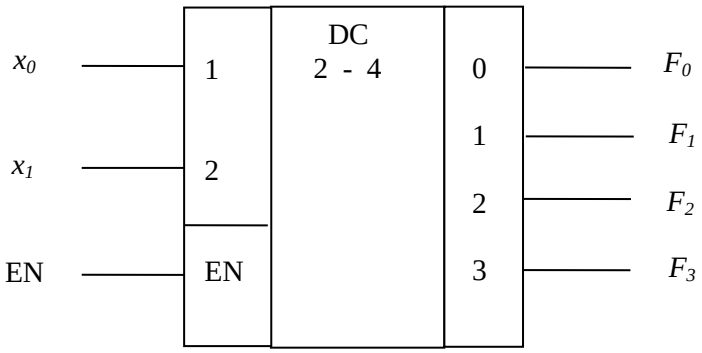
$$F_2 = \bar{x}_{n-1}\bar{x}_{n-2}\ldots x_1\bar{x}_0 EN$$

$$\ldots\ldots\ldots\ldots\ldots\ldots$$

$$F_{m-1} = x_{n-1}x_{n-2}\ldots x_1 x_0 EN\,.$$



а)



б)

Рис.3.1. Двухвходовый дешифратор
а) логическая схема; *б)* условное обозначение

Из формул видно, что поданный на информационные входы дешифратора двоичный код порождает (при разрешающем значении сигнала $EN = 1$) единственный активный выходной сигнал F_i с индексом i, равным десятичному значению входного кода.

Помимо выполнения основной функции дешифрации двоичных кодов (в микросхемах памяти, микропроцессорах и многих других цифровых устройствах) дешифратор может иметь и другое применение. Формируя на выходах все конъюнктивные термы, он позволяет в сочетании с элементами ИЛИ воспроизводить произвольные логические функции от n аргументов. Для этого надо подать на схемы ИЛИ выходы дешифратора с конъюнкциями, входящими в СДНФ реализуемых логических функций.

Мультиплексор (управляемый кодом коммутатор) выполняет функцию коммутации одного из m информационных входов $x_0, x_1, \ldots, x_{m-1}$ на выход F

под управлением n адресных (управляющих) входов a_0, a_1, …, a_{n-1}, число которых определяется соотношением $2^n = m$. Каждой управляющей двоичной комбинации соответствует свой вход, подключаемый к выходу. Индекс входного сигнала, поступающего на выход, равен десятичному значению управляющего кода.

Функционирование мультиплексора описывается так называемой *мультиплексной формулой*:

$$F = x_0\bar{a}_{n-1}\bar{a}_{n-2}..\bar{a}_1\bar{a}_0 \vee x_1\bar{a}_{n-1}\bar{a}_{n-2}..\bar{a}_1a_0 \vee ... \vee x_{m-1}a_{n-1}a_{n-2}..a_1a_0,$$

для реализации которой требуется n инверторов, m (n+1)-входовых конъюнкторов и один m-входовый дизъюнктор.

Логическая схема мультиплексора для случая $n = 2$ и $m = 4$ приведена на рис. 3.2, а его условное обозначение – на рис. 3.3.

Основное применение мультиплексоров состоит в обеспечении мультиплексированного подключения к единой шине различных источников сигналов для поочередного ее использования.

С помощью мультиплексора можно также осуществлять преобразование поданного на его информационные входы m-разрядного параллельного кода в последовательный код, активизируя такт за тактом управляющие линии значениями адреса от 0 до 2^n-1.

Мультиплексор можно использовать как *универсальный логический модуль* (УЛМ), настраиваемый на выполнение любой логической функции от n переменных. Простейшая схема настройки представлена на рис. 3.4. На информационных входах мультиплексора зафиксированы значения функции, которые передаются на выход при подаче на управляющие входы соответствующих значений переменных.

Демультиплексор в противоположность мультиплексору коммутирует под управлением n адресных линий единственный входной сигнал с одним из $m = 2^n$ выходов. Данную операцию может выполнить дешифратор, если на его вход разрешения (*EN*) подать информационный сигнал, а входной двоичный код использовать в качестве сигналов управления. Дешифратор, имеющий вход разрешения и выполняющий функцию демультиплексора, называют *дешифратором-демультиплексором*. Его условное обозначение приведено на рис. 3.5.

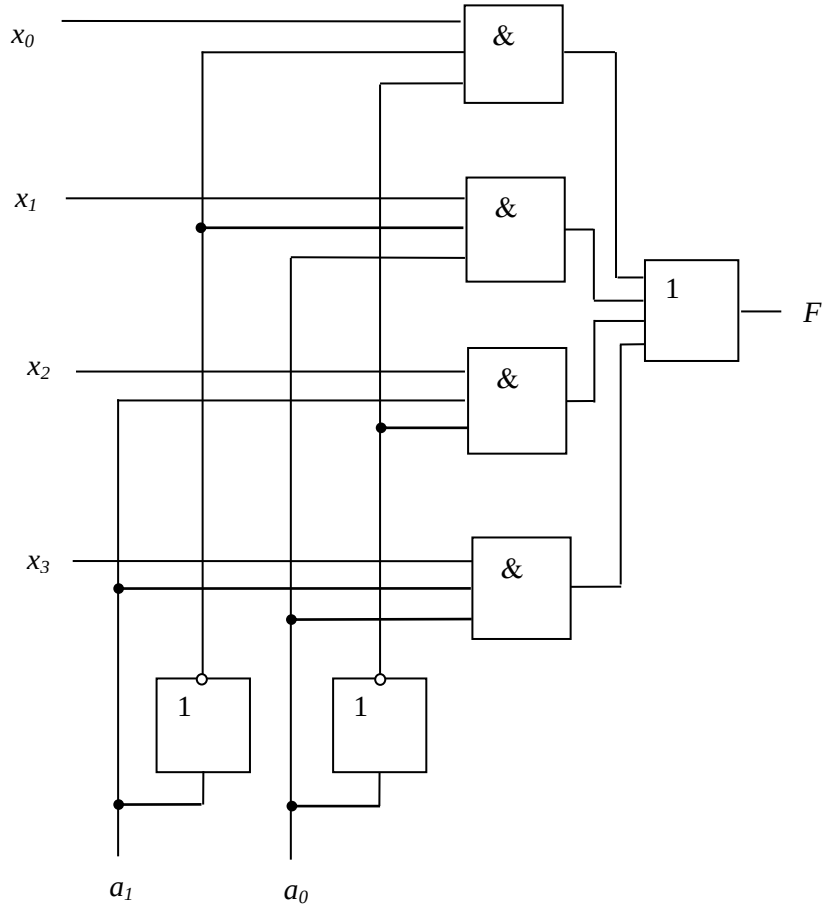


Рис.3.2. Логическая схема мультиплексора с четырьмя информационными входами

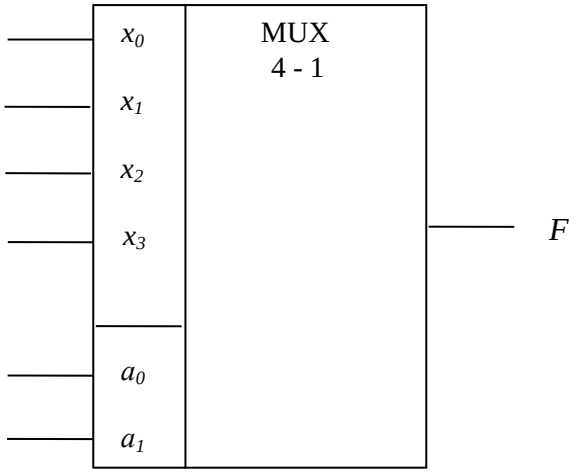


Рис. 3.3. Условное обозначение мультиплексора с четырьмя информационными входами

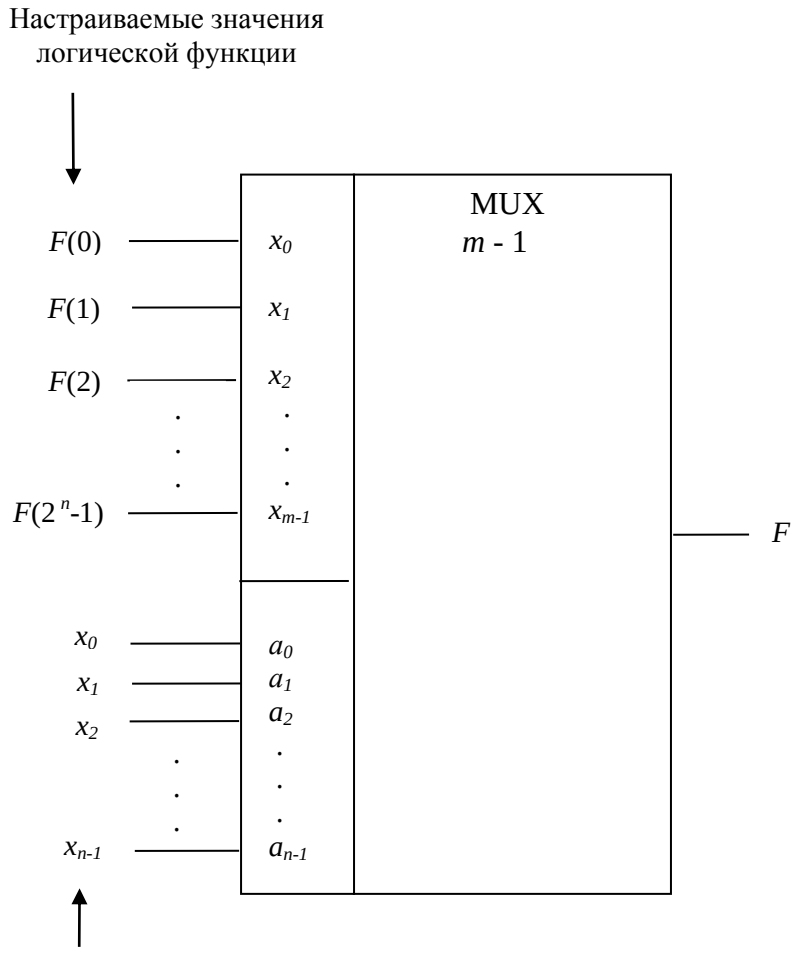


Рис. 3.4. Схема использования мультиплексора в качестве УЛМ

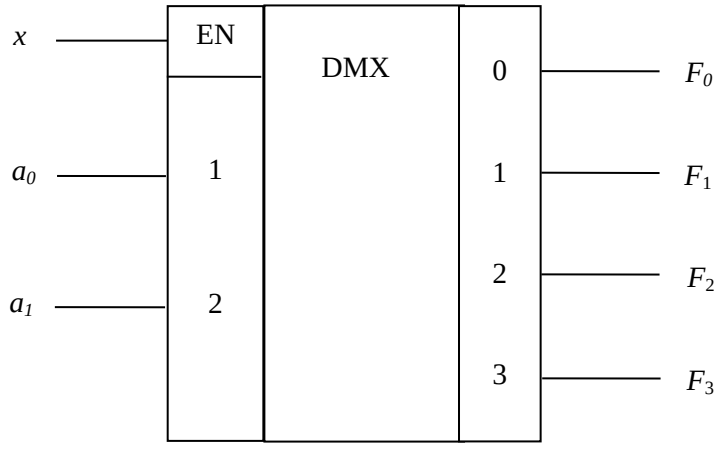


Рис. 3.5. Условное обозначение дешифратора-демультиплексора

Компаратор (СМР) – это устройство, выполняющее функцию сравнения двух слов.

Для слов $a = a_{n-1}a_{n-2}..a_1a_0$ и $b = b_{n-1}b_{n-2}..b_1b_0$ компаратор определяет три типа отношений: отношение равенства $F_{a=b}$, отношение «больше» $F_{a>b}$, отношение «меньше» $F_{a<b}$. Признаком выполнения равенства по i-у разряду слов является логическая функция $r_{i=} = \overline{a_i \oplus b_i} = \overline{a_i\bar{b}_i \vee \bar{a}_i b_i}$, признаком «больше» - функция $r_{i>} = a_i\bar{b}_i$, признаком «меньше» - функция $r_{i<} = \bar{a}_i b_i$.

Логические функции, выражающие отношения равенства, «больше» и «меньше» между n-разрядными числами a и b, можно записать следующим образом:

$$F_{a=b} = r_{n-1=}r_{n-2=}F'_{a=b}$$
$$F_{a>b} = r_{n-1>} \vee r_{n-1=}r_{n-2>} \vee F'_{a>b}$$
$$F_{a<b} = r_{n-1<} \vee r_{n-1=}r_{n-2<} \vee F'_{a<b}$$

Штрихами помечены функции отношений по младшим разрядам слов.

Фрагмент логической схемы компаратора, соответствующий вышеприведенным формулам, представлен на рис. 3.6.

Схема сдвига осуществляет сдвиг n-разрядного кода $x_{n-1}x_{n-2}..x_1x_0$ на один разряд влево или вправо в зависимости от заданного сигнала управления C = 0 или C = 1, соответственно. Пример реализации схемы для случая четырех входных битов приведен на рис. 3.7.

Схема контроля четности выполняет функцию проверки поступающего на ее n входов двоичного кода $x_{n-1}x_{n-2}..x_1x_0$ на присутствие в нем четного (нечетного) числа единиц. Схема реализует сумматор по модулю 2 битов входного кода, формируя на выходе «1», если число единиц во входном коде нечетное, и «0», если число единиц четное.

Восьмиразрядная схема контроля четности, построенная на базе XOR-элементов, представлена на рис. 3.8. XOR-элемент, обозначаемый символом «=1», в свою очередь может быть реализован по схеме, обведенной пунктиром на рис. 3.6 (с исключением инверсии на выходе).

Схемы контроля четности применяются для организации контролируемой (в плане обнаружения ошибок) передачи кодов между устройствами цифровой ВС.

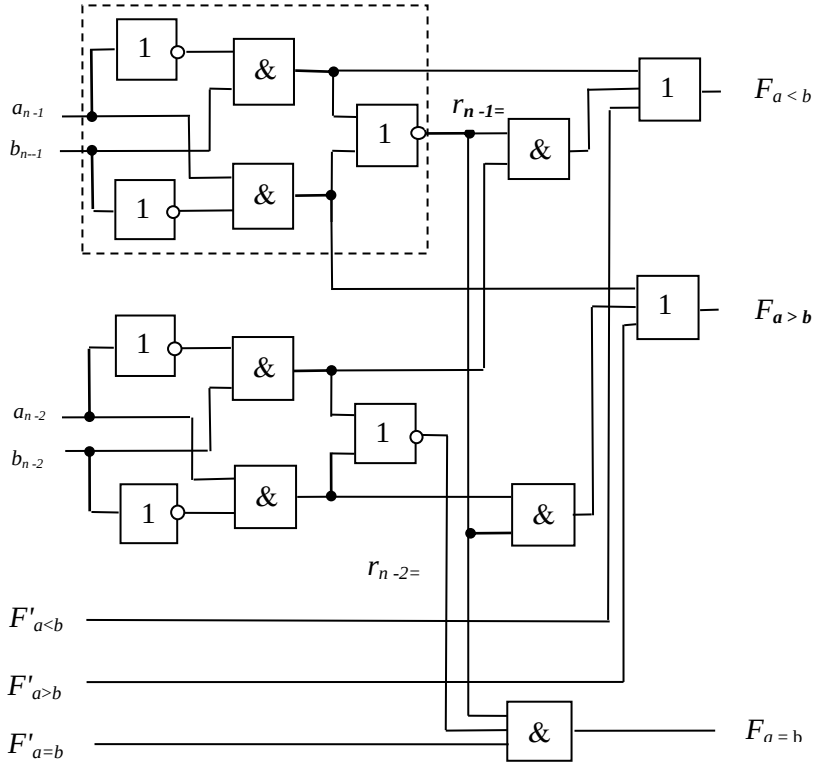


Рис. 3.6. Фрагмент логической схемы компаратора

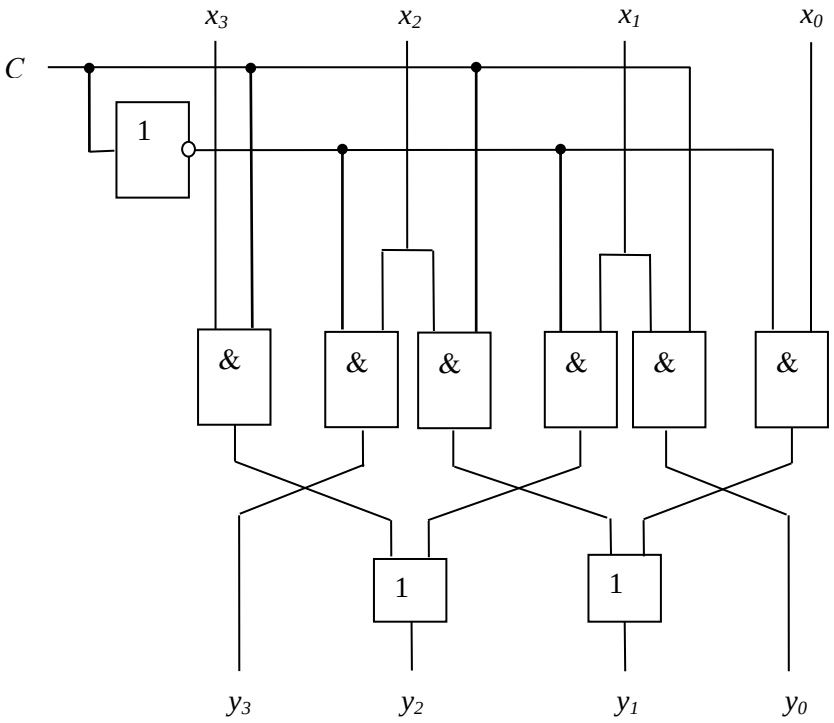


Рис. 3.7. Схема сдвига четырехразрядного кода

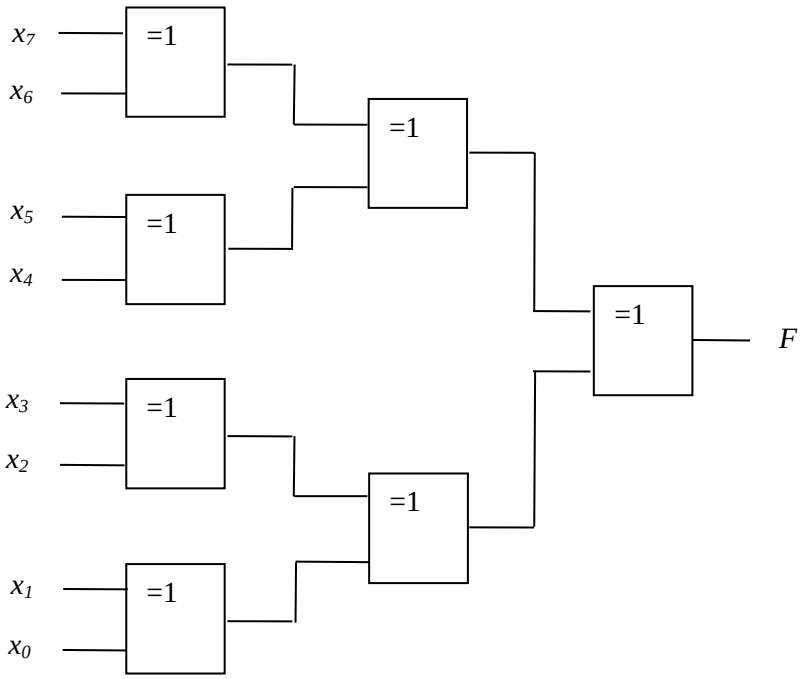


Рис. 3.8. Восьмиразрядная схема контроля четности

Сумматор выполняет арифметическое сложение двух целых чисел. *Накапливающий сумматор* поданное на его вход число суммирует к хранящемуся в его памяти предыдущему результату. *Комбинационный сумматор* не имеет памяти и формирует на выходе результат сложения поступивших на его входы двух чисел.

Многоразрядный комбинационный сумматор складывается из одноразрядных сумматоров.

Одноразрядный комбинационный сумматор имеет три входа a_i , b_i (i-е разряды складываемых чисел), p_{i-1} (сигнал переноса из соседнего младшего разряда) и два выхода s_i (i-й разряд суммы), p_i (сигнал переноса в соседний старший разряд). Соотношения значений входных и выходных сигналов представлены в таблице1.

По таблице истинности легко построить логические функции выходов сумматора следующего вида:

$$p_i = \bar{a}_i b_i p_{i-1} \vee a_i \bar{b}_i p_{i-1} \vee a_i b_i \bar{p}_{i-1} \vee a_i b_i p_{i-1} =$$
$$= p_{i-1}(\bar{a}_i b_i \vee a_i \bar{b}_i) \vee a_i b_i (\bar{p}_{i-1} \vee p_{i-1}) = p_{i-1}(a_i \oplus b_i) \vee a_i b_i$$
$$s_i = \bar{a}_i \bar{b}_i p_{i-1} \vee \bar{a}_i b_i \bar{p}_{i-1} \vee a_i \bar{b}_i \bar{p}_{i-1} \vee a_i b_i p_{i-1} =$$
$$= \bar{a}_i (\bar{b}_i p_{i-1} \vee b_i \bar{p}_{i-1}) \vee a_i (\bar{b}_i \bar{p}_{i-1} \vee b_i p_{i-1}) =$$
$$= \bar{a}_i (b_i \oplus p_{i-1}) \vee a_i \overline{(b_i \oplus p_{i-1})} = a_i \oplus b_i \oplus p_{i-1}$$

Таблица 1
Таблица истинности для одноразрядного сумматора

a_i	b_i	p_{i-1}	p_i	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Схемная реализация полученных логических функций представлена на рис. 3.9. Условное обозначение одноразрядного комбинационного сумматора изображено на рис. 3.10.

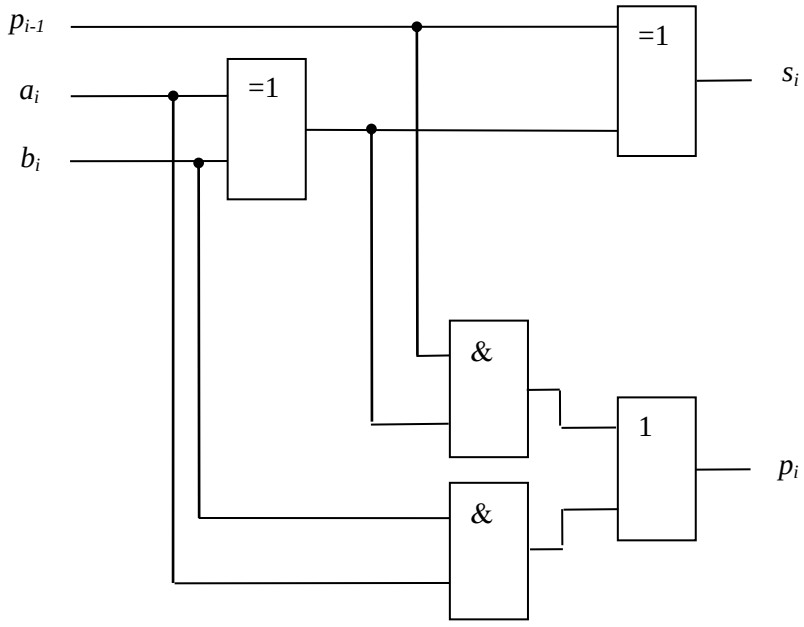


Рис. 3.9. Схема одноразрядного комбинационного сумматора

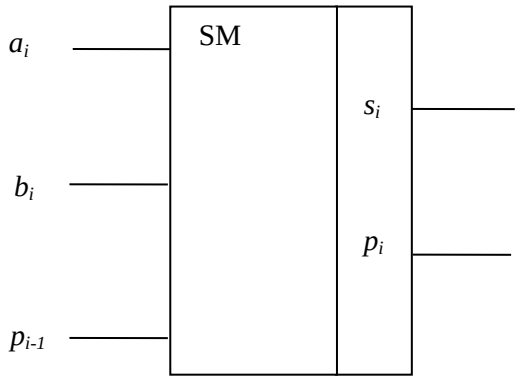


Рис. 3.10. Условное обозначение одноразрядного комбинационного сумматора

Многоразрядные комбинационные сумматоры, формируемые из одноразрядных, по принципу организации переноса подразделяются на сумматоры *с последовательным (сквозным), параллельным и комбинированным переносом.*

Сумматор *с последовательным переносом* строится путем последовательного (каскадного) соединения по цепям переноса необходимого числа одноразрядных сумматоров (рис. 3.11). При выполнении операции одноразрядные сумматоры срабатывают поочередно слева направо, и время суммирования определяется временем распространения сигнала переноса через всю схему. Поэтому с увеличением разрядности сумматора его быстродействие снижается.

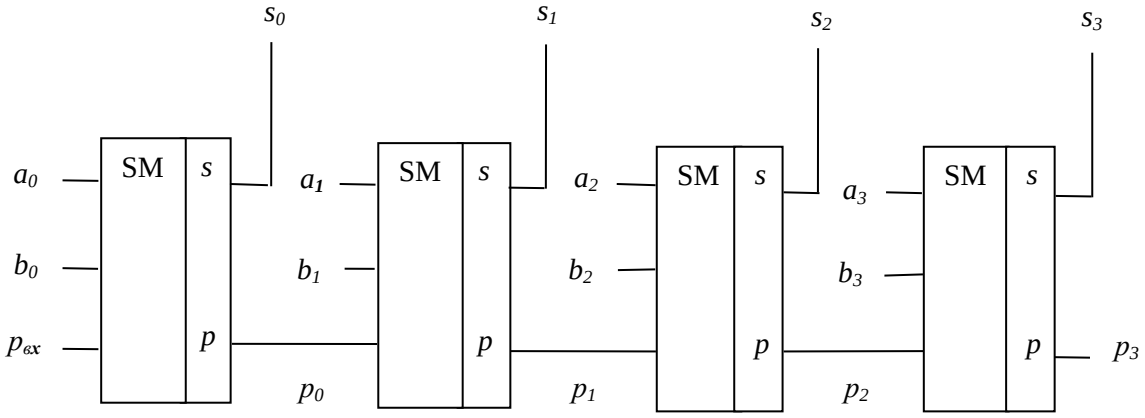


Рис. 3.11. Структура четырехразрядного сумматора с последовательным переносом

Максимально увеличить быстродействие позволяет сумматор *с параллельным переносом* (рис. 3.12). Составляющие его одноразрядные

сумматоры работают параллельно, получая входные сигналы переноса от специальной схемы, называемой *схемой ускоренного переноса*. Эта схема формирует сигналы переноса во все разряды одновременно.

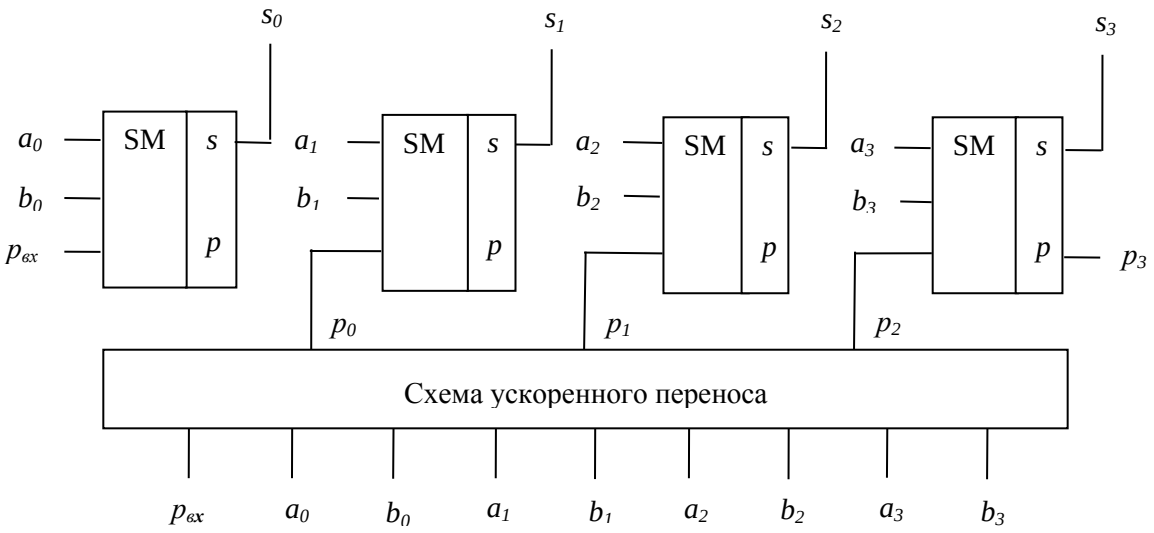


Рис. 3.12. Структура четырехразрядного сумматора с параллельным переносом

Схемы ускоренного переноса для сумматоров с числом разрядов более четырех оказываются очень громоздкими. Для достижения приемлемого компромисса между быстродействием и аппаратной сложностью применяются схемы *комбинированного переноса*. Это так называемые *сумматоры групповой структуры*. В них схема общей разрядности n делится на k групп разрядности m ($n = k \cdot m$). Внутри групп и между группами могут использоваться различные виды переноса. Детально с основными вариантами организации групповых сумматоров можно ознакомиться по источнику [6].

Арифметико-логическое устройство (АЛУ, Arithmetic-Logic Unit - ALU) выполняет ряд базовых логических и арифметических операций (*микроопераций*) над словами, позволяющих воспроизводить функции произвольной сложности. АЛУ имеет в своем составе сумматор, схемы базовых логических операций, а также схемную логику, обеспечивающую дополнительные функциональные возможности и перестройку с одной операции на другую. АЛУ, оперирующее n-разрядными словами, обычно строится из одноразрядных АЛУ с формированием последовательных или параллельных переносов. Для иллюстрации базовых принципов построения и функционирования АЛУ рассмотрим простейшую одноразрядную структуру, представленную на рис. 3.13.

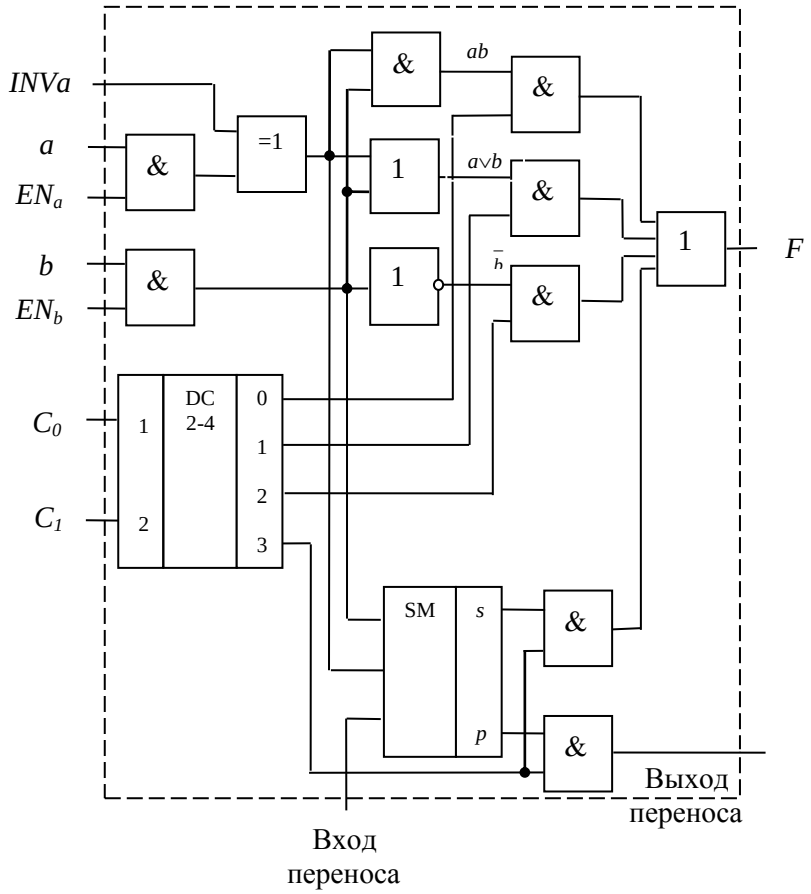


Рис. 3.13. Одноразрядное АЛУ

Под управлением кода C_1C_0 данная схема способна формировать на выходе F одну из четырех функций над одноразрядными словами a и b:

$$ab, \quad a \vee b, \quad \bar{b}, \quad a + b.$$

В результате дешифрации управляющего кода активизируется соответствующая линия разрешения, пропускающая на выход вычисленное значение выбранной функции.

В схеме предусмотрена также возможность делать a и b равными нулю с помощью отрицаний сигналов разрешения EN_a (Enable a) и EN_b (Enable b), соответственно. Активизируя сигнал INV_a (инверсия a), можно получить на выходе при разрешающих сигналах EN_a, EN_b и управляющем коде $C_1C_0 = 01$ значение функции $\bar{a} \vee b$.

Объединение необходимого числа одноразрядных АЛУ с формированием последовательных или параллельных переносов (для обеспечения операции арифметического сложения) позволяет получить АЛУ требуемой разрядности. Управление полноразрядным АЛУ осуществляется шестью сигналами: INV_a,

EN_a, EN_b, C_1, C_0 и INC. Сигнал INC (увеличение на единицу) подается на вход переноса младшего одноразрядного АЛУ. Комбинируя значения управляющих сигналов можно настраивать АЛУ на выполнение различных функций, некоторые из которых представлены в таблице 2.

Таблица 2.

Некоторые из возможных комбинаций значений управляющих сигналов АЛУ и соответствующие им функции на выходе

INVa	EN_a	EN_b	C_0	C_1	*INC*	Функция (*F*)
0	1	1	0	0	0	ab
0	1	1	1	0	0	$a \vee b$
0	1	1	0	1	0	$\overline{b}$
0	1	1	1	1	0	$a + b$
0	1	1	1	1	1	$a+b+1$
0	1	0	1	1	1	$a + 1$
1	1	0	1	1	1	$- a$
1	1	1	1	1	1	$b - a$
1	0	1	1	1	0	$b - 1$

3.2. Функциональные узлы последовательностной логики

Функциональные узлы и устройства *последовательностной логики* [8], [9] в отличие от комбинационных цепей содержат элементы памяти. Поэтому их называют *автоматами с памятью* (АП).

АП можно представить в виде двух взаимосвязанных составляющих: *памяти* и *комбинационной* цепи. Совокупность состояний всех элементов памяти определяет внутреннее состояние Q автомата.

Вектор входных сигналов *x* в зависимости от текущего состояния АП Q^k переводит его в новое состояние Q^{k+1} и порождает на выходе вектор сигналов y. Поведение такого автомата, называемого *автоматом Мили*, описывается функциями вида:

$$Q^{k+1} = \delta(Q^k, x); \quad y = \lambda(Q^k, x), \quad k = 0, 1, 2, \ldots,$$

где δ называется *функцией переходов*, а λ - *функцией выходов.*

В отличие от автомата Мили, *автомат Мура* характеризуется зависимостью выходных сигналов только от его нового состояния, т.е. описывается уравнениями вида:

$$Q^{k+1} = \delta(Q^k, x); \quad y = \mu(Q^{k+1}), \quad k = 0, 1, 2, \ldots,$$

где μ называют *сдвинутой функцией выходов*.

Для АП характерно наличие в их структуре обратных связей, посредством которых реализуются функции запоминания состояний. АП делятся на *асинхронные* и *синхронные*.

Асинхронный АП переходит из одного состояния в другое под воздействием изменений входных сигналов. Роль элементов памяти в нем выполняют элементы задержки сигналов обратных связей (сигналов состояния) или асинхронные элементы памяти (конденсатор с ключевым транзистором, асинхронные защелки и триггеры), управляемые комбинационной цепью. Сигналы состояния вместе с новым набором входных сигналов определяют следующее состояние и вектор выходных сигналов АП.

Синхронный АП меняет свое состояние под воздействием входных сигналов и сигналов состояния в строго определенные моменты времени в соответствии с частотой следования синхронизирующих импульсов тактового генератора. Для фиксации состояния автомата используются синхронные элементы памяти (конденсатор с синхронизируемым ключевым транзистором, синхронные защелки и триггеры).

АП может быть *автономным*, т.е. не иметь информационных входов и функционировать в соответствии с реализованным в нем алгоритмом под воздействием тактовых сигналов синхрогенератора.

Простейшими (элементарными) АП являются запоминающие элементы: конденсатор с ключевым транзистором, защелки и триггеры различных типов.

Схема запоминающего элемента в виде конденсатора с ключевым МДП-транзистором представлена на рис. 3.14. Исток транзистора подсоединен к линии записи-считывания. Сток транзистора не имеет внешнего вывода и образует одну из обкладок конденсатора. Другой обкладкой является подложка. Линия выборки, управляя затвором транзистора, обеспечивает отключение или подключение запоминающего конденсатора к линии записи-считывания.

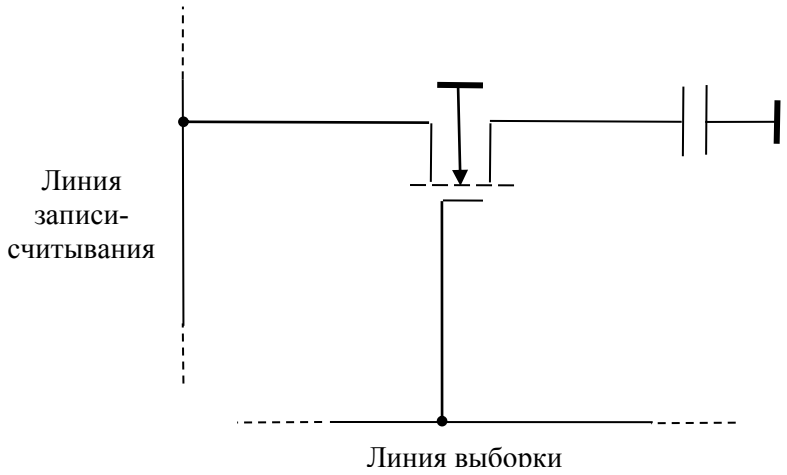


Рис. 3.14. Схема запоминающего элемента в виде конденсатора с ключевым МДП-транзистором

При записи логической единицы конденсатор заряжается, при записи нуля – разряжается. При считывании логической единицы линия записи-считывания, предварительно заряженная до половины напряжения питания, принимает на себя часть заряда конденсатора, что приводит к увеличению ее напряжения. При считывании логического нуля предзаряженная линия записи-считывания отдает часть заряда конденсатору, снижая свое напряжение. Измененное напряжение линии преобразуется специальной схемой усилителя-регенератора в сигнал логической «1» или логического «0», который одновременно является считанной информацией и сигналом восстановления исходного состояния конденсатора.

Простейшая *асинхронная RS-защелка* может быть построена на двух логических элементах ИЛИ-НЕ по схеме, представленной на рис. 3.15. S (*Set*) – вход установки защелки в единичное состояние ($Q = 1$). R (*Reset*) – вход сброса защелки в нулевое состояние ($Q = 0$). Условное обозначение RS-защелки и ее таблица состояний приведены, соответственно, на рисунке 3.16 и в таблице 3.

По аналогичной схеме может быть построена асинхронная RS-защелка на основе логических элементов И-НЕ, только входы и выходы у нее поменяются на инверсные.

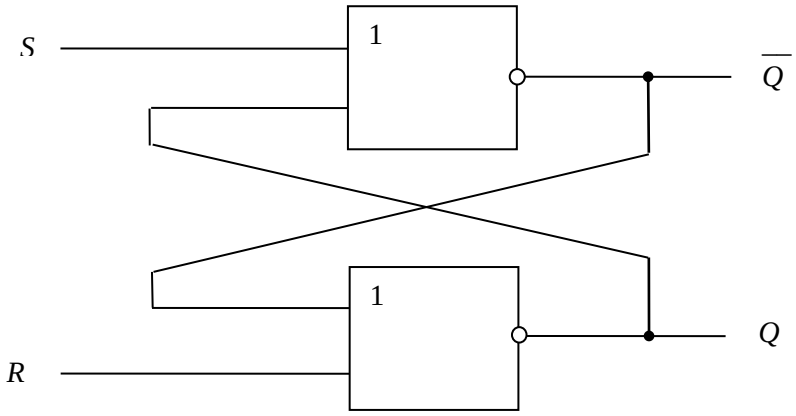


Рис. 3.15. RS-защелка на логических элементах ИЛИ-НЕ

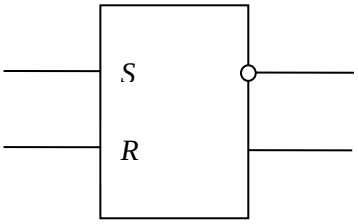


Рис.3.16. Условное обозначение RS-защелки на логических элементах ИЛИ-НЕ

Таблица 3.
Таблица состояний RS-защелки на логических элементах ИЛИ-НЕ

S^k	R^k	Q^{k+1}	$\overline{Q}^{k+1}$	Примечание
1	1	0	0	Запрещенная комбинация
1	0	1	0	Установка в «1»
0	1	0	1	Сброс в «0»
0	0	Q^k	$\overline{Q}^k$	Хранение

Синхронная RS-защелка помимо информационных входов R и S имеет вход C (*Clock*) для сигнала синхронизации. Его назначение: обеспечить одновременное (синхронное) изменение состояний многих защелок или выделить из информационного потока нужную часть сигнала.

Один из возможных вариантов функциональной схемы синхронной RS-защелки приведен на рис. 3.17. При $C = 0$ защелка находится в состоянии хранения информации, не реагируя на информационные сигналы. При $C = 1$ схема функционирует как асинхронная RS-защелка. Условное обозначение синхронной RS-защелки представлено на рис. 3.18.

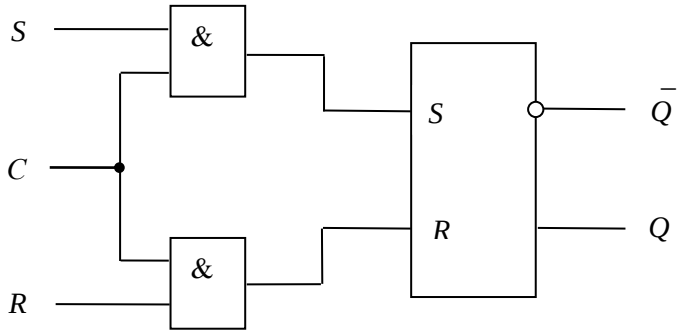


Рис. 3.17. Синхронная RS-защелка

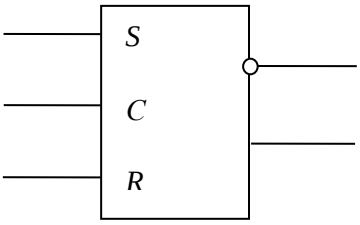


Рис. 3.18. Условное обозначение синхронной RS-защелки

Синхронная D-защелка (схема на рис. 3.19) представляет собой память объемом 1 бит. Она обеспечивает возможность записи информации по одному входу D, когда разрешающий сигнал C принимает значение 1. Ее условное обозначение приведено на рис. 3.20.

В отличие от синхронных защелок, управляемых уровнем разрешающего сигнала, *триггер* реагирует на информационные сигналы под воздействием перепада тактового сигнала (нарастающего или спадающего фронта импульса). Поэтому его можно назвать защелкой с динамическим управлением. Одна из возможных логических схем триггера изображена на рис. 3.21.

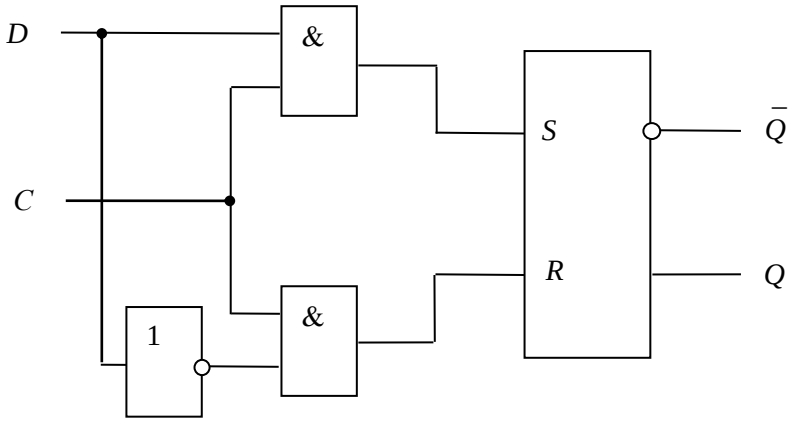


Рис. 3.19. Синхронная D-защелка

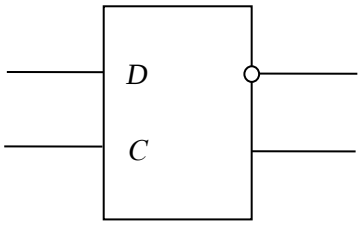


Рис. 3.20. Условное обозначение синхронной D-защелки

RS-триггер, схема которого представлена на рис. 3.21, управляется нарастающим фронтом тактового импульса. При $C = 0$ RS-защелка на выходе триггера поддерживается в режиме хранения. При переходе C из 0 в 1 состояние защелки устанавливается в соответствии со значениями сигналов на информационных входах $\overline{S}, \overline{R}$, после чего информационные входы блокируются до перехода C в состояние 0. Условное обозначение триггера, управляемого нарастающим фронтом тактового импульса, приведено на рис. 3.22 (метка «/» или «▷» на входе для сигнала синхронизации означает управление нарастающим фронтом тактового импульса).

На основе RS-триггера по схеме, приведенной на рис. 3.23, легко получается D-триггер, условное обозначение которого представлено на рис. 3.24.

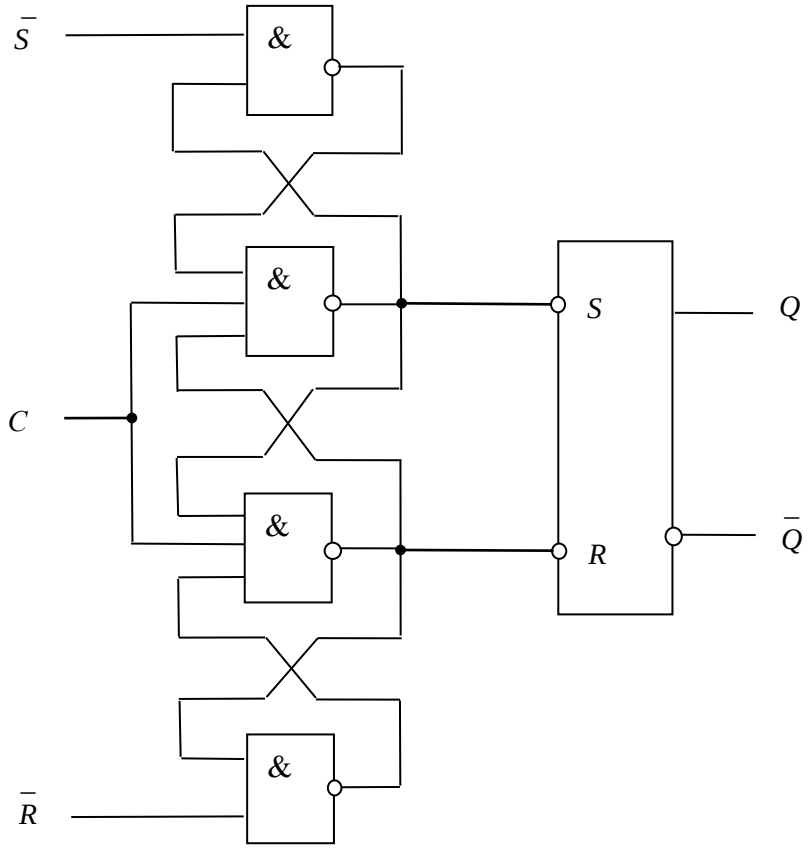


Рис. 3.21. Схема RS-триггера, управляемого нарастающим фронтом тактового импульса

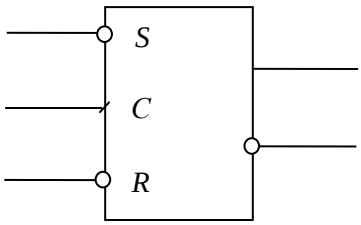


Рис. 3.22. Условное обозначение RS-триггера, управляемого нарастающим фронтом тактового импульса

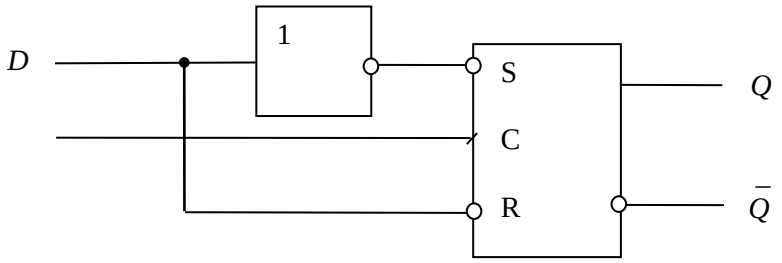


Рис. 3.23. Схема D-триггера, управляемого нарастающим фронтом тактового импульса

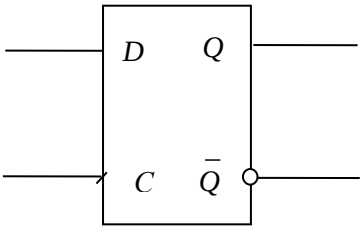


Рис. 3.24. Условное обозначение D-триггера, управляемого нарастающим фронтом тактового импульса

Перечень триггерных устройств может быть дополнен так называемыми *счетными триггерами* (JK-триггер, T-триггер), *двухступенчатыми триггерами* с различными способами управления ступенями и т.д. Информацию о них можно найти в источнике [6].

К наиболее распространенным типовым функциональным узлам последовательностной логики относятся *регистры* различного назначения, *регистровые файлы, счетчики*.

Регистр – многоразрядное (словарное) устройство, построенное из триггеров (или синхронных защелок) и дополнительных логических элементов, обеспечивающих выполнение возложенных на него функций при приеме, хранении и выдаче информации, сдвиге информации в разрядной сетке, выполнении поразрядных логических операций над словами.

По способам приема и выдачи кода, характеру выполняемых над ним преобразований различают *параллельные (статические)* регистры, *последовательные (сдвигающие)* регистры, *параллельно-последовательные* регистры и *универсальные* регистры.

Параллельные регистры принимают и выдают слова по всем разрядам одновременно, что обусловливает их использование для хранения слов при поразрядных логических преобразованиях.

Последовательные регистры принимают и выдают слова разряд за разрядом, сдвигая их в разрядной сетке. Различают последовательные регистры с однонаправленным и реверсивным сдвигом. Пример схемы последовательного n-разрядного регистра с потактовым сдвигом на один разряд вправо, построенного на базе D-триггеров и управляемого синхронизирующим сигналом C, приведен на рис. 3.25.

Параллельно-последовательные регистры комбинируют в себе параллельный и последовательный принципы обмена информацией как по

входу, так и по выходу, реализуя всевозможные сочетания способов приема и выдачи слов.

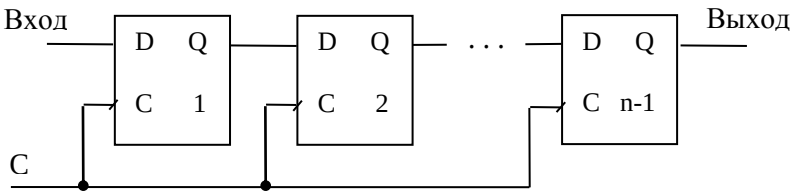


Рис. 3.25. Схема последовательного n-разрядного регистра с потактовым сдвигом на один разряд вправо

Универсальные регистры интегрируют в себе аппаратные возможности выполнения набора микроопераций, например, параллельной загрузки, сдвига влево, сдвига вправо, сброса, хранения. Конкретный вид исполняемой в данный момент операции определяется соответствующими значениями управляющих сигналов.

Регистровый файл представляет собой блок регистровой памяти, составленный из статических регистров и обеспечивающий возможность независимой одновременной записи и считывания слов двух разных регистров.

Регистровая память может наращиваться как по количеству хранимых слов, так и по их разрядности путем объединения готовых блоков по правилам, изложенным в [6].

Счетчик - это автомат с памятью, фиксирующей в определенном коде число поступивших на его вход импульсов.

Характерными для счетчиков различных типов являются микрооперации сброса, установки, инкрементирования, декрементирования и другие.

Число возможных состояний счетчика определяет его *емкость* (*модуль счета)* и длину одного цикла работы. После поступления на вход очередного импульса с порядковым номером, равным модулю счета, начинается новый цикл, аналогичный предыдущему.

Цифровые элементы и функциональные узлы являются основой для построения цифровых устройств более сложного функционального назначения. Это могут быть микропроцессоры (универсальные или специального назначения), запоминающие устройства, интерфейсные схемы и другая цифровая аппаратура ВС.

4. АРХИТЕКТУРА ПАМЯТИ ЦИФРОВЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

4.1. Назначение, основные параметры и общее представление иерархии запоминающих устройств

Память современных компьютеров – это сложная иерархическая система запоминающих устройств, отличающихся друг от друга выполняемыми функциями, принципом действия, конструктивно-технологическим исполнением, быстродействием, информационной емкостью, энергопотреблением и другими параметрами.

Запоминающие устройства (ЗУ) компьютера предназначены для хранения информации и обмена ею с другими его устройствами (процессором, периферийным оборудованием). В качестве основных параметров, принимаемых во внимание при выборе той или иной архитектуры запоминающих устройств, рассматриваются следующие.

Информационная емкость определяется как выраженный в битах (байтах, словах) максимально возможный объем хранимой информации. Этот объем зависит от организации ЗУ, т.е. от его архитектуры, определяющей тип используемых элементов памяти, принципы доступа к ним, разрядность хранимых слов, объем адресного пространства.

Быстродействие памяти характеризуется *временем доступа* и *длительностью цикла***.** *Время доступа* (access time) – это задержка появления действительных данных на выходе памяти относительно начала цикла чтения. Т.е. это длительность активной фазы обращения к памяти, после которой следует фаза восстановления. *Длительность цикла* определяется как минимальный период следования обращений к памяти.

Производительность памяти оценивается как скорость потока записываемых или считываемых данных, измеряемая в Мбайтах (или других масштабных единицах информации) в секунду и определяемая тремя факторами:

типом и *быстродействием* применяемых запоминающих устройств (микросхем памяти, модулей памяти);

разрядностью шины памяти (количеством одновременно считываемых или записываемых битов информации);

особенностями архитектуры памяти (быстрый страничный режим, конвейеризация доступа к памяти, чередование банков и т.д.).

Немаловажным параметром, принимаемым во внимание при планировании архитектуры памяти, является ее *стоимость*. Именно этот параметр часто сдерживает стремление увеличить информационную емкость памяти или повысить ее быстродействие.

Для обеспечения разумного компромисса между основными параметрами память современных вычислительных систем организована в виде многоуровневой иерархической структуры различных типов ЗУ, представленной на рис. 4.1.

Наименьшим временем доступа характеризуются *регистры* процессора. Они расположены ближе всего к его исполнительному тракту. Некоторые из них, можно сказать, входят в его состав, т.к. непосредственно задействованы в процессе интерпретации машинных команд на микроархитектурном уровне. Они избавляют от необходимости обращения к более медленным уровням памяти (кэш-памяти, основной памяти) для записи промежуточных результатов, получаемых в ходе выполнения инструкций. Таких регистров немного, ибо увеличение их количества на кристалле процессора может осуществляться только за счет сокращения его аппаратных функциональных возможностей.

Продолжением реализации идеи сочетания маленькой, но быстрой памяти с большой, но медленной, является *кэш-память*. Это небольшая, но высокоскоростная память, предназначенная для хранения копий наиболее часто используемой информации из основной (оперативной) памяти (команд, данных). Любую необходимую информацию процессор прежде всего ищет в кэш-памяти и только при отсутствии ее там обращается к основной памяти.

Кэш-память строится по многоуровневому принципу. На рис. 4.1 представлены три уровня кэш-памяти.

Самой близкой к процессору (расположенной на его кристалле) и самой быстрой является *кэш-память 1-го уровня* (L1-cache). Она подразделяется на отдельные *кэш-память команд* и *кэш-память данных*. Это позволяет независимо оперировать обоими кэшами, что способствует увеличению пропускной способности памяти и применению индивидуальных технологий для оперирования данными и командами. Так в процессоре Pentium IY использован нетрадиционный подход к организации внутренней кэш-памяти команд. Его так называемый Trace Cache хранит не машинные инструкции, а готовые микропрограммы их интерпретации, с которыми непосредственно оперирует вычислительный тракт процессора, в том порядке, в котором они должны выполняться.

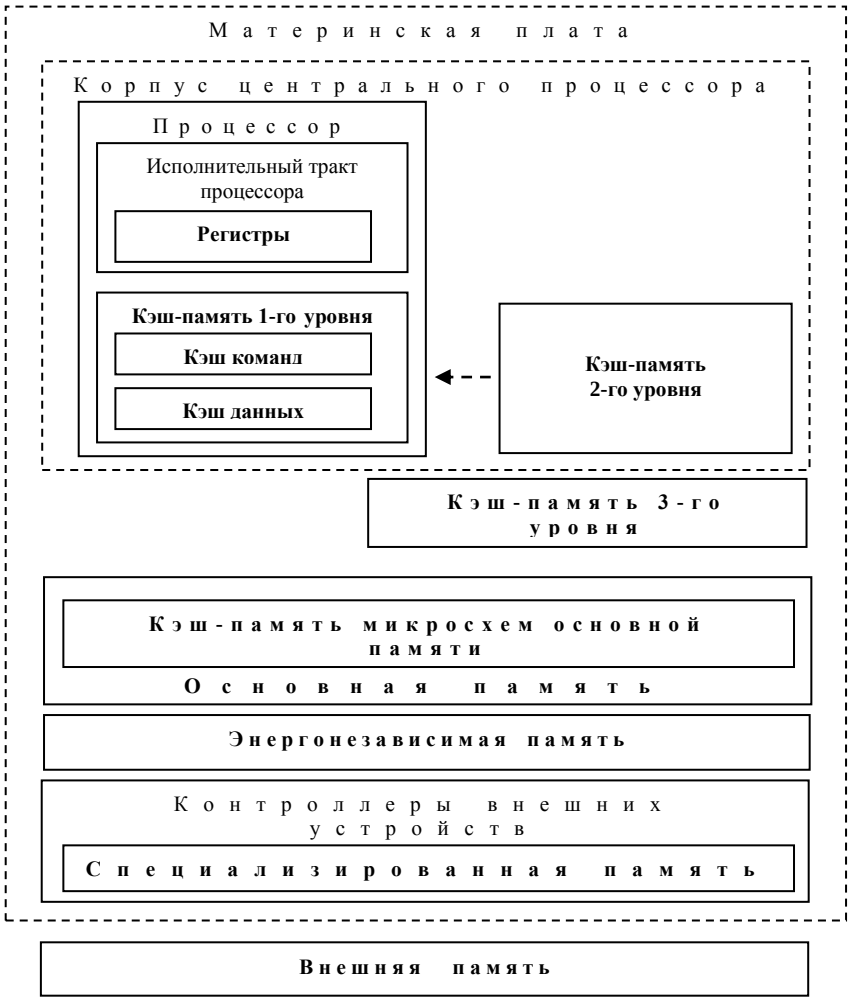


Рис. 4.1. Многоуровневая иерархическая структура памяти

Кэш-память 2-го уровня (L2-cache) традиционно дальше от процессора. Ее кристалл либо интегрирован в одном корпусе с кристаллом процессора (Pentium Pro, Pentium II, Pentium Ш) и соединен с ним высокоскоростным трактом данных, работающим на частоте его ядра. Либо, как это сделано в Pentium IY, объединенный L2-кэш команд и данных (256 Кбайт) интегрирован прямо на кристалле процессора (на рис. 4.1 этот вариант расположения кэш-памяти 2-го уровня отмечен пунктирной стрелкой).

Кэш-память 3-го уровня (L3-cache) – это статическая память объемом в несколько Мбайт, размещенная на материнской плате, но имеющая время доступа, значительно меньшее, чем основная динамическая память. У процессоров класса Pentium кэш-память 3-го уровня отсутствует (исключение составляют некоторые реализации процессора Pentium IY).

Таким образом, последовательное копирование наиболее часто используемой информации из основной памяти в кэш-память 3-го уровня, далее

в кэш-память 2-го уровня, а затем в кэш 1-го уровня при рациональной организации механизмов доступа к копируемым данным способствует повышению общей производительности памяти в целом.

Следует заметить, что кэш-память не адресуема пользователем, для него она «прозрачна». Кэш-память доступна процессору.

Наиболее сбалансированным звеном с точки зрения основных характеристик (информационная емкость, быстродействие, стоимость) является *основная (оперативная)* память. Это адресуемая пользователем память с произвольным доступом (RAM – Random Access Memory), допускающая возможность записи и чтения любых ее ячеек в произвольном порядке. Она хранит исполняемую в текущий момент программу, связанные с ней данные и является средством оперативного обмена информацией (командами, данными) между процессором, внешней памятью и периферийными устройствами.

Одним из основных требований, предъявляемых к оперативной памяти, является большая (для полупроводниковой памяти) информационная емкость. Она может исчисляться сотнями Мбайт и уходить в Гбайты, увеличиваясь до предела физического адресного пространства процессора с учетом архитектурных и конструктивно-технологических особенностей материнской платы, на которой она располагается.

Для обеспечения необходимой разрядности ячеек, соответствующей ширине шины данных, формируется так называемый *банк памяти* (bank). Это комплект одинаковых по типу и объему микросхем или модулей памяти с их посадочными местами. Работоспособным является только полностью заполненный банк. В банк может входить *микросхема четности*, предназначенная для проверки четности остальных микросхем, входящих в состав банка (например, 8 1Мбитных чипов и 9-й 1Мбитный *чип четности*, т.е. каждый байт фактически дополняется девятым битом – *битом паритета* – parity bit).

Необходимый объем оперативной памяти может набираться несколькими банками, причем с применением механизма *чередования банков* (bank interleaving). При чередовании банков смежные блоки данных располагаются поочередно в разных банках. Тогда при последовательном обращении к данным банки работают поочередно, и активная фаза обращения к одному банку выполняется параллельно с фазой восстановления другого банка. Для этого предусматриваются специальные средства перекоммутации адресных линий памяти и раздельные линии управляющих сигналов для банков.

Энергонезависимая память хранит записанную информацию при отсутствии питающего напряжения от сети, а *постоянная память*, как разновидность энергонезависимой памяти, не использует даже автономные источники питания. Основным режимом работы такой памяти является считывание данных. Энергонезависимая память, как правило, применяется для хранения неизменяемой или редко изменяемой информации системного плана, требующей защиты от несанкционированного изменения. Такой информацией прежде всего является BIOS (Basic Input-Output System – *базовая система ввода-вывода*) – самый нижний уровень программного обеспечения, «оживляющий» железо компьютера. Сюда следует также отнести память конфигурации устройств компьютера, таблицы знакогенераторов графических адаптеров.

К разряду *специализированной памяти* на рис. 4.1 отнесена *последовательная полупроводниковая память*, используемая контроллерами периферийных устройств. Это буферы памяти FIFO, LIFO, файловые и циклические запоминающие устройства видеопамяти.

Последний, самый отдаленный от процессора и самый объемный уровень памяти - *внешняя память*. Она ориентирована на хранение больших объемов информации, реализуется на основе устройств с подвижными носителями информации (например, магнитные и оптические диски). Внешняя память принципиально отличается от внутренней (оперативной, энергонезависимой) памяти способом доступа процессора к ее содержимому. Ее устройства оперируют блоками информации, а не байтами и словами, как оперативная память.

4.2. Базовые принципы организации адресной памяти

К *адресной памяти* относятся все типы ЗУ, доступ к любой ячейке которых осуществляется по заданному адресному коду (*физическому адресу ячейки*). Под *ячейкой памяти* традиционно понимается совокупность элементов памяти, имеющих общий адрес и предназначенных для записи, хранения и считывания информации словами. В случае одноразрядной ячейки памяти ЗУ называется *одноразрядным*, при многоразрядной организации ячеек - *словарным (многоразрядным)*.

К адресной памяти в первую очередь относится оперативная память (RAM), которая делится на *динамическую* (DRAM – Dynamic RAM) и *статическую* (SRAM – Static RAM). Эти два типа оперативной памяти в своей

основе отличаются тем, что используется в качестве *элемента памяти* (хранителя одного бита информации) матрицы накопителя микросхемы памяти.

DRAM использует для запоминания одного бита информации конденсатор с ключевым транзистором (рис. 3.14). Однако конденсаторы не способны сохранять заряд длительное время, поэтому они требуют периодического восстановления (регенерации) своего состояния (memory refresh). Это может осуществляться в режиме холостых (без выдачи данных на выходные буферы) построчных циклов чтения, организация которых приводит к усложнению механизма управления памятью. Но с другой стороны, аппаратные затраты на реализацию одного элемента памяти микросхемы DRAM значительно меньше затрат на реализацию триггера статической памяти. В итоге микросхемы DRAM оказываются самыми дешёвыми, обладают большей информационной ёмкостью по сравнению со статическими микросхемами при одинаковых размерах кристалла, обеспечивают приемлемое быстродействие (характерное время доступа 60-70 нс) и умеренное энергопотребление. Поэтому они используются в качестве элементной базы основной памяти.

Элементы памяти SRAM**,** реализуемые на триггерах, занимают больше места на кристалле (отсюда небольшая информационная емкость), но проще в управлении и не требуют регенерации. Время доступа к ним, в зависимости от схемотехники и технологии изготовления запоминающих ячеек, определяется диапазоном 3 – 6 нс. Именно это обусловливает применение SRAM в качестве элементной базы кэш-памяти различных уровней.

К адресной памяти относится и энергонезависимая память, которая, в отличие от динамической памяти, имеет гораздо больше общего в архитектуре со статической памятью.

С точки зрения базовых принципов организации доступа к запоминающим элементам целесообразно рассмотреть обобщенные структуры памяти 2D, 3D и 2DM [6]. Структуры 2D и 2DM характерны для микросхем SRAM и энергонезависимой памяти, 3D чаще используется в микросхемах DRAM.

Структура 2D (D – аббревиатура английского слова Dimension – измерение) на примере статической памяти с максимально возможной степенью детализации представлена на рис. 4.2.

Матрица накопителя приведенной схемы объединяет в себе 2^n строк D-триггеров, каждая из которых идентифицируется своим n- разрядным адресом и содержит слово, разрядность которого определяется числом столбцов матрицы накопителя m. Код адреса, поступающий с адресных линий на вход

дешифратора, активизирует соответствующую ему строку (ячейку памяти) для записи или считывания информации.

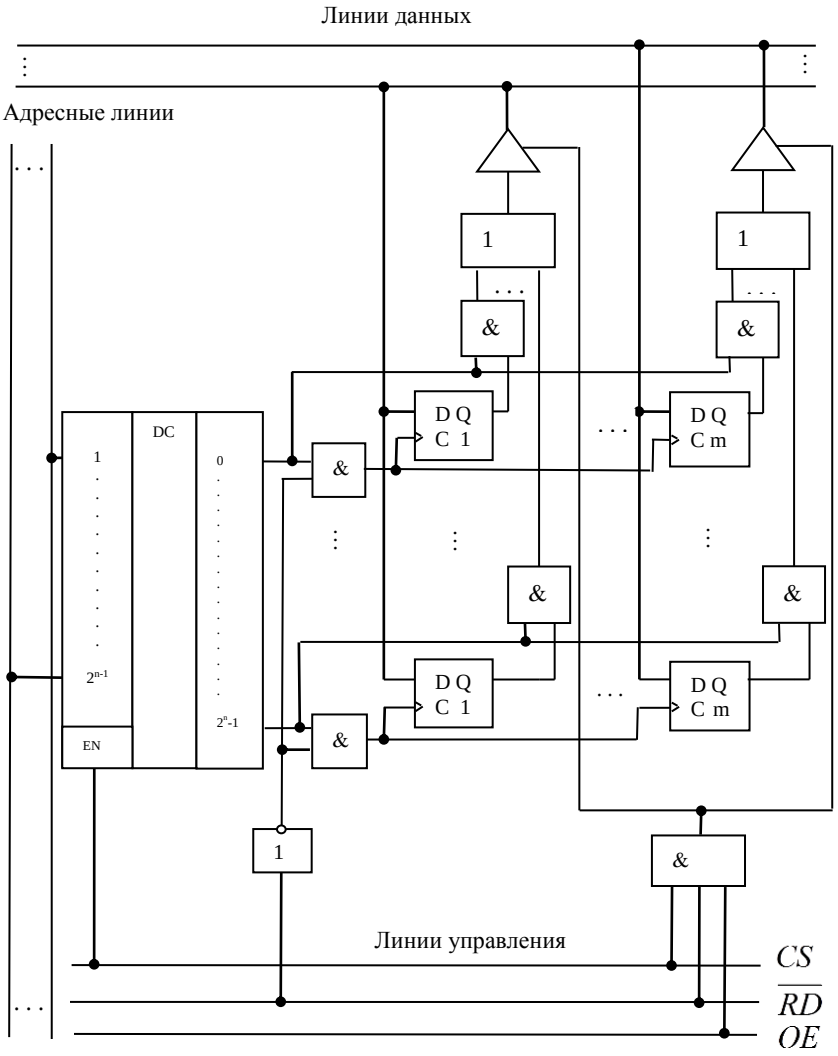


Рис. 4.2. Структура статического ЗУ типа 2D

Режим записи определяется активными сигналами CS (*Chip Select*) и $\overline{RD}$ ($\overline{\mathrm{Re}\,ad}$). Сигнал CS разрешает дешифрацию адресного кода. Сигнал $\overline{RD}$ в сочетании с активным сигналом на одном из выходов дешифратора устанавливает активные сигналы C для триггеров соответствующей строки, которые считывают информацию с линий данных. Другие выходы на эти линии в это время блокированы буферными элементами (обозначены треугольниками с сигналами управления сбоку), которые находятся в третьем состоянии (сигналы управления буферами имеют значения «0» из-за $\overline{RD} = 0$).

Считывание информации происходит при активных сигналах CS, OE(*Output Enable*) и пассивном сигнале $\overline{RD}$. Сигнал CS все так же активизирует дешифратор на выбор строки для считывания. Кроме того, в

сочетании с пассивным сигналом $\overline{RD}$ и активным сигналом *OE* он приводит в активное состояние буферные элементы выхода на линии данных. В это время сигнал $\overline{RD}$ блокирует входы триггеров читаемой строки от записи.

Недостатком структуры 2D является чрезмерное усложнение дешифратора адреса при увеличении количества хранимых слов, что ограничивает ее применение для создания ЗУ большой емкости.

Использовать более простые дешифраторы позволяет *структура* 3D, реализующая принцип двухкоординатной адресации к элементам памяти. Одноразрядное ЗУ данного типа без развернутого представления матрицы накопителя и схем выбора элемента памяти изображено на рис. 4.3.

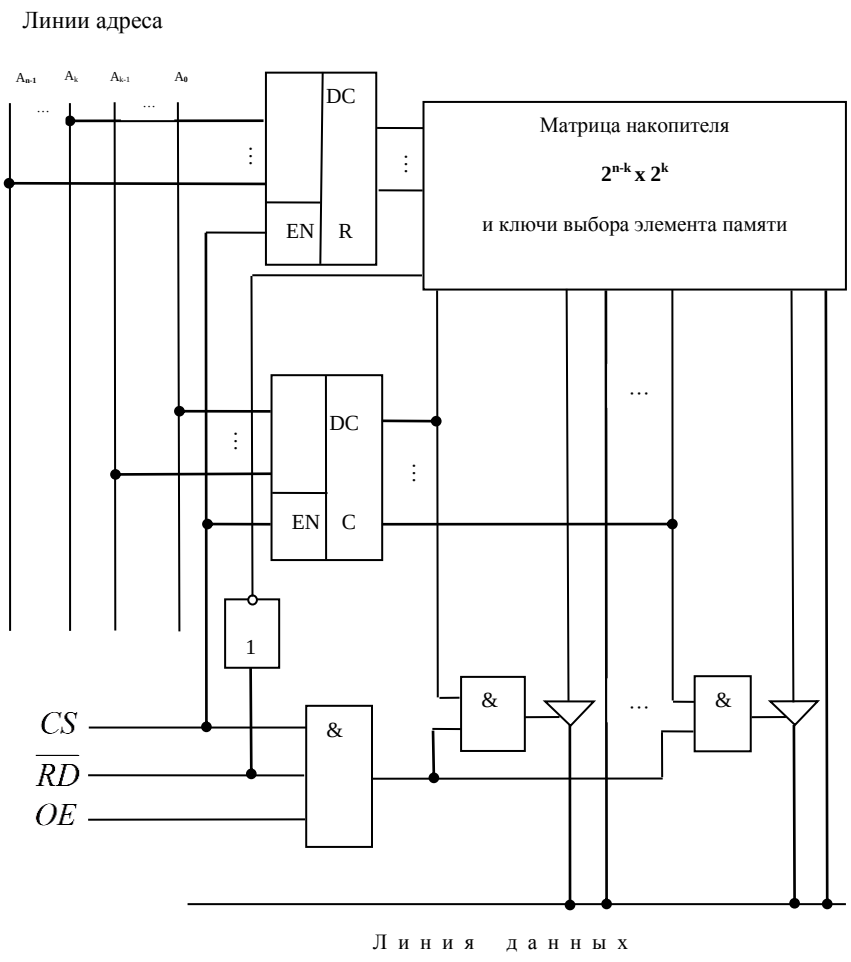


Рис. 4.3. Структура ЗУ типа 3D

Каждый элемент памяти матрицы накопителя структуры 3D, хранящий один бит информации, имеет свой адрес. При обращении к элементу для записи или считывания его адрес подаётся в виде двух составляющих $A_{n-1},...,A_k$ и $A_{k-1},...,A_0$ (не обязательно равной разрядности) на входы дешифраторов кода

адреса строк (DC_R) и кода адреса столбцов (DC_C), соответственно. В случае n - разрядного адреса информационная ёмкость матрицы накопителя равна $2^{n-k} \cdot 2^k = 2^n$ бит. При этом общее число выходов дешифраторов $2^{n-k} + 2^k = 2^n(2^{-k} + 2^{k-n})$ при определенном k оказывается существенно меньше, чем 2^n структуры 2D (например, при $k = \frac{n}{2}$ меньше в $2^{\frac{n}{2}-1}$ раз). Кстати, деление адреса на две равные части позволяет сократить в два раза число адресных выводов микросхемы памяти. На них последовательно могут подаваться и демультиплексироваться на соответствующие разряды внутреннего входного регистра старшая и младшая половины адреса, распределяемые между дешифраторами DC_R и DC_C, соответственно.

Параллельное включение относительно дешифраторов m одноразрядных матриц накопителей, управляемых одними и теми же сигналами, позволяет получить m - разрядное ЗУ.

Наращивание одноразрядных компонент структуры 3D для обеспечения большой разрядности слов может оказаться проблематичным. Поэтому в качестве компромиссного варианта, компенсирующего недостатки архитектур 2D и 3D, используется структура 2DM (2D Modified), представленная на рис. 4.4. В отличие от архитектуры 2D, структура 2DM хранит в строке матрицы накопителя не одно m - разрядное слово, а 2^k m - разрядных слов. Это уменьшает число выходов дешифратора адреса строк DC_1 , т.к. строка выбирается по $n-k$ старшим разрядам адресного кода. Доступ к слову в строке при чтении осуществляется мультиплексированием выходов триггеров строки под управлением младших разрядов адреса. Доступ к слову в строке при записи реализуется с помощью дешифратора младших разрядов адреса DC_2.

Реализуя те или иные способы организации матрицы накопителя и механизма доступа к элементам памяти, разные типы ЗУ (статические, энергонезависимые, динамические) отличаются схемотехникой запоминающих элементов, схемами управления чтением и записью, а также аппаратной реализацией различных режимов их функционирования.

Наиболее фундаментальные отличия наблюдаются у микросхем DRAM. Использование в качестве запоминающего элемента конденсатора с ключевым транзистором экономит аппаратные затраты на реализацию матрицы накопителя, но требует дополнительных схем регенерации данных при считывании, а также специальных аппаратных средств Memory Refresh. Последние могут быть реализованы внешним контроллером динамической памяти или быть интегрированными на кристалле микросхемы памяти

(*квазистатическая память*). Рассмотрению базовых принципов организации и тенденций развития микросхем динамической памяти посвящен материал следующего подраздела.

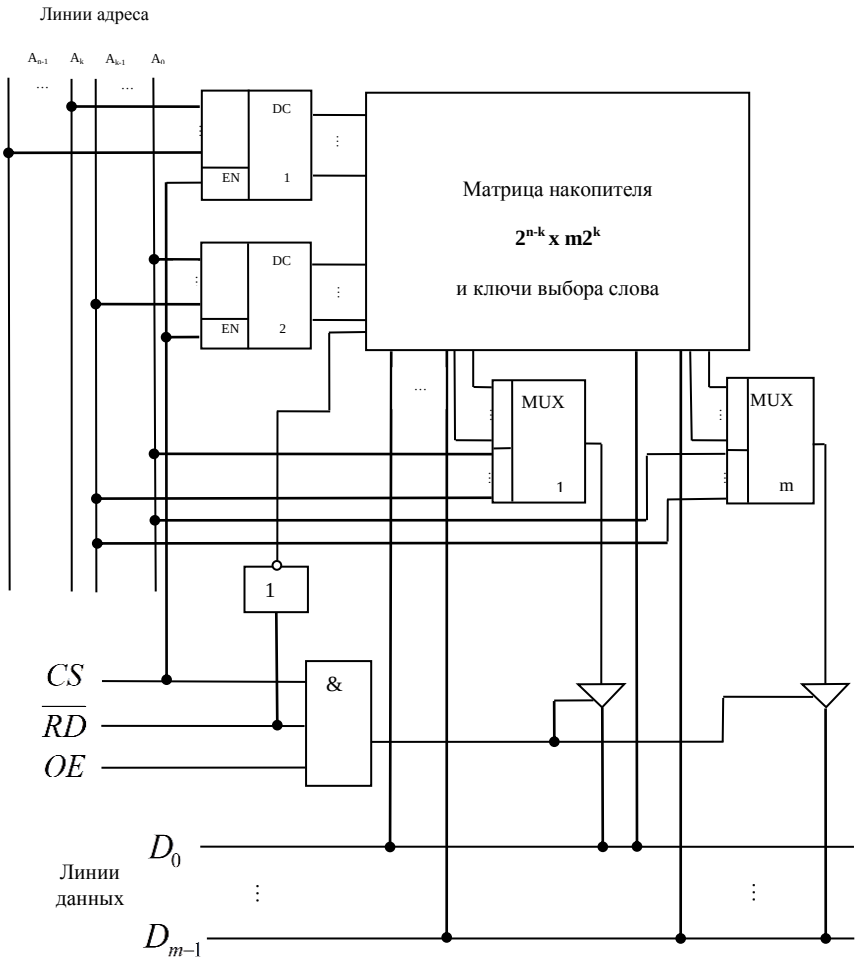


Рис. 4.4. Структура ЗУ типа 2DM

4.3. Микросхемы и модули динамической памяти

Из рассмотренных ранее структур памяти для микросхем динамической памяти характерна структура 3D. Микросхема DRAM, как функционально и конструктивно законченное микроэлектронное изделие, определяется следующим составом входных и выходных сигналов.

$\overline{RAS}$ ($\overline{Row\ \ Address\ \ Strobe}$) – строб выборки адреса строки. По спаду этого низкоактивного входного сигнала начинается цикл обращения и отсчет времени доступа к памяти.

$\overline{CAS}$ ($\overline{Column\ \ Address\ \ Strobe}$) – строб выборки адреса столбца. Это тоже низкоактивный входной сигнал, по спаду которого начинается запись или чтение идентифицированной ячейки памяти. По сути, активное состояние этого сигнала во время активности сигнала $\overline{RAS}$ и определяет выбор микросхемы (активное состояние сигнала *CS*) .

MA (*Multiplexed Address*) – сигналы мультиплексированных входных линий адреса. Во время спада сигнала $\overline{RAS}$ на этих линиях уже должен присутствовать адрес строки, а во время спада сигнала $\overline{CAS}$ – адрес столбца. Как уже отмечалось, разрядности адресов строки и столбца могут не совпадать.

$\overline{WE}$ ($\overline{Write\ \ Enable}$) – низкоактивный входной сигнал разрешения записи. Есть два варианта записи в выбранную ячейку. Так называемый *обычный* вариант (*Early Write – ранняя запись*) предусматривает запись данных по спаду $\overline{CAS}$ при низком уровне $\overline{WE}$. Другой вариант (*Delayed Write – задержанная запись*) предполагает осуществление записи по спаду $\overline{WE}$ при низком уровне $\overline{CAS}$. При высоком уровне $\overline{CAS}$ изменение сигнала $\overline{WE}$ записи не вызывает, а только переводит выходной буфер (если таковой есть) в третье состояние.

$\overline{OE}$ ($\overline{Output\ \ Enable}$) – низкоактивный входной сигнал разрешения открытия выходного буфера для чтения данных.

DB (*Data Bit*) – выводы объединенных внутри микросхемы линий записи-считывания данных. Единые линии ввода-вывода характерны для микросхем со словарной организацией. Одноразрядные микросхемы имеют два отдельных вывода: *DI* (*Data Input*) – вход данных и *DO* (*Data Output*) – выход данных.

Обычное управление микросхемой DRAM предполагает при каждом обращении к ней поочерёдную активацию строки и столбца матрицы накопителя по спаду импульсов $\overline{RAS}$ и $\overline{CAS}$, соответственно, и чтение (при высоком уровне $\overline{WE}$) или запись (при низком уровне $\overline{WE}$) информации. Время доступа, характерное для современных микросхем DRAM, составляет 60 – 70 нс.

Вместе с тем, в свое время широкое распространение получил так называемый *режим быстрого страничного обмена* FPM (Fast Page Mode). В этом режиме обычную микросхему DRAM заставляет работать контроллер памяти. Он обеспечивает ускоренный доступ к последовательности элементов данных, расположенных в одной строке матрицы накопителя микросхемы памяти. Обращение к памяти начинается с активизации строки матрицы накопителя путём выдачи адреса строки и сопровождающего его строба $\overline{RAS}$, удерживаемого далее на низком уровне на время всех последующих циклов обращений к столбцам (для записи или считывания) по выставляемым адресам

столбцов и сопровождающим их стробам $\overline{CAS}$. Такой режим существенно экономит время за счет однократного использования фазы установки адреса строки и ее активизации (в первом цикле) и исключения этой фазы из последующих циклов обращений к элементам данной строки. При этом время доступа к элементам пакета, начиная со второго, сокращается почти вдвое. *Лучший пакетный цикл чтения* (параметр, указывающий последовательно число тактов системной шины, необходимое для считывания первого, второго, третьего и четвертого смежных элементов в режиме страничного обмена) для памяти с временем доступа 70 нс при частоте системной шины 66 МГц выглядит следующим образом: 5 – 3 – 3 – 3.

Дальнейшее свое развитие динамическая память получила в микросхемах EDO (Extended Data Out) DRAM. В отличие от обычных микросхем DRAM, информация на выходе которых держится только при низком уровне сигнала $\overline{CAS}$ и сбрасывается при переходе его на высокий уровень (выходной буфер переходит в третье состояние), в EDO DRAM для сохранения выходной информации предусмотрен статический регистр-защёлка. При низком уровне сигнала $\overline{CAS}$ он принимает выходные данные и при переходе $\overline{CAS}$ на высокий уровень фиксирует (защелкивает) их. По сигналам, открывающим выходной буфер регистра, данные могут считываться вплоть до следующего спада импульса $\overline{CAS}$. Введенное усовершенствование позволяет сократить длительность импульса $\overline{CAS}$ и обеспечивает возможность определенной конвейеризации работы памяти при чтении. Страничный режим работы с EDO-памятью называют *гиперстраничным режимом обмена* HPM (Hyper Page Mode), т.к. подъем сигнала $\overline{CAS}$ начинается до появления действительных данных на выходе микросхемы и адрес следующего столбца может выставляться раньше. Такой конвейерный режим не уменьшает время доступа к памяти, но увеличивает ее пропускную способность. EDO-память с временем доступа 70 нс в гиперстраничном режиме обмена при частоте системной шины 66 МГц обеспечивает лучший пакетный цикл 5 – 2 – 2 – 2. Однако надо заметить, что конвейеризация с использованием выходных буферов затрудняет применение технологии чередования банков. По этой причине некоторые системные платы не поддерживают Bank Interleaving для EDO-памяти.

Следующей ступенью развития конвейерной архитектуры памяти явилась BEDO (Burst EDO) DRAM. Микросхема этого типа, в отличие от EDO DRAM, содержит дополнительно специальный внутренний счётчик адреса столбцов. В режиме пакетного цикла на нее подается только первый адрес столбца, а в последующих передачах импульсы $\overline{CAS}$ только запрашивают очередные

адреса, формируемые счетчиком. Это позволяет при введении искусственного запаздывания первых выходных данных в пределах одного такта обеспечить появление последующих данных в соответствии с тактовой частотой системной шины. Лучший пакетный цикл чтения для BEDO-памяти с временем доступа 70 нс при частоте системной шины до 66 МГц 5 – 1 – 1 – 1.

Среди микросхем асинхронной памяти заслуживает внимания многобанковая архитектура MDRAM (Multibank DRAM). Она предусматривает поочередное обращение к банкам в случаях кучности адресов, исключая задержки на подготовку памяти к очередному обращению. При нарушении очередности и повторном обращении к тому же банку выполняется полный цикл обращения к памяти.

Более широкими возможностями в плане увеличения быстродействия и пропускной способности основной памяти обладают микросхемы *синхронной динамической памяти* (SDRAM – Synchronous DRAM). В отличие от рассмотренных микросхем асинхронной памяти, в которых основное распределение событий во времени определяется импульсами $\overline{RAS}$ и $\overline{CAS}$, микросхемы SDRAM представляют собой конвейеризованные устройства, тактируемые синхроимпульсами частоты системной шины. В интерфейсе SDRAM наряду с выводами, характерными для обычных микросхем DRAM, предусмотрены входы синхронизации и некоторые дополнительные выводы управления конвейером, многобанковой структурой памяти, настроечными параметрами и т.п. Все сигналы управления (включая $\overline{RAS}$ и $\overline{CAS}$) стробируются по положительному перепаду синхроимпульсов. Их комбинация в каждом такте рассматривается как код определенной *команды*. С использованием этих команд реализуется последовательность преобразований, аналогичная той, которая была рассмотрена для микросхем асинхронной памяти DRAM (FPM, EDO, BEDO, MDRAM).

В состав команд, используемых для программирования конвейера SDRAM, могут входить:

ACT (*Active*) – команда выбора строки матрицы накопителя по старшему полуадресу;

WR (*Write*) – команда выбора столбца для записи;

RD (*READ*) – команда выбора столбца для чтения;

PRE (*Precharge*) – команда предзаряда шин;

REF (*Refresh*) - команда регенерации данных.

Синхронный интерфейс SDRAM снимает многие проблемы взаимного согласования сигналов во времени. Их значения фиксируются положительными

перепадами тактовых импульсов. Особенно важно это при реализации микросхем с многобанковой структурой.

Сочетание конвейерности с мультибанковой организацией оказывается наиболее эффективным при обслуживании множественных обращений. Такие ситуации, во-первых, обусловлены характерными для современных процессоров конвейерностью, суперскалярностью, динамическим предсказанием ветвлений. Во-вторых, к обращениям, инициируемым процессором, добавляются запросы к памяти со стороны других активных устройств.

В микросхемах SDRAM с многобанковой организацией можно активизировать несколько банков соответствующими командами ACT. При этом обращение к открытой строке нужного банка осуществляется по командам *WR* и *RD*, среди параметров которых предусмотрен номер банка. Рациональное использование этих возможностей при планировании транзакций (групп логически объединённых последовательных операций по работе с данными, обрабатываемых или отменяемых целиком) позволяет организовать плотный поток данных в шине не только в рамках одного пакета обмена, но и при обслуживании множественных обращений.

Микросхемы SDRAM – это устройства с программируемыми параметрами. Настройка системы на конкретные значения параметров осуществляется на этапе ее инициализации путем программирования содержимого специального регистра режимов. Команды чтения и записи предусматривают возможность работы с программируемой длиной пакетного цикла (1, 2, 4, 8 или 256 элементов). Предусмотрена возможность прерывания последующими командами пакетных циклов предыдущих команд с отбрасыванием непереданных данных. Возможно блокирование записи или чтения любого элемента пакета установкой в соответствующем такте специального сигнала маскирования. Предусмотрено программирование задержки данных с целью оптимального согласования быстродействия памяти с частотой системной шины, а также перевод микросхемы в режим хранения данных с пониженным энергопотреблением.

Память SDRAM может обеспечивать лучший пакетный цикл чтения 5 – 1 – 1 – 1 на частотах до 133 МГц. Использование ее в составе модуля DIMM (о модулях памяти будет сказано несколько позже) с 8-байтной разрядностью при частоте системной шины 133 МГц обеспечивает максимальную производительность 1064 Мбайт/с.

В настоящее время можно выделить два основных направления развития архитектуры синхронной динамической памяти.

Одно из них, DDR (Double Data Rate – удвоенная скорость данных) SDRAM, является продолжением развития базовых принципов, заложенных в SDRAM.

В микросхемах DDR SDRAM передача данных в пакетном режиме осуществляется с удвоенной частотой – по нарастанию и спаду синхронизирующих импульсов. Ядро памяти и буфер ввода-вывода функционируют на частоте системной шины, а частота передачи данных по внешней шине (эффективная частота) в два раза ее превышает. При этом ширина внешней шины данных вдвое меньше разрядности буферного регистра. Преобразование широкого низкоскоростного потока данных в узкий высокоскоростной и обратно выполняется схемами мультиплексирования/демультиплексирования буфера ввода-вывода.

Высокая частота следования данных в DDR SDRAM предъявляет жесткие требования к точности их синхронизации. Поэтому тактирующие импульсы подаются в дифференциальной форме, как разность между прямым и обратным сигналами синхрогенератора. Кроме того, с целью синхронизации потока данных с тактами системной шины предусмотрен дополнительный двунаправленный вывод для стробирующего сигнала, сопровождающего данные. Этот сигнал генерируется источником данных (микросхемой памяти при чтении и контроллером памяти при записи). Центры нарастания и спада стробирующих импульсов при чтении определяют моменты смены данных, а при записи – середины окон действительных данных. Для согласования системы синхронизации данных с частотой системной шины в микросхеме предусмотрена специальная схема фазовой автоподстройки DLL (Delay Locked Loop), обеспечивающая согласование фронтов импульсов основной и дополнительной системы синхронизации. Наличие DLL повышает надежность и гибкость системы, но усложняет процесс программирования ее настроечных параметров.

Таким образом, архитектура DDR SDRAM обеспечивает ее 64-разрядному модулю при частоте системной шины 133 МГц пиковую производительность 2128 Мбайт/с. Модули той же разрядности, комплектуемые микросхемами DDR-400 (стандарт PC3200), при частоте системной шины 200 МГц имеют пиковую скорость обмена данными 3200 Мбайт/с.

Как развитие направления DDR SDRAM рассматриваются микросхемы DDR2 SDRAM, предусматривающие четырехэлементную выборку

последовательных данных за тактовый период синхронизации ядра памяти в пакетном режиме. Их схемы преобразования потоков данных используют мультиплексоры/демультиплексоры типов 4-1/1-4, соответственно (в микросхемах DDR SDRAM применяются менее сложные мультиплексоры/демультиплексоры типов 2-1/1-2). При этом соотношения параметров (частоты, разрядности) буферного регистра и внешней шины данных оставлены такими же, как у DDR SDRAM, но частота синхронизации ядра памяти снижена в два раза. Это способствует повышению надежности и снижению энергопотребления памяти.

Дальнейшее развитие стандартов DDR SDRAM продолжалось по пути уменьшения энергопотребления и увеличения ширины внутренней шины памяти. Характеристики различных поколений DDR SDRAM приведены в таблице 4.

Таблица 4.
Таблица характеристик поколений памяти DDR SDRAM

Характеристика	DDR	DDR2	DDR3	DDR4
Частота шины памяти, МГц	100-350	200-600	400-1200	800-1600
Пропускная способность, Мбайт/с	1600-5600	3200-9600	6400-19200	12800-25600
Напряжение, В	2,5	1,8	1,5	1,2
Размер предвыборки	2	4	8	16
Количество банков	4	4	8	16
Техпроцесс, нм	130, 90	110, 90, 75	90, 65, 50, 40	36, 32

Альтернативным направлением развития синхронной памяти является архитектура RDRAM (Rambus DRAM). Ее идея состоит в уменьшении числа линий интерфейса микросхем памяти и организации чередования большого числа банков в сочетании с высоким темпом передачи потока однобайтных или двухбайтных данных.

Запоминающее ядро микросхемы RDRAM, построено на обычных для динамической памяти элементах в виде многобанковой структуры с числом банков 8 – для 64-мегабитных микросхем и 32 – для микросхем емкостью в 256 Мбит. Для обеспечения возможности параллельной работы банков и придания максимальной гибкости системе каждый банк снабжен своими усилителями записи/считывания (исключая случаи совместного использования двумя смежными банками одних усилителей). Ядро имеет разрядность 16 байт, работает на частоте, в 8 раз меньшей частоты внешнего канала, доступ к нему осуществляется по внутренним сигналам $\overline{RAS}$ и $\overline{CAS}$.

Необходимый объем памяти может набираться из нескольких микросхем RDRAM, подключаемых через свой интерфейс к *каналу* RDRAM Rambus Channel. К этому каналу с одной стороны подключен контроллер памяти, выполняющий функции обслуживания подсоединенных к каналу микросхем памяти по запросам со стороны интерфейса системной шины. С другой стороны канал заканчивается терминаторами, не дающими сигналам отражаться от конца канала. Rambus Channel - это параллельно-последовательная шина с 30 основными линиями, тактируемая частотой до 400 МГц, предполагающая синхронизацию обоими фронтами тактовых импульсов и использование сверхбыстродействующих интерфейсных схем. Шина включает три линии строк и пять линий колонок, по которым передаются пакеты команд, программирующих обмен данными в условиях конвейерной многобанковой организации памяти RDRAM. Пакет (по смыслу не имеющий ничего общего с пакетом SDRAM) занимает во времени 4 такта системной шины и содержит 8 элементов, разграниченных моментами нарастания и спада синхронизирующих импульсов (точнее моментами нулевых значений импульсов, подаваемых в дифференциальной форме). При этом емкость пакета строк составляет 24 бит, пакета колонок – 40 бит. В канале предусмотрено также шестнадцать линий (в случае без контроля паритета) для двухбайтовой передачи данных пакетами емкостью в 16 байт (при плотном потоке данных на частоте системной шины 400 МГц память RDRAM способна обеспечить производительность 1600 Мбайт/с). Наряду с этим, в канале присутствуют две линии синхронизации (одна для стробирования данных, посылаемых микросхемами контроллеру при чтении, другая для синхронизации информации, идущей от контроллера к микросхемам памяти) и четыре вспомогательные линии для управления энергопотреблением и программирования управляющих регистров при инициализации памяти.

Управляющие регистры содержат информацию об адресе микросхемы, параметрах настройки временных циклов и т.п.

Как способ повышения производительности динамической памяти применяется помещение статической кэш-памяти прямо на кристалл памяти. Примером такой архитектуры являются микросхемы CDRAM (Cached DRAM – *кэшированная* DRAM**)**, выпускаемые фирмами Mitsubishi, Samsung с ёмкостью 4 и 16 Мбит, 16 Кбайтным кэшем статической памяти,128-битной внутренней шиной данных. Другой пример – EDRAM (Enhanced DRAM) (фирма Ramtron International) – микросхемы ёмкостью 4 Мбит с 8-Кбайтным кэшем статической памяти и разрядностью внутренней шины данных 2048 бит. Эти виды памяти намного эффективнее комбинации обычной DRAM с вторичным кэшем (особенно в многозадачных системах, где высока частота кэш-промахов при переключении задач).

Микросхемы памяти, как конструктивно законченные изделия, могут упаковываться в корпуса следующих типов:

SOJ (Small Outline J-Lead) **–** малогабаритный пластмассовый корпус с выводами J-образной формы (микросхемы в таких корпусах устанавливаются в специальные «кроватки» на графических адаптерах, а также используются для поверхностного монтажа модулей памяти SIMM и DIMM);

TSOP (Thin Small Outline Package**) –** тонкий пластмассовый корпус для поверхностного монтажа с шагом выводов 0,8 мм;

TSOP-II – корпус, аналогичный TSOP, но с шагом выводов 0,65 мм;

BGA (Ball Grid Array) – корпус с матрицей шариковых выводов на нижней плоскости, применяемый для упаковки микросхем RDRAM.

Формирование основной памяти компьютера путем установки микросхем непосредственно на плату процессора является конструктивно, технологически и экономически накладным. Поэтому микросхемы памяти предварительно организуются в модули, т.е. наборы микросхем, компонуемых на отдельной печатной плате с краевым разъемом *методом поверхностного монтажа* (SMT – Surface Mounting Technology) и обеспечивающих необходимую разрядность ячеек памяти. Использование модулей, подключаемых к плате процессора через контактную колодку из третьего измерения, позволяет экономить площадь основной платы, повышает скорость и надёжность монтажа памяти, совершенствует систему ее энергоснабжения.

Немного об основных типах модулей динамической памяти.

SIPP (Single In-line Pin Package) – модуль памяти с однорядным расположением штырьковых выводов, выпущенный одним из первых в

единственном варианте с 30 выводами и однобайтной организацией. Модули этого типа оказались непрактичными из-за ненадежности контактных выводов.

SIMM (Single In-line Memory Module) – модуль памяти с однорядным расположением печатных выводов в виде площадок, расположенных по обеим поверхностям печатной платы вдоль одной из ее сторон. Вариант SIMM 30-pin, как и модуль SIPP, имеет 30 выводов, однобайтную организацию и компонуется микросхемами памяти FPM, EDO. Вариант SIMM 72-pin – это модуль более современного типа с 72 печатными выводами и 4-байтной организацией с возможностью независимого побайтного обращения. Существуют два вида модулей SIMM 72-pin, отличающихся компоновкой микросхем памяти (FPM, EDO, BEDO) и логической организацией. *Односторонний* (single side) SIMM 72-pin логически организован как один банк памяти, реализуемый микросхемами, смонтированными на одной его поверхности (информационная емкость модуля может составлять 1, 4, 16 или 32 Мбайт). *Двусторонний* (double side) SIMM 72-pin имеет второй комплект микросхем, смонтированный на другой его поверхности и логически организованный как второй банк (информационная емкость модуля может быть 2, 8, 32 или 64 Мбайт). Обращение к строке каждого банка организовано как обращение к двум 16-разрядным словам по сигналам RAS0, RAS2 для первого банка и сигналам RAS1, RAS3 для второго (в односторонних модулях используются только два сигнала RAS0, RAS2). Независимый доступ к байтам слов осуществляется по сигналам CASx.

DIMM (Dual In-line Memory Module) – модуль памяти с двумя рядами независимых печатных выводов, расположенных с обеих сторон печатной платы. Модуль DIMM 168-pin имеет 168 печатных выводов, 8-байтную шину данных и может организовываться в виде двух 4-байтных банков с возможным их чередованием. По архитектуре он близок к SIMM 72-pin, но в связи с удвоенной разрядностью имеет удвоенное число линий CAS, сигналов разрешения записи и управления выходными буферами. DIMM 168-pin может укомплектовываться как микросхемами FPM и EDO (1-ое поколение), так и микросхемами SDRAM (2-ое поколение). Различают Unbuffered DIMM 168-pin (с небуферизованными входными и выходными цепями), Buffered DIMM 168-pin (с буферизованными входными адресными и управляющими сигналами, кроме RAS) и Registered DIMM 168-pin (комплектуемые микросхемами SDRAM и имеющие буферные регистры для адресных и управляющих сигналов, синхронизируемые тактовыми импульсами системной шины).

Информационная емкость модулей DIMM 168-pin может быть от 8 до 512 Мбайт.

DIMM 184-pin имеет 184 печатных вывода, разрядность 8 байт. Состав его сигналов в основном схож с DIMM 168-pin, но их больше, т.к. модуль комплектуется микросхемами DDR SDRAM. Модули данного типа могут иметь информационную емкость 128, 256 Мбайт и выше.

SO DIMM (Small-Outline DIMM) – малогабаритный модуль, комплектуемый микросхемами DRAM в корпусах TSOP. Модули SO DIMM 72-pin емкостью 2-32 Мбайт и четырехбайтовой разрядностью данных организованы с использованием двух двухбайтных слов с возможностью побайтного обращения и организации четырехбайтных применений. Они комплектуются микросхемами FPM, EDO. Модули SO DIMM 144-pin, емкостью 8-64 Мбайт и 8-байтовой разрядностью, обеспечивают возможность организации двух банков (выбираемых по сигналам RAS0 и RAS1) с побайтным обращением к строкам с использованием 8 сигналов CAS. Модули этого типа могут укомплектовываться как микросхемами FPM, EDO, так и микросхемами SDRAM.

DRAM cards 88-pin – миниатюрные модули в пластиковом корпусе, емкостью 2-36 Мбайт, разрядностью 18, 32, 36 бит, комплектующиеся микросхемами DRAM в корпусах TSOP.

RIMM (Rambus Interface Memory Module) – модуль памяти, специально предназначенный для компоновки микросхем RDRAD. Микросхемы (их может быть до 16), упакованные в корпуса BGA, припаяны параллельно (кроме двух выводов) к 30-проводной шине Rambus Channel и закрыты пластиной радиатора. Объем памяти модулей RIMM не обязательно кратен 2, т.к. она может формироваться произвольно из микросхем различной емкости и составлять, например, 64, 96, 128 или 256 Мбайт.

4.4. Статическая память и ее применение для кэширования основной памяти вычислительной системы

Менее емкой, но обладающей более высоким быстродействием по сравнению с динамической памятью (время доступа может составлять несколько наносекунд) является *статическая память* (SRAM). Как правило, она имеет структуру 2D или 2DM. В качестве запоминающих элементов в ней используются триггеры, способные сохранять информацию сколь угодно долго при наличии питания (отсюда и название – статическая) и отличающиеся простотой в управлении. Но вместе с тем эти элементы памяти

схемотехнически более сложны и занимают больше места на кристалле (отсюда и меньшая информационная емкость микросхем статической памяти).

В зависимости от применяемой базовой схемотехнологии построения запоминающих элементов для статической памяти четко просматривается противоречие между такими параметрами как быстродействие и энергопотребление. Как правило, самая быстродействующая статическая память, допускающая работу на частоте системной шины процессора, отличается высоким энергопотреблением, не говоря уж о низком уровне ее упаковки. Поэтому главное ее применение – кэширование основной памяти компьютера (RAM). Выполняя функции сверхоперативного буфера между процессором и RAM, кэш-память позволяет сократить время ожидания процессора при обращении к относительно медленной памяти на микросхемах DRAM, храня в себе копии блоков RAM, к которым происходили последние обращения. В случае последующего обращения к этим блокам информация берется непосредственно из быстрой кэш-памяти.

Как уже отмечалось, в своей многоуровневой организации кэш-память подразделяется на внутреннюю (расположенную непосредственно на кристалле процессора) и внешнюю, интегрированную на своей подложке и монтируемую либо в одном корпусе с процессором, либо отдельно в своем корпусе на материнской плате.

Если говорить о микросхемах SRAM, то именно они используются для реализации внешнего кэша и в зависимости от архитектурных особенностей могут представлять собой следующие разновидности [10].

Async SRAM (Asynchronous SRAM или просто SRAM) – это *асинхронная статическая память* с временем доступа 12, 15, 20 нс. Помимо шин адреса и данных, интерфейсы этих микросхем имеют выводы управления $\overline{CS}$, $\overline{OE}$ и $\overline{WE}$, выполняющие обычные свои роли с некоторыми нюансами. Запись выходными буферами может осуществляться как по высокому уровню сигнала $\overline{OE}$, так и по активному сигналу $\overline{WE}$ (при любом значении $\overline{OE}$).

SB (Synchronous Burst) SRAM – синхронная статическая память, ориентированная на эффективное выполнение пакетных циклов обмена, характерных для кэш-памяти. С этой целью в ее структуре предусмотрен внутренний двухбитный счетчик адреса и расширенный интерфейс, включающий дополнительно сигнал *CLC* для синхронизации с системной шиной, сигналы $\overline{CADS}$ (Cache ADdress Strobe) и $\overline{ADSP}$ (ADdress Status of Processor), которыми процессор или кэш-контроллер стробируют исходный адрес очередного цикла, а также сигнал $\overline{ADV}$ (ADVance) для перехода к

следующему адресу пакета. Память данного типа при задержке данных относительно положительного перепада синхронизирующих импульсов в 8,5; 10; 13,5 нс обеспечивает пакетный цикл 2-1-1-1 на частотах 66, 60, 50 МГц, соответственно.

PB (Pipelined Burst) SRAM – это так называемая *конвейерно-пакетная синхронная статическая память* с усовершенствованным конвейером, использующая дополнительный внутренний регистр данных и способная уменьшить задержку данных относительно перепада синхронизирующего импульса внутри пакета до 4,5 - 8 нс. При этом пакетный цикл 3-1-1-1 обеспечивается на частоте системной шины 133 МГц и выше.

Микросхемы рассмотренных типов статической памяти составляют элементную базу для построения кэш-памяти вычислительных систем самых различных архитектур. При этом в качестве базовых принципов организации кэш-памяти используются следующие.

Основная (кэшируемая) память делится на блоки фиксированного размера, совпадающего с размером *строки кэш-памяти*. Строка кэша (cache line) – это последовательность байтов, хранящая копию блока основной памяти. Ее длина обычно определяется количеством байтов, передаваемых за один пакетный цикл обмена между основной и кэш-памятью (например, для процессоров Pentium 4×8байт = 32 байт). Каждой строке кэша, имеющей свой номер, ставится в соответствие бит ее состояния (действительности, достоверности) и адрес скопированного в нее блока основной памяти. Бит достоверности определяет на текущий момент адекватность содержимого строки состоянию соответствующего блока основной памяти. Адрес блока (*тег*, от англ. **tag**) представляет собой старшую составляющую физического адреса блока основной памяти. Разрядность тега определяется размером адресного пространства кэшируемой памяти, числом байтов в строке и общим количеством строк кэш-памяти. Надо заметить, что копирование строки иногда может осуществляться не целиком, а *секторами* (*секторированный кэш*), т.е. минимальными порциями обмена между кэшем и основной памятью. При этом, бит достоверности предусматривается для каждого сектора строки.

Таким образом, кэш-память хранит ограниченное общим числом ее строк количество блоков оперативной памяти и *каталог* (cache directory), определяющий соответствие строк конкретным блокам посредством указания тегов и битов достоверности.

Следует обратить внимание на то, что в силу ряда архитектурных ограничений объем кэшированной основной памяти может быть ограничен или из него могут быть исключены некоторые конкретные области.

При каждом обращении к кэшируемой памяти по каталогу проверяется наличие действительных ее копий в кэш-памяти. В положительном случае фиксируется кэш-попадание (cache hit) и необходимые данные (при чтении) берутся непосредственно из кэш-памяти. При отсутствии копии отмечается кэш-промах (cache miss), и данные берутся из основной памяти, а блок, которому они принадлежат, в соответствии с алгоритмом кэширования замещает один из блоков кэш-памяти (например, это может быть один из наиболее редко используемых блоков). Здесь уместно обратить внимание на два важных фактора, оказавших существенное влияние на организацию и базовые принципы функционирования кэш-памяти. Первый – это фактор *пространственной локализации* востребуемых адресов памяти, выражающий большую вероятность последовательного использования соседних адресов, чем далеко отстоящих друг от друга. Именно это обусловило определенную избыточность переносимой в кэш информации (не конкретных, необходимых на данный момент данных, а целых блоков, в пределах которых они находятся). Вторым является фактор *временной локализации*, отражающий закономерность повторного использования недавно задействованных адресов (например, команд внутри цикла или данных, расположенных вблизи вершины стека). Именно поэтому целесообразно заменять в кэш-памяти давно не используемые строки.

Кэш-промах при записи данных просто сопровождается помещением их (возможно через буфер отложенной записи) в основную память. Кэш-попадание при записи предполагает использование одной из двух основных стратегий.

Стратегия *сквозной записи* (WT – Write Through) предусматривает запись данных и в строку кэша, и в основную память. При этом, легко обеспечивается целостность данных и отпадает необходимость использования признаков достоверности данных. Но, вместе с тем, снижается скорость записи из-за необходимости обращения к основной памяти. Однако, в какой-то степени этот недостаток можно компенсировать, используя буфер отложенной записи (FIFO-буфер, о котором речь пойдет позже).

Стратегия *обратной записи* (WB – Write Back) в случае кэш-попадания предполагает запись данных в *действительную* строку кэша и объявление ее *недействительной* до выгрузки ее содержимого в основную память. Такая

выгрузка может выполняться в любое свободное время после модификации строки, если в ней нет острой необходимости, связанной с сохранением целостности данных.

Микросхемы статической памяти, из которых набирается необходимый объем кэша, организуются в один или несколько *банков*. Каждый банк заполняется микросхемами одинаковой информационной емкости, с одинаковыми техническими характеристиками, согласованными с разрядностью и частотой системной шины.

Каталог кэша реализуется одной или двумя словарными микросхемами асинхронной статической памяти для хранения тегов (Tag SRAM) и, возможно, отдельной микросхемой для хранения признаков действительности (чистоты) строк (Dirty SRAM).

С точки зрения организации взаимного соответствия между строкой кэш-памяти и блоком основной памяти представляют интерес три известные архитектуры: *кэш прямого отображения* (direct-mapped cache), *наборно-ассоциативный кэш* (set-associative cache) и *полностью ассоциативный кэш* (fully associative cache).

Архитектура *кэша прямого отображения* для идентификации слова (или байта) в кэш-памяти использует представление адреса основной памяти в виде четырех компонент (рис. 4.5).

Рис. 4.5. Представление адреса для кэш-памяти прямого отображения

Поле старших разрядов ТЕГ определяет адрес блока основной памяти, которому принадлежит адресуемый элемент. Поле средних разрядов СТРОКА идентифицирует номер строки кэша, в которой находится элемент. Младшие разряды СЛОВО, БАЙТ определяют, соответственно, место элемента памяти (слова, байта) в строке.

Очевидно, конкретный адресуемый элемент может храниться только в одном месте кэш-памяти прямого отображения, определяемом средними и младшими разрядами адреса основной памяти, т.е. разрядами, поступающими на адресную шину кэш-памяти. При этом, на одно и то же место в кэше могут претендовать элементы данных, имеющие одинаковые средние и младшие составляющие адреса, но принадлежащие различным блокам основной памяти,

т.е. отличающиеся полем ТЕГ. Информация о том, какой блок в текущий момент занимает данную строку кэша и является ли эта строка действительной, содержится в каталоге кэш-памяти. Поиск нужного элемента начинается с выделения битов СТРОКА, по которым определяется единственная строка кэш-памяти, которой может принадлежать адресуемый элемент. Если найденная строка, согласно каталогу, является действительной и имеет идентичный старшим разрядам адреса тег, то имеет место кэш-попадание.

Надо заметить, что для сокращения времени доступа к кэш-памяти прямого отображения идентификацию строки и выборку слова (байта) целесообразно осуществлять параллельно с проверкой тега и установления действительности строки. С точки зрения аппаратных затрат данная архитектура является наиболее экономичной.

Стремление уладить конфликт между блоками, претендующими на размещение в определенной строке кэш-памяти прямого отображения, привело к появлению архитектуры *наборно-ассоциативного кэша.* Эта архитектура допускает возможность помещения блока кэшируемой памяти в одну из строк *набора*, идентифицируемого средней частью адреса (НАБОР), представленного на рис. 4.6. Для каждой из строк набора в каталоге кэш-памяти ставится в соответствие свой тег и бит действительности. При количестве n строк в наборе память называется n-входовой. По сути, эту архитектуру можно рассматривать как совокупность параллельно организованных банков прямого отображения, в которой принцип ассоциативного поиска применяется в пределах адресуемого набора строк. Она успешно применяется для реализации внутреннего кэша современных процессоров.

Рис. 4.6. Представление адреса для кэш-памяти наборно-ассоциативного типа

Полную свободу задействования любой строки для кэширования блока основной памяти предоставляет архитектура *полностью ассоциативного кэша*. Она использует представление адреса, приведенное на рис. 4.7.

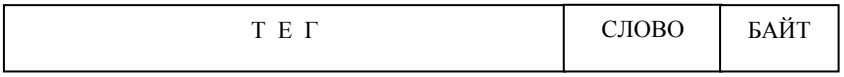


Рис. 4.7. Представление адреса в случае полностью ассоциативной кэш-памяти

Каждой строке кэш-памяти ее каталог ставит в соответствие тег – полный физический адрес кэшируемого блока основной памяти. Доступ к строке осуществляется путем сравнения тега адреса с тегом строки кэша, которое выполняется параллельно для всех строк в соответствии с принципами организации ассоциативной памяти.

Являясь наиболее гибким, полностью ассоциативный кэш требует значительных аппаратных затрат на его реализацию. Поэтому он применяется в структурах кэш-памяти 1-го уровня, не претендуя на использование в более объемных кэшах второго и более высоких уровней.

4.5. Энергонезависимая память

В отличие от динамической и статической полупроводниковой памяти, *энергонезависимая память* хранит записанную информацию при отсутствии питающего напряжения от сети, а *постоянная память*, как разновидность энергонезависимой памяти, не использует даже автономные источники питания. Основным режимом работы такой памяти является считывание данных. В зависимости от способов построения и программирования запоминающих ячеек, а также сфер применения различают следующие типы постоянной памяти.

ROM (Read Only Memory – *память только для чтения*) – постоянная память, программируемая при изготовлении и в дальнейшем допускающая только считывание. Это так называемые *масочные постоянные запоминающие устройства* **(***масочные ПЗУ***)**, программируемые при изготовлении путем «прошивки» их проводниками считывания и характеризуемые временем доступа 30-70 нс. Из-за невозможности модификации содержимого этот вид памяти не нашел широкого применения и использовался в основном для реализации знакогенераторов в некоторых моделях графических адаптеров.

PROM (Programmable ROM) – постоянная память, программируемая однократно после изготовления перед установкой в целевое устройство на специальном программаторе (прожигаемые ПЗУ). Обладая аналогичными

параметрами и возможностью программирования изготовителем целевого оборудования, эта память получила более широкое применение в графических адаптерах, а также для хранения не перезаписываемых кодов BIOS – Basic Input/Output System – *базовой системы ввода-вывода***,** выполняющей функции:

изоляции операционной системы и прикладных программ от специфических особенностей конкретной аппаратуры;

программной поддержки стандартных ресурсов ПК и конфигурирования аппаратных средств;

диагностики ПК с применением POST (Power On Self Test) и вызова загрузчика операционной системы.

EPROM (Erasable PROM) – стираемая и программируемая многократно память с временем доступа в диапазоне 50-250 нс, представленная микросхемами:

UV EPROM (Ultra-Violet EPROM), стираемые ультрафиолетовым облучением через специальное окно в корпусе микросхемы;

EEPROM (Electrical EPROM) – электрически стираемые микросхемы, широко применяемые в качестве носителя BIOS в современных персональных компьютерах.

В качестве запоминающих элементов матрицы накопителя микросхем EPROM используются МНОП-*транзисторы* или *транзисторы с плавающим затвором* [6].

МНОП-*транзистор* (металл-нитрид-окисел-полупроводник) имеет под затвором двухслойный диэлектрик (нитрид кремния – окисел кремния). На границе слоев возникают центры захвата заряда, в которых под действием электрического поля достаточно высокой напряженности благодаря туннельному эффекту создается заряд, влияющий на пороговое напряжение транзистора. Наличие высокого порога не позволяет транзистору открыться в рабочем режиме. Транзистор, в котором заряд отсутствует или имеет противоположный знак, легко открывается рабочим напряжением на затворе. Одно из состояний трактуется как логический «0», другое – как логическая «1».

Транзистор с плавающим затвором имеет в диэлектрике замкнутую проводящую область, называемую плавающим затвором, в которую может быть введен электрический заряд. По принципу работы он похож на МНОП-транзистор, но есть отличия в механизмах его программирования. Как примеры транзисторов с плавающим затвором можно рассматривать версии ЛИЗМОП (транзистор с лавинной инжекцией заряда) и FLOTOX (Floating Gate Tunnel Oxide - плавающий затвор с туннелированием в окисле).

Флэш-память (Flash-Memory) по основным принципам работы аналогична EEPROM. В современных материнских платах используются микросхемы Flash BIOS, программы в которых могут перезаписываться при помощи специальных средств, обеспечивающих возможность модернизации этих программ при появлении новых устройств. Запись и стирание информации производится электрическими сигналами, но не на уровне отдельных слов. Это делается либо для всей памяти одновременно, либо для достаточно больших ее блоков.

Полупостоянная память – это статическая память с малым энергопотреблением, питаемая при выключенном компьютере от батарейки. Самой экономичной является КМОП-*память* (CMOS Memory) со временем доступа более 100 нс, но очень малым энергопотреблением, что оправдывает ее применение для хранения информации о конфигурации компьютера и обеспечения часов-календаря (RTC – Real Time Clock). При загрузке компьютера BIOS берет необходимую для своей работы информацию об изменяемых параметрах устройств компьютера именно из этой памяти.

5. АРХИТЕКТУРА МИКРОПРОЦЕССОРОВ

5.1. Базовые принципы организации микропроцессора

Любой процессор с точки зрения канонической схемы, отражающей базовые принципы его организации, объединяет в себе два устройства: *операционное* (*операционный автомат*) и *управляющее* (*управляющий автомат*) (рис. 5.1).

Рис. 5.1. Каноническая схема процессора

Операционный автомат – это устройство, выполняющее непосредственно операции над информацией под воздействием управляющих сигналов и вырабатывающее определенные признаки результатов этих операций. Он включает в себя в качестве составляющих узлов регистры, сумматоры, дешифраторы, мультиплексоры, АЛУ и другие функциональные блоки, реализующие элементарные действия, из которых складывается процесс выполнения команд. Такими действиями могут быть запись в регистр, инвертирование содержимого регистра, сдвиг кода влево или вправо, дешифрация двоичного кода, сложение двух чисел и другие. Каждое элементарное действие, выполняемое в одном из узлов операционного автомата в течение одного тактового периода, называется *микрооперацией*.

Управляющий автомат представляет собой устройство, координирующее действия узлов операционного автомата посредством управляющих сигналов. Эти сигналы вырабатываются управляющим автоматом с учетом признаков результатов выполнения текущих операций, поступающих с выхода операционного автомата. Их состав зависит от способности операционного автомата выполнять параллельно (в один такт) несколько микроопераций. Совокупность одновременно выполняемых микроопераций называют *микрокомандой*. Последовательность микрокоманд, интерпретирующая сложную команду (выполняемую не за один такт), образует *микропрограмму*. Таким образом, управляющий автомат определяет микропрограмму, регламентирующую последовательность наборов управляющих сигналов (микрокоманд), активизирующих операционный автомат на выполнение необходимых действий, связанных с выполнением очередной команды.

Различают два основных способа реализации микропрограммного управления. Один базируется на *жесткой (схемной) логике*, другой использует *программируемую логику*.

В случае *жесткой логики* микропрограммы интерпретации команд реализуются аппаратно соответствующими логическими схемами управляющего автомата. При этом можно достичь максимального быстродействия процессора, но столкнуться с проблемой экономической целесообразности разработки такого проекта из-за недостаточной его универсальности, а значит малой тиражируемости.

Принцип *программируемой логики* (рис. 5.2) предполагает хранение кодовых комбинаций микропрограмм в специальной *управляющей памяти* и последовательное извлечение их для выдачи такт за тактом управляющих сигналов в операционный автомат при выполнении команды. Каждой команде соответствует своя микропрограмма в управляющей памяти. При выборке очередной команды из оперативной памяти по ее коду находится соответствующая микропрограмма в управляющей памяти, которая выполняется путем последовательного считывания кодовых комбинаций микрокоманд и подачей сигналов управления на операционный автомат.

Общий вид формата микрокоманды представлен на рис. 5.3. Первое поле (слева направо) формата содержит адрес следующей микрокоманды в случае безусловного перехода (адрес первой микрокоманды микропрограммы в управляющей памяти находится по значению кода выполняемой операции). Второе поле отводится под биты, определяющие условия перехода к следующей микрокоманде в зависимости от признаков результатов,

вырабатываемых операционным автоматом. В третьем поле размещаются управляющие сигналы, активизирующие операционный автомат на выполнение микрокоманды.

Использование принципа программируемой логики обеспечивает возможность переориентации процессора на новый состав команд и более совершенные алгоритмы их интерпретации путем замены содержимого управляющей памяти. Это придает универсальность процессору на уровне проектного решения и, возможно, на уровне готового изделия (в зависимости от его конструктивно-технологического исполнения).

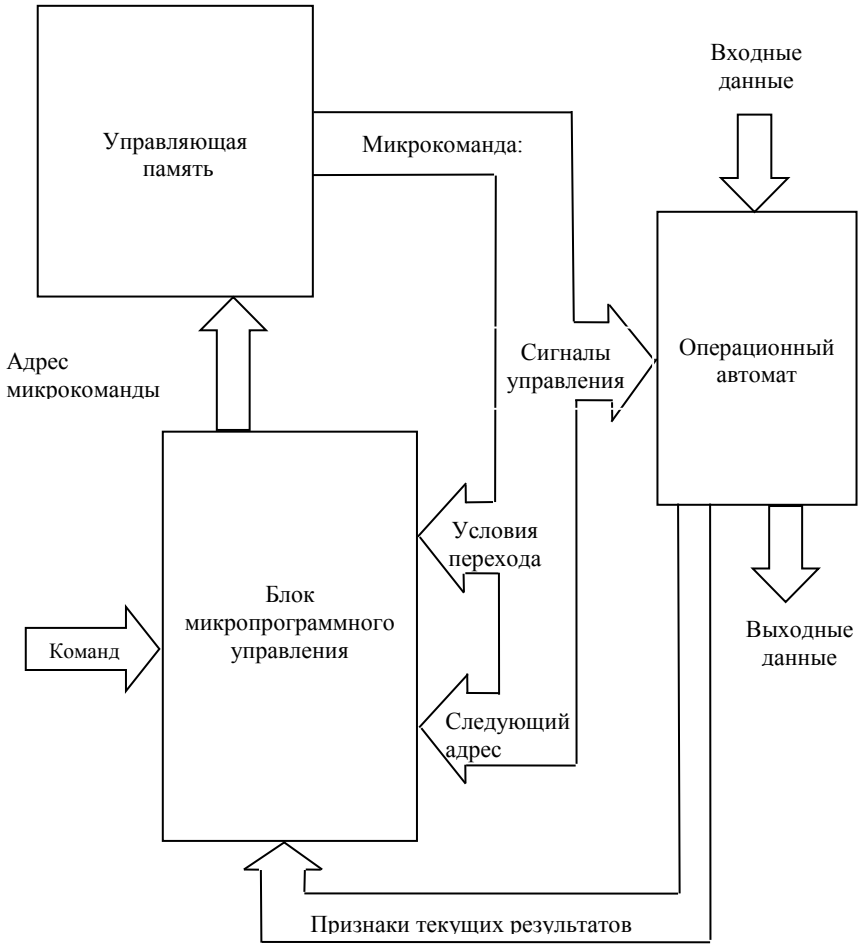


Рис. 5.2. Организация процессора с программируемой логикой

Следующий адрес	Условия перехода	Сигналы управления

Рис. 5.3. Формат микрокоманды

Универсальные микропроцессоры, используемые в современных вычислительных системах, отличаются наличием эффективных средств для выполнения сложных расчетов. Они обладают широким спектром команд, некоторые из которых (так называемые *внутренние команды*) выполняются операционным автоматом за один такт (цикл) его работы. Другие (*сложные*) команды интерпретируются микропрограммами, хранящимися в энергонезависимой управляющей памяти процессора.

При проектировании микропроцессора к разряду внутренних относятся команды, удовлетворяющие приемлемому компромиссу между требуемой скоростью их выполнения и сложностью аппаратной реализации. Такой компромисс легко обеспечивается для простых команд, не требующих включения сложных логических схем в состав операционного автомата. Но по мере усложнения команд альтернативный путь интерпретации может играть решающую роль. Интерпретатор разбивает команды на элементарные шаги (микрокоманды, микрооперации), аппаратная реализация которых обходится гораздо дешевле. При этом основная функциональная нагрузка переносится с аппаратного обеспечения на программное. Это удешевляет процессор, но снижает его производительность.

Интерпретация, как технология развития архитектуры вычислительных систем, используемая с начала 50-х годов прошлого века, позволяет:

- расширять систему команд процессора без изменения (усложнения) его аппаратной части;
- отрабатывать новые сложные команды, восполняющие недостатки аппаратных средств, экспериментируя с интерпретаторами;
- следовать принципу преемственности в плане обеспечения возможности использования без изменений старого программного обеспечения;
- накапливая статистику, выявлять сложные команды, целесообразность перевода которых в разряд внутренних (аппаратная реализация) не вызывает сомнений.

Эти положительные моменты обусловили развитие технологии интерпретации в архитектурах компьютеров DEC VAX, Intel, IBM, получивших название CISC (Complex Instruction Set Computer – компьютер с полным набором команд). Это название появилось не сразу, а только после возникновения концепции RISC, о которой речь пойдет далее.

Большое число сложных (интерпретируемых) команд различного формата (общее число команд может достигать нескольких сотен) обеспечивает на сравнительно дешевом оборудовании необходимую функциональность

вычислительной системы, но никак не способствует повышению ее производительности. Вместе с тем, анализ кода программ, генерируемых компиляторами архитектуры CISC, позволил выделить относительно небольшое количество (порядка нескольких десятков) практически используемых простых команд, которые способны обеспечить требуемую функциональность и могут быть реализованы как внутренние, т.е. выполняемые за один такт операционным автоматом и не требующие интерпретации. Это явилось отправной точкой возникновения (примерно с начала 80-х годов прошлого века) и развития концепции (архитектуры) RISC (Reduced Instruction Set Computer - компьютер с сокращенным набором команд), основные принципы которой в идеале можно сформулировать следующим образом:

- все команды должны выполняться аппаратно операционным автоматом (должны быть внутренними);

- как можно большее число команд должно поступать в вычислительный тракт процессора в единицу времени, т.е. выполняться параллельно;

- в целях быстрого декодирования команды должны иметь однотипный формат с небольшим числом полей фиксированной длины;

- операции непосредственных вычислений над данными должны выполняться в формате команд типа «регистр, регистр → регистр» и должны быть отделены от функций обмена информацией между регистрами и оперативной памятью (команды «регистр ↔ память»);

- для эффективного функционирования системы в ней должно быть предусмотрено достаточное количество регистров.

Основная идея RISC-архитектуры – обеспечить высокую производительность вычислительной системы за счет выполнения пусть большего количества, но аппаратно реализуемых и параллельно обслуживаемых высокоскоростным конвейером (конвейерами) команд. Эта идея, к примеру, лежит в основе архитектуры SPARC (Scalable Processor ARChitecture – наращиваемая архитектура процессора), развиваемой фирмой Sun Microsystems в последовательности версий процессоров UltraSPARC I - UltraSPARC IV. В качестве RISC-ядра эта технология применена и в микропроцессорах Intel шестого поколения (P6). Для этого потребовалось дополнительное аппаратное обеспечение сведения традиционных команд архитектуры IA-32 (Intel Architecture 32 bit) к командам RISC-процессора. Кроме того, пришлось решать проблемы зависимости по данным, остро обозначившиеся из-за недостаточного количества регистров общего назначения и многое другое. Все эти недостатки были учтены при разработке совместными

усилиями компаний Intel и Hewlett Packard архитектуры IA-64, на особенностях которой мы остановимся позже.

5.2. Архитектура уровня команд процессора

Система команд процессора – это уровень его архитектуры, связывающий аппаратные средства компьютера с его программным обеспечением. Удачная система команд с одной стороны создает благоприятные условия для рациональной организации аппаратных средств, обеспечивающей необходимое их быстродействие и перспективы развития с учетом особенностей существующих и будущих технологий. С другой стороны, отличаясь регулярностью и полнотой, она облегчает построение эффективных компиляторов, переводящих программы с языков высокого уровня на язык машинных команд для последующей их интерпретации.

Состав команд, их содержание и форматы тесно взаимосвязаны с реализуемой моделью памяти, используемыми типами данных, способами их адресации, набором имеющихся в распоряжении регистров.

Модель памяти прежде всего определяет ее *структурируемость*, т.е. возможность обращения к байтам, словам (32-разрядным или 64-разрядным). При этом может быть задействован принцип выравнивания адресов слов в естественных границах, начиная с 0. Например, в процессоре Pentium IV обмен с памятью осуществляется 64-битными словами (по одному каналу) с выровненными адресами. Для этого из 41-разрядной шины адреса используются 33 адресных бита, обеспечивающих обращение к 8 Гбайтам основной памяти (остальные 8 линий идентифицируют байты). Выравнивание адресов существенно экономит аппаратные средства и повышает производительность вычислительной системы. Однако принцип обратной совместимости требует обеспечения возможности обращения к словам, начинающимся с произвольного адреса (как это было в процессорах ранних поколений). Такая возможность реализуется на базе новой модели ценой дополнительных аппаратных затрат.

Другой характеристикой модели памяти является возможность дублирования адресных пространств с использованием доступного для пользователя *механизма сегментации*. Отдельные линейные адресные пространства могут открываться для программ, различных типов данных, стековых и других структур. При этом для каждого сегмента могут быть использованы свои способы защиты хранимой информации, что повышает надежность функционирования вычислительной системы.

Помимо прочего модель памяти должна предусматривать определенные способы предотвращения конфликтов между операциями обмена (чтения и записи), возникающих в результате переупорядочения микрокоманд в ходе конвейерного, суперскалярного выполнения инструкций программы.

Состав команд процессора во многом зависит от типов данных, которыми должен оперировать процессор, и возможности реализации их аппаратной поддержки. Различают *числовые* и *нечисловые* типы данных. Первые включают целые числа различной разрядности (8, 16, 32 или 64 бита), целые числа со знаком (один бит у них отведен под знак), числа с плавающей точкой (для их представления могут использоваться 32, 64 или 128 битов), а также целые числа двоично-десятичного формата (каждый десятичный разряд числа кодируется четырьмя битами – тетрадой). Ко вторым относятся символы (для их представления могут использоваться символьные коды ASCII или UNICODE), цепочки символов, значения булевой алгебры (истина и ложь), а также указатели, представляющие машинные адреса.

Конкретные данные указанных типов, выступая в роли операндов той или иной операции, должны извлекаться из мест их хранения (регистров, ячеек основной или кэш-памяти) или помещаться туда, если под операндом понимается результат выполнения операции (выходной операнд). Для адресации операндов (определения мест их нахождения) в спецификации команды предусматриваются так называемые поля операндов. В зависимости от числа таких полей, различают трехадресные (три поля операндов), двухадресные (два поля операндов), одноадресные (одно поле операнда) и безадресные (поля операндов отсутствуют) команды.

Очевидно, чем больше полей операндов в команде, тем она длиннее. Причем эта зависимость особенно ощутима при работе с данными, хранящимися в основной памяти, и не так существенна при адресации регистров (их число невелико, поэтому адрес регистра короткий). Однако предпочтение в пользу регистров в этом плане оправдывается только при многократном использовании хранимых в них операндов. Для выполнения этого условия должно быть предусмотрено достаточное количество регистров. Так в архитектуре процессора UltraSPARC III предпочтение отдано трехадресным командам, выполняемым с использованием 200 регистров общего назначения, образующих восемь 32-регистровых групп (регистровых окон) с 8-регистровым перекрытием соседних окон (регистры 24-31 предыдущего окна являются одновременно регистрами 0–7 последующего окна). В любой момент программа имеет доступ только к одному (видимому)

окну. Смена окон организуется по принципу стека с использованием специального регистра CWP (Current Window Pointer – указатель текущего окна). Перекрытие окон обеспечивает эффективный способ передачи параметров между вызывающей и вызываемой процедурами.

Другой путь сокращения длины команды – определение одного или нескольких операндов неявным образом, т.е. использование двухадресных, одноадресных и безадресных команд. Примером использования двухадресных команд является Pentium IV со всеми предыдущими версиями архитектуры IA-32 (Intel Architecture-32). Как вариант с безадресной архитектурой команд можно рассматривать процессор picoJava II (архитектура JVM - Java Virtual Machine) со стековой организацией операндов.

Биты адресных полей команды по-разному могут быть использованы для адресации операндов. Из известных способов адресации базовыми (принципиально отличающимися друг от друга и применяемыми в различных сочетаниях) являются:

- непосредственная адресация;
- прямая адресация;
- регистровая адресация;
- косвенная регистровая адресация;
- индексная адресация;
- относительная индексная адресация;
- относительная адресация;
- стековая адресация.

Непосредственная адресация предусматривает размещение в адресном поле самого операнда, который автоматически становится доступным сразу после считывания команды из памяти. Однако так можно оперировать только константами, причем значения их ограничиваются размером поля. Данный способ адресации принято использовать для целочисленных констант.

Прямая адресация предполагает задание в поле операнда его полного адреса. Для этого требуется значительное количество битов. Кроме того, адрес должен быть строго фиксирован. Поэтому данный способ применим только к глобальным переменным программы, адреса которых известны компилятору.

При *регистровой адресации* поле операнда интерпретируется как адрес регистра, в котором расположен операнд. Быстрота доступа к регистрам и короткие поля операндов обусловили широкое применение этого способа адресации, особенно в процессорах с RISC-архитектурой.

В случае *косвенной регистровой адресации* доступ к операнду осуществляется по адресу, содержащемуся в регистре, на который указывает поле операнда. Такой способ адресации позволяет динамически управлять командой, т.е. использовать в ней при разных выполнениях разные слова памяти.

Индексная адресация предполагает формирование адреса как результата сложения содержимого определенного регистра с константой смещения, заданной в поле операнда.

Относительная индексная адресация предусматривает вычисление адреса путем суммирования содержимого двух регистров и смещения, которое не обязательно. Один регистр называется *базовым*, другой – *индексным*. Очевидно, использование только этих двух регистров (без смещения) для формирования адреса позволяет существенно сократить длину команды.

Относительная адресация широко применяется в командах управления (условные и безусловные переходы, обращения к подпрограммам, управление циклами) для адресации ячейки следующей выполняемой команды. Этот адрес получается в результате сложения содержимого регистра-указателя команд (регистр EIP – Extended Instruction Pointer в архитектуре IA-32), хранящего адрес текущей команды, с заданным в поле операнда смещением, определяющим положение следующей команды. Надо заметить, что, наряду с относительной адресацией, в командах управления могут применяться и другие способы адресации, имеющие смысл в конкретных ситуациях.

Стековая адресация не предполагает обращений к памяти, видимых на уровне команд. Она использует неявные адреса указателей основания и вершины текущего стека, хранящиеся в соответствующих регистрах.

Помимо рассмотренных способов адресации, существует множество их комбинаций и модификаций, среди которых относительная индексная адресация без смещения, индексная адресация с масштабированием, относительная индексная адресация с масштабированием и многие другие способы, количество которых может исчисляться сотнями.

Особенности уровня архитектуры команд процессора во многом определяются составом его регистров, доступных программисту. Эти регистры используются для хранения промежуточных результатов, организации доступа к кодам программы и данных, контроля и управления вычислительным процессом. В процессорах архитектуры IA-32 (включая Pentium IV) они подразделяются на регистры общего назначения, регистры сегментов, указатель инструкций, регистр флагов.

Восемь 32-разрядных *регистров общего назначения* (EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP) предназначены для хранения данных, адресов, указателей. Регистры EAX, EBX, ECX и EDX (E – Extended – расширенный) допускают отдельное обращение к их младшей 16-разрядной половине по именам AX, BX, CX и DX, соответственно, а также к составляющим ее байтам AL (L – Low, младший байт) и AH (H – High, старший байт), BL и BH, CL и CH, DL и DH. Регистры ESI (Extended Source Index), EDI (Extended Destination Index), EBP (Extended Base Pointer) и ESP (Extended Stack Pointer) позволяют отдельно обращаться к их младшим половинам, обозначаемым, соответственно, SI, DI, BP и SP.

Шесть *регистров сегментов* CS (Code Segment), SS (Stack Segment), DS (Data Segment), ES, FS, GS (дополнительные сегменты) используются для размещения 16-разрядных указателей на базовые адреса сегментов (в реальном режиме) или 16-разрядных селекторов дескрипторов соответствующих сегментов (в защищенном режиме). Сегментные регистры могут использоваться по умолчанию, согласно их названиям, для указания на сегменты соответствующих типов информации (кодов команд, стека, данных). Возможно также их альтернативное использование, определяемое префиксами команд CS:, SS:, DS:, ES:, FS: и GS:.

EIP (Extended Instruction Pointer) - это 32-битный *указатель инструкций*, содержащий смещение следующей исполняемой команды относительно базового адреса сегмента кодов. Предусмотрена возможность отдельного обращения к младшей половине регистра EIP по имени IP при работе в режиме 16-разрядной адресации.

Регистр флагов (IFLAGS) в своих 32 битах содержит информацию о состоянии процесса выполнения программы. Не все биты данного регистра доступны пользователю. Многие из них используются только в привилегированном режиме для управления обработкой исключений и маскируемых прерываний, последовательностью вызываемых задач, отладочными процессами и другими системными процедурами. К битам, доступным программисту, относятся коды условий, устанавливаемые в каждом цикле операционного автомата и отражающие состояние результата очередной выполненной операции. Это следующие биты:

OF (Overflow Flag) – *флаг переполнения*, принимающий значение «1», если результат арифметической операции выходит за рамки разрядной сетки операнда назначения;

SF (Sign Flag) – *флаг знака*, устанавливающийся в состояние «1» при единичном значении старшего бита результата (признаке отрицательного числа);

ZF (Zero Flag) – *флаг нуля*, принимающий значение «1», если все разряды результата нулевые;

AF (Auxiliary Flag) – *флаг дополнительного переноса* (*заема*) в тетраде для десятичной арифметики;

PF (Parity Flag) – *флаг паритета*, устанавливающийся в состояние «1» при четном числе единиц в коде результата;

CF (Carry Flag) – *флаг переноса* (*заема*) старшего бита в арифметических операциях.

Помимо указанных регистров, в архитектуре IA-32 предусмотрено множество регистров, обеспечивающих вычисления с плавающей запятой, а также большое количество системных (доступных в привилегированном режиме) регистров, используемых для адресации и контроля различных уровней памяти, устройств ввода-вывода и поддержки ряда других функций, реализуемых аппаратными и системными программными средствами процессора.

Прежде, чем перейти к рассмотрению форматов команд наиболее распространенных архитектур процессоров, остановимся на основных типах команд, характерных для большинства компьютеров.

Команды перемещения данных, осуществляя копирование их из одного места в другое, могут преследовать следующие цели:

- дублировать данные под новыми именами;
- обеспечить возможность быстрого доступа к данным;
- поддержать стековую организацию вычислительного процесса.

В зависимости от видов источника и пункта назначения данных возможны следующие категории команд: регистр - регистр, регистр - память, регистр - стек, стек - регистр, память - регистр, стек - память, память - стек, память - память (команды, связанные со стеком, выделены в отдельные категории в предположении не программной, а аппаратной реализации стека). Перемещаемые командой данные могут быть разной длины – от одного бита до всей памяти. Чаще всего передаются байт, несколько байтов, слово или несколько слов.

Команды, реализующие *бинарные операции*, выполняют действия над двумя операндами и получают результат. К ним относятся арифметические команды сложения, вычитания, умножения и деления целых чисел и чисел с

плавающей запятой (включая числа с удвоенной точностью), а также логические команды И, ИЛИ, И-НЕ, ИЛИ-НЕ и т.д.

Команды, реализующие *унарные операции*, производят результат действия над одним операндом. В их числе различные варианты команд сдвига и циклического сдвига, команда установки нуля (очистки), команда инкрементирования (увеличения значения операнда на единицу), команда аддитивной инверсии числа (получения его значения с противоположным знаком), команда логического отрицания (инверсии двоичного кода).

Команды сравнения и переходов в комплексе предназначены для изменения последовательности выполнения команд программы. Они подразделяются на *команды безусловного перехода*, *команды условного перехода* и *команды сравнения*.

Команды безусловного перехода определяют адрес следующей выполняемой команды вне зависимости от каких-либо условий. Конкретный вид команды связан с используемым в ней способом адресации команды, к которой осуществляется переход.

Команды условного перехода определяют адрес следующей команды, проверяя выполнение (или невыполнение) некоторого условия. Условие может быть связано с состоянием определенного бита (битов) регистра флагов, значением знакового бита числа, содержанием полей специальных регистров и других источников информации, используемых для формирования и проверки условий выполнения переходов. Различные варианты команд условного перехода отличаются способами формирования условий и адресации команд, к которым осуществляется переход.

Информация, используемая в условии перехода, может представлять собой признак результата обычной арифметической операции над целыми числами или числами с плавающей запятой, предшествующей команде условного перехода. Она может также вырабатываться как результат выполнения одной из *команд сравнения* двух операндов. В случае трехадресной машины операции сравнения и условного перехода могут быть объединены в одной команде. При двухадресной архитектуре команд команда сравнения, вырабатывающая результат сравнения, должна предварять команду условного перехода.

Команды обращения к процедурам реализуют передачу управления первой выполняемой команде вызываемой процедуры (подпрограммы) и сохраняют адрес возврата в вызывающую программу после завершения выполнения подпрограммы. Механизмы помещения адреса возврата в фиксированную ячейку памяти или в определенное место подпрограммы (таким местом может

быть первое слово процедуры, к которому осуществляется безусловный переход после ее выполнения; при этом выполнение подпрограммы начинается со второго слова) не срабатывают в случае рекурсии, т.е. вызова процедурой самой себя. Самым эффективным решением с этой точки зрения является размещение адреса возврата в стеке. Использование этой динамической структуры позволяет эффективно обрабатывать рекурсии практически любой глубины вложенности.

Команды управления циклом организуют выполнение выделенной группы команд заданное количество раз. При этом используется счетчик, увеличивающий или уменьшающий свое значение на определенную константу при каждом проходе цикла. Значения счетчика контролируются с целью отслеживания момента выхода из цикла в соответствии с заданным числом его выполнения. Конкретные варианты команд данного типа отличаются местом проверки окончания цикла (в его начале или конце), а также условиями повторения цикла.

В отдельные типы следует выделить *команды ввода-вывода, команды прерываний* и ряд команд системного уровня, связанных с управлением процессором.

В содержательном аспекте (без конкретизации расположения и длины составляющих кодов) обобщенный формат команды можно представить в виде совокупности полей, изображенной на рис. 5.4.

Рис. 5.4. Обобщенный формат команды

Единственным обязательным полем в представленном формате является *код операции*, идентифицирующий действия, выполняемые командой. Код операции может расширяться за счет полей адресов (например, при введении дополнительных двухадресных команд за счет поля адреса 3) и оказываться рассредоточенным по разным полям формата.

Префикс – это необязательный код, предназначенный для изменения действия команды.

Поля адресов (адрес 1, адрес 2, адрес 3) также являются необязательными, но те, которые присутствуют в команде, содержат информацию, определяющую используемый способ адресации и компоненты, необходимые

для вычисления полного адреса. Как исключение, при непосредственной адресации поле адреса содержит сам операнд.

Команды одних архитектур могут иметь одинаковую длину. Это упрощает их декодирование, но может потребовать больших затрат памяти из-за необходимости выравнивания всех команд по самой длинной из них, а также по адресам памяти. Примером такой архитектуры являются процессоры UltraSparc, оперирующие набором простых 32-битных команд, выровненных в памяти. Типичные базовые форматы команд приведены на рис. 5.5 (над форматами целые числа указывают размеры соответствующих полей в битах).

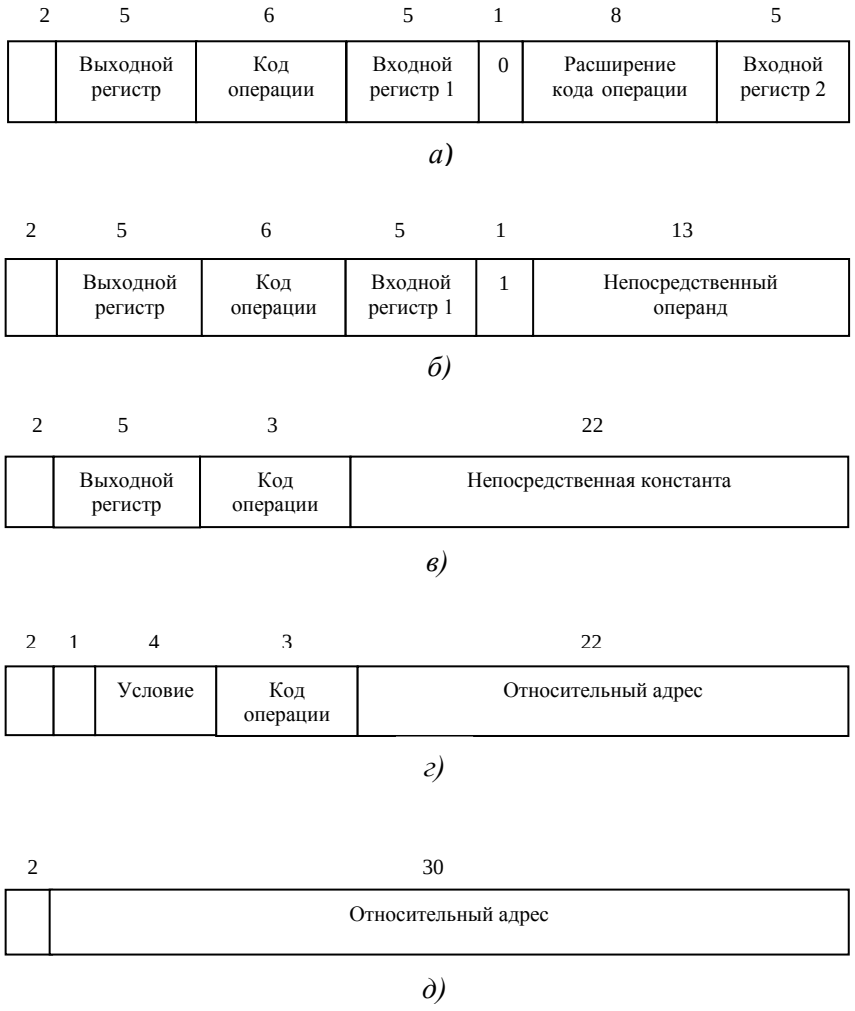


Рис. 5.5. Типичные базовые форматы команд процессоров UltraSparc

Формат *а)* представляет типичную бинарную операцию над операндами, находящимися в двух регистрах (входной регистр 1 и входной регистр 2), с помещением результата в третий регистр (выходной регистр). Под адреса регистров отведено по 5 бит в соответствии с размером регистрового окна (32

регистра). Поле расширения кода операции (8 бит) предусмотрено для команд с плавающей запятой. Два крайних левых бита в данном и других типах команд используются для идентификации формата команды.

Формат *б)*, в отличие от формата *а)*, вместо одного из входных регистров использует непосредственно заданную в 13-битном поле константу со знаком. Для различения этих двух форматов предусмотрен бит 13, содержащий 0 в случае формата *а)* и 1 в случае формата *б)*.

В формате *в)* представляется команда SETHI, которая используется как вспомогательная для передачи непосредственного 32-битного операнда в регистр. Она устанавливает биты с 10 по 31, а следующая команда передает оставшиеся биты, заданные в непосредственном формате.

Формат *г)* используется для команд непрогнозируемого перехода с 22-битным смещением. При этом поле Условие определяет, какое условие нужно проверять. Существует модификация данного формата для команд переходов с прогнозированием, в которой под поле смещения оставлено 19 битов, а 3 освободившихся бита используются для прогнозирования (1 бит санкционирования перехода) и определения набора используемых битов кода условия (2 бита). Бит 29 в данных форматах позволяет избегать пустых операций при определенных условиях.

Формат *д)* используется для команды обращения к процедуре и комментариев не требует.

В отличие от рассмотренной архитектуры команд, команды других архитектур по длине могут отличаться и занимать часть слова, целое слово или несколько слов. В качестве примера, рассмотрим форматы команд IA-32. Для этого используем обобщенное их представление, изображенное на рис. 5.6.

Согласно рис. 5.6, форматы команд IA-32 могут содержать до шести полей разной длины, пять из которых (или некоторые из них) могут отсутствовать. Перед кодом операции могут задаваться от 1 до 5 префиксных байтов, модифицирующих выполняемую операцию.

В некоторых командах младший бит кода операции указывает разрядность участвующих в ней операндов. При нулевом значении этого бита выполняются операции с байтами, при единичном - операции со словами. Разрядность слов, 16 или 32, определяется режимом работы процессора и значением бита D в дескрипторе сегмента кода. Для некоторых команд разрядность операндов может меняться соответствующим префиксом.

Бит, следующий после младшего бита в коде операции, может быть задействован в качестве указателя способов использования непосредственно

задаваемых операндов или может определять, является ли адрес памяти (если он есть в коде команды) источником или приемником.

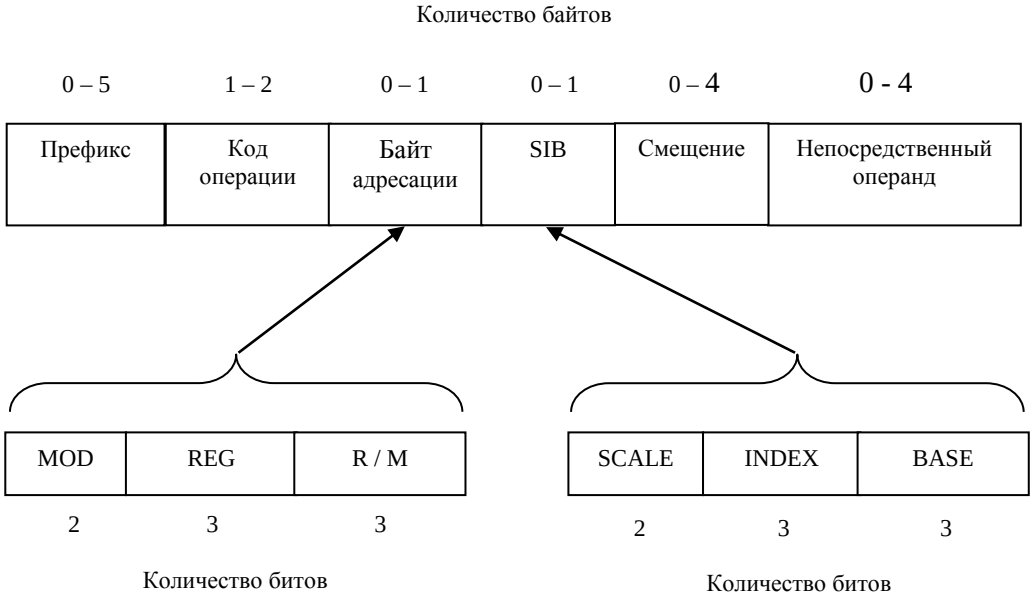


Рис. 5.6. Обобщенное представление форматов команд процессоров IA-32

Байт адресации состоит из трех полей. Коды полей MOD и R / M определяют адрес одного из операндов команды в соответствии с выбираемым способом адресации. Для реализации некоторых способов адресации (например, базово-индексной адресации с масштабированием) дополнительно используются поля байта SIB: SCALE, INDEX, BASE – масштаб, индекс, база. Трехбитные поля INDEX и BASE указывают регистры, используемые в качестве индексного и базового регистров, соответственно. В двухбитном поле SCALE задается масштабный коэффициент, на который умножается содержимое индексного регистра перед сложением с остальной частью адреса. Одним из слагаемых адреса может быть содержимое поля Смещение (например, в случае использования базово-индексной адресации со смещением и масштабированием).

Поле REG байта адресации двухадресных команд содержит код регистра, в котором находится второй операнд. В случае одноадресных команд это поле используется как расширение (дополнительные биты) кода операции.

Сложность и нерегулярность форматов команд IA-32, проиллюстрированная вышеизложенным материалом, объясняется тем, что данная архитектура развивалась на протяжении нескольких поколений процессоров, и ранее в нее были включены не очень удачные варианты команд. Принцип обратной совместимости, которого строго придерживалась IA-32, не позволял их менять.

Следует особо остановиться на расширении набора команд IA-32 MMX (MultiMedia eXtesion – мультимедийное расширение). Команды MMX реализуют групповую обработку нескольких целочисленных операндов разрядностью 1, 2, 4 или 8 байт (принцип SIMD – Single Instruction stream Multiple Data stream). Включение их в состав команд IA-32 и реализация (начиная с Pentium MMX и выше) специального аппаратного блока для их выполнения позволили значительно повысить эффективность обработки аудио и видеоинформации. Выполнение команд MMX осуществляется с использованием восьми 80-разрядных регистров, разделяемых с FPU (Floating-Point Unit – блок выполнения операций с плавающей точкой). FPU задействует эти регистры под именами ST0-ST7 как кольцевой стек для хранения чисел с плавающей точкой расширенной точности. MMX использует их как 64-разрядные регистры MM0-MM7 для хранения восьми 8-разрядных, четырех 16-разрядных, двух 32-разрядных или одного 64-разрядного операндов, над которыми одновременно выполняется очередная команда. Совпадение регистров FPU и MMX накладывает определенные ограничения на чередование кодов указанных расширений в программе (для переключения с блока MMX на блок FPU процессору требуется несколько десятков тактов).

Заслуживает также внимания расширение команд IA-32 SSE (Streaming SIMD Extensions – потоковое SIMD-расширение), берущее свое начало от процессора Pentium III и обеспечивающее групповое выполнение операций над числами с плавающей точкой. Блок SSE-2 процессора Pentium IY реализует исполнение 144 векторных команд с плавающей точкой, используя независимый блок из восьми 128-разрядных регистров XMM0–XMM7. В указанных регистрах могут размещаться и параллельно обрабатываться два числа с плавающей точкой двойной точности или четыре числа одинарной точности. Блок может выполнять как векторные, так и скалярные команды, для выполнения которых используются 32 младших разряда регистров XMM0–XMM7. В SSE-2 предусмотрена также параллельная обработка целочисленных операндов: шестнадцати 8-разрядных, восьми 16-разрядных, четырех 32-разрядных или двух 64-разрядных. Поскольку SSE-2 имеет свое отдельное оборудование, его команды могут спокойно смешиваться с инструкциями MMX и FPU.

5.3. Микроархитектурный уровень процессора

Перед уходом на микроархитектурный уровень, определяющий технологию выполнения (интерпретации) машинных команд с использованием

конвейеризации, суперскалярных вычислений, предсказания переходов, переименования регистров, переупорядочивания команд и других приемов, позволяющих рационально задействовать имеющиеся аппаратные средства, остановимся на вопросе: в какой степени архитектурный уровень машинных команд должен влиять на планирование внутренней структуры вычислительного процесса?

Существуют две альтернативных концепции.

Согласно первой система команд не должна содержать каких-либо указаний по рациональному использованию функциональных блоков процессора. Планирование динамики вычислений должно осуществляться самим процессором в соответствии с технологиями его микроархитектурного уровня, реализованными гибким сочетанием аппаратных и компилятивных средств. Наиболее строго придерживаются этой концепции RISC-архитектуры (например, процессоры UltraSPARC).

Вторая концепция, напротив, предполагает открытое управление на уровне машинных команд внутренними аппаратными ресурсами процессора с целью наиболее рационального их использования. Реализация такого подхода требует введения в коды инструкций дополнительных полей, которые в отличие от традиционных, указывающих на то, *что надо делать*, будут информировать о том, *как это надо делать*. Процессоры, следующие данной концепции, называют процессорами с длинным командным словом (архитектура VLIW – Very Long Instruction Word). Очевидно, что при реализации таких процессоров основная часть нагрузки ложится скорее на компилятивные средства, нежели аппаратные, которые для выполнения многих функций могут оказываться слишком громоздкими.

Концепция VLIW положена в основу 64-битной аппаратной платформы IA-64 (Intel Architecture-64), разработанной совместно компаниями Intel и Hewlett Packard и реализованной в микропроцессорах Itanium и Itanium 2. Эта архитектура (в терминах Intel – EPIC - Explicity Parallel Instruction Computing - вычисления с явной параллельностью инструкций) изначально не планировалась как совместимая с архитектурой x86, хотя и не исключала использование в качестве платформы для домашних компьютеров. Но после выпуска фирмой AMD 64-битной архитектуры AMD64, сохранившей совместимость с x86, актуальность использования платформы IA-64 где-либо, кроме серверов, пропала.

Микроархитектурный уровень связан с интерпретацией команд, предусмотренных архитектурой команд процессора. Его организация зависит

от выбора компромисса между критериями, основными из которых являются производительность и стоимость (аппаратная сложность) процессора. В качестве характеристик, определяющих производительность процессора, принимаются во внимание:

- количество циклов (тактов), необходимое для выполнения команд;
- длительность цикла (тактовая частота процессора);
- возможность параллельного выполнения нескольких операций с применением конвейеризации и суперскалярных вычислений.

Организация простейшей микроархитектуры процессора представлена в общем виде на рис. 5.7.

Схема, изображенная на рис. 5.7, реализует процесс поочередного выполнения извлекаемых из памяти команд под контролем блока микропрограммного управления, в функции которого входят:

- определение адреса и инициация выборки очередной команды из памяти;
- дешифрация команды;

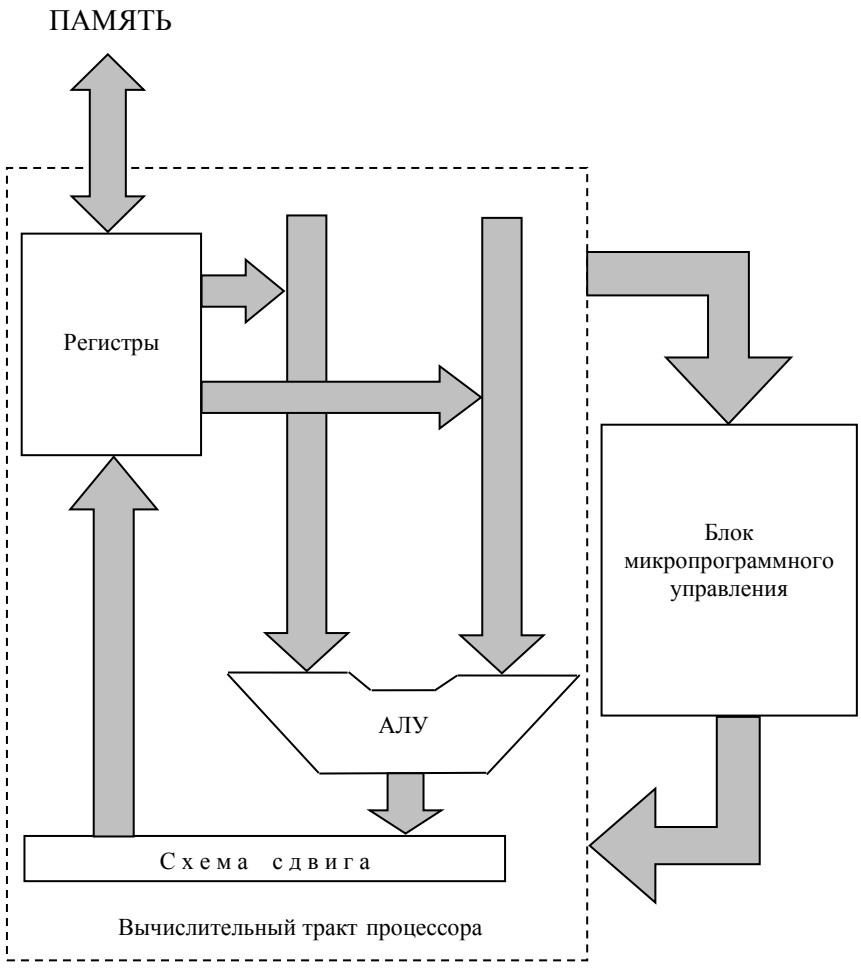


Рис. 5.7. Общее представление простейшей микроархитектуры процессора

- последовательное считывание из управляющей памяти соответствующих микрокоманд и инициация их выполнения вычислительным трактом процессора.Сосредоточение всех указанных функций в едином блоке микропрограммного управления, а также использование вычислительного тракта процессора для определения адреса и выборки очередного кода команды из памяти, ограничивают возможность конвейеризации выполнения команд.

Конвейеризация (pipelining) предполагает структурирование процесса обработки каждой инструкции (команды) в виде последовательности этапов, каждый из которых связан с определенной функциональной ступенью конвейера. Такими ступенями могут быть: *предвыборка команды*, *декодирование*, *формирование адресов и выборка операндов*, *выполнение команды*, *запись результата*. Это естественный (классический) конвейер, для реализации которого схема микроархитектурного уровня должна быть преобразована к виду, представленному на рис. 5.8. В этой схеме процессы выборки и дешифрации (декодирования) команд выделены в отдельные аппаратные блоки, выполняющие свои функции самостоятельно, без использования вычислительного тракта процессора и блока микропрограммного управления. При этом последний оперирует микропрограммами меньшей длины, включающими только микрокоманды непосредственного выполнения инструкций.

Блок выборки команд может быть построен с использованием сдвигающего регистра, загружаемого из памяти кодами команд в порядке их следования, и ряда дополнительных регистров для копирования в них в порядке поступления в сдвигающий регистр полей определенной длины. Это могут быть однобайтовые или двухбайтовые поля выделяемых кодов операций, словарные поля извлекаемых непосредственных операндов или смещений и другие. В результате в готовом виде получаются все данные, необходимые для функционирования последующих блоков процессора.

Блок дешифрации команд может быть организован с использованием в качестве основы постоянного запоминающего устройства (ПЗУ), например, архитектуры 2D. Каждая строка ПЗУ, адресуемая кодом операции, содержит два поля. В первом поле указана длина соответствующей команды, во втором – адрес ее первой микрокоманды в управляющей памяти процессора. Длина команды необходима для правильного использования регистров, загруженных блоком выборки команд. Указанный адрес первой микрокоманды служит отправной точкой для формирования на основании содержимого управляющей памяти очереди микрокоманд последовательного выполнения инструкций. При

этом из кодов микрокоманд, включаемых в очередь, исключается за ненадобностью поле «Следующий адрес». Поле «Условие» оставляется только в микрокомандах с условным переходом, имеющих специальный формат с полем для указания адреса микроперехода. Такие микрокоманды помечаются установкой специально отведенного для этого бита. В формате микрокоманд предусмотрен также специальный бит завершения, отмечающий последнюю микрокоманду микропрограммы.

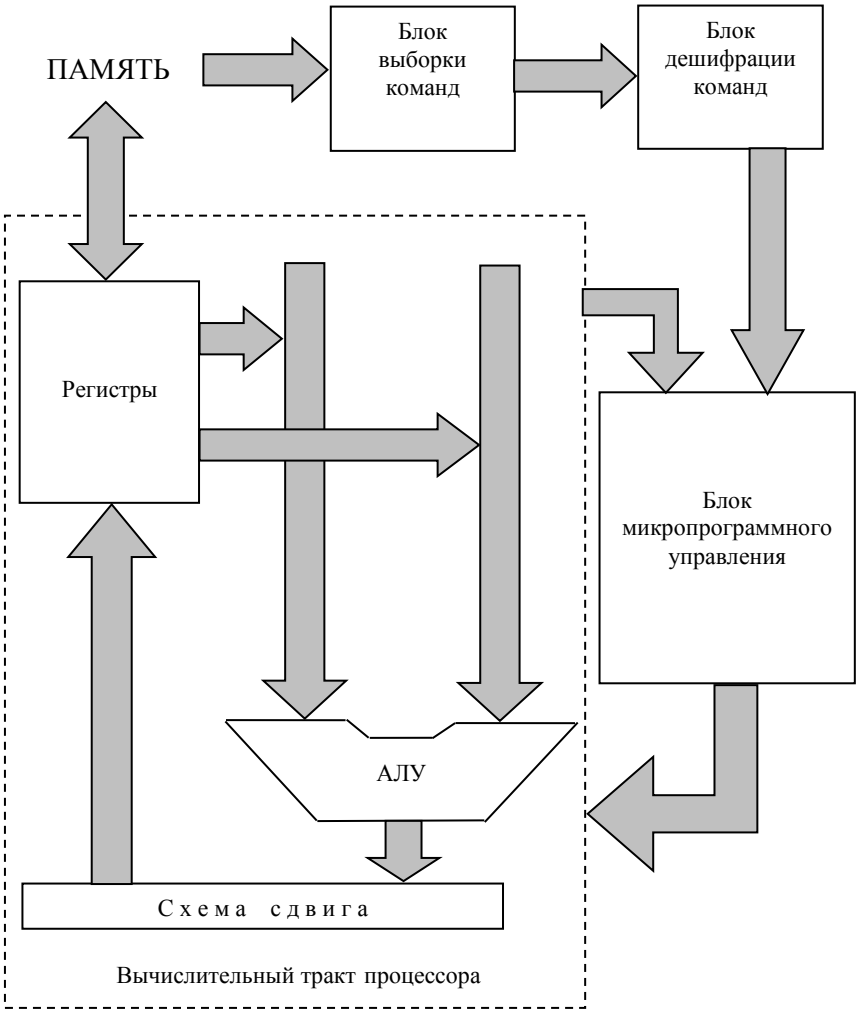


Рис. 5.8. Конвейеризованная микроархитектура процессора

Очевидно, рассмотренный процесс формирования очереди микрокоманд может быть выделен в отдельный аппаратный блок (блок формирования очереди), реализующий дополнительную ступень конвейера.

Блок микропрограммного управления в каждом своем цикле (такте) выполняет с использованием вычислительного тракта процессора очередную

(взятую из очереди) микрокоманду. При этом, цикл складывается из следующих последовательных этапов:

- считывание из очереди кода микрокоманды и установление сигналов управления вычислительным трактом процессора (сигналов активизации регистров, управления АЛУ и других);
- загрузка во входные шины АЛУ содержимого активизированных регистров;
- срабатывание АЛУ и схемы сдвига;
- стабилизация сигналов на выходной шине АЛУ и загрузка результатов в активизированные регистры.

Заметим, что каждый из перечисленных этапов выполняется своей частью аппаратуры блока микропрограммного управления и вычислительного тракта процессора. На каждом этапе задействована только одна аппаратная часть, другие в это время простаивают. Это можно использовать для увеличения глубины (числа ступеней) конвейера, если непрерывный процесс выполнения указанных шагов в одном цикле разорвать введением дополнительных регистров-защелок для фиксации состояний шин вычислительного тракта (рис. 5.9) и поочередной (потактовой) активизации управляющих сигналов кода микрокоманды. Сначала активизируются сигналы управления доступом к входным шинам, затем сигналы управления АЛУ и схемой сдвига и только потом сигналы управления доступом к выходной шине. Каждый такой шаг выполняется быстрее, чем полный цикл вычислительного тракта. Поэтому тактовую частоту процессора можно повысить, и пустить по новым ступеням конвейера более быстрый поток микрокоманд.

Конвейеризация, как способ повышения производительности процессора, хорошо работает с линейным кодом программы. Продвижение по ступеням конвейера последовательно выбираемых команд одна за другой экономит общее время, затрачиваемое на выполнение программы. Число ступеней (*глубина)* конвейера от процессора к процессору может меняться. Так, у процессора Pentium конвейер пятиступенчатый, глубина конвейера Pentium II составляет 12, Pentium III – 10, а у Pentium IY она достигает 20 ступеней (Hyper Pipelined Technology). Увеличение длины конвейера, связанное с декомпозицией процесса выполнения команды на более мелкие (а значит более быстрые) этапы, позволяет существенно увеличить частоту процессора. Однако надо заметить, что каждая команда при этом выполняется дольше, т.к. проходит большее число этапов. Это приводит к потерям времени при очистке и перезагрузке конвейера в случаях ошибок в предсказании переходов.

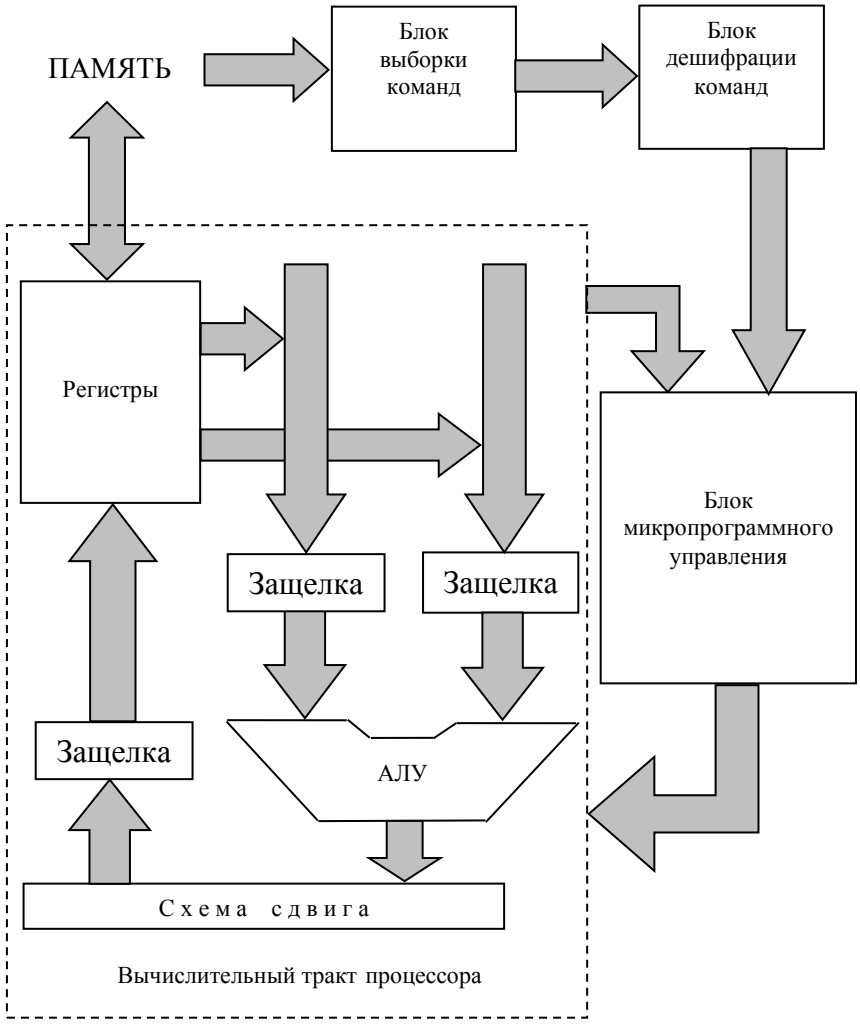


Рис. 5.9. Конвейеризованная микроархитектура процессора с увеличенной глубиной конвейера

Технология *предсказания переходов* (*прогнозирования ветвлений*), широко применяемая в современных процессорах, направлена на то, чтобы сохранить сформированный конвейер, не снижая его производительности. Поскольку декодирование происходит на следующей стадии после выборки команды, то до распознания команды перехода и определения нового направления выборки команд уже могут быть вызваны некоторые коды ненужных команд. Для исключения таких ситуаций может быть применена технология *отсрочки ветвления*, согласно которой компилятор заполняет одну или несколько позиций после команды перехода полезными с его точки зрения командами или пустыми командами, не производящими никаких действий. Это не нарушает правильности выполнения программы, но приводит к значительным потерям на холостых циклах. Поэтому в современных процессорах предусматриваются специальные аппаратные средства предсказания переходов.

Различают технологии *динамического* и *статического* прогнозирования переходов. Первые реализуют прогнозы непосредственно в ходе выполнения программы и связаны с построением довольно сложных аппаратных средств. Вторые основную нагрузку возлагают на компиляторы, которые априори анализируют ситуации, связанные с условными переходами и с помощью определенных битов кода (по сути дела новой команды) сообщают аппаратному обеспечению направление перехода. Иногда статическое прогнозирование выполняется компилятором на основании результатов прокрутки программы с фиксацией всех ее переходов. После этого компилятор вносит в программу все необходимые коррективы.

Динамическое предсказание переходов предусматривает наличие в составе аппаратного обеспечения процессора специального блока BTB (Branch Target Buffer – буфер цели перехода). Широко распространенным подходом к организации такого блока является фиксация всех переходов в специальной *таблице динамики переходов*. Строка таблицы содержит поля для указания ее достоверности (один бит), адреса (или тега) команды перехода и одного или нескольких битов прогнозирования перехода. Возможно также наличие поля для непосредственного сохранения целевого адреса последнего перехода. По m младшим разрядам адреса очередной выбранной команды перехода определяется строка таблицы (общее число строк 2^m) и проверяется совпадение указанного в ней тега со старшей составляющей адреса команды. Если совпадение имеет место и строка достоверна, то в соответствии с битом (битами) прогноза определяется переход. При несовпадении срабатывает та или иная предусмотренная эвристика (например, можно пойти вперед по линейному коду или отдать предпочтение переходу назад). Рассмотренная организация таблицы аналогична организации кэш-памяти прямого отображения. Очевидно, для придания большей гибкости системе в плане разрешения конфликтных ситуаций (на одно и то же место в таблице могут претендовать многие команды перехода) целесообразно использовать более сложную таблицу с многовходовой наборно-ассоциативной или полностью ассоциативной организацией.

Процессор с единственным конвейером называется *скалярным* в отличие от *суперскалярного* процессора, имеющего два и более конвейеров, обрабатывающих инструкции параллельно. Основная идея *суперскалярной архитектуры* состоит в преобразовании исходной последовательной программы в как можно большее количество параллельных динамических

вычислительных структур, одновременная реализация которых ускоряет выполнение программы. При этом речь идет о параллельности уровня команды.

Увеличение числа конвейеров путем дублирования полного (или почти полного) состава блоков требует создания громоздкого аппаратного обеспечения. Поэтому, начиная с микроархитектуры Pentium Pro, используется более рациональный подход, реализующий идею одного (главного) конвейера с распараллеливанием только самых сложных его этапов. Этот могут быть этапы декодирования команд и непосредственного выполнения микрокоманд функциональными блоками. Пример такого конвейера [11], лежащего в основе микроархитектуры процессора Pentium II, представлен на рис. 5.10.

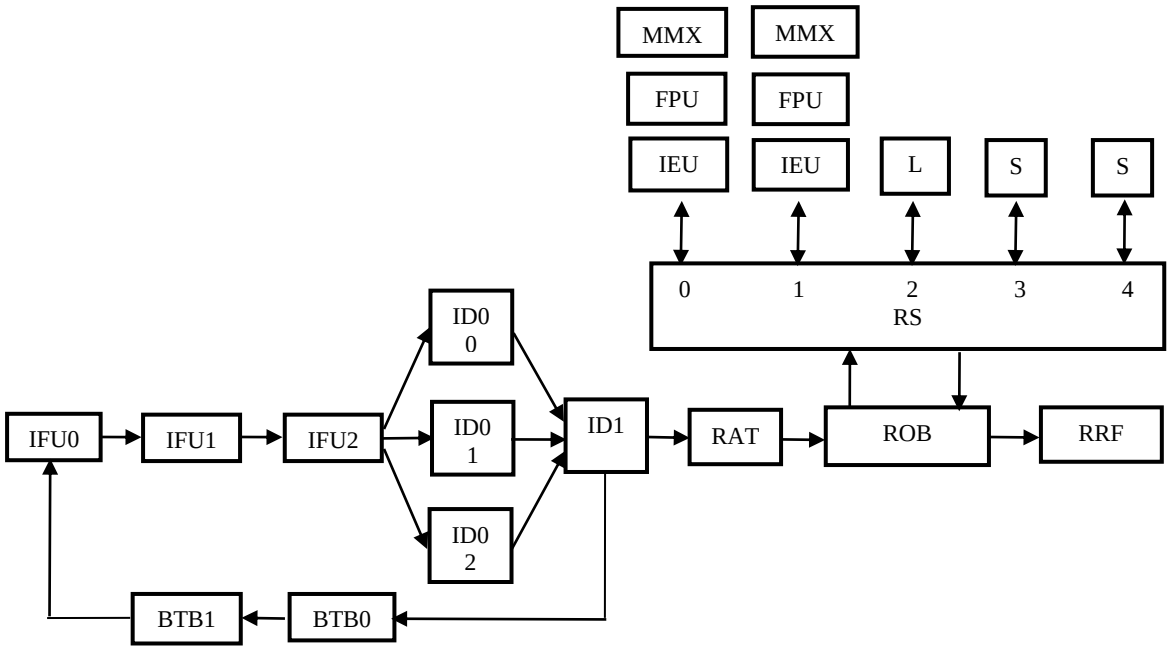


Рис. 5.10. Структура конвейера процессора Pentium II

Каждая из 12 ступеней конвейера реализуется своей частью аппаратного обеспечения.

Блок выборки команд (IFU – Instruction Fetch Unit) обеспечивает первые три ступени конвейера: выборку строк из кэша команд (IFU0), декодирование длины команд (IFU1) и выравнивание команд (IFU2). Четвертую и пятую ступени реализует дешифратор команд (ID – Instruction Decoder). Причем на стадии ID0 работают три параллельных декодера (ID00, ID01, ID02), превращающие команды в последовательности микрокоманд в соответствии с содержимым управляющего ПЗУ процессора. Два декодера дешифруют простые команды, реализуемые одной микрооперацией. Третий обслуживает

сложные команды, выполняемые микропрограммно. На стадии ID1 формируется очередь микрокоманд. При этом отслеживаются условные микропереходы и осуществляется статическое или динамическое их прогнозирование. Эти две ступени конвейера реализуются блоком прогнозирования переходов (BTB), включающим устройство статического прогнозирования BTB0 и устройство динамического предсказания ветвлений BTB1. Для динамического предсказания используются четыре бита прогноза.

Выполнение команд в последовательности, предписанной программой, не всегда оказывается рациональным с точки зрения производительности вычислительной системы. Причиной является взаимозависимость команд, из-за которой последующая команда не может быть выполнена до того, как выполнится предыдущая. Вместе с тем в последовательности команд можно выделить независимые команды, внеочередное выполнение которых (пока другие команды ждут друг друга) будет способствовать повышению эффективности вычислительного процесса.

Очевидно, что для запуска очередной команды необходимо иметь информацию о состоянии задействованных в ней регистров. Такую информацию дает предусматриваемый в микроархитектуре процессора *счетчик обращений* (scoreboard). Каждый регистр, используемый в качестве источника, он обеспечивает счетчиком небольшой разрядности, в котором фиксируется число обращений к этому регистру со стороны активизированных команд. При запуске очередной команды, использующей регистр в качестве источника, значение его счетчика увеличивается на единицу. При завершении выполнения команды – уменьшается на единицу.

Регистры, используемые в качестве пункта назначения, снабжены также однобитными счетчиками для фиксации факта использования их для записи одной из выполняемых команд. Еще один бит должен быть предусмотрен для контроля за записями, которые обязаны сделать пропущенные команды. В дополнение ко всему, счетчик обращений отслеживает занятость функциональных блоков вычислительного тракта процессора.

Состояния вышеуказанных счетчиков позволяют определить возможность выдачи на исполнение очередной команды, руководствуясь следующими правилами:

- нельзя читать записываемый источник (RAW-зависимость, Read After Write);

- нельзя записывать в читаемый источник (WAR-зависимость, Write After Read);

- нельзя записывать в пункт назначения, задействованный другой командой (WAW-зависимость, Write After Write).

В отличие от RAW-зависимости, носящей принципиальный характер, WAR- и WAW-зависимости можно устранить, введя дополнительные ресурсы для записи результатов.

Эффективная загрузка параллельно функционирующих блоков может осуществляться аппаратным путем, программными средствами компиляции или комбинацией аппаратного и программного обеспечения. Компиляторы, как правило, используют довольно тонкие технологии распараллеливания последовательных программ. Аппаратными средствами выделяются более простые формы параллелизма. Например, естественный параллелизм целочисленных вычислений адресов операндов и выполнения операций с плавающей запятой над ними.

Ограничение возможности параллельного выполнения инструкций зачастую связано с использованием общих регистров. Обходить это ограничение позволяет применяемая в современных процессорах технология *переименования регистров* (register renaming). Такие процессоры фактически имеют более восьми общих регистров и предусматривают возможность использования параллельными инструкциями одинаковых логических имен, но соответствующих разным физическим регистрам (у Pentium IY таких регистров, обслуживающих только целочисленные вычисления, 128). Однако при этом предполагается отсутствие фактических зависимостей по данным.

Блок переименования регистров в конвейере на рис. 5.10 для динамического отображения логических имен на конкретные физические регистры использует *таблицу псевдонимов регистров* (RAT – Register Alias Table). В строке таблицы, соответствующей логическому регистру, при каждом его переименовании указывается новое имя используемого физического регистра, на которое подменяется логическое имя в последующих командах. С каждым физическим регистром связан счетчик, который отслеживает состояние его занятости. Значение счетчика увеличивается на единицу при каждом переименовании операнда, использующего данный физический регистр. После отработки операнда счетчик уменьшается на единицу. Равенство значения счетчика нулю свидетельствует о том, что регистр свободен и может быть использован для других переименований.

Микрооперации с переименованными регистрами поступают в *буфер переупорядочивания команд* (ROB – ReOrder Buffer). Далее с ними работает *резервирующая станция* (RS – Reservation Station). Отслеживая готовность

операндов, занятость функциональных блоков и разрешая конфликты ресурсов, сложный счетчик обращений RS формирует очередь выполнения микроопераций. При этом микрооперации могут подаваться на исполнение не в том порядке, как они поступили в ROB.

Доступ к функциональным блокам обеспечивается пятью выходными портами RS. Порты 0 и 1 разделены тремя, дважды продублированными, функциональными блоками, каждый. Это блок выполнения целочисленных операций (IEU), блок выполнения операций с плавающей точкой (FPU) и блок выполнения команд MMX. Порт 2 обеспечивает выход на блок загрузки (L – Load). Для блоков сохранения (S – Store) предусмотрены порты 3 и 4.

Выполненная в одном из функциональных блоков микрооперация через RS возвращается обратно в ROB, где ожидает возврата. Блок возврата, сохраняя первоначальный порядок следования микроопераций, принимает выполненные микрооперации в файл регистров выгрузки (RRF – Retirement Register File) и возвращает результаты выполнения в места из назначения. Результаты спекулятивного выполнения команд он хранит в специальных регистрах и аннулирует их в случае неподтвержденного (спекулятивного) направления перехода. Восстанавливая первоначальный порядок следования микроопераций, блок возврата гарантирует правильную обработку прерываний и устранение результатов, полученных после неправильного предсказания ветвлений.

Рассмотренная организация конвейера Pentium II наглядно демонстрирует так называемую технологию *продвижения данных* (data forwarding). Она подразумевает выполнение всех возможных действий по исполнению инструкции до готовности ее операндов. При этом декодированная, готовая к выполнению инструкция дожидается своих операндов с выходов соответствующих блоков, после чего может быть передана на свое исполнительное устройство.

6. СИСТЕМОТЕХНИЧЕСКИЙ УРОВЕНЬ ПРЕДСТАВЛЕНИЯ АРХИТЕКТУРЫ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

6.1. Магистрально-модульный принцип организации вычислительной системы

В отличие от свойственного первым ЭВМ принципа централизованного управления, в основе организации современных цифровых ВС лежит так называемый *магистрально-модульный* принцип. Суть его заключается в наличии набора магистралей (шин, Bus), к которым свободно подсоединяются функциональные модули, включающиеся в процесс обмена информацией по ВС в режиме разделения времени. Такой подход обеспечивает открытость системы, т.е. возможность достаточно оперативно расширять ее функциональность и наращивать производительность путем изменения состава задействованных в ней модулей.

Шинный интерфейс – это совокупность аппаратных, программно поддерживаемых средств, обеспечивающих совместимость функциональных модулей (процессора, памяти, периферийных устройств), объединяемых в единую вычислительную систему в соответствии с магистрально-модульным принципом ее организации.

Обеспечение совместимости рассматривается в трех аспектах.

Совместимость в функциональном аспекте (*функциональная совместимость*) предполагает наличие определенного состава сигналов, правил их взаимообусловленности и распределения во времени, обеспечивающих обмен информацией по шине. Все это определяет так называемый *протокол шины*.

Электрическая совместимость обеспечивается определенными уровнями сигналов, напряжений питания, нагрузочной способности и других характеристик, определяющих возможность взаимодействия устройств со схемотехнической точки зрения.

Механическая совместимость определяется требованиями к конструкторскому исполнению (по форме, габаритам и т.п.) как самих функциональных модулей, вступающих в информационный обмен по шине, так и средств их сопряжения с шиной.

Центральным для шинного интерфейса является понятие шины. *Шина* - это группа проводников, объединяющая в себе линии, предназначенные для обмена информацией между устройствами компьютера. Следует различать *внутренние шины процессора*, обеспечивающие информационный обмен между его узлами на микроархитектурном уровне, и *внешние* (по отношению к процессору), так называемые *системные шины*. Каждому типу шин присущи

две характеристики: разрядность и частота передачи данных, по которым можно вычислить пропускную способность шины.

По характеру циркулирующей в них информации линии шины подразделяются на адресные линии, линии данных и линии управления (рис. 6.1.).

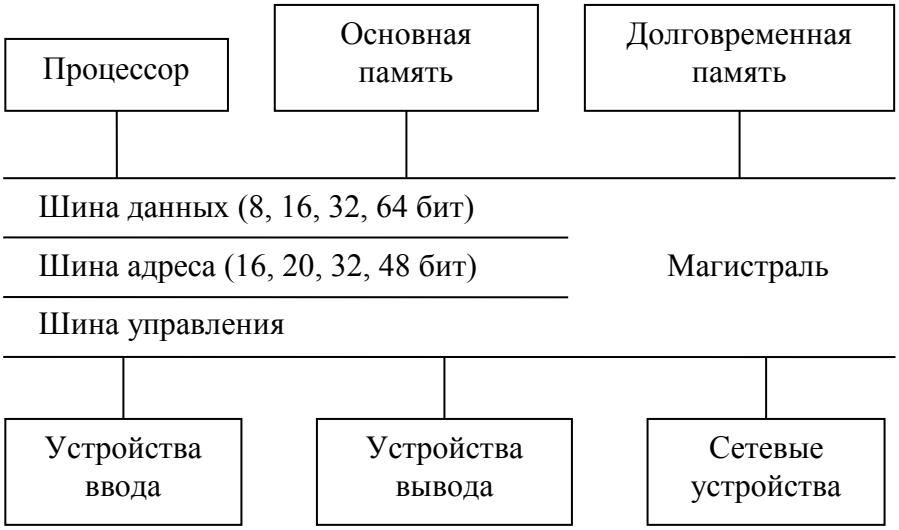


Рис. 6.1. Общая схема магистрально-модульной организации ВС

Линии данных (шина данных) предназначены для передачи информации (данных различных типов, кодов выполняемых или генерируемых программ) между устройствами вычислительной системы. Количество таких линий определяется разрядностью процессора, т.е. числом двоичных разрядов, которые могут быть обработаны или переданы процессором одновременно.

Адресные линии (шина адреса) призваны обеспечить два основных адресных пространства: адресное пространство памяти и адресное пространство портов периферийных устройств. Первое связано с адресацией ячеек оперативной и постоянной памяти вычислительной системы, второе – с организацией доступа к *портам* (регистрам) интерфейсных схем периферийного оборудования. Число задействованных адресных линий n определяет размер используемого адресного пространства 2^n. В современных компьютерах в 64-битных процессорах используется 48-разрядная шина адреса, позволяющая адресовать 2^{48} = 256 терабайт памяти. Теоретически возможная ширина шины адреса в 64 бита не используется из практических соображений - нет потребности и физической возможности использования 16 эксабайт памяти.

Линии управления (шина управления) служат для передачи управляющих сигналов, реализующих протокол шины. К примеру, сигналы управления указывают, какую операцию (запись или считывание информации из памяти) нужно производить, синхронизируют процессы обмена информацией между устройствами и т.д.

Устройства, вступающие в информационный обмен по шине, подразделяют на *задающие* (*активные*) и *подчиненные* (*пассивные*). Первые инициируют вторые на выполнение определенных действий.

Задающее устройство может подключаться к шине через интерфейсную схему, называемую *шинным драйвером*. По сути это усилитель, обеспечивающий достаточную для активизации шины мощность поступающих на нее сигналов. Подчиненное устройство связываются с шиной *приемником шины*. Устройства, способные быть и задающими и подчиненными, подсоединяются к ней через *приемопередатчик*.

Указанные типы интерфейсных схем называют *шинными формирователями.* Имея выводы с третьим состоянием, они способны не только формировать требуемые уровни логических сигналов, но и отключать при необходимости устройство от шины.

Аналогичные функции могут выполнять так называемые *буферные регистры*, которые, в отличие от шинных формирователей, способны сохранять информацию, т.е. осуществлять ее временную буферизацию. Буферный регистр, выполняющий функции *порта ввода-вывода*, как правило, имеет разъем для подключения внешнего устройства.

К некоторым линиям шины подключение устройств осуществляется через интерфейсные схемы, имеющие выходы с открытым коллектором. Это делается там, где необходимо обеспечить операцию монтажного ИЛИ для выходов на общую линию (например, на линию запроса шины).

Шинные формирователи и буферные регистры решают схемотехнические проблемы управляемого выхода устройств (в том числе и процессора) на системную шину. Более сложные функции выполняют *периферийные адаптеры*. Они способны поддерживать несколько оперативно программируемых режимов обмена между внешним устройством и шиной. Реализующие их микросхемы интегрируют в себе функции шинных формирователей, портов ввода-вывода и средств программного управления режимами обмена.

Наиболее сложными устройствами сопряжения с шиной являются *контроллеры*. Наряду с функциями адаптера контроллер способен выполнять некоторые самостоятельные действия, связанные с выполнением очередной команды центрального процессора. Для этих целей он может иметь в своем составе собственный процессор. Показательным в этом плане является контроллер DMA (Direct Memory Access – прямой доступ к памяти). Этот контроллер освобождает центральный процессор от выполнения функций программного обмена внешних устройств с памятью. Он осуществляет прямую пересылку данных, минуя центральный процессор, получая от него в свои регистры начальный адрес и размер пересылаемого блока памяти, а также информацию о направлении и режиме обмена. Далее инициатором обмена становится внешнее устройство, а центральный процессор может продолжать выполнение своих функций, используя ресурсы, не задействованные DMA.

Периферийные адаптеры и контроллеры могут располагаться непосредственно на системной плате компьютера или на картах расширения, устанавливаемых своими краевыми печатными разъемами в универсальные щелевые разъемы системной шины, называемые *слотами расширения*. На системной плате обычно монтируются стандартные (обязательные) интерфейсы устройств ввода-вывода (например, контроллер DMA). Карты расширения используются либо нестандартными интерфейсами, либо стандартными, но имеющими качественное отличие модификаций и допускающими возможность выбора (например, видеокарты, отличающиеся объемом памяти, наличием или отсутствием графического ускорителя и т.п.).

По способу упорядочения действий, выполняемых в соответствии с протоколом шины, различают *синхронные* и *асинхронные* шины. В синхронной шине все события привязываются к тактовым импульсам синхрогенератора, для которого выделена специальная линия управления. Любая транзакция (обмен информацией) по такой шине занимает целое число циклов шины, определяемых периодом следования тактовых импульсов. Асинхронная шина не предусматривает наличия тактового генератора. Координация событий осуществляется посредством управляющих сигналов, которыми обмениваются взаимодействующие устройства. Циклы такой шины могут иметь различную длину, зависящую от типа транзакции и взаимного расположения во времени реализующих ее управляющих сигналов.

Необходимость воспользоваться шиной может появиться одновременно у нескольких задающих устройств, подсоединенных к ней через тот или иной порт. Для разрешения таких конфликтных ситуаций в архитектуре шины предусматривается специальный механизм, называемый *арбитражем шины*.

Механизм *централизованного арбитража*, реализуемый специальной схемой-арбитром, интегрированной на кристалле центрального процессора или представленной отдельной микросхемой чипсета, может работать по принципу системы последовательного опроса. Такая система использует две линии управления. На одну из них, называемую *линией запроса шины*, выход устройств организован по схеме монтажного ИЛИ (выходы с открытым коллектором). По состоянию этой линии арбитр определяет наличие или отсутствие запросов шины. Другая линия, *линия предоставления шины*, последовательно связывает все устройства ввода-вывода. По ней арбитр посылает сигнал предоставления шины. Очередное устройство цепочки, получившее сигнал и обнаружившее наличие запроса шины с его стороны, не распространяет этот сигнал дальше по линии и начинает пользоваться шиной. Устройства, не выставлявшие сигнал запроса шины, передают сигнал предоставления шины дальше по цепочке. Такой механизм упорядочения пользования шиной дает максимальный приоритет устройству, расположенному ближе всех остальных к арбитру по линии предоставления шины.

Более гибкой является система последовательного опроса на разных уровнях приоритета. Такая система обслуживается несколькими парами линий

запроса и предоставления шины (таких пар может быть 4, 8 или 16). Каждая пара имеет свой уровень приоритета. Каждое устройство связано с одной из пар линий, причем на линии более высокого уровня выведено большее количество устройств. Последовательный опрос арбитр начинает с самой высокой линии приоритета.

Иногда арбитр использует дополнительную *линию подтверждения приема*, при активизации которой устройством, получившим в распоряжение шину, происходит отключение его от линий запроса и предоставления шины. Это позволяет другим устройствам, не теряя времени, запрашивать шину во время ее использования и получать право выхода на нее по окончании текущей транзакции (об этом информирует сброс линии подтверждения приема).

В целях экономии оборудования арбитр, не отличающийся особой «интеллектуальностью» в системе последовательного опроса, может быть исключен из нее. Его функции способен выполнить источник питания, используя три линии управления. Одна из них - *линия запроса шины* по монтажному ИЛИ, вторая – *линия занятости* (BUSY) и третья – *линия арбитража* шины. Линия арбитража, выведенная через ключ на источник питания, последовательно соединяет все устройства. Каждое устройство по линии арбитража имеет вход IN и выход OUT, соединенный с входом IN следующего устройства в цепочке. Когда шина не затребована ни одним устройством, на линию подается напряжение, активизирующее входы и выходы всех устройств. Каждое устройство, стремящееся получить доступ к шине, при ее освобождении сбрасывает сигнал своего выхода OUT, тем самым сбрасывая вход IN следующего устройства в цепочке. Шину в свое распоряжение получает устройство, у которого вход IN остался активным при сброшенном сигнале OUT. Оно активизирует линию BUSY и сигнал своего выхода OUT и приступает к работе с шиной.

Все устройства компьютера так или иначе общаются с памятью. Чаще других к ней обращается центральный процессор. Поэтому во избежание постоянных конфликтов на общей шине, из-за которых процессор вынужден довольствоваться самым низким приоритетом, целесообразно обеспечить его отдельным трактом выхода на память. Другие устройства могут использовать при этом децентрализованный арбитраж шины. Суть его состоит в обеспечении каждого устройства своей приоритетной линией запроса шины и возможностью контроля всех линий запроса. В конце очередной транзакции, каждое устройство должно определить, обладает ли оно в текущий момент высшим приоритетом (предполагается сброс к этому времени сигнала запроса устройства, владеющего шиной). Устройство, получившее положительный результат, по окончании транзакции начинает пользоваться шиной. Такая система арбитража не требует присутствия в ней арбитра, но довольно громоздка из-за большого количества линий запроса.

Подключение отдельных модулей компьютера к магистрали, осуществляемое на физическом уровне с помощью адаптеров и контроллеров, на

программном уровне обеспечивается загрузкой в оперативную память драйверов устройств, которые обычно входят в состав операционной системы.

Набор адаптеров и контроллеров, спроектированных для совместной работы, называют чипсетом (chipset). В компьютерах чипсет, расположенный на материнской плате, играет роль связующего компонента (моста), обеспечивающего взаимодействие процессора с памятью, устройствами ввода-вывода и другими адаптерами (рис. 6.2). Чаще всего чипсет материнских плат современных компьютеров состоит из двух основных микросхем:

- *Контроллер-концентратор памяти* или *северный мост (northbridge);*
- *Контроллер-концентратор ввода-вывода* или *южный мост (southbridge).*

Северный мост обеспечивает взаимодействие процессора с оперативной памятью и графическим адаптером, соединяясь с ними высокоскоростной шиной (в современных процессорах контроллер памяти зачастую интегрируется непосредственно в корпус процессора).

Южный мост обеспечивает взаимодействие процессора и оперативной памяти с низкоскоростными интерфейсами PCI, IDE, SATA, USB.

Разделение чипсета на два моста обусловлено разностью в скоростях передачи данных между устройствами. Процессор, оперативная память и графический адаптер имеют скорость взаимодействия значительно выше в сравнении с жестким диском и периферийными устройствами. Ситуация также осложняется существованием различных интерфейсов шин, которые имеют свои собственные характеристики разрядности и частоты передачи данных.

6.2. Шинные интерфейсы вычислительных систем

Для подключения модулей к вычислительной системе необходимо определить их место в ее шинной организации [12], [13]. Большинство компьютеров имеет как внутренние, так и внешние шины. Внутренняя шина (локальная шина) подключает все внутренние компоненты компьютера к процессору и памяти. Внешняя шина подключает внешнюю периферию к материнской плате.

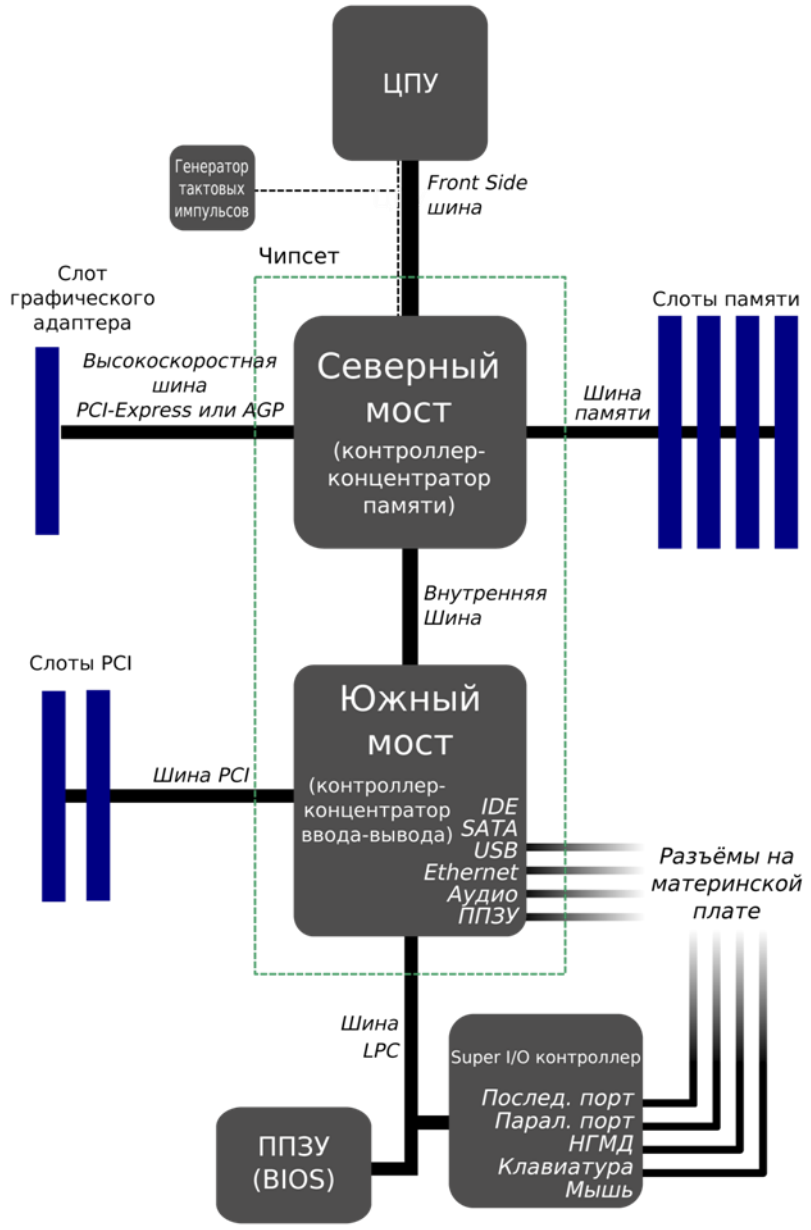


Рис. 6.2. Схема магистрально-модульной организации персонального компьютера

Front Side шина (Front Side Bus - FSB, системная шина) — шина, обеспечивающая соединение между x86/x86-64-совместимым центральным процессором (ЦПУ - центральное процессорное устройство) и внутренними устройствами. Микропроцессор через FSB подключается к системному контроллеру (северному мосту). Системный контроллер имеет в своём составе контроллер памяти, а также контроллеры шин, к которым подключаются периферийные устройства. Северный мост обеспечивает взаимодействие процессора с оперативной памятью и графическим адаптером, соединяясь с ними высокоскоростной шиной. В современных процессорах контроллер

памяти зачастую интегрируется непосредственно в корпус процессора. Подключение к северному мосту графического адаптера осуществляется посредством шин PCI-Express или AGP (Accelerated Graphics Port, ускоренный графический порт.

Менее производительные устройства, такие как микросхема ППЗУ (программируемое постоянное запоминающее устройство) BIOS, устройства с шиной PCI, подключаются к южному мосту, который соединяется с северным мостом специальной внутренней шиной.

Таким образом, шина FSB работает в качестве магистрального канала между процессором и чипсетом.

Некоторые компьютеры имеют внешнюю кэш-память, подключённую через «заднюю» шину *(Back Side Bus - BSB)* процессора. В этом случае говорят о технологии *двойной независимой шины (Dual Independent Bus — DIB)*, которая впервые была реализована в процессорах шестого поколения и предназначалась для увеличения пропускной способности шины процессора и повышения его производительности. BSB обеспечивает более быструю работу кэш-памяти второго уровня (на скорости ядра процессора), чем в том случае, если бы ей пришлось использовать (совместно с процессором) основную шину FSB. Для реализации архитектуры DIB кэш-память второго уровня была перемещена с системной платы в один корпус с процессором.

Каждая из вторичных (после системной) шин вычислительной системы работает на своей частоте, которая может быть как выше, так и ниже частоты FSB. Иногда частота вторичной шины является производной от частоты FSB, иногда задаётся независимо.

Среди интерфейсов внутренней шины можно выделить следующие.

PCI (Peripheral Component Interconnect) – шина ввода-вывода для подключения периферийных устройств к материнской плате, разработанная компанией Intel в 1992 году. Первоначально шина была 32-разрядной, с частотой 33 МГц, пиковая пропускная способность составляла 132 Мбайт/с. Позже появились 64-разрядные версии и версии с частотой 66 МГц. Модификация PCI 2.1 работает на тактовой частоте до 66 МГц и при разрядности 64 имеет пропускную способность до 528 Мбайт/с.

Характерной особенностью шины является возможность параллельного подключения к ней нескольких устройств (при этом скорость передачи данных разделяется между всеми устройствами на одной шине). В шине предусмотрена поддержка режимов Plug and Play и Bus Mastering. Plug and Play (дословно переводится как «Подключай и играй») - технология, предназначенная для быстрого определения и конфигурирования устройств в компьютере. В зависимости от аппаратного интерфейса и программной платформы (ОС, BIOS) этот режим может включаться на этапе начальной загрузки системы или в ходе

горячей замены (без отключения питания) устройств. Bus Mastering позволяет любому PCI-устройству переходить в режим управления (пересылки данных) шиной без участия центрального процессора. Устройство захватывает шину и становится главным, освобождая центральный процессор на время передачи информации для выполнения других приложений.

AGP (Accelerated Graphics Port, ускоренный графический порт) – интерфейс для подключения видеоадаптера к отдельной магистрали AGP, имеющей выход непосредственно на системную память. Разработка данной шины компанией Intel в 1996 году обусловлена быстрым ростом производительности видеокарт. Им стало не хватать пропускной способности существующей на тот момент шины PCI. Шина AGP может работать на частоте системной шины, обеспечивая высокую скорость передачи 32-разрядных графических данных. В отличие от шины PCI 2.1, на основе которой она разрабатывалась, в ней устранена мультиплексированность линий адреса и данных, а также усилена конвейеризация операций чтения-записи. Шина AGP имеет два режима работы: DMA и Execute. В режиме DMA основной памятью является память видеокарты. Графические объекты хранятся в системной памяти, но перед использованием копируются в локальную память карты. Обмен осуществляется большими последовательными пакетами. В режиме Execute системная и локальная память видеокарты логически равноправны. Графические объекты не копируются в локальную память, а выбираются непосредственно из системной памяти.

Следует отметить, что к 2004 году шина AGP стала недостаточной для поддержки новейших видеокарт, и интерфейс стал замещаться более новой шиной PCI-Express.

PCI-Express (PCI-E) - компьютерная шина, использующая программную модель шины PCI и высокопроизводительный физический протокол последовательной передачи данных. Стандарт разрабатывается компанией Intel с 2002 года. В отличие от PCI, где все устройства подключались параллельно к общей шине данных, PCI-Express использует соединение «точка-точка» между двумя устройствами и в общем случае образует сеть с топологией типа звезда, где центральным элементом выступает коммутатор. Каждое такое соединение называется линией (lane) и является двунаправленным, причем прием и передача информации осуществляется по двум отдельным проводникам. Стандарт PCI-Express описывает порты, состоящие из одной (x1) или нескольких (x2, x4, x8, x16, x32) линий. Пропускная способность PCI-Express различных версий отображена в таблице 5.

Таблица 5.
Пропускная способность шин PCI-Express

Год	Версия	Пропускная способность на x линий				
		x1	x2	x4	x8	x16
2002	1.0	250 Мбайт/с	0.50 Гбайт/с	1.0 Гбайт/с	2.0 Гбайт/с	4.0 Гбайт/с
2007	2.0	500 Мбайт/с	1.0 Гбайт/с	2.0 Гбайт/с	4.0 Гбайт/с	8.0 Гбайт/с
2010	3.0	984.6 Мбайт/с	1.97 Гбайт/с	3.94 Гбайт/с	7.88 Гбайт/с	15.8 Гбайт/с
2017	4.0	1969 Мбайт/с	3.94 Гбайт/с	7.88 Гбайт/с	15.75 Гбайт/с	31.5 Гбайт/с
2019	5.0	3938 Мбайт/с	7.88 Гбайт/с	15.75 Гбайт/с	31.51 Гбайт/с	63.0 Гбайт/с

Шина LPC (Low Pin Count, малое число контактов) - внешняя шина, разработанная компанией Intel и предназначенная главным образом для подключения стандартных устройств, унаследованных от ранних версий персональных компьютеров. Раньше такие устройства подключались к шине *ISA* (*Industry Standard Architecture*). Разработка новой шины обусловлена следующими причинами:

- унаследованные устройства используют заранее определённые диапазоны адресов памяти и ввода-вывода и не обладают возможностью конфигурирования, предусмотренной технологией Plug and Play, поэтому они не могут быть напрямую подключены к шинам (PCI, AGP, PCI-Express) современных персональных компьютеров;
- традиционно использовавшаяся для подключения таких устройств шина ISA, выполнявшая роль единой системной шины расширения в первых персональных компьютерах, имеет много сигналов, что усложняет и удорожает как контроллеры, так и системные платы, поэтому сохранение её в прежнем виде представлялось нецелесообразным.

В современных ПК шина LPC через южный мост подключается к шине PCI или PCI-Express. К LPC обычно присоединяется располагаемая на материнской плате микросхема многофункционального контроллера Super I/O, интегрирующая в себе функции многих контроллеров: параллельного порта (LPT-порт, Line Print Terminal), последовательных (COM, **com**munications) портов, портов клавиатуры и мыши, дисковода гибких дисков (floppy).

Среди интерфейсов внешней шины заслуживают внимания следующие.

ATA (Advanced Technology Attachment) - параллельный интерфейс подключения жестких дисков и оптических дисководов. Первоначальная

версия стандарта была разработана компанией Western Digital в 1986 году и получила название IDE (Integrated Drive Electronics). Для подключения жестких дисков с данным интерфейсом используется 40-проводной кабель (также именуемый шлейфом). Каждый шлейф имеет два или три разъема, один из которых подключается к контроллеру на материнской плате, а один или два оставшихся - к дискам. Если к одному шлейфу подключены два устройства, то одно из них называется ведущим (master), а второе - ведомым (slave). Следует заметить, что разделение устройств на ведущее и ведомое не влияет на способ их работы. Однако в BIOS и операционной системе ведущее устройство отображается в списке устройств перед ведомым.

SATA (Serial ATA) - последовательный интерфейс обмена данными с накопителями информации. Является развитием параллельного интерфейса ATA. В данном интерфейсе используется 7-контактный разъем. В отличие от ATA, один кабель SATA может подключать только одно устройство к магистрали. Интерфейс поддерживает горячее подключение устройства.

USB (Universal Serial Bus) - последовательный интерфейс для подключения различных периферийных устройств. Получил широкое распространение и фактически стал основным интерфейсом подключения периферии к бытовой цифровой технике. Интерфейс позволяет осуществлять не только обмен информацией, но и обеспечивать электропитание периферийного устройства. До версии 2.0 включительно интерфейс представляет собой 4-проводной кабель, 2 провода для передачи данных и еще 2 для питания. В версии 3.0 было добавлено еще 4 линии данных и одна линия питания, за счет чего значительно повысилась скорость передачи данных и попутно возросла толщина кабеля.

Пропускные способности различных версий интерфейсов внешней шины приведены в таблице 6.

Таблица 6.
Пропускная способность интерфейсов внешней шины

Интерфейс	Пропускная способность
ATA-1	4,2 Мбайт/с
ATA-2	13,3 Мбайт/с
ATA-4	16,7 Мбайт/с
ATA-4	25,0 Мбайт/с
ATA-4	33,3 Мбайт/с
ATA-5	44,4 Мбайт/с
ATA-5	66,6 Мбайт/с
ATA-6	100 Мбайт/с

Интерфейс	Пропускная способность
ATA-7	133 Мбайт/с
SATA 1.x	150 Мбайт/с
SATA 2.x	300 Мбайт/с
SATA 3.x	600 Мбайт/с
USB 1.0	1 Мбайт/с
USB 2.0	40 Мбайт/с
USB 3.0	400 Мбайт/с

СПИСОК ЛИТЕРАТУРЫ

1. Таненбаум Э., Остин Т. Архитектура компьютера, 6-е издание. – СПб.: Питер, 2013. – 816 с.
2. Сиротинина Н.Ю., Непомнящий О.В., Коршун К.В., Васильев В.С. Параллельные вычислительные системы: учеб. Пособие. – Красноярск: Сиб. федер. ун-т, 2019. – 178 с.
3. Петров К.С. Радиоматериалы, радиокомпоненты и электроника: Учебное пособие. – СПб.: Питер, 2004. – 522 с.
4. Хайкин, С. Нейронные сети: полный курс, 2-е издание.: Пер. с англ.. – М.: Издательский дом «Вильямс», 2008. – 1104 с.
5. Комарцова Л.Г., Максимов А.В. Нейрокомпьютеры: Учебное пособие для вузов, 2-е изд., перераб. и доп. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2004. – 400 с.
6. Угрюмов Е.П. Цифровая схемотехника: Учебное пособие для вузов, 3-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2010. – 816 с.
7. Лобач В.Т., Потипак М.В. Основы проектирования цифровых устройств радиоэлектронных систем: учебное пособие. – Ростов-на-Дону, Таганрог: Издательство Южного федерального университета, 2020. – 140 с.
8. Харрис Д.Х., Харрис Д.Х. Цифровая схемотехника и архитектура компьютера: RISC-V. – ДМК Пресс, 2022. – 810 с.
9. Глушков В.М. Синтез цифровых автоматов. – М.: Физматгиз, 1962. – 476 с.
10. Гук М. Аппаратные средства IBM PC. Энциклопедия, 3-е изд. – СПб.: Питер, 2008. – 1072 с.
11. Гук М. Процессоры Intel: от 8086 до Pentium П. – СПб. Питер, 1997. – 224с.
12. Бройдо В.Л., Ильина О.П. Архитектура ЭВМ и систем: Учебник для вузов.2-е изд. – СПб.: Питер, 2021. – 720 с.
13. Бройдо В.Л., Ильина О.П. Вычислительные системы, сети и телекоммуникации: Учебник для вузов. 4-е изд. – СПб.: Питер, 2021. – 560 с.

Павел Дмитриевич **Басалин**
Алексей Евгеньевич **Тимофеев**

СХЕМОТЕХНИКА И ОРГАНИЗАЦИЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Учебное пособие

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»

603950, Нижний Новгород, пр. Гагарина, 23.