Министерство образования и науки Российской Федерации

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

В.В. Шаляпин

Основы микропроцессорной техники

Учебное пособие

Санкт-Петербург
Издательство Политехнического университета
2011

УДК 004.4’426(075.8)
ББК 32.973.26-04я73
Ш 189

Шаляпин В. В. Основы микропроцессорной техники: учеб пособие / Шаляпин В. В. / 2011. — 214 с.

Учебное пособие посвящено основам микропроцессорной техники, в котором рассматриваются принципы организации микропроцессорных систем различной сложности, алгоритмы их функционирования, а также методы проектирования устройств на основе микроконтроллеров.

Данное пособие предназначено для студентов обучающихся по направлению «Информатика и вычислительная техника» - 230100, по специальностям: 230102 и 230105.

Оглавление

Введение

Микропроцессорная техника сейчас активно вошла в нашу жизнь. Универсальность, гибкость, простота проектирования аппаратуры, практически неограниченные возможности по усложнению алгоритмов обработки информации - все это обещает микропроцессорной технике большое будущее. Микропроцессоры используются как в бытовых приборах для простейшей обработки сигналов и формирования команд, так и в сложнейших измерительных системах для цифровой обработки сигналов.

Учебное пособие посвящено основам микропроцессорной техники, ее азбуке, базовым понятиям, принципам. От читателя не требуется знаний о микропроцессорах, но необходимы хотя бы начальные знания по цифровой схемотехнике.

Содержащиеся в нем начальные сведения представляют собой тот необходимый минимум знаний, который должен иметь и которым должен свободно и активно пользоваться каждый разработчик микропроцессорных систем. Любые дополнительные знания, конечно же, не повредят, но и заменить то, что изложено здесь, они не смогут.

Достаточно часто препятствием в освоении микропроцессорной техники становится непонимание того, как работает сам микропроцессор или микроконтроллер. В пособии сделана попытка объяснить принципы устройства этих микросхем на рассмотрении примеров упрощенных вариантов внутренней структуры. Только после этого происходит переход к обсуждению особенностей применения реально существующего семейства микросхем.

При подготовке учебного пособия были использованы материалы лекций, читаемых автором в течение ряда лет в Санкт-Петербургском Государственном Политехническом Университете.

1. Основные положения

Микропроцессорная система может рассматриваться как частный случай электронной системы, предназначенной для обработки входных сигналов и выдачи выходных сигналов (рис. 1). В качестве входных и выходных сигналов при этом могут использоваться аналоговые сигналы, одиночные цифровые сигналы, цифровые коды, последовательности цифровых кодов. Внутри системы может производиться хранение, накопление сигналов (или информации), но суть от этого не меняется. Если система цифровая (а микропроцессорные системы относятся к разряду цифровых), то входные аналоговые сигналы преобразуются в последовательности кодов выборок с помощью АЦП, а выходные аналоговые сигналы формируются из последовательности кодов выборок с помощью ЦАП. Обработка и хранение информации производятся в цифровом виде.

Характерная особенность традиционной цифровой системы состоит в том, что алгоритмы обработки и хранения информации в ней жестко связаны со схемотехникой системы. То есть изменение этих алгоритмов возможно только путем изменения структуры системы, замены электронных узлов, входящих в систему, и/или связей между ними. Например, если нам нужна дополнительная операция суммирования, то необходимо добавить в структуру системы лишний сумматор. Или если нужна дополнительная функция хранения кода в течение одного такта, то мы должны добавить в структуру еще один регистр. Естественно, это практически невозможно сделать в процессе эксплуатации, обязательно нужен новый производственный цикл проектирования, изготовления, отладки всей системы. Именно поэтому традиционная цифровая система часто называется системой на «жесткой логике».

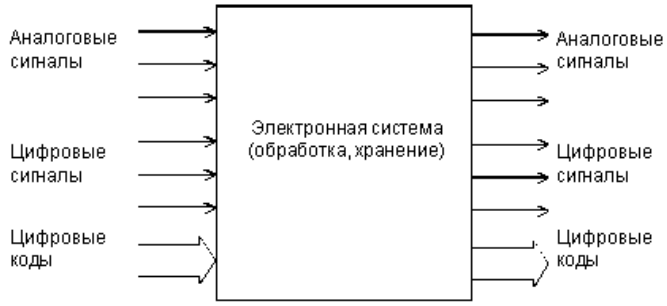


Рис. 1. Электронная система.

Любая система на «жесткой логике» обязательно представляет собой специализированную систему, настроенную исключительно на одну задачу или (реже) на несколько близких, заранее известных задач. Это имеет свои бесспорные преимущества.

Во-первых, специализированная система (в отличие от универсальной) никогда не имеет аппаратурной избыточности, то есть каждый ее элемент обязательно работает в полную силу (конечно, если эта система грамотно спроектирована).

Во-вторых, именно специализированная система может обеспечить максимально высокое быстродействие, так как скорость выполнения алгоритмов обработки информации определяется в ней только быстродействием отдельных логических элементов и выбранной схемой путей прохождения информации. А именно логические элементы всегда обладают максимальным на данный момент быстродействием.

Но в то же время большим недостатком цифровой системы на «жесткой логике» является то, что для каждой новой задачи ее надо проектировать и изготавливать заново. Это процесс длительный, дорогостоящий, требующий высокой квалификации исполнителей. А если решаемая задача вдруг изменяется, то вся аппаратура должна быть полностью заменена. В нашем быстро меняющемся мире это довольно расточительно.

Путь преодоления этого недостатка довольно очевиден: надо построить такую систему, которая могла бы легко адаптироваться под любую задачу, перестраиваться с одного алгоритма работы на другой без изменения

аппаратуры. И задавать тот или иной алгоритм мы тогда могли бы путем ввода в систему некой дополнительной управляющей информации, программы работы системы (рис. 2). Тогда система станет универсальной, или программируемой, не жесткой, а гибкой. Именно это и обеспечивает микропроцессорная система.

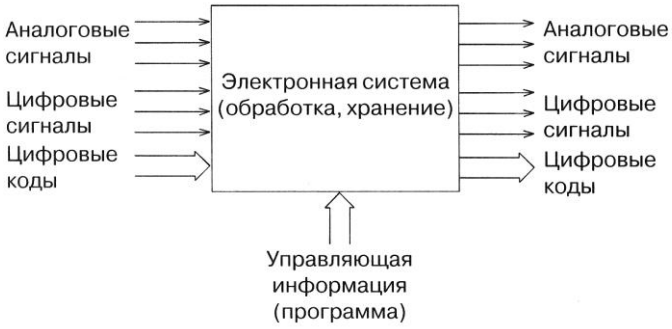


Рис.2. Программируемая (она же универсальная) электронная система.

Но любая универсальность обязательно приводит к избыточности. Ведь решение максимально трудной задачи требует гораздо больше средств, чем решение максимально простой задачи. Поэтому сложность универсальной системы должна быть такой, чтобы обеспечивать решение самой трудной задачи, а при решении простой задачи система будет работать далеко не в полную силу, будет использовать не все свои ресурсы. И чем проще решаемая задача, тем больше избыточность, и тем менее оправданной становится универсальность. Избыточность ведет к увеличению стоимости системы, снижению ее надежности, увеличению потребляемой мощности и т.д. Кроме того, универсальность, как правило, приводит к существенному снижению быстродействия. Оптимизировать универсальную систему так, чтобы каждая новая задача решалась максимально быстро, попросту невозможно. Общее правило таково: чем больше универсальность, гибкость, тем меньше быстродействие. Более того, для универсальных систем не существует таких

задач (пусть даже и самых простых), которые бы они решали с максимально возможным быстродействием. За все приходится платить.

Таким образом, можно сделать следующий вывод. Системы на «жесткой логике» хороши там, где решаемая задача не меняется длительное время, где требуется самое высокое быстродействие, где алгоритмы обработки информации предельно просты. А универсальные, программируемые системы хороши там, где часто меняются решаемые задачи, где высокое быстродействие не слишком важно, где алгоритмы обработки информации сложные. То есть любая система хороша на своем месте.

Однако за последние десятилетия быстродействие универсальных (микропроцессорных) систем сильно выросло (на несколько порядков). К тому же большой объем выпуска микросхем для этих систем привел к резкому снижению их стоимости. В результате область применения систем на «жесткой логике» резко сузилась. Более того, высокими темпами развиваются сейчас программируемые системы, предназначенные для решения одной задачи или нескольких близких задач. Они удачно совмещают в себе как достоинства систем на «жесткой логике», так и программируемых систем, обеспечивая сочетание достаточно высокого быстродействия и необходимой гибкости. Так что вытеснение «жесткой логики» продолжается.

1.1. Определение микропроцессора.

Микропроцессор (МП) - это программно-управляемое электронное цифровое устройство, предназначенное для обработки цифровой информации и управления процессом этой обработки, выполненное на одной или нескольких интегральных схемах с высокой степенью интеграции электронных элементов. Ядром любой микропроцессорной системы является микропроцессор или просто процессор (от английского processor). Перевести на русский язык это слово правильнее всего как «обработчик», так как именно микропроцессор —

это тот узел, блок, который производит всю обработку информации внутри микропроцессорной системы. Остальные узлы выполняют всего лишь вспомогательные функции: хранение информации (в том числе и управляющей информации, то есть программы), связи с внешними устройствами, связи с пользователем и т.д. Процессор заменяет практически всю «жесткую логику», которая понадобилась бы в случае традиционной цифровой системы. Он выполняет арифметические функции (сложение, умножение и т.д.), логические функции (сдвиг, сравнение, маскирование кодов и т.д.), временное хранение кодов (во внутренних регистрах), пересылку кодов между узлами микропроцессорной системы и многое другое. Количество таких элементарных операций, выполняемых процессором, может достигать нескольких сотен. Процессор можно сравнить с мозгом системы. Но при этом надо учитывать, что все свои операции процессор выполняет последовательно, то есть одну за другой, по очереди. Конечно, существуют процессоры с параллельным выполнением некоторых операций, встречаются также микропроцессорные системы, в которых несколько процессоров работают над одной задачей параллельно, но это редкие исключения. С одной стороны, последовательное выполнение операций — несомненное достоинство, так как позволяет с помощью всего одного процессора выполнять любые, самые сложные алгоритмы обработки информации. Но, с другой стороны, последовательное выполнение операций приводит к тому, что время выполнения алгоритма зависит от его сложности. Простые алгоритмы выполняются быстрее сложных. То есть микропроцессорная система способна сделать все, но работает она не слишком быстро, ведь все информационные потоки приходится пропускать через один-единственный узел — микропроцессор (рис. 3). В традиционной цифровой системе можно легко организовать параллельную обработку всех потоков информации, правда, ценой усложнения схемы.

Итак, микропроцессор способен выполнять множество операций. Но откуда он узнает, какую операцию ему надо выполнять в данный момент? Именно это определяется управляющей информацией, программой. Программа представляет собой набор команд (инструкций**)**, то есть цифровых кодов,

Рис. 3. Информационные потоки в микропроцессорной системе.

расшифровав которые, процессор узнает, что ему надо делать. Программа от начала и до конца составляется человеком, программистом, а процессор выступает в роли послушного исполнителя этой программы, никакой инициативы он не проявляет (если, конечно, исправен). Поэтому сравнение процессора с мозгом не слишком корректно. Он всего лишь исполнитель того алгоритма, который заранее составил для него человек. Любое отклонение от этого алгоритма может быть вызвано только неисправностью процессора или каких-нибудь других узлов микропроцессорной системы. Все команды, выполняемые процессором, образуют систему команд процессора.

Логическая структура микропроцессора, т. е. конфигурация составляющих микропроцессор логических схем и связей между ними, определяется функциональным назначением. Именно структура задает состав логических блоков микропроцессора и то, как эти блоки должны быть связаны между собой, чтобы полностью отвечать архитектурным требованиям. Срабатывание электронных блоков микропроцессора в определенной последовательности приводит к выполнению заданных архитектурой микропроцессора функций, т. е. к реализации вычислительных алгоритмов. Одни и те же функции можно

выполнить в микропроцессорах со структурой, отличающейся набором, количеством и порядком срабатывания логических блоков. Различные структуры микропроцессоров, как правило, обеспечивают их различные возможности, в том числе и различную скорость обработки данных. Основные логические блоки микропроцессора показаны на рис. 4.

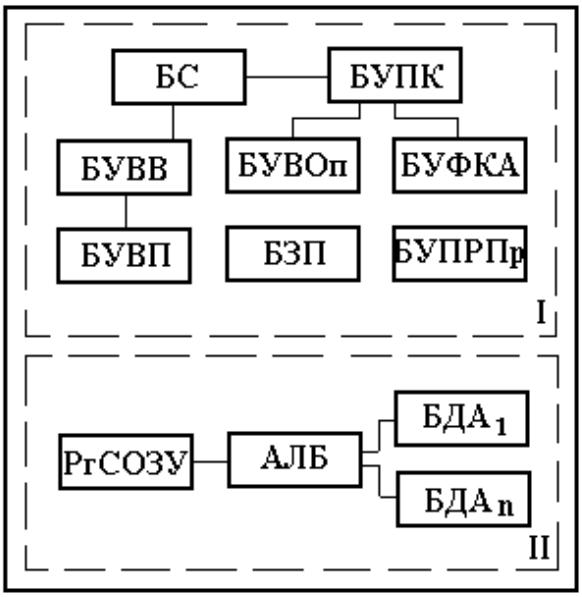


Рис. 1.4. Общая логическая структура микропроцессора: I - управляющая часть, II - операционная часть; БУПК - блок управления последовательностью команд; БУВОп - блок управления выполнением операций; БУФКА - блок управления формированием кодов адресов; БУВП - блок управления виртуальной памятью; БЗП - блок защиты памяти; БУПРПр - блок управления прерыванием работы процессора; БУВВ - блок управления вводом/выводом; РгСОЗУ - регистровое сверхоперативное запоминающее устройство; АЛБ - арифметико-логический блок; БДА - блок дополнительной арифметики; БС - блок синхронизации.

Для разработчика микропроцессорных систем информация о тонкостях внутренней структуры процессора не слишком важна. Разработчик должен

рассматривать процессор как «черный ящик», который в ответ на входные и управляющие коды производит ту или иную операцию и выдает выходные сигналы. Разработчику необходимо знать систему команд, режимы работы процессора, а также правила взаимодействия процессора с внешним миром или, как их еще называют, протоколы обмена информацией. О внутренней структуре процессора надо знать только то, что необходимо для выбора той или иной команды, того или иного режима работы.

Микропроцессор характеризуется большим числом параметров, поскольку он с одной стороны, является процессором, то есть частью компьютера, а с другой – интегральной схемой. Поэтому для микропроцессора важны такие параметры как тип корпуса, количество выводов, мощность рассеивания, уровни сигналов, нагрузочная способность и т. д. Для описания микропроцессора как функционального устройства необходимо задать форматы команд и данных, количество и тип команд, методы адресации данных организацию и емкость стека и т. д.

Создание микропроцессорных устройств систем все в большей степени становится функцией специалистов в конкретной предметной области, а не профессиональных программистов и специалистов по вычислительной технике. Это вызывает большую потребность в инженерных кадрах, которые, кроме своей предметной области, дополнительно разбираются в микропроцессорной технике.

Как известно, МПС состоит из двух специфических частей: аппаратурных средств (АС) и прикладного программного обеспечения (ППО). При этом, рассматривая общий процесс проектирования МПС, можно отметить, что в большинстве случаев доля общей трудоемкости разработки ППО значительно превосходит трудоемкость разработки АС. Указанное обстоятельство объясняется тем, что разработка аппаратурной части МПУ на базе типовых микропроцессорных БИС сводится (чаще всего) к выполнению стандартных

операций в соответствии с рекомендациями, изложенными в технической документации на используемые БИС. Совсем по-другому выглядит инженерный труд при разработке ППО. Проектная работа носит здесь творческий характер, изобилует решениями, имеющими "волевую" окраску, и решениями, продиктованными конъюнктурными соображениями. В силу перечисленных обстоятельств именно при проектировании ППО разработчик сталкивается с наибольшим количеством проблем и от того, как они будут решены, зависит успех разработки МПУ в целом. Таким образом, весьма актуальным представляется ориентация будущих инженеров на более глубокое изучение вопросов, связанных именно с программированием МПС, в частности языков программирования, средств автоматизации программирования (ассемблеров, компиляторов с языков высокого уровня, линкеров и др.), структурных особенностей микропроцессорной элементной базы.

1.2. Принципы построения МП-систем.

При построении МП-систем используются следующие принципы:

1. микропрограммное управление;
2. модульность построения системы;
3. магистральный способ обмена информацией.

Микропрограммное управление. Существуют 2 способа построения устройства управления (УУ) микропроцессора:

а) с использованием комбинационных схем – дешифраторов и программируемых логических матриц;

б) с использованием микропрограммного запоминающего устройства (ПЗУ МК).

В первом случае каждое входное воздействие на УУ (команда) жестко связано с выходными сигналами и их изменения возможны только при изменении электрической схемы УУ. Использование такого метода жестко

фиксирует систему команд, но достигается максимальное быстродействие УУ. Микропроцессоры, использующие комбинационные УУ,
называются микропроцессорами с фиксированным набором команд.

Во втором случае смена выполняемой команды обеспечивается заменой микропрограммы. При считывании каждой микрокоманды требуется обращение к ПЗУ МК, что снижает быстродействие УУ. Микропрограммное управление заменяет аппаратные средства программными и обеспечивает высокую гибкость устройства, но при снижении быстродействия.

Модульность. Принцип модульной организации предполагает построение микропроцессорных устройств (МПУ) и систем на основе модулей. Под модулем понимается конструктивно, функционально и электрически законченное устройство, позволяющее самостоятельно или в совокупности с другими модулями решать задачи заданного класса. При выборе функционального состава МПУ или системы необходимо учитывать многофункциональность (универсальность) и специализацию модулей. Повышение универсальности модулей обеспечивает снижение номенклатуры, снижение затрат на проектирование и изготовление, высокую серийность и, следовательно, низкую стоимость модулей. Специализация модулей является средством достижения соответствия структуры МПУ или системы выполняемым алгоритмам и тем самым повышения быстродействия и эффективности применения МПУ.

Магистральный способ обмена информацией. Среди способов организации связей между модулями МПУ или системы можно выделить два:

а) с помощью произвольных связей, реализующих принцип «каждый с каждым»;

б) с помощью упорядоченных связей (магистралей);

Для достижения максимальной универсальности и упрощения протоколов обмена информацией в микропроцессорных системах применяется так называемая шинная структура связей между отдельными устройствами, входящими в систему. Суть шинной структуры связей сводится к следующему.

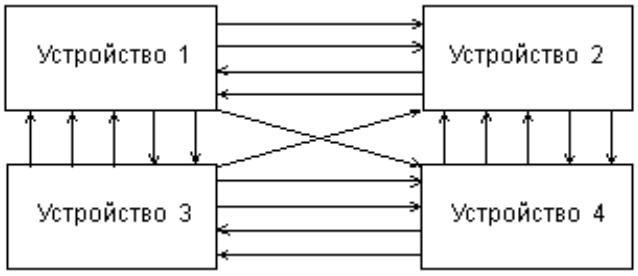


Рис. 5. Классическая структура связей.

При классической структуре связей (рис. 5) все сигналы и коды между устройствами передаются по отдельным линиям связи. Каждое устройство, входящее в систему, передает свои сигналы и коды независимо от других устройств. При этом в системе получается очень много линий связи и разных протоколов обмена информацией. В этом случае необходим значительный объем дополнительной аппаратуры (дешифраторы, коммутаторы), увеличивается сложность отладки системы;

При шинной структуре связей (рис. 6) все сигналы между устройствами передаются по одним и тем же линиям связи, но в разное время (это называется мультиплексированной передачей). Причем передача по всем линиям связи может осуществляться в обоих направлениях (так называемая двунаправленная передача). В результате количество линий связи существенно сокращается, а правила обмена (протоколы) упрощаются. Группа линий связи, по которым передаются сигналы или коды как раз и называется шиной (англ. bus) или магистралью.

Понятно, что при шинной структуре связей легко осуществляется пересылка всех информационных потоков в нужном направлении, например, их можно

пропустить через один процессор, что очень важно для микропроцессорной системы. Однако при шинной структуре связей вся информация передается по линиям связи последовательно во времени, по очереди, что снижает быстродействие системы по сравнению с классической структурой связей.

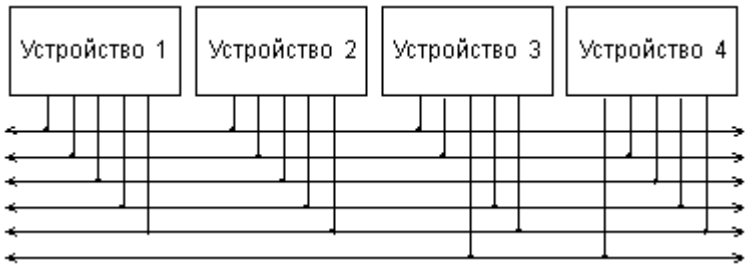


Рис. 6. Шинная структура связей.

Большое достоинство шинной структуры связей состоит в том, что все устройства, подключенные к шине, должны принимать и передавать информацию по одним и тем же правилам (протоколам обмена информацией по шине). Соответственно, все узлы, отвечающие за обмен с шиной в этих устройствах, должны быть единообразны, унифицированы. Магистральный способ позволяет минимизировать количество связей между модулями, обеспечить стандартизацию интерфейсов, сократить число выводов БИС микропроцессора. Существенный недостаток шинной структуры связан с тем, что все устройства подключаются к каждой линии связи параллельно. Поэтому любая неисправность любого устройства может вывести из строя всю систему, если она портит линию связи. По этой же причине отладка системы с шинной структурой связей довольно сложна и обычно требует специального оборудования.

В МПС выделяются следующие магистрали: шина данных, адресная шина, шина управления. При выборе структуры МПС необходимо учитывать, что при уменьшении числа шин увеличивается площадь кристалла или модуля, отводимая под функциональные элементы и тем самым повышаются функциональные возможности МПС. Вместе с тем применение временного

мультиплексирования обмена информацией приводит к снижению быстродействия и необходимости использования дополнительных буферных регистров.

2. Классификация микропроцессоров

1. По назначению различают универсальные, сигнальные, медийные и специализированные микропроцессоры.

Универсальные микропроцессоры предназначаются для применения в вычислительных системах: персональных компьютерах, рабочих станциях, в параллельных суперкомпьютерах.

Цифровые сигнальные микропроцессоры рассчитаны на обработку на обработку в реальном времени цифровых потоков, образованных путем оцифровывания аналоговых сигналов. Современные сигнальные микро-процессоры способны проводить вычисления с плавающей точкой над 32-40-разрядными операндами.

Медийные микропроцессоры представляют собой законченные системы для обработки аудио- и видеоинформации.

Специализированные микропроцессоры предназначаются для решения определенного класса задач или одной конкретной задачи. Среди специализированных микропроцессоров можно выделить микроконтроллеры, ориентированные на выполнение сложных последовательностей логических операций, математические МП, предназначенные для повышения производительности при выполнении арифметических операций за счет, например, матричных методов их выполнения, МП для обработки данных в различных областях применений. С помощью специализированных МП можно эффективно решать сложные задачи параллельной обработки данных.

2. По составу команд различают RISC- и CISC- процессоры.

Анализ кода программ, генерируемого компиляторами языков высокого уровня, показал, что практически используется только ограниченный набор простых команд форматов «регистр↔регистр, регистр↔память». Компиляторы не в состоянии эффективно использовать сложные команды. Это способствовало формированию концепции процессоров с сокращенным набором команд(RISC-процессоров). RISC- процессоры эффективны в тех областях применения, в которых можно продуктивно использовать структурные способы уменьшения времени доступа к оперативной памяти. Если программа генерирует произвольные последовательности адресов обращения к памяти и каждая единица данных используется только для выполнения одной команды, то фактически производительность процессора определяется временем обращения к основной памяти. В этом случае использование сокращенного набора команд только ухудшает эффективность, так как требует пересылки операндов между памятью и регистром вместо выполнения команд «память, память- память».

Процессоры с традиционными наборами команд называют CISC-процессорами с полными наборами команд. Как правило, в этих процессорах команды имеют много разных форматов и требуют для своего представления различного числа ячеек памяти. Это обуславливает определение типа команды в ходе ее дешифрации при исполнении, что усложняет устройство управления процессора и препятствует повышению тактовой частоты до уровня, достижимого в RISC-процессорах на той же элементной базе.

3. По числу больших интегральных схем (БИС) в микропроцессорном комплекте различают микропроцессоры однокристальные, многокристальные и многокристальные секционные.

Однокристальные микропроцессоры получаются при реализации всех аппаратных средств процессора в виде одной БИС или СБИС (сверхбольшой интегральной схемы). По мере увеличения степени интеграции элементов в

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

Разбиение логической структуры можно выполнить двумя способами. Первый способ состоит в разделении логической структуры на функционально законченные части, например, операционное устройство (ОУ), память и устройство управления (УУ), как было сделано микропроцессоре i432(рис. 7, а)). Второй способ состоит в размещении на кристалле всех функциональных блоков (ОУ, память, УУ), но небольшой разрядности, например 2 или 4 разряда. На рис. 7, б) данный способ иллюстрируется вертикальными штриховыми линиями. Каждая микросхема дополняется входами/выходами для соединения друг с другом. В результате рассмотренного функционального разделения структуры микропроцессора на функционально и конструктивно законченные части создаются условия реализации каждой из них в виде БИС. Все они образуют комплект секционных БИС МП. Секционность БИС МП определяет возможность "наращивания" разрядности обрабатываемых данных или усложнения устройств управления микропроцессора при "параллельном" включении большего числа БИС.

Однокристальные и трехкристальные БИС МП, как правило, изготовляют на основе микроэлектронных технологий униполярных полупроводниковых приборов, а многокристальные секционные БИС МП на основе технологии биполярных полупроводниковых приборов. Использование многокристальных микропроцессорных высокоскоростных биполярных БИС, имеющих

функциональную законченность при малой физической разрядности обрабатываемых данных и монтируемых в корпус с большим числом выводов,

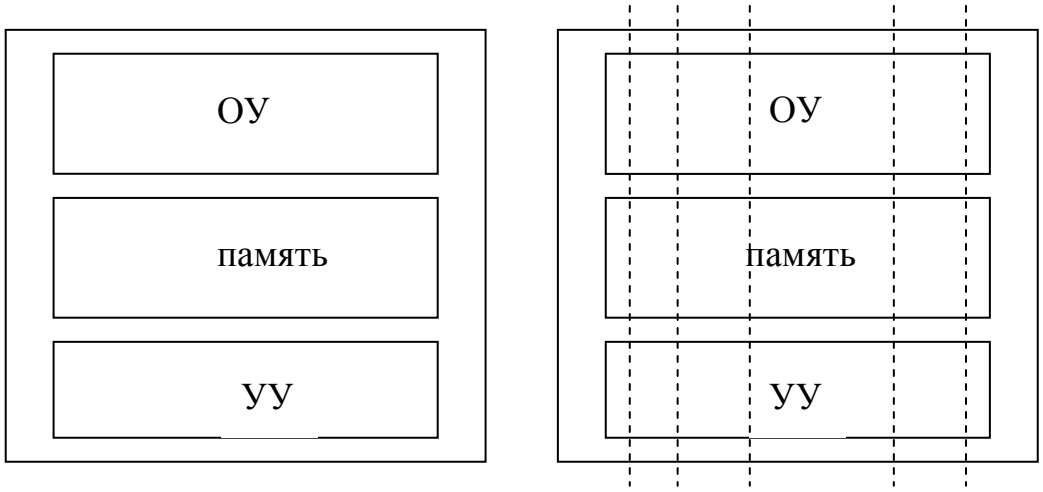


Рис. 7. Способы разделения логической структуры микропроцессора

позволяет организовать разветвление связи в процессоре, а также осуществить конвейерные принципы обработки информации для повышения его производительности.

4. По виду обрабатываемых входных сигналов различают цифровые и аналоговые микропроцессоры. Сами микропроцессоры цифровые устройства, могут иметь встроенные аналого-цифровые и цифро-аналоговые преобразователи. Поэтому входные аналоговые сигналы передаются в МП через преобразователь в цифровой форме, обрабатываются. После обратного преобразования в аналоговую форму поступают на выход. С архитектурной точки зрения такие микропроцессоры представляют собой аналоговые функциональные преобразователи сигналов и называются аналоговыми микропроцессорами. Они выполняют функции любой аналоговой схемы (например, производят генерацию колебаний, модуляцию, смещение, фильтрацию, кодирование и декодирование сигналов в реальном масштабе

времени и т.д., заменяя сложные схемы, состоящие из операционных усилителей, катушек индуктивности, конденсаторов и т.д.). При этом применение аналогового микропроцессора значительно повышает точность обработки аналоговых сигналов и их воспроизводимость, а также расширяет функциональные возможности за счет программной "настройки" цифровой части микропроцессора на различные алгоритмы обработки сигналов.

Обычно в составе однокристальных аналоговых МП имеется несколько каналов аналого-цифрового и цифро-аналогового преобразования. В аналоговом микропроцессоре разрядность обрабатываемых данных достигает 24 бит и более, большое значение уделяется увеличению скорости выполнения арифметических операций. Отличительная черта аналоговых микропроцессоров - способность к переработке большого объема числовых данных, т. е. к выполнению операций сложения и умножения с большой скоростью при необходимости даже за счет отказа от операций прерываний и переходов. Аналоговый сигнал, преобразованный в цифровую форму, обрабатывается в реальном масштабе времени и передается на выход обычно в аналоговой форме через цифро-аналоговый преобразователь. Производительность аналогового микропроцессора определяется его способностью быстро выполнять операции умножения. Одним из направлений дальнейшего совершенствования аналоговых микропроцессоров является повышение их универсальности и гибкости. Поэтому вместе с повышением скорости обработки большого объема цифровых данных будут развиваться средства обеспечения развитых вычислительных процессов обработки цифровой информации за счет реализации аппаратных блоков прерывания программ и программных переходов.

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

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

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

6. По организации структуры микропроцессорных систем различают системы с одной, двумя и тремя шинами.

7. По числу разрядов микропроцессоры делят на 8-ми, 16-ти и 32-х разрядные.

3. Структурная схема МПС

На рис. 8 приведена общая структурная схема микропроцессорной системы.

Все устройства микропроцессорной системы объединяются общей системной шиной (она же называется еще системной магистралью или каналом).

Системная магистраль включает в себя три основные шины:

---шина адреса (Address Bus);

---шина данных (Data Bus);

---шина управления (Control Bus).

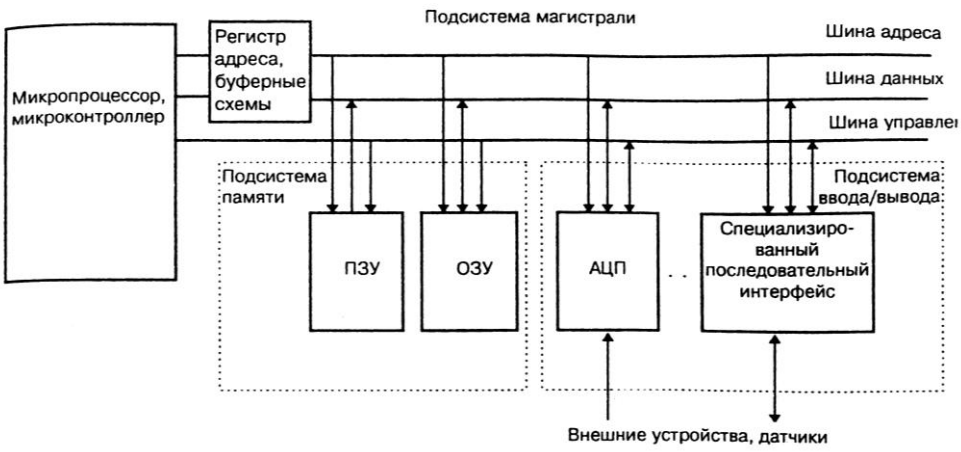


Рис.8. Структурная схема микропроцессорной системы.

Шина данных — это основная шина, которая используется для передачи информационных кодов между всеми устройствами микропроцессорной системы. Обычно в пересылке информации участвует процессор, который передает код данных в какое-то устройство или в ячейку памяти или же принимает код данных из какого-то устройства или из ячейки памяти. Но возможна также и передача информации между устройствами без участия процессора. Количество ее разрядов (линий связи) определяет скорость и эффективность информационного обмена, а также максимально возможное количество команд.

Шина данных всегда двунаправленная, так как предполагает передачу информации в обоих направлениях. Наиболее часто встречающийся тип выходного каскада для линий этой шины — выход с тремя состояниями.

Обычно шина данных имеет 8, 16, 32 или 64 разряда. За один цикл обмена по 64-разрядной шине может передаваться 8 байт информации, а по 8-разрядной — только один байт. Разрядность шины данных определяет и разрядность всей магистрали. Например, когда говорят о 32-разрядной системной магистрали, подразумевается, что она имеет 32-разрядную шину данных.

Шина адреса служит для определения адреса (номера) устройства, с которым процессор обменивается информацией в данный момент. Каждому устройству (кроме процессора), каждой ячейке памяти в микропроцессорной системе присваивается собственный адрес. Когда код какого-то адреса выставляется процессором на шине адреса, устройство, которому этот адрес приписан, понимает, что ему предстоит обмен информацией.

Шина адреса вторая по важности шина, которая определяет максимально возможную сложность микропроцессорной системы, то есть допустимый объем памяти и, следовательно, максимально возможный размер программы и максимально возможный объем запоминаемых данных. Количество адресов, обеспечиваемых шиной адреса, определяется как 2^N, где N — количество разрядов. Например, 16-разрядная шина адреса обеспечивает 65 536 адресов. Разрядность шины адреса обычно кратна 4 и может достигать 32 и даже 64. Шина адреса может быть однонаправленной (когда магистралью всегда управляет только процессор) или двунаправленной (когда процессор может временно передавать управление магистралью другому устройству, например контроллеру ПДП). Как в шине данных, так и в шине адреса может использоваться положительная логика или отрицательная логика. При положительной логике высокий уровень напряжения соответствует логической единице на соответствующей линии связи, низкий — логическому нулю. При отрицательной логике наоборот. В большинстве случаев уровни сигналов на шинах — ТТЛ.

Для снижения общего количества линий связи магистрали часто применяется мультиплексирование шин адреса и данных. То есть одни и те же линии связи используются в разные моменты времени для передачи как адреса, так и данных (в начале цикла — адрес, в конце цикла — данные). Для фиксации этих моментов (стробирования) служат специальные сигналы на шине управления. Понятно, что мультиплексированная шина адреса/данных обеспечивает

меньшую скорость обмена, требует более длительного цикла обмена (рис. 9). По типу шины адреса и шины данных все магистрали также делятся на мультиплексированные и немультиплексированные.

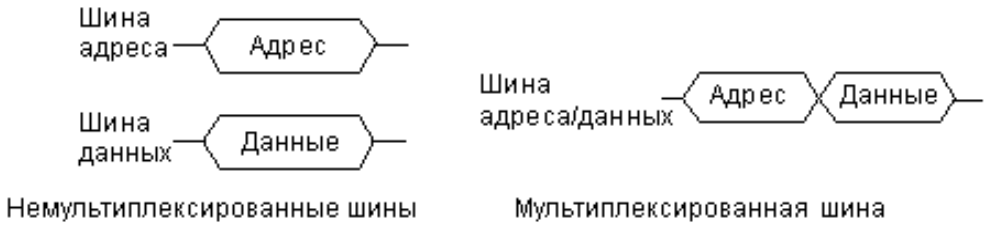


Рис. 9. Мультиплексирование шин адреса и данных.

В некоторых мультиплексированных магистралях после одного кода адреса передается несколько кодов данных (массив данных). Это позволяет существенно повысить быстродействие магистрали. Иногда в магистралях применяется частичное мультиплексирование, то есть часть разрядов данных передается по немультиплексированным линиям, а другая часть — по мультиплексированным с адресом линиям.

Шина управления в отличие от шины адреса и шины данных состоит из отдельных управляющих сигналов. Каждый из этих сигналов во время обмена информацией имеет свою функцию. Некоторые сигналы служат для стробирования передаваемых или принимаемых данных (то есть определяют моменты времени, когда информационный код выставлен на шину данных). Другие управляющие сигналы могут использоваться для подтверждения приема данных, для сброса всех устройств в исходное состояние, для тактирования всех устройств и т.д. Линии шины управления могут быть однонаправленными или двунаправленными.

Шина управления — это вспомогательная шина, управляющие сигналы на которой определяют тип текущего цикла и фиксируют моменты времени, соответствующие разным частям или стадиям цикла. Кроме того, управляющие сигналы обеспечивают согласование работы процессора (или другого хозяина магистрали, задатчика, master) с работой памяти или устройства ввода/вывода

(устройства-исполнителя, slave). Управляющие сигналы также обслуживают запрос и предоставление прерываний, запрос и предоставление прямого доступа.

Сигналы шины управления могут передаваться как в положительной логике (реже), так и в отрицательной логике (чаще). Линии шины управления могут быть как однонаправленными, так и двунаправленными. Типы выходных каскадов могут быть самыми разными: с двумя состояниями (для однонаправленных линий), с тремя состояниями (для двунаправленных линий), с открытым коллектором (для двунаправленных и мультиплексированных линий).

Самые главные управляющие сигналы — это стробы обмена, то есть сигналы, формируемые процессором и определяющие моменты времени, в которые производится пересылка данных по шине данных, обмен данными. Чаще всего в магистрали используются два различных строба обмена:

Строб записи (вывода), определяет момент времени, когда устройство-исполнитель может принимать данные, выставленные процессором на шину данных. Строб чтения (ввода), определяет момент времени, когда устройство-исполнитель должно выдать на шину данных код данных, который будет прочитан процессором. При этом большое значение имеет то, как процессор заканчивает обмен в пределах цикла, в какой момент он снимает свой строб обмена. Возможны два пути решения (рис. 10):

--- синхронный обмен;

---асинхронный обмен.

При синхронном обмене процессор заканчивает обмен данными самостоятельно, через раз и навсегда установленный временной интервал выдержки ($t_{выд}$), то есть без учета интересов устройства-исполнителя;

При асинхронном обмене процессор заканчивает обмен только тогда, когда устройство-исполнитель подтверждает выполнение операции специальным сигналом (так называемый режим handshake — рукопожатие).

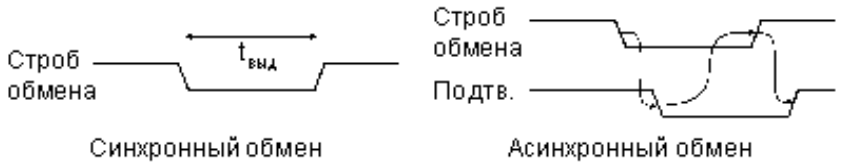


Рис. 10. Синхронный обмен и асинхронный обмен.

Достоинства синхронного обмена — более простой протокол обмена, меньшее количество управляющих сигналов. Недостатки — отсутствие гарантии, что исполнитель выполнил требуемую операцию, а также высокие требования к быстродействию исполнителя.

Достоинства асинхронного обмена — более надежная пересылка данных, возможность работы с самыми разными по быстродействию исполнителями. Недостаток — необходимость формирования сигнала подтверждения всеми исполнителями, то есть дополнительные аппаратурные затраты.

Какой тип обмена быстрее, синхронный или асинхронный? Ответ на этот вопрос неоднозначен. С одной стороны, при асинхронном обмене требуется какое-то время на выработку, передачу дополнительного сигнала и на его обработку процессором. С другой стороны, при синхронном обмене приходится искусственно увеличивать длительность строба обмена для соответствия требованиям большего числа исполнителей, чтобы они успевали обмениваться информацией в темпе процессора. Поэтому иногда в магистрали предусматривают возможность как синхронного, так и асинхронного обмена, причем синхронный обмен является основным и довольно быстрым, а асинхронный применяется только для медленных исполнителей.

По используемому типу обмена магистрали микропроцессорных систем также делятся на синхронные и асинхронные.

Если в микропроцессорную систему надо ввести входной код (или входной сигнал), то процессор по шине адреса обращается к нужному устройству ввода/вывода и принимает по шине данных входную информацию. Если из микропроцессорной системы надо вывести выходной код (или выходной сигнал), то процессор обращается по шине адреса к нужному устройству ввода/вывода и передает ему по шине данных выходную информацию.

Если информация должна пройти сложную многоступенчатую обработку, то процессор может хранить промежуточные результаты в системной оперативной памяти. Для обращения к любой ячейке памяти процессор выставляет ее адрес на шину адреса и передает в нее информационный код по шине данных или же принимает из нее информационный код по шине данных. В памяти (оперативной и постоянной) находятся также и управляющие коды (команды выполняемой процессором программы), которые процессор также читает по шине данных с адресацией по шине адреса. Постоянная память используется в основном для хранения программы начального пуска микропроцессорной системы, которая выполняется каждый раз после включения питания. Информация в нее заносится изготовителем раз и навсегда.

Таким образом, в микропроцессорной системе все информационные коды и коды команд передаются по шинам последовательно, по очереди. Это определяет сравнительно невысокое быстродействие микропроцессорной системы. Оно ограничено обычно даже не быстродействием процессора (которое тоже очень важно) и не скоростью обмена по системной шине (магистрали), а именно последовательным характером передачи информации по системной шине (магистрали).

Важно учитывать, что устройства ввода/вывода чаще всего представляют собой устройства на «жесткой логике». На них может быть возложена часть функций, выполняемых микропроцессорной системой. Поэтому у разработчика всегда

имеется возможность перераспределять функции системы между аппаратной и программной реализациями оптимальным образом. Аппаратная реализация ускоряет выполнение функции, но имеет недостаточную гибкость. Программная реализация значительно медленнее, но обеспечивает высокую гибкость. Аппаратная реализация функций увеличивает стоимость системы и ее энергопотребление, программная — не увеличивает. Чаще всего применяется комбинирование аппаратных и программных функций. Иногда устройства ввода/вывода имеют в своем составе процессор, то есть представляют собой небольшую специализированную микропроцессорную систему. Это позволяет переложить часть программных функций на устройства ввода/вывода, разгрузив центральный процессор системы.

В системах с шинной структурой связей применяют все три существующие разновидности выходных каскадов цифровых микросхем:

---стандартный выход или выход с двумя состояниями (обозначается 2С, 2S, реже ТТЛ, TTL);

---выход с открытым коллектором (обозначается ОК, ОС);

---выход с тремя состояниями или (что, то же самое) с возможностью отключения (обозначается 3С, 3S).

Упрощенно эти три типа выходных каскадов могут быть представлены в виде схем на рис. 11.

У выхода 2С два ключа замыкаются по очереди, что соответствует уровням логической единицы (верхний ключ замкнут) и логического нуля (нижний ключ замкнут). У выхода ОК замкнутый ключ формирует уровень логического нуля, разомкнутый — логической единицы. У выхода 3С ключи могут замыкаться по очереди (как в случае 2С), а могут размыкаться одновременно, образуя третье, высокоимпедансное, состояние. Переход в третье состояние (Z-состояние) управляется сигналом на специальном входе EZ.

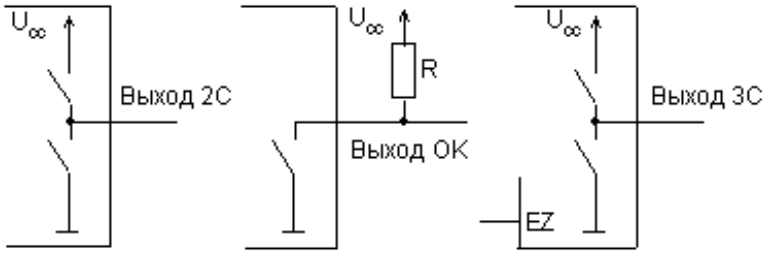


Рис. 11. Три типа выходов цифровых микросхем.

Выходные каскады типов 3С и ОК позволяют объединять несколько выходов микросхем для получения мультиплексированных (рис. 12) или двунаправленных (рис. 13) линий.

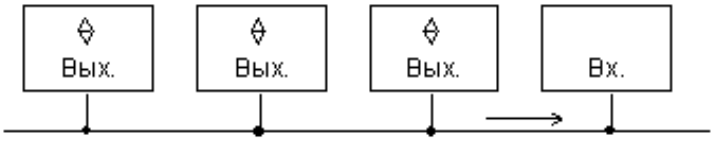


Рис. 12. Мультиплексированная линия.

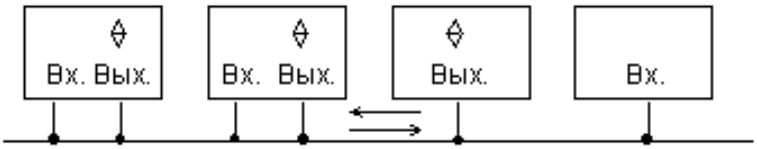


Рис. 13. Двунаправленная линия.

При этом в случае выходов 3С необходимо обеспечить, чтобы на линии всегда работал только один активный выход, а все остальные выходы находились бы в это время в третьем состоянии, иначе возможны конфликты. Объединенные выходы ОК могут работать все одновременно, без всяких конфликтов.
На рис.14 показано подключение ОЗУ, ПЗУ и порта ввода/вывода к системной магистрали.

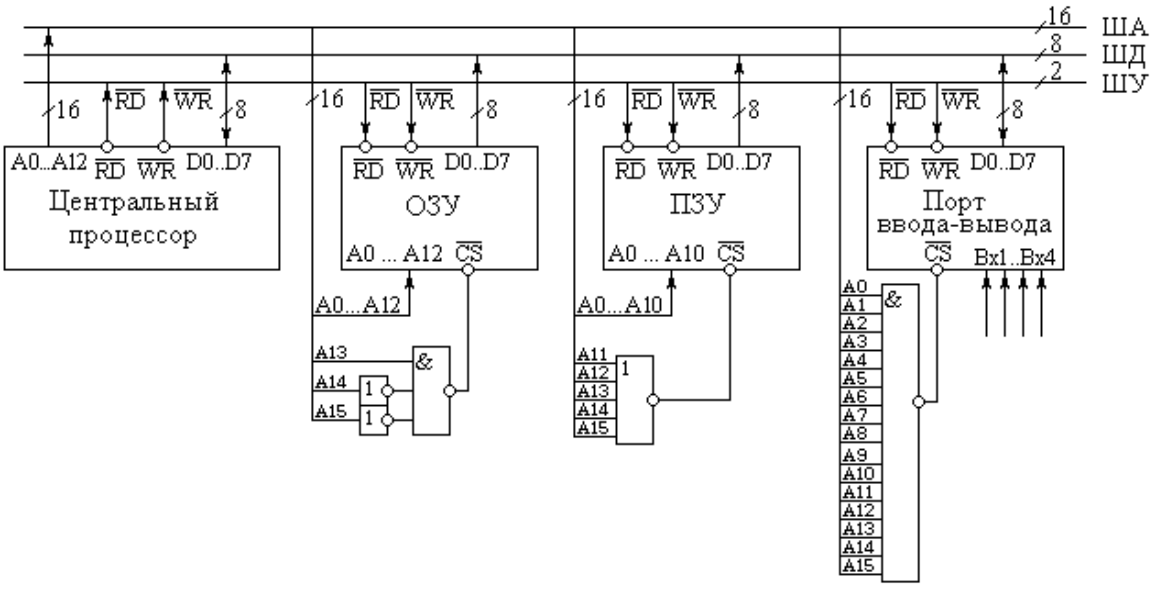


Рис. 14. Структурная схема подключения устройств микропроцессорной системы к магистрали**.**

3.1. Архитектура микропроцессорных систем

Существует два основных вида микропроцессорных систем: фон-неймановская и гарвардская.

Основной особенностью фон-неймановской архитектуры является использование общей памяти для хранения программ и данных, как показано на рис. 15 (трехшинный вариант).

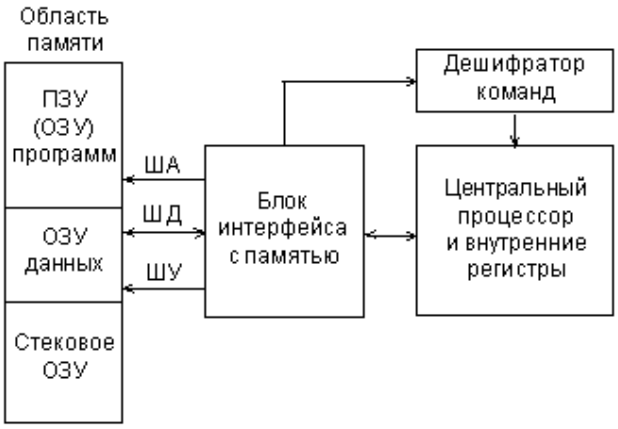


Рис. 15. Структура МПС с фон-неймановской архитектурой.

Основное преимущество архитектуры Фон-Неймана – упрощение устройства МПС, так как реализуется обращение только к одной общей памяти. Кроме того, использование единой области памяти позволило оперативно перераспределять ресурсы между областями программ и данных, что существенно повысило гибкость МПС с точки зрения разработчика программного обеспечения. Размещение стека в общей памяти облегчило доступ к его содержимому. Неслучайно поэтому фон-неймановская архитектура стала основной архитектурой универсальных компьютеров, включая персональные.

Основной особенностью гарвардской архитектуры является использование раздельных адресных пространств для хранения команд и данных, как показано на рис. 16.

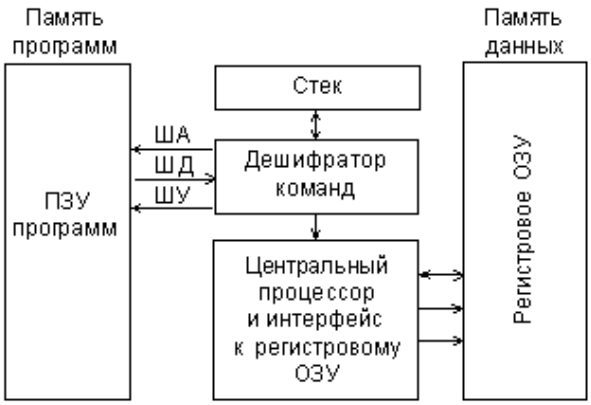


Рис. 16. Структура МПС с гарвардской архитектурой.

Гарвардская архитектура почти не использовалась до конца 70-х годов, пока производители МК не поняли, что она дает определенные преимущества разработчикам автономных систем управления. Дело в том, что, судя по опыту использования МПС для управления различными объектами, для реализации большинства алгоритмов управления такие преимущества фон-неймановской архитектуры как гибкость и универсальность не имеют большого значения. Анализ реальных программ управления показал, что необходимый объем

памяти данных МК, используемый для хранения промежуточных результатов, как правило, на порядок меньше требуемого объема памяти программ. В этих условиях использование единого адресного пространства приводило к увеличению формата команд за счет увеличения числа разрядов для адресации операндов. Применение отдельной небольшой по объему памяти данных способствовало сокращению длины команд и ускорению поиска информации в памяти данных.

Кроме того, гарвардская архитектура обеспечивает потенциально более высокую скорость выполнения программы по сравнению с фон-неймановской за счет возможности реализации параллельных операций. Выборка следующей команды может происходить одновременно с выполнением предыдущей, и нет необходимости останавливать процессор на время выборки команды. Этот метод реализации операций позволяет обеспечивать выполнение различных команд за одинаковое число тактов, что дает возможность более просто определить время выполнения циклов и критичных участков программы.

Большинство производителей современных 8-разрядных МК используют гарвардскую архитектуру. Однако гарвардская архитектура является недостаточно гибкой для реализации некоторых программных процедур. Поэтому сравнение МК, выполненных по разным архитектурам, следует проводить применительно к конкретному приложению.

Фон-неймановская архитектура с общей, единой шиной для данных и команд (одношинную, или принстонская, архитектура) приведена на рис. 17.

Рис. 17. Архитектура с общей шиной данных и команд.

Соответственно, в составе системы в этом случае присутствует одна общая память, как для данных, так и для команд.

Но существует также тип архитектуры микропроцессорной системы — это архитектура с раздельными шинами данных и команд (двухшинная, или гарвардская, архитектура). Эта архитектура предполагает наличие в системе отдельной памяти для данных и отдельной памяти для команд (рис. 18). Обмен процессора с каждым из двух типов памяти происходит по своей шине.

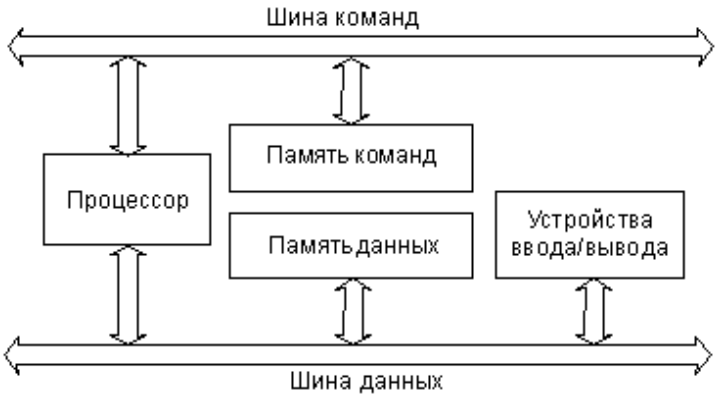


Рис. 18. Архитектура с раздельными шинами данных и команд.

Архитектура с общей шиной распространена гораздо больше, она применяется, например, в персональных компьютерах и в сложных микрокомпьютерах. Архитектура с раздельными шинами применяется в основном в однокристальных микроконтроллерах.

Рассмотрим некоторые достоинства и недостатки обоих архитектурных решений.

Архитектура с общей шиной (принстонская, фон-неймановская) проще, она не требует от процессора одновременного обслуживания двух шин, контроля обмена по двум шинам сразу. Наличие единой памяти данных и команд позволяет гибко распределять ее объем между кодами данных и команд. Например, в некоторых случаях нужна большая и сложная программа, а данных в памяти надо хранить не слишком много. В других случаях, наоборот, программа требуется простая, но необходимы большие объемы хранимых

данных. Перераспределение памяти не вызывает никаких проблем, главное — чтобы программа и данные вместе помещались в памяти системы. Как правило, в системах с такой архитектурой память бывает довольно большого объема (до десятков и сотен мегабайт). Это позволяет решать самые сложные задачи.

Архитектура с раздельными шинами данных и команд сложнее, она заставляет процессор одновременно с двумя потоками кодов, обслуживать обмен по двум шинам одновременно. Программа может размещаться только в памяти команд, данные — только в памяти данных. Такая узкая специализация ограничивает круг задач, решаемых системой, так как не дает возможности гибкого перераспределения памяти. Память данных и память команд в этом случае имеют не слишком большой объем, поэтому применение систем с данной архитектурой ограничивается обычно не слишком сложными задачами.

В чем же преимущество архитектуры с двумя шинами (гарвардской)? В первую очередь, в быстродействии. Дело в том, что при единственной шине команд и данных процессор вынужден по одной этой шине принимать данные (из памяти или устройства ввода/вывода) и передавать данные (в память или в устройство ввода/вывода), а также читать команды из памяти. Естественно, одновременно эти пересылки кодов по магистрали происходить не могут, они должны производиться по очереди. Современные процессоры способны совместить во времени выполнение команд и проведение циклов обмена по системной шине. Использование конвейерных технологий и быстрой кэш-памяти позволяет им ускорить процесс взаимодействия со сравнительно медленной системной памятью. Повышение тактовой частоты и совершенствование структуры процессоров дают возможность сократить время выполнения команд. Но дальнейшее увеличение быстродействия системы возможно только при совмещении пересылки данных и чтения команд, то есть при переходе к архитектуре с двумя шинами.

В случае двухшинной архитектуры обмен по обеим шинам может быть независимым, параллельным во времени. Соответственно, структуры шин (количество разрядов кода адреса и кода данных, порядок и скорость обмена информацией и т.д.) могут быть выбраны оптимально для той задачи, которая решается каждой шиной. Поэтому при прочих равных условиях переход на двухшинную архитектуру ускоряет работу микропроцессорной системы, хотя и требует дополнительных затрат на аппаратуру, усложнения структуры процессора. Память данных в этом случае имеет свое распределение адресов, а память команд — свое.

Проще всего преимущества двухшинной архитектуры реализуются внутри одной микросхемы. В этом случае можно также существенно уменьшить влияние недостатков этой архитектуры. Поэтому основное ее применение — в микроконтроллерах, от которых не требуется решения слишком сложных задач, но зато необходимо максимальное быстродействие при заданной тактовой частоте.

3.2. Типы микропроцессорных систем

Диапазон применения микропроцессорной техники сейчас очень широк, требования к микропроцессорным системам предъявляются самые разные. Поэтому сформировалось несколько типов микропроцессорных систем, различающихся мощностью, универсальностью, быстродействием и структурными отличиями. Основные типы следующие:

---микроконтроллеры — наиболее простой тип микропроцессорных систем, в которых все или большинство узлов системы выполнены в виде одной микросхемы;

---контроллеры — управляющие микропроцессорные системы, выполненные в виде отдельных модулей;

---микрокомпьютеры — более мощные микропроцессорные системы с развитыми средствами сопряжения с внешними устройствами.

---компьютеры (в том числе персональные) — самые мощные и наиболее универсальные микропроцессорные системы.

Четкую границу между этими типами иногда провести довольно сложно. Быстродействие всех типов микропроцессоров постоянно растет, и нередки ситуации, когда новый микроконтроллер оказывается быстрее, например, устаревшего персонального компьютера. Но кое-какие принципиальные отличия все-таки имеются.

Микроконтроллеры представляют собой универсальные устройства, которые практически всегда используются не сами по себе, а в составе более сложных устройств, в том числе и контроллеров. Системная шина микроконтроллера скрыта от пользователя внутри микросхемы. Возможности подключения внешних устройств к микроконтроллеру ограничены. Устройства на микроконтроллерах обычно предназначены для решения одной задачи.

Контроллеры, как правило, создаются для решения какой-то отдельной задачи или группы близких задач. Они обычно не имеют возможностей подключения дополнительных узлов и устройств, например, большой памяти, средств ввода/вывода. Их системная шина чаще всего недоступна пользователю. Структура контроллера проста и оптимизирована под максимальное быстродействие. В большинстве случаев выполняемые программы хранятся в постоянной памяти и не меняются. Конструктивно контроллеры выпускаются в одноплатном варианте.

Микрокомпьютеры отличаются от контроллеров более открытой структурой, они допускают подключение к системной шине нескольких дополнительных устройств. Производятся микрокомпьютеры в каркасе, корпусе с разъемами системной магистрали, доступными пользователю. Микрокомпьютеры могут иметь средства хранения информации на магнитных носителях (например,

магнитные диски) и довольно развитые средства связи с пользователем (видеомонитор, клавиатура). Микрокомпьютеры рассчитаны на широкий круг задач, но в отличие от контроллеров, к каждой новой задаче его надо приспосабливать заново. Выполняемые микрокомпьютером программы можно легко менять.

Наконец, компьютеры и самые распространенные из них — персональные компьютеры — это самые универсальные из микропроцессорных систем. Они обязательно предусматривают возможность модернизации, а также широкие возможности подключения новых устройств. Их системная шина, конечно, доступна пользователю. Кроме того, внешние устройства могут подключаться к компьютеру через несколько встроенных портов связи (количество портов доходит иногда до 10). Компьютер всегда имеет сильно развитые средства связи с пользователем, средства длительного хранения информации большого объема, средства связи с другими компьютерами по информационным сетям. Области применения компьютеров могут быть самыми разными: математические расчеты, обслуживание доступа к базам данных, управление работой сложных электронных систем, компьютерные игры, подготовка документов и т.д.

Любую задачу в принципе можно выполнить с помощью каждого из перечисленных типов микропроцессорных систем. Но при выборе типа надо по возможности избегать избыточности и предусматривать необходимую для данной задачи гибкость системы.

В настоящее время при разработке новых микропроцессорных систем чаще всего выбирают путь использования микроконтроллеров (примерно в 80% случаев). При этом микроконтроллеры применяются или самостоятельно, в микропроцессорных комплектах выпускаются сейчас довольно редко, с минимальной дополнительной аппаратурой, или в составе более сложных контроллеров с развитыми средствами ввода/вывода.

Классические микропроцессорные системы на базе микросхем процессоров и первую очередь, из-за сложности процесса разработки и отладки этих систем. Данный тип микропроцессорных систем выбирают в основном тогда, когда микроконтроллеры не могут обеспечить требуемых характеристик.

Наконец, заметное место занимают сейчас микропроцессорные системы на основе персонального компьютера. Разработчику в этом случае нужно только оснастить персональный компьютер дополнительными устройствами сопряжения, а ядро микропроцессорной системы уже готово. Персональный компьютер имеет развитые средства программирования, что существенно упрощает задачу разработчика. К тому же он может обеспечить самые сложные алгоритмы обработки информации. Основные недостатки персонального компьютера — большие размеры корпуса и аппаратурная избыточность для простых задач. Недостатком является и неприспособленность большинства персональных компьютеров к работе в сложных условиях (запыленность, высокая влажность, вибрации, высокие температуры и т.д.). Однако выпускаются и специальные персональные компьютеры, приспособленные к различным условиям эксплуатации.

4. Функционирование процессора

4.1. Функции процессора

Процессор (рис.19) обычно представляет собой отдельную микросхему или же часть микросхемы (в случае микроконтроллера). В прежние годы процессор иногда выполнялся на комплектах из нескольких микросхем, но сейчас от такого подхода уже практически отказались. Микросхема процессора обязательно имеет выводы трех шин: шины адреса, шины данных и шины

управления. Иногда некоторые сигналы и шины мультиплексируются, чтобы уменьшить количество выводов микросхемы процессора.

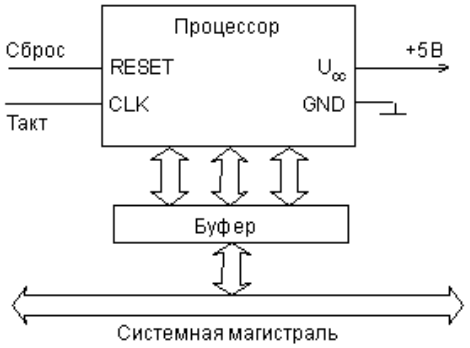


Рис. 19. Схема включения процессора.

Важнейшие характеристики процессора — это количество разрядов его шины данных, количество разрядов его шины адреса и количество управляющих сигналов в шине управления. Разрядность шины данных определяет скорость работы системы. Разрядность шины адреса определяет допустимую сложность системы. Количество линий управления определяет разнообразие режимов обмена и эффективность обмена процессора с другими устройствами системы.

Кроме выводов для сигналов трех основных шин процессор всегда имеет вывод (или два вывода) для подключения внешнего тактового сигнала или кварцевого резонатора (CLK), так как процессор всегда представляет собой тактируемое устройство. Чем больше тактовая частота процессора, тем он быстрее работает, то есть тем быстрее выполняет команды. Впрочем, быстродействие процессора определяется не только тактовой частотой, но и особенностями его структуры. Современные процессоры выполняют большинство команд за один такт и имеют средства для параллельного выполнения нескольких команд. Тактовая частота процессора не связана прямо и жестко со скоростью обмена по магистрали, так как скорость обмена по магистрали ограничена задержками распространения сигналов и искажениями сигналов на магистрали. То есть

тактовая частота процессора определяет только его внутреннее быстродействие, а не внешнее. Иногда тактовая частота процессора имеет нижний и верхний пределы. При превышении верхнего предела частоты возможно перегревание процессора, а также сбои, причем, что самое неприятное, возникающие не всегда и нерегулярно. Так что с изменением этой частоты надо быть очень осторожным.

Еще один важный сигнал, который имеется в каждом процессоре, — это сигнал начального сброса RESET. При включении питания, при аварийной ситуации или зависании процессора подача этого сигнала приводит к инициализации процессора, заставляет его приступить к выполнению программы начального запуска. Аварийная ситуация может быть вызвана помехами по цепям питания и "земли", сбоями в работе памяти, внешними ионизирующими излучениями и еще множеством причин. В результате процессор может потерять контроль над выполняемой программой и остановиться в каком-то адресе. Для выхода из этого состояния как раз и используется сигнал начального сброса. Этот же вход начального сброса может использоваться для оповещения процессора о том, что напряжение питания стало ниже установленного предела. В таком случае процессор переходит к выполнению программы сохранения важных данных. По сути, этот вход представляет собой особую разновидность радиального прерывания. Иногда у микросхемы процессора имеется еще один-два входа радиальных прерываний для обработки особых ситуаций (например, для прерывания от внешнего таймера).

Шина питания современного процессора обычно имеет одно напряжение питания (+5В или +3,3В) и общий провод ("землю"). Первые процессоры нередко требовали нескольких напряжений питания. В некоторых процессорах предусмотрен режим пониженного энергопотребления. Вообще, современные микросхемы процессоров, особенно с высокими тактовыми частотами, потребляют довольно большую мощность. В результате для поддержания

нормальной рабочей температуры корпуса на них нередко приходится устанавливать радиаторы, вентиляторы или даже специальные микрохолодильники.

Для подключения процессора к магистрали используются буферные микросхемы, обеспечивающие, если необходимо, демультиплексирование сигналов и электрическое буферирование сигналов магистрали. Иногда протоколы обмена по системной магистрали и по шинам процессора не совпадают между собой, тогда буферные микросхемы еще и согласуют эти протоколы друг с другом. Иногда в микропроцессорной системе используется несколько магистралей (системных и локальных), тогда для каждой из магистралей применяется свой буферный узел. Такая структура характерна, например, для персональных компьютеров.

После включения питания процессор переходит в первый адрес программы начального пуска и выполняет эту программу. Данная программа предварительно записана в постоянную (энергонезависимую) память. После завершения программы начального пуска процессор начинает выполнять основную программу, находящуюся в постоянной или оперативной памяти, для чего выбирает по очереди все команды. От этой программы процессор могут отвлекать внешние прерывания или запросы на ПДП. Команды из памяти процессор выбирает с помощью циклов чтения по магистрали. При необходимости процессор записывает данные в память или в устройства ввода/вывода с помощью циклов записи или же читает данные из памяти или из устройств ввода/вывода с помощью циклов чтения.

Таким образом, основные функции любого процессора следующие:

--- выборка (чтение) выполняемых команд;

--- ввод (чтение) данных из памяти или устройства ввода/вывода;

--- вывод (запись) данных в память или в устройства ввода/вывода;

--- обработка данных (операндов), в том числе арифметические операции над ними;

--- адресация памяти, то есть задание адреса памяти, с которым будет производиться обмен;

--- обработка прерываний и режима прямого доступа.

Упрощенно структуру микропроцессора можно представить в следующем виде (рис. 20).

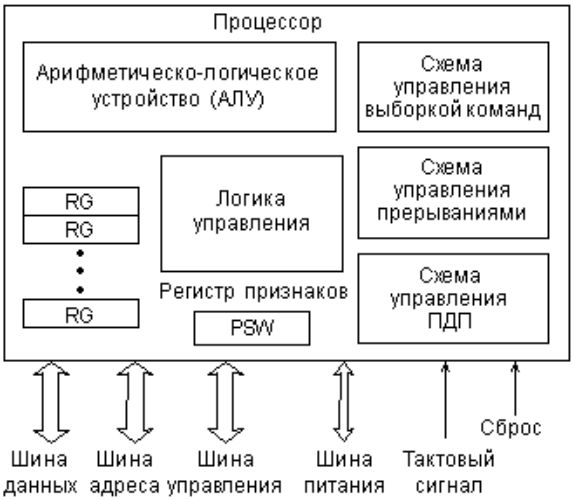


Рис. 20. Внутренняя структура микропроцессора.

Основные функции показанных узлов следующие.

Схема управления выборкой команд выполняет чтение команд из памяти и их дешифрацию. В первых микропроцессорах было невозможно одновременное выполнение предыдущей команды и выборка следующей команды, так как процессор не мог совмещать эти операции. В 16 - ти разрядных процессорах появляется конвейер (очередь) команд, позволяющий выбирать несколько следующих команд, пока выполняется предыдущая. Два процесса идут параллельно, что ускоряет работу процессора. Конвейер представляет собой небольшую внутреннюю память процессора, в которую при малейшей возможности (при освобождении внешней шины) записывается несколько команд, следующих за исполняемой. Читаются эти команды

процессором в том же порядке, что и записываются в конвейер (это память типа FIFO, First In — First Out, первый вошел — первый вышел). Правда, если выполняемая команда предполагает переход не на следующую ячейку памяти, а на удаленную (с меньшим или большим адресом), конвейер не помогает, и его приходится сбрасывать. Но такие команды встречаются в программах сравнительно редко.

Развитием идеи конвейера стало использование внутренней кэш-памяти процессора, которая заполняется командами, пока процессор занят выполнением предыдущих команд. Чем больше объем кэш-памяти, тем меньше вероятность того, что ее содержимое придется сбросить при команде перехода. Понятно, что обрабатывать команды, находящиеся во внутренней памяти, процессор может гораздо быстрее, чем те, которые расположены во внешней памяти. В кэш-памяти могут храниться и данные, которые обрабатываются в данный момент, это также ускоряет работу. Для большего ускорения выборки команд в современных процессорах применяют совмещение выборки и дешифрации, одновременную дешифрацию нескольких команд, несколько параллельных конвейеров команд, предсказание команд переходов и некоторые другие методы.

Арифметико-логическое устройство (или АЛУ, ALU) предназначено для обработки информации в соответствии с полученной процессором командой. Примерами обработки могут служить логические операции (логическое "И", "ИЛИ", "Исключающего ИЛИ" и т.д.), побитные операции над операндами и арифметические операции (сложение, вычитание, умножение, деление и т.д.). Над какими кодами производится операция, куда помещается ее результат — определяется выполняемой командой. Если команда сводится всего лишь к пересылке данных без их обработки, то АЛУ не участвует в ее выполнении.

Быстродействие АЛУ во многом определяет производительность процессора. Причем важна не только частота тактового сигнала, которым тактируется АЛУ,

но и количество тактов, необходимое для выполнения той или иной команды. Для повышения производительности разработчики стремятся довести время выполнения команды до одного такта, а также обеспечить работу АЛУ на возможно более высокой частоте. Один из путей решения этой задачи состоит в уменьшении количества выполняемых АЛУ команд, создание процессоров с уменьшенным набором команд (так называемые RISC-процессоры). Другой путь повышения производительности процессора — использование нескольких параллельно работающих АЛУ.

Что касается операций над числами с плавающей точкой и других специальных сложных операций, то в системах на базе первых процессоров их реализовали последовательностью более простых команд, специальными подпрограммами, однако затем были разработаны специальные вычислители — математические сопроцессоры, которые заменяли основной процессор на время выполнения таких команд. В современных микропроцессорах математические сопроцессоры входят в структуру как составная часть.

Регистры процессора представляют собой по сути ячейки очень быстрой памяти и служат для временного хранения различных кодов: данных, адресов, служебных кодов. Операции с этими кодами выполняются предельно быстро, поэтому, в общем случае, чем больше внутренних регистров, тем лучше. Кроме того, на быстродействие процессора сильно влияет разрядность регистров. Именно разрядность регистров и АЛУ называется внутренней разрядностью процессора, которая может не совпадать с внешней разрядностью.

По отношению к назначению внутренних регистров существует два основных подхода. Первого придерживается, например, компания Intel, которая каждому регистру отводит строго определенную функцию. С одной стороны, это упрощает организацию процессора и уменьшает время выполнения команды, но с другой — снижает гибкость, а иногда и замедляет работу программы. Например, некоторые арифметические операции и обмен с устройствами

ввода/вывода проводятся только через один регистр — **аккумулятор**, в результате чего при выполнении некоторых процедур может потребоваться несколько дополнительных пересылок между регистрами. Второй подход состоит в том, чтобы все (или почти все) регистры сделать равноправными, как, например, в 16-разрядных процессорах Т-11 фирмы DEC. При этом достигается высокая гибкость, но необходимо усложнение структуры процессора. Существуют и промежуточные решения, в частности, в процессоре MC68000 фирмы Motorola половина регистров использовалась для данных, и они были взаимозаменяемы, а другая половина — для адресов, и они также взаимозаменяемы.

Регистр признаков (регистр состояния) занимает особое место, хотя он также является внутренним регистром процессора. Содержащаяся в нем информация — это не данные, не адрес, а слово состояния процессора (ССП, PSW — Processor Status Word). Каждый бит этого слова (флаг) содержит информацию о результате предыдущей команды. Например, есть бит нулевого результата, который устанавливается в том случае, когда результат выполнения предыдущей команды — нуль, и очищается в том случае, когда результат выполнения команды отличен от нуля. Эти биты (флаги) используются командами условных переходов, например, командой перехода в случае нулевого результата. В этом же регистре иногда содержатся флаги управления, определяющие режим выполнения некоторых команд.

Схема управления прерываниями обрабатывает поступающий на процес- сор запрос прерывания, определяет адрес начала программы обработки прерывания (адрес вектора прерывания), обеспечивает переход к этой программе после выполнения текущей команды и сохранения в памяти (в стеке) текущего состояния регистров процессора. По окончании программы обработки прерывания процессор возвращается к прерванной программе с

восстановленными из памяти (из стека) значениями внутренних регистров. Подробнее о стеке будет рассказано в разделе функции памяти.

Схема управления прямым доступом к памяти служит для временного отключения процессора от внешних шин и приостановки работы процессора на время предоставления прямого доступа запросившему его устройству.

Логика управления организует взаимодействие всех узлов процессора, перенаправляет данные, синхронизирует работу процессора с внешними сигналами, а также реализует процедуры ввода и вывода информации.

Таким образом, в ходе работы процессора схема выборки команд выбирает последовательно команды из памяти, затем эти команды выполняются, причем в случае необходимости обработки данных подключается АЛУ. На входы АЛУ могут подаваться обрабатываемые данные из памяти или из внутренних регистров. Во внутренних регистрах хранятся также коды адресов обрабатываемых данных, расположенных в памяти. Результат обработки в АЛУ изменяет состояние регистра признаков и записывается во внутренний регистр или в память (как источник, так и приемник данных указывается в составе кода команды). При необходимости информация может переписываться из памяти (или из устройства ввода/вывода) во внутренний регистр или из внутреннего регистра в память (или в устройство ввода/вывода).

Внутренние регистры любого микропроцессора обязательно выполняют две служебные функции:

--- определяют адрес в памяти, где находится выполняемая в данный момент команда (функция **счетчика команд** или **указателя команд**);

--- определяют текущий адрес стека (функция указателя стека).

В разных процессорах для каждой из этих функций может отводиться один или два внутренних регистра. Эти два указателя отличаются от других не только своим специфическим, служебным, системным назначением, но и особым способом изменения содержимого. Их содержимое программы могут менять

только в случае крайней необходимости, так как любая ошибка при этом грозит нарушением работы компьютера, зависанием и порчей содержимого памяти.

Содержимое указателя (счетчика) команд изменяется следующим образом. В начале работы системы (при включении питания) в него заносится раз и навсегда установленное значение. Это первый адрес программы начального запуска. Затем после выборки из памяти каждой следующей команды значение указателя команд автоматически увеличивается (инкрементируется) на единицу (или на два в зависимости от формата команд и типа процессора). То есть следующая команда будет выбираться из следующего по порядку адреса памяти. При выполнении команд перехода, нарушающих последовательный перебор адресов памяти, в указатель команд принудительно записывается новое значение — новый адрес в памяти, начиная с которого адреса команд опять же будут перебираться последовательно. Такая же смена содержимого указателя команд производится при вызове подпрограммы и возврате из нее или при начале обработки прерывания и после его окончания.

О стеке будет подробнее рассказано в разделе функции памяти.

Основная функция любого процессора, ради которой он и создается, — это выполнение команд. Система команд, выполняемых процессором, представляет собой нечто подобное таблице истинности логических элементов или таблице режимов работы более сложных логических микросхем. То есть она определяет логику работы процессора и его реакцию на те или иные комбинации внешних событий.

Написание программ для микропроцессорной системы — важнейший и часто наиболее трудоемкий этап разработки такой системы. А для создания эффективных программ необходимо иметь хотя бы самое общее представление о системе команд используемого процессора. Самые компактные и быстрые программы и подпрограммы создаются на языке Ассемблер, использование которого без знания системы команд абсолютно невозможно, ведь язык

Ассемблер представляет собой символьную запись цифровых кодов машинного языка, кодов команд процессора. Конечно, для разработки программного обеспечения существуют всевозможные программные средства. Пользоваться ими обычно можно и без знания системы команд процессора. Чаще всего применяются языки программирования высокого уровня, такие как Паскаль и Си. Однако знание системы команд и языка Ассемблер позволяет в несколько раз повысить эффективность некоторых наиболее важных частей программного обеспечения любой микропроцессорной системы — от микроконтроллера до персонального компьютера.

Каждая команда, выбираемая (читаемая) из памяти процессором, определяет алгоритм поведения процессора на ближайшие несколько тактов. Код команды говорит о том, какую операцию предстоит выполнить процессору и с какими операндами (то есть кодами данных), где взять исходную информацию для выполнения команды и куда поместить результат (если необходимо). Код команды может занимать от одного до нескольких байт, причем процессор узнает о том, сколько байт команды ему надо читать, из первого прочитанного им байта или слова. В процессоре код команды расшифровывается и преобразуется в набор микроопераций, выполняемых отдельными узлами процессора. Но разработчику микропроцессорных систем это знание не слишком важно, ему важен только результат выполнения той или иной команды.

4.2. Методы адресации

Большая часть команд процессора работает с кодами данных (операндами). Одни команды требуют входных операндов (одного или двух), другие выдают выходные операнды (чаще один операнд). Входные операнды называются еще операндами-источниками, а выходные называются операндами-приемниками.

Все эти коды операндов (входные и выходные) должны где-то располагаться. Они могут находиться во внутренних регистрах процессора (наиболее удобный и быстрый вариант). Они могут располагаться в системной памяти (самый распространенный вариант). Наконец, они могут находиться в устройствах ввода/вывода (наиболее редкий случай). Определение места положения операндов производится кодом команды. Причем существуют разные методы, с помощью которых код команды может определить, откуда брать входной операнд и куда помещать выходной операнд. Эти методы называются **методами адресации**. Эффективность выбранных методов адресации во многом определяет эффективность работы всего процессора в целом.

Количество методов адресации в различных процессорах может быть от 4 до 16. Рассмотрим несколько типичных методов адресации операндов, используемых сейчас в большинстве микропроцессоров.

Непосредственная адресация (рис. 21) предполагает, что операнд (входной) находится в памяти непосредственно за кодом команды. Операнд обычно представляет собой константу, которую надо куда-то переслать, к чему-то прибавить и т.д. Например, команда может состоять в том, чтобы прибавить число 6 к содержимому какого-то внутреннего регистра процессора. Это число 6 будет располагаться в памяти, внутри программы в адресе, следующем за кодом данной команды сложения.

Рис. 21. Непосредственная адресация.

Прямая (она же абсолютная) адресация (рис. 22) предполагает, что операнд (входной или выходной) находится в памяти по адресу, код которого находится внутри программы сразу же за кодом команды. Например, команда может состоять в том, чтобы очистить (сделать нулевым) содержимое ячейки памяти с адресом 1000000. Код этого адреса 1000000 будет располагаться в памяти, внутри программы в следующем адресе за кодом данной команды очистки.

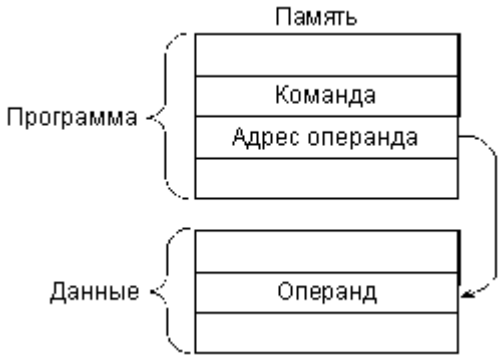


Рис. 22. Прямая адресация.

Регистровая адресация (рис. 23) предполагает, что операнд (входной или выходной) находится во внутреннем регистре процессора. Например, команда может состоять в том, чтобы переслать число из нулевого регистра в первый. Номера обоих регистров (0 и 1) будут определяться кодом команды пересылки.

Рис. 23. Регистровая адресация.

Косвенно-регистровая (она же косвенная) адресация предполагает, что во внутреннем регистре процессора находится не сам операнд, а его адрес в памяти (рис. 24). Например, команда может состоять в том, чтобы очистить ячейку памяти с адресом, находящимся в нулевом регистре. Номер этого регистра (0) будет определяться кодом команды очистки.

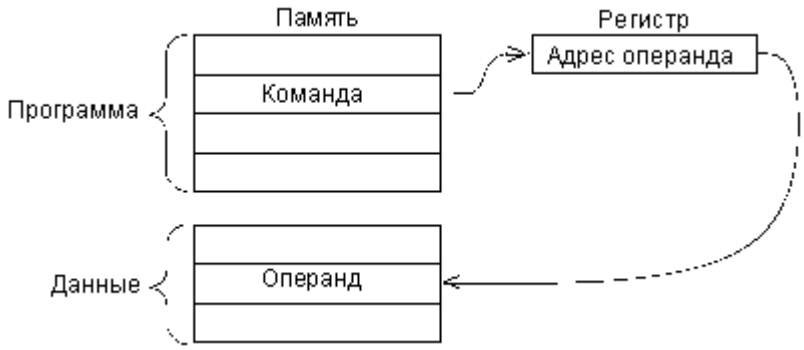


Рис. 24. Косвенная адресация.

Реже встречаются еще два метода адресации.

Автоинкрементная адресация очень близка к косвенной адресации, но отличается от нее тем, что после выполнения команды содержимое используемого регистра увеличивается на единицу или на два. Этот метод адресации очень удобен, например, при последовательной обработке кодов из массива данных, находящегося в памяти. После обработки какого-то кода адрес в регистре будет указывать уже на следующий код из массива. При использовании косвенной адресации в данном случае пришлось бы увеличивать содержимое этого регистра отдельной командой.

Автодекрементная адресация работает как автоинкрементная, но только содержимое выбранного регистра уменьшается на единицу или на два перед выполнением команды. Эта адресация также удобна при обработке массивов данных. Совместное использование автоинкрементной и автодекрементной адресаций позволяет организовать память стекового типа.

Из других распространенных методов адресации можно упомянуть об индексных методах, которые предполагают для вычисления адреса операнда прибавление к содержимому регистра заданной константы (индекса). Код этой константы располагается в памяти непосредственно за кодом команды.

Отметим, что выбор того или иного метода адресации в значительной степени определяет время выполнения команды. Самая быстрая адресация — это

регистровая, так как она не требует дополнительных циклов обмена по магистрали. Если же адресация требует обращения к памяти, то время выполнения команды будет увеличиваться за счет длительности необходимых циклов обращения к памяти. Понятно, что чем больше внутренних регистров у процессора, тем чаще и свободнее можно применять регистровую адресацию, и тем быстрее будет работать система в целом.

4.3. Сегментирование памяти

Говоря об адресации, нельзя обойти вопрос о сегментировании памяти, применяемой в некоторых процессорах, например в процессорах IBM PC-совместимых персональных компьютеров.

В процессоре Intel 8086 сегментирование памяти организовано следующим образом.

Вся память системы представляется не в виде непрерывного пространства, а в виде нескольких кусков — сегментов заданного размера (по 64 Кбайта), положение которых в пространстве памяти можно изменять программным путем. Для хранения кодов адресов памяти используются не отдельные регистры, а пары регистров:

---сегментный регистр определяет адрес начала сегмента (то есть положение сегмента в памяти);

---регистр указателя (регистр смещения) определяет положение рабочего адреса внутри сегмента.

При этом физический 20-разрядный адрес памяти, выставляемый на внешнюю шину адреса, образуется так, как показано на рис. 25, то есть путем сложения смещения и адреса сегмента со сдвигом на 4 бита. Положение этого адреса в памяти показано на рис. 26.

Сегмент может начинаться только на 16-байтной границе памяти (так как адрес начала сегмента, по сути, имеет четыре младших нулевых разряда, как видно из

рис. 25), то есть с адреса, кратного 16. Эти допустимые границы сегментов называются границами параграфов.

Отметим, что введение сегментирования, прежде всего, связано с тем, что внутренние регистры процессора 16-разрядные, а физический адрес памяти 20-разрядный (16-разрядный адрес позволяет использовать память только в 64 Кбайт, что явно недостаточно). В процессоре MC68000 фирмы Motorola внутренние регистры 32-разрядные, поэтому там проблемы сегментирования памяти не возникает.

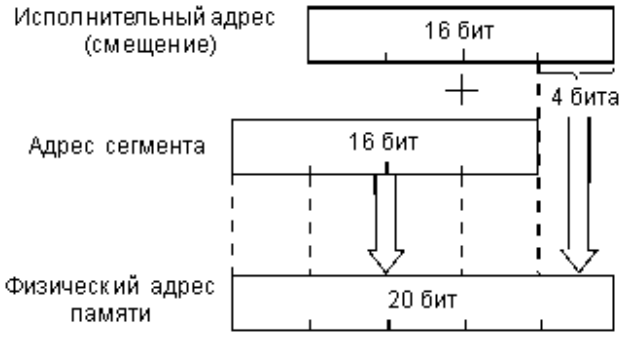


Рис. 25. Формирование физического адреса памяти из адреса сегмента и смещения.

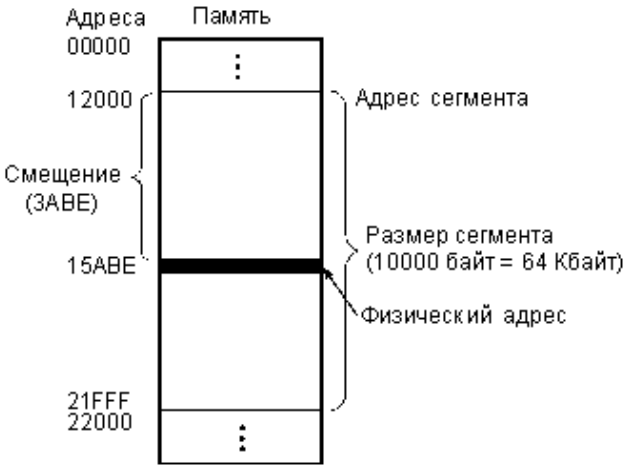


Рис. 26. Физический адрес в сегменте (все коды — шестнадцатеричные).

Применяются и более сложные методы сегментирования памяти. Например, в процессоре Intel 80286 в так называемом защищенном режиме адрес памяти вычисляется в соответствии с рис. 27.

В сегментном регистре в данном случае хранится не базовый (начальный) адрес сегментов, а коды селекторов, определяющие адреса в памяти, по которым хранятся дескрипторы (то есть описатели) сегментов. Область памяти с дескрипторами называется таблицей дескрипторов. Каждый дескриптор сегмента содержит базовый адрес сегмента, размер сегмента (от 1 до 64 Кбайт) и его атрибуты. Базовый адрес сегмента имеет разрядность 24 бит, что обеспечивает адресацию 16 Мбайт физической памяти.

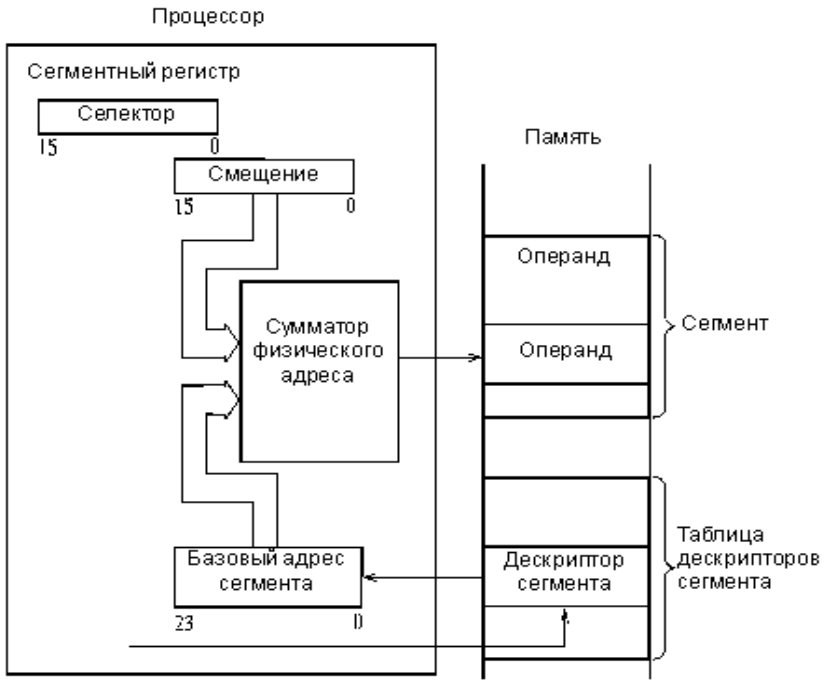


Рис. 27. Адресация памяти в защищенном режиме процессора Intel 80286.

Таким образом, на сумматор, вычисляющий физический адрес памяти, подается не содержимое сегментного регистра, как в предыдущем случае, а базовый адрес сегмента из таблицы дескрипторов.

Еще более сложный метод адресации памяти с сегментированием использован в процессоре Intel 80386 и в более поздних моделях процессоров фирмы Intel. Этот метод иллюстрируется рис. 28.

Адрес памяти (физический адрес) вычисляется в три этапа. Сначала вычисляется так называемый **эффективный адрес** (32-разрядный) путем

суммирования трех компонентов: базы, индекса и смещения (Base, Index, Displacement), причем возможно умножение индекса на масштаб (Scale). Эти компоненты имеют следующий смысл:

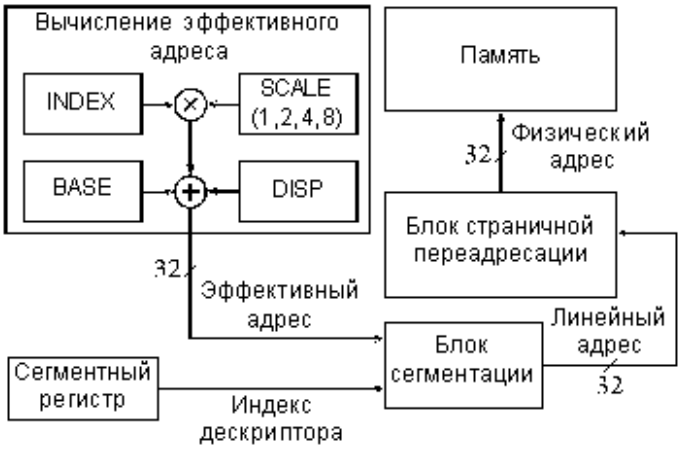


Рис. 28. Формирование физического адреса памяти процессора 80386 в защищенном режиме.

--- смещение — это 8-, 16- или 32-разрядное число, включенное в команду.

--- база — это содержимое базового регистра процессора. Обычно оно используется для указания на начало некоторого массива.

--- индекс — это содержимое индексного регистра процессора. Обычно оно используется для выбора одного из элементов массива.

--- масштаб — это множитель (он может быть равен 1, 2, 4 или 8), указанный в коде команды, на который перед суммированием с другими компонентами умножается индекс. Он используется для указания размера элемента массива.

Затем специальный блок сегментации вычисляет 32-разрядный линейный адрес, который представляет собой сумму базового адреса сегмента из сегментного регистра с эффективным адресом. Наконец, физический 32-битный адрес памяти образуется путем преобразования линейного адреса блоком страничной переадресации, который осуществляет перевод линейного адреса в физический страницами по 4 Кбайта.

В любом случае сегментирование позволяет выделить в памяти один или несколько сегментов для данных и один или несколько сегментов для программ. Переход от одного сегмента к другому сводится всего лишь к изменению содержимого сегментного регистра. Иногда это бывает очень удобно. Но для программиста работать с сегментированной памятью обычно сложнее, чем с непрерывной, несегментированной памятью, так как приходится следить за границами сегментов, за их описанием, переключением и т.д.

4.4. Адресация байтов и слов

Многие процессоры, имеющие разрядность 16 или 32, способны адресовать не только целое слово в памяти (16-разрядное или 32-разрядное), но и отдельные байты. Каждому байту в каждом слове при этом отводится свой адрес.

Так, в случае 16-разрядных процессоров все слова в памяти (16-разрядные) имеют четные адреса. А байты, входящие в эти слова, могут иметь как четные адреса, так и нечетные.

Например, пусть 16-разрядная ячейка памяти имеет адрес 23420, и в ней хранится код 2A5E (рис. 29).

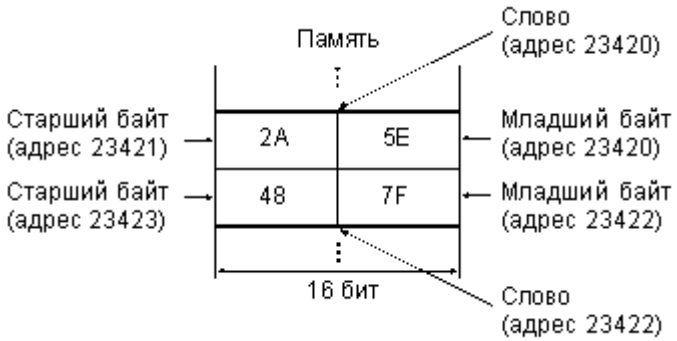


Рис. 29. Адресация слов и байтов.

При обращении к целому слову (с содержимым 2A5E) процессор выставляет адрес 23420. При обращении к младшему байту этой ячейки (с содержимым 5E)

процессор выставляет тот же самый адрес 23420, но использует команду, адресующую байт, а не слово. При обращении к старшему байту этой же ячейки (с содержимым 2A) процессор выставляет адрес 23421 и использует команду, адресующую байт. Следующая по порядку 16-разрядная ячейка памяти с содержимым 487F будет иметь адрес 23422, то есть опять же четный. Ее байты будут иметь адреса 23422 и 23423.

Для различия байтовых и словных циклов обмена на магистрали в шине управления предусматривается специальный сигнал байтового обмена. Для работы с байтами в систему команд процессора вводятся специальные команды или предусматриваются методы байтовой адресации.

4.5. Регистры процессора

Как уже упоминалось, внутренние регистры процессора представляют собой сверхоперативную память небольшого размера, которая предназначена для временного хранения служебной информации или данных. Количество регистров в разных процессорах может быть от 6—8 до нескольких десятков. Регистры могут быть универсальными и специализированными. Специализированные регистры, которые присутствуют в большинстве процессоров, — это регистр-счетчик команд, регистр состояния (PSW), регистр указателя стека. Остальные регистры процессора могут быть как универсальными, так и специализированными.

Например, в 16-разрядном процессоре T-11 фирмы DEC было 8 регистров общего назначения (РОН) и один регистр состояния. Все регистры имели по 16 разрядов. Из регистров общего назначения один отводился под счетчик команд, другой — под указатель стека. Все остальные регистры общего назначения полностью взаимозаменяемы, то есть имеют универсальное назначение, могут хранить как данные, так и адреса (указатели), индексы и т.д. Максимально

допустимый объем памяти для данного процессора составлял 64 Кбайт (адрес памяти 16-разрядный).

В 16-разрядном процессоре MC68000 фирмы Motorola было 19 регистров: 16-разрядный регистр состояния, 32-разрядный регистр счетчика команд, 9 регистров адреса (32-разрядных) и 8 регистров данных (32-разрядных). Два регистра адреса отведены под указатели стека. Максимально допустимый объем адресуемой памяти — 16 Мбайт (внешняя шина адреса 24-разрядная). Все 8 регистров данных взаимозаменяемы. 7 регистров адреса – тоже взаимозаменяемы.

В 16-разрядном процессоре Intel 8086, который стал базовым в линии процессоров, используемых в персональных компьютерах, реализован принципиально другой подход. Каждый регистр этого процессора имеет свое особое назначение, и заменять друг друга регистры могут только частично или же не могут вообще. Остановимся на особенностях этого процессора подробнее.

Процессор 8086 имеет 14 регистров разрядностью по 16 бит. Из них четыре регистра (AX, BX, CX, DX) — это регистры данных, каждый из которых помимо хранения операндов и результатов операций имеет еще и свое специфическое назначение:

--- регистр AX — умножение, деление, обмен с устройствами ввода/вывода (команды ввода и вывода);

--- регистр BX — базовый регистр в вычислениях адреса;

--- регистр CX — счетчик циклов;

--- регистр DX — определение адреса ввода/вывода.

Для регистров данных существует возможность раздельного использования обоих байтов (например, для регистра AX они имеют обозначения AL – младший байт и AH — старший байт).

Следующие четыре внутренних регистра процессора — это сегментные регистры, каждый из которых определяет положение одного из рабочих сегментов (рис. 30):

--- регистр CS (Code Segment) соответствует сегменту команд, исполняемых в данный момент;

--- регистр DS (Data Segment) соответствует сегменту данных, с которыми работает процессор;

--- регистр ES (Extra Segment) соответствует дополнительному сегменту данных;

--- регистр SS (Stack Segment) соответствует сегменту стека.

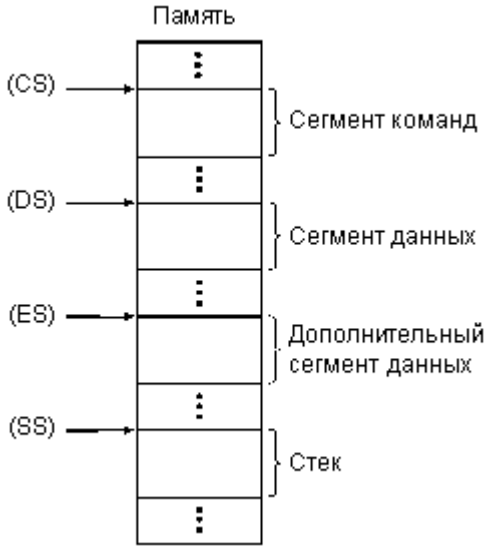


Рис. 30. Сегменты команд, данных и стека в памяти.

В принципе, все эти сегменты могут и перекрываться для оптимального использования пространства памяти. Например, если программа занимает только часть сегмента, то сегмент данных может начинаться сразу после завершения работы программы (с точностью 16 байт), а не после окончания всего сегмента программы.

Следующие пять регистров процессора (SP — Stack Pointer, BP — Base Pointer, SI — Source Index, DI — Destination Index, IP —Instruction Pointer) служат

указателями (то есть определяют смещение в пределах сегмента). Например, счетчик команд процессора образуется парой регистров CS и IP, а указатель стека — парой регистров SP и SS. Регистры SI, DI используются в строковых операциях, то есть при последовательной обработке нескольких ячеек памяти одной командой.

Последний регистр FLAGS — это регистр состояния процессора (PSW). Из его 16 разрядов используются только девять (рис. 31): CF (Carry Flag) — флаг переноса при арифметических операциях; PF (Parity Flag) — флаг четности результата, AF (Auxiliary Flag) — флаг дополнительного переноса; ZF (Zero Flag) — флаг нулевого результата; SF (Sign Flag) — флаг знака (совпадает со старшим битом результата); TF (Trap Flag) — флаг пошагового режима (используется при отладке),; IF (Interrupt-enable Flag) — флаг разрешения аппаратных прерываний; DF (Direction Flag) — флаг направления при строковых операциях,; OF (Overflow Flag) — флаг переполнения.

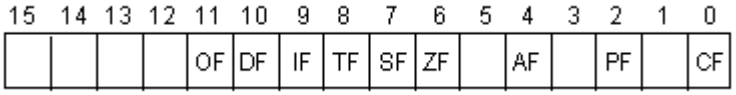


Рис. 31. Регистр состояния процессора 8086.

Биты регистра состояния устанавливаются или очищаются в зависимости от результата исполнения предыдущей команды и используются некоторыми командами процессора. Биты регистра состояния могут также устанавливаться и очищаться специальными командами процессора (о системе команд процессора будет рассказано в следующем разделе).

Во многих процессорах выделяется специальный регистр, называемый **аккумулятором** (то есть накопителем). При этом, как правило, только этот регистр-аккумулятор может участвовать во всех операциях, только через него может производиться взаимодействие с устройствами ввода/вывода. Иногда в него же помещается результат любой выполненной команды (в этом случае

говорят даже об "аккумуляторной" архитектуре процессора). Например, в процессоре 8086 регистр данных AX можно считать своеобразным аккумулятором, так как именно он обязательно участвует в командах умножения и деления, а также только через него можно пересылать данные в устройство ввода/вывода и из устройства ввода/вывода. Выделение специального регистра-аккумулятора упрощает структуру процессора и ускоряет пересылки кодов внутри процессора, но в некоторых случаях замедляет работу системы в целом, так как весь поток информации должен пройти через один регистр-аккумулятор. В случае, когда несколько регистров процессора полностью взаимозаменяемы, таких проблем не возникает.

4.6. Система команд процессора

В общем случае система команд процессора включает в себя следующие четыре основные группы команд:

--- команды пересылки данных;

--- арифметические команды;

--- логические команды;

--- команды переходов.

Команды пересылки данных не требуют выполнения никаких операций над операндами. Операнды просто пересылаются (точнее, копируются) из источника (Source) в приемник (Destination). Источником и приемником могут быть внутренние регистры процессора, ячейки памяти или устройства ввода/вывода. АЛУ в данном случае не используется.

Команды пересылки данных занимают очень важное место в системе команд любого процессора. Они выполняют следующие важнейшие функции:

загрузка (запись) содержимого во внутренние регистры процессора;

сохранение в памяти содержимого внутренних регистров процессора;

копирование содержимого из одной области памяти в другую;

запись в устройства ввода/вывода и чтение из устройств ввода/вывода.

В некоторых процессорах все эти функции выполняются одной единственной командой MOV (для байтовых пересылок — MOVB) но с различными методами адресации операндов.

В других процессорах помимо команды MOV имеется еще несколько команд для выполнения перечисленных функций. Например, для загрузки регистров могут использоваться команды загрузки, причем для разных регистров — разные команды (их обозначения обычно строятся с использованием слова LOAD — загрузка). Часто выделяются специальные команды для сохранения в стеке и для извлечения из стека (PUSH — сохранить в стеке, POP — извлечь из стека). Эти команды выполняют пересылку с автоинкрементной и с автодекрементной адресацией (даже если эти режимы адресации не предусмотрены в процессоре в явном виде).

Иногда в систему команд вводится специальная команда MOVS для строчной (или цепочечной) пересылки данных (например, в процессоре 8086). Эта команда пересылает не одно слово или байт, а заданное количество слов или байтов (MOVSB), то есть инициирует не один цикл обмена по магистрали, а несколько. При этом адрес памяти, с которым происходит взаимодействие, увеличивается на 1 или на 2 после каждого обращения или же уменьшается на 1 или на 2 после каждого обращения. То есть в неявном виде применяется автоинкрементная или автодекрементная адресация.

В некоторых процессорах (например, в процессоре 8086) специально выделяются функции обмена с устройствами ввода/вывода. Команда IN используется для ввода (чтения) информации из устройства ввода/вывода, а команда OUT используется для вывода (записи) в устройство ввода/вывода. Обмен информацией в этом случае производится между регистром-аккумулятором и устройством ввода/вывода. Начиная с процессора 80286, добавлены команды строчного (цепочечного) ввода (команда INS) и строчного

вывода (команда OUTS). Эти команды позволяют пересылать целый массив (строку) данных из памяти в устройство ввода/вывода (OUTS) или из устройства ввода/вывода в память (INS). Адрес памяти после каждого обращения увеличивается или уменьшается (как и в случае с командой MOVS).

Также к командам пересылки данных относятся команды обмена информацией (их обозначение строится на основе слова Exchange). Может быть предусмотрен обмен информацией между внутренними регистрами, между двумя половинами одного регистра (SWAP) или между регистром и ячейкой памяти.

Арифметические команды выполняют операции сложения, вычитания, умножения, деления, увеличения на единицу (инкрементирования), уменьшения на единицу (декрементирования) и т.д. Этим командам требуется один или два входных операнда. Формируют команды один выходной операнд.

Арифметические команды рассматривают коды операндов как числовые двоичные или двоично-десятичные коды. Эти команды могут быть разделены на пять основных групп:

--- команды операций с фиксированной запятой (сложение, вычитание, умножение, деление);

--- команды операций с плавающей запятой (сложение, вычитание, умножение, деление);

--- команды очистки;

--- команды инкремента и декремента;

--- команда сравнения.

Команды операций с фиксированной запятой работают с кодами в регистрах процессора или в памяти как с обычными двоичными кодами. Команда сложения (ADD) вычисляет сумму двух кодов. Команда вычитания (SUB) вычисляет разность двух кодов. Команда умножения (MUL) вычисляет произведение двух кодов (разрядность результата вдвое больше разрядности

сомножителей). Команда деления (DIV) вычисляет частное от деления одного кода на другой. Причем все эти команды могут работать как с числами со знаком, так и с числами без знака.

Команды операций с плавающей запятой (точкой) используют формат представления чисел с порядком и мантиссой (обычно эти числа занимают две последовательные ячейки памяти). В современных мощных процессорах набор команд с плавающей запятой не ограничивается только четырьмя арифметическими действиями, а содержит и множество других более сложных команд, например, вычисление тригонометрических функций, логарифмических функций, а также сложных функций, необходимых при обработке звука и изображения.

Команды очистки (CLR) предназначены для записи нулевого кода в регистр или ячейку памяти. Эти команды могут быть заменены командами пересылки нулевого кода, но специальные команды очистки обычно выполняются быстрее, чем команды пересылки. Команды очистки иногда относят к группе логических команд, но суть их от этого не меняется.

Команды инкремента (увеличения на единицу, INC) и декремента (уменьшения на единицу, DEC) также бывают очень удобны. Их можно в принципе заменить командами суммирования с единицей или вычитания единицы, но инкремент и декремент выполняются быстрее, чем суммирование и вычитание. Эти команды требуют одного входного операнда, который одновременно является и выходным операндом.

Наконец, команда сравнения (обозначается CMP) предназначена для сравнения двух входных операндов. По сути, она вычисляет разность этих двух операндов, но выходного операнда не формирует, а всего лишь изменяет биты в регистре состояния процессора (PSW) по результату этого вычитания. Следующая за командой сравнения команда (обычно это команда перехода) будет анализировать биты в регистре состояния процессора и выполнять

действия в зависимости от их значений. В некоторых процессорах предусмотрены команды цепочечного сравнения двух последовательностей операндов, находящихся в памяти (например, в процессоре 8086 и совместимых с ним).

Логические команды производят над операндами логические операции, например, логическое И, логическое ИЛИ, исключающее ИЛИ, очистку, инверсию, разнообразные сдвиги (вправо, влево, арифметический сдвиг, циклический сдвиг). Этим командам, как и арифметическим, требуется один или два входных операнда, и формируют они один выходной операнд.

Логические команды выполняют над операндами логические (побитовые) операции, то есть они рассматривают коды операндов не как единое число, а как набор отдельных битов. Этим они отличаются от арифметических команд. Логические команды выполняют следующие основные операции:

--- логическое И, логическое ИЛИ, сложение по модулю 2 (Исключающее ИЛИ);

--- логические, арифметические и циклические сдвиги;

--- проверка битов и операндов;

--- установка и очистка битов (флагов) регистра состояния процессора (PSW).

Команды логических операций позволяют побитно вычислять основные логические функции от двух входных операндов. Кроме того, операция И (AND) используется для принудительной очистки заданных битов (в качестве одного из операндов при этом используется код маски, в котором разряды, требующие очистки, установлены в нуль). Операция ИЛИ (OR) применяется для принудительной установки заданных битов (в качестве одного из операндов при этом используется код маски, в котором разряды, требующие установки в единицу, равны единице). Операция «Исключающее ИЛИ» (XOR) используется для инверсии заданных битов (в качестве одного из операндов при этом применяется код маски, в котором биты, подлежащие инверсии, установлены в

единицу). Команды требуют двух входных операндов и формируют один выходной операнд.

Команды сдвигов позволяют побитно сдвигать код операнда вправо (в сторону младших разрядов) или влево (в сторону старших разрядов). Тип сдвига (логический, арифметический или циклический) определяет, каково будет новое значение старшего бита (при сдвиге вправо) или младшего бита (при сдвиге влево), а также определяет, будет ли где-то сохранено прежнее значение старшего бита (при сдвиге влево) или младшего бита (при сдвиге вправо). Например, при логическом сдвиге вправо в старшем разряде кода операнда устанавливается нуль, а младший разряд записывается в качестве флага переноса в регистр состояния процессора. А при арифметическом сдвиге вправо значение старшего разряда сохраняется прежним (нулем или единицей), младший разряд также записывается в качестве флага переноса.

Циклические сдвиги позволяют сдвигать биты кода операнда по кругу (по часовой стрелке при сдвиге вправо или против часовой стрелки при сдвиге влево). При этом в кольцо сдвига может входить или не входить флаг переноса. В бит флага переноса (если он используется) записывается значение старшего бита при циклическом сдвиге влево и младшего бита при циклическом сдвиге вправо. Соответственно, значение бита флага переноса будет переписываться в младший разряд при циклическом сдвиге влево и в старший разряд при циклическом сдвиге вправо.

Для примера на рис. 32 показаны действия, выполняемые командами сдвигов вправо.

Команды проверки битов и операндов предназначены для установки или очистки битов регистра состояния процессора в зависимости от значения выбранных битов или всего операнда в целом. Выходного операнда команды не формируют. Команда проверки операнда (TST) проверяет весь код операнда в целом на равенство нулю и на знак (на значение старшего бита), она требует

только одного входного операнда. Команда проверки бита (BIT) проверяет только отдельные биты, для выбора которых в качестве второго операнда используется код маски. В коде маски проверяемым битам основного операнда должны соответствовать единичные разряды.

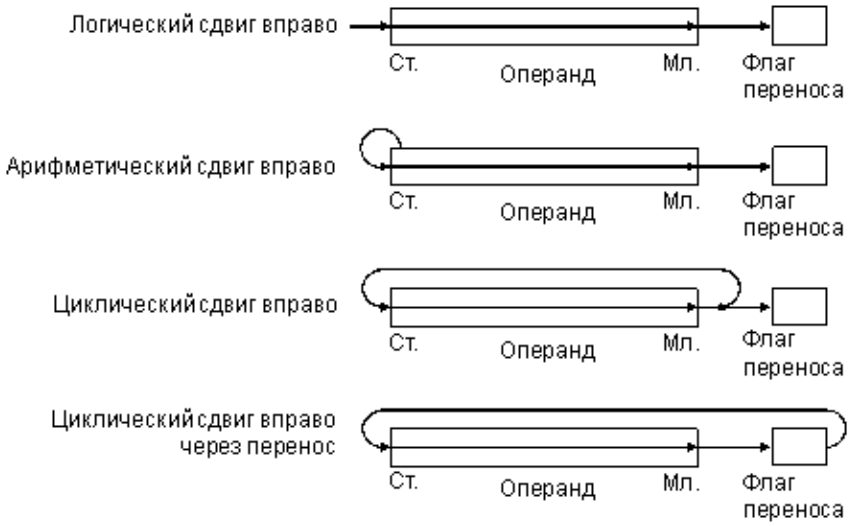


Рис. 32. Команды сдвигов вправо.

Наконец, команды установки и очистки битов регистра состояния процессора (то есть флагов) позволяют установить или очистить любой флаг, что бывает очень удобно. Каждому флагу обычно соответствуют две команды, одна из которых устанавливает его в единицу, а другая сбрасывает в нуль. Например, флагу переноса C (Carry) будут соответствовать команды CLC (очистка) и SEC или STC (установка).

Команды переходов предназначены для изменения обычного порядка последовательного выполнения команд. С их помощью организуются переходы на подпрограммы и возвраты из них, всевозможные циклы, ветвления программ, пропуски фрагментов программ и т.д. Команды переходов всегда меняют содержимое счетчика команд. Переходы могут быть условными и безусловными. Именно эти команды позволяют строить сложные алгоритмы обработки информации.

В соответствии с результатом каждой выполненной команды устанавливаются или очищаются биты регистра состояния процессора (PSW). Но надо помнить, что не все команды изменяют все имеющиеся в PSW флаги. Это определяется особенностями каждого конкретного процессора.

У разных процессоров системы команд существенно различаются, но в основе своей они очень похожи. Количество команд у процессоров также различно. Например, у процессора MC68000 всего 61 команда, а у процессора 8086 — 133 команды. У современных мощных процессоров количество команд достигает нескольких сотен. В то же время существуют процессоры с сокращенным набором команд (так называемые RISC-процессоры), в которых за счет максимального сокращения количества команд достигается увеличение эффективности и скорости их выполнения.

Рассмотрим теперь особенности четырех выделенных групп команд процессора более подробно.

Команды переходов предназначены для организации всевозможных циклов, ветвлений, вызовов подпрограмм и т.д., то есть они нарушают последовательный ход выполнения программы. Эти команды записывают в регистр-счетчик команд новое значение и тем самым вызывают переход процессора не к следующей по порядку команде, а к любой другой команде в памяти программ. Некоторые команды переходов предусматривают в дальнейшем возврат назад, в точку, из которой был сделан переход, другие не предусматривают этого. Если возврат предусмотрен, то текущие параметры процессора сохраняются в стеке. Если возврат не предусмотрен, то текущие параметры процессора не сохраняются.

Команды переходов без возврата делятся на две группы:

--- команды безусловных переходов;

--- команды условных переходов.

В обозначениях этих команд используются слова Branch (ветвление) и Jump (прыжок).

Команды безусловных переходов вызывают переход в новый адрес независимо ни от чего. Они могут вызывать переход на указанную величину смещения (вперед или назад) или же на указанный адрес памяти. Величина смещения или новое значение адреса указываются в качестве входного операнда.

Команды условных переходов вызывают переход не всегда, а только при выполнении заданных условий. В качестве таких условий обычно выступают значения флагов в регистре состояния процессора (PSW). То есть условием перехода является результат предыдущей операции, меняющей значения флагов. Всего таких условий перехода может быть от 4 до 16. Несколько примеров команд условных переходов:

--- переход, если равно нулю;

--- переход, если не равно нулю;

--- переход, если есть переполнение;

--- переход, если нет переполнения;

--- переход, если больше нуля;

--- переход, если меньше или равно нулю.

Если условие перехода выполняется, то производится загрузка в регистр-счетчик команд нового значения. Если же условие перехода не выполняется, счетчик команд просто наращивается, и процессор выбирает и выполняет следующую по порядку команду.

Специально для проверки условий перехода применяется команда сравнения (CMP), предшествующая команде условного перехода (или даже нескольким командам условных переходов). Но флаги могут устанавливаться и любой другой командой, например командой пересылки данных, любой арифметической или логической командой. Отметим, что сами команды

переходов флаги не меняют, что как раз и позволяет ставить несколько команд переходов одну за другой.

Совместное использование нескольких команд условных и безусловных переходов позволяет процессору выполнять разветвленные алгоритмы любой сложности. Для примера на рис. 33 показано разветвление программы на две ветки с последующим соединением, а на рис. 34 — разветвление на три ветки с последующим соединением.

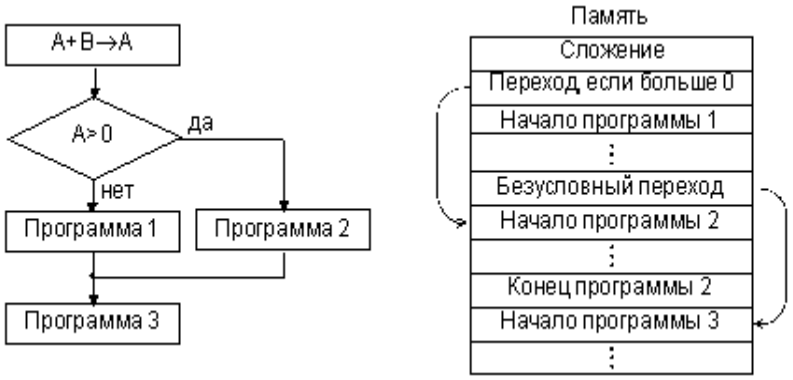


Рис. 33. Реализация разветвления на две ветви.

Команды переходов с дальнейшим возвратом в точку, из которой был произведен переход, применяются для выполнения подпрограмм, то есть вспомогательных программ. Эти команды называются также командами вызова подпрограмм (распространенное название — CALL). Использование подпрограмм позволяет упростить структуру основной программы, сделать ее более логичной, гибкой, легкой для написания и отладки. В то же время надо учитывать, что широкое использование подпрограмм, как правило, увеличивает время выполнения программы.

Все команды переходов с возвратом предполагают безусловный переход (они не проверяют никаких флагов). При этом они требуют одного входного операнда, который может указывать как абсолютное значение нового адреса, так и смещение, складываемое с текущим значением адреса. Текущее значение

счетчика команд (текущий адрес) сохраняется перед выполнением перехода в стеке.

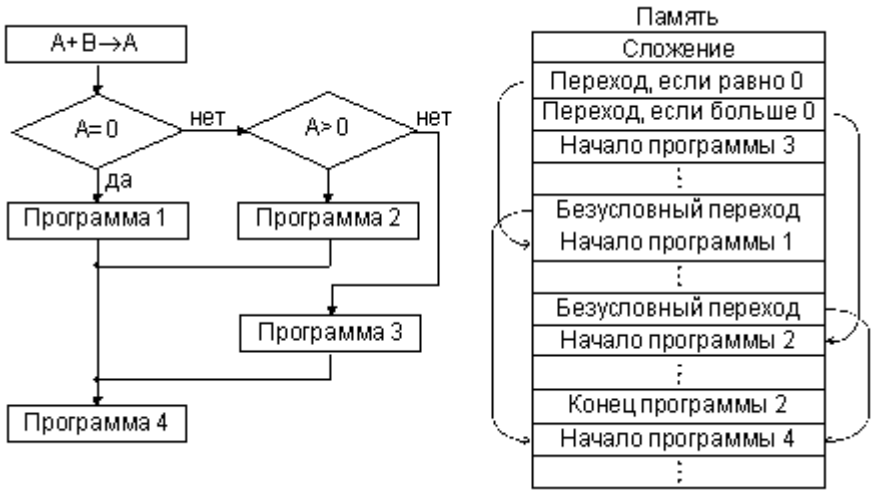


Рис. 34. Реализация разветвления на три ветви.

Для обратного возврата в точку вызова подпрограммы (точку перехода) используется специальная команда возврата (RET или RTS). Эта команда извлекает из стека значение адреса команды перехода и записывает его в регистр-счетчик команд.

Особое место среди команд перехода с возвратом занимают команды прерываний (распространенное название — INT). Эти команды в качестве входного операнда требуют номер прерывания (адрес вектора). Обслуживание таких переходов осуществляется точно так же, как и аппаратных прерываний. То есть для выполнения данного перехода процессор обращается к таблице векторов прерываний и получает из нее по номеру прерывания адрес памяти, в который ему необходимо перейти. Адрес вызова прерывания и содержимое регистра состояния процессора (PSW) сохраняются в стеке. Сохранение PSW — важное отличие команд прерывания от команд переходов с возвратом.

Команды прерываний во многих случаях оказываются удобнее, чем обычные команды переходов с возвратом. Сформировать таблицу векторов прерываний можно один раз, а потом уже обращаться к ней по мере необходимости. Номер

прерывания соответствует номеру подпрограммы, то есть номеру функции, выполняемой подпрограммой. Поэтому команды прерывания гораздо чаще включаются в системы команд процессоров, чем обычные команды переходов с возвратом.

Для возврата из подпрограммы, вызванной командой прерывания, используется команда возврата из прерывания (IRET или RTI). Эта команда извлекает из стека сохраненное там значение счетчика команд и регистра состояния процессора (PSW).

Отметим, что у некоторых процессоров предусмотрены также команды условных прерываний, например, команда прерывания при переполнении.

Конечно, в данном разделе мы рассмотрели только основные команды, наиболее часто встречающиеся в процессорах. У конкретных процессоров могут быть и многие другие команды, не относящиеся к перечисленным группам команд. Но изучать их надо уже после того, как выбран тип процессора, подходящий для задачи, решаемой данной микропроцессорной системой.

4.7. Быстродействие процессора

Быстродействие процессора — это одна из важнейших его характеристик, определяющая эффективность работы всей микропроцессорной системы в целом. Быстродействие процессора зависит от множества факторов, что затрудняет сравнение быстродействия даже разных процессоров внутри одного семейства, не говоря уже о процессорах разных фирм и разного назначения.

Выделим важнейшие факторы, влияющие на быстродействие процессора.

Прежде всего, быстродействие зависит от тактовой частоты процессора. Все операции внутри процессора выполняются синхронно, тактируются единым тактовым сигналом. Понятно, что чем больше тактовая частота, тем быстрее работает процессор, причем, например, двукратное увеличение тактовой

частоты какого-то процессора снижает вдвое время выполнения команд этим процессором.

Однако надо учитывать, что разные процессоры выполняют одинаковые команды за разное количество тактов, причем количество тактов, затрачиваемых на команду, может изменяться от одного такта до десятков или даже сотен. В некоторых процессорах за счет распараллеливания микроопераций на команду тратится даже меньше одного такта.

Количество тактов, затрачиваемых на выполнение команды, зависит от сложности этой команды и от методов адресации операндов. Например, быстрее всего (за меньшее число тактов) выполняются команды пересылки данных между внутренними регистрами процессора. Медленнее всего (за большое число тактов) выполняются сложные арифметические команды с плавающей запятой, операнды которых хранятся в памяти.

Первоначально для количественной оценки производительности процессоров применялась единица измерения MIPS (Mega Instruction Per Second), соответствовавшая количеству миллионов выполняемых инструкций (команд) за секунду. Естественно, изготовители микропроцессоров старались ориентироваться на самые быстрые команды. Понятно, что подобный показатель не слишком удачен. Для измерения производительности при выполнении вычислений с плавающей запятой (точкой) чуть позже была предложена единица FLOPS (Floating point Operations Per Second), но она по определению узкоспециальная, так как в некоторых системах операции с плавающей запятой просто не используются.

Другой аналогичный показатель быстродействия процессора — время выполнения коротких (быстрых) операций. Время выполнения команд — важный, но далеко не единственный фактор, определяющий быстродействие. Большое значение имеет также структура системы команд процессора. Например, некоторым процессорам для выполнения какой-то операции

понадобится одна команда, а другим процессорам — несколько команд. Какие-то процессоры имеют систему команд, позволяющую быстро решать задачи одного типа, а какие-то — задачи другого типа. Важны и методы адресации, разрешенные в данном процессоре, и наличие сегментирования памяти, и способы взаимодействия процессора с устройствами ввода/вывода и т.д.

Существенно влияет на быстродействие системы в целом и то, как процессор «общается» с памятью команд и памятью данных, применяется ли совмещение выборки команд из памяти с выполнением ранее выбранных команд.

Быстродействие системы в целом определяется также и разрядностью процессора. Например, 8-разрядный процессор будет медленнее пересылать и обрабатывать большие массивы данных, чем 16-разрядный процессор. Точно так же 16-разрядный процессор будет значительно медленнее работать с большими числами (большими, чем 65536), чем 32-разрядный процессор.

При высокой сложности решаемых задач быстродействие системы зависит и от общего объема системной памяти. Ведь если системной памяти мало, системе приходится сохранять данные во внешней памяти (например, на магнитном диске), а это очень сильно (на несколько порядков) замедляет работу. Так что разрядность шины адреса процессора тоже важна.

Поэтому количественные показатели производительности процессоров очень условны, они лишь косвенно характеризуют быстродействие системы на базе этого процессора. Тем не менее, некоторые производители предлагают количественные показатели для своих процессоров, которые характеризуют время выполнения специально составленных тестовых программ, содержащих самые различные команды в тех или иных соотношениях.

Так, для сравнения производительности 32-разрядных процессоров фирма Intel, производящая процессоры для персональных компьютеров, в 1992 году предложила свою единицу измерения iCOMP Index (Intel COmparative Microprocessor Performance). Для вычисления этого показателя используется

смесь 16- и 32-битных целочисленных команд, команд с плавающей точкой, команд обработки графики и видео. В качестве базового взят процессор i486SX-25, чей индекс принят равным 100. В 1996 году разработчиками Intel был предложен другой показатель — iCOMP Index 2.0, для вычисления которого не используются 16-разрядные команды, зато введен мультимедийный тест, а за базу взят Pentium-120, чей индекс принят равным 100. При этом надо учитывать, что измерения проводятся в составе системы, настроенной на максимальное быстродействие именно данных процессоров, и только самой фирмой Intel.

Ценность этих показателей и всех им подобных не слишком велика. Для конкретного компьютера и разных процессоров величина показателя может предоставить вполне объективные данные, позволяющие оценить, например, целесообразность замены процессора на более мощный. Но усредненность показателей iCOMP не позволяет точно сказать, как будет себя вести процессор в различных задачах, которые ориентированы на преимущественное использование разных типов команд.

Точная оценка быстродействия процессора возможна только в составе конкретной системы при решении определенной задачи. Но все перечисленные здесь факторы можно и нужно учитывать при выборе процессора. А количественные показатели помогают сделать выбор.

5. Функции памяти

Память микропроцессорной системы выполняет функцию временного или постоянного хранения данных и команд. Объем памяти определяет допустимую сложность выполняемых системой алгоритмов, а также в некоторой степени и скорость работы системы в целом. Модули памяти выполняются на микросхемах памяти (оперативной или постоянной). Все чаще в составе микропроцессорных систем используется флэш-память (англ. — flash memory),

которая представляет собой энергонезависимую память с возможностью многократной перезаписи содержимого.

Информация в памяти хранится в ячейках, количество разрядов которых равно количеству разрядов шины данных процессора. Обычно оно кратно восьми (например, 8, 16, 32, 64). Допустимое количество ячеек памяти определяется количеством разрядов шины адреса как 2^N, где N — количество разрядов шины адреса. Чаще всего объем памяти измеряется в байтах независимо от разрядности ячейки памяти. Используются также следующие более крупные единицы объема памяти: килобайт — 2^{10} или 1024 байта (обозначается Кбайт), мегабайт — 2^{20} или 1 048 576 байт (обозначается Мбайт), гигабайт — 2^{30} байт (обозначается Гбайт), терабайт — 2^{40} (обозначается Тбайт). Совокупность ячеек памяти называется обычно пространством памяти системы.

Для подключения модуля памяти к системной магистрали используются блоки сопряжения, которые включают в себя дешифратор (селектор) адреса, схему обработки управляющих сигналов магистрали и буферы данных (рис. 35).

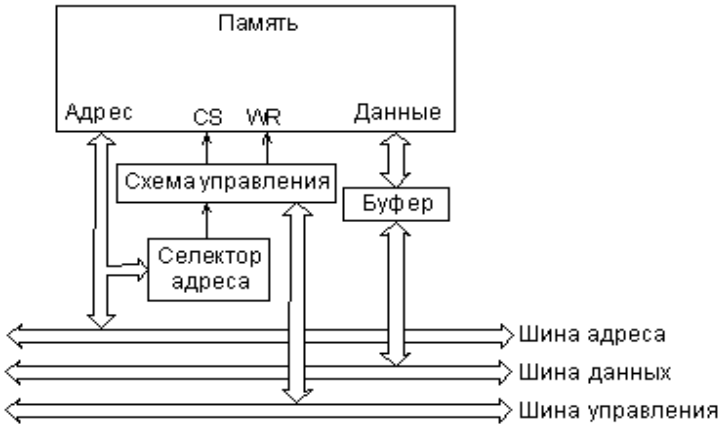


Рис. 35. Структура модуля памяти.

Оперативная память общается с системной магистралью в циклах чтения и записи, постоянная память — только в циклах чтения. Обычно в составе системы имеется несколько модулей памяти, каждый из которых работает в

своей области пространства памяти. Селектор адреса как раз и определяет, какая область адресов пространства памяти отведена данному модулю памяти. Схема управления вырабатывает в нужные моменты сигналы разрешения работы памяти (CS) и сигналы разрешения записи в память (WR). Буферы данных передают данные от памяти к магистрали или от магистрали к памяти.

В пространстве памяти микропроцессорной системы обычно выделяются несколько особых областей, которые выполняют специальные функции.

Память программы начального запуска всегда выполняется на ПЗУ или флэш-памяти. Именно с этой области процессор начинает работу после включения питания и после сброса его с помощью сигнала RESET.

Память для стека или стек (Stack) — это часть оперативной памяти, предназначенная для временного хранения данных в режиме LIFO (Last In — First Out).

Особенность стека по сравнению с другой оперативной памятью — это заданный и неизменяемый способ адресации. При записи любого числа (кода) в стек число записывается по адресу, определяемому как содержимое регистра указателя стека, предварительно уменьшенное (декрементированное) на единицу (или на два, если 16-разрядные слова расположены в памяти по четным адресам). При чтении из стека число читается из адреса, определяемого содержимым указателя стека, после чего это содержимое указателя стека увеличивается (инкрементируется) на единицу (или на два). В результате получается, что число, записанное последним, будет прочитано первым, а число, записанное первым, будет прочитано последним. Такая память называется LIFO или памятью магазинного типа (например, в магазине автомата патрон, установленный последним, будет извлечен первым).

Принцип действия стека показан на рис.36 (адреса ячеек памяти выбраны условно).

Пусть, например, текущее состояние указателя стека 1000008, и в него надо записать два числа (слова). Первое слово будет записано по адресу 1000006 (перед записью указатель стека уменьшится на два). Второе — по адресу 1000004. После записи содержимое указателя стека — 1000004. Если затем прочитать из стека два слова, то первым будет прочитано слово из адреса 1000004, а после чтения указатель стека станет равным 1000006. Вторым будет прочитано слово из адреса 1000006, а указатель стека станет равным 1000008. Все вернулось к исходному состоянию. Первое записанное слово читается вторым, а второе — первым.

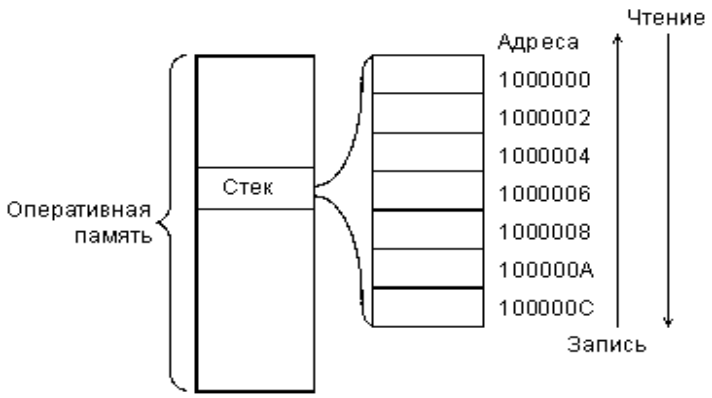


Рис. 36. Принцип работы стека.

Необходимость такой адресации становится очевидной в случае многократно вложенных подпрограмм. Пусть, например, выполняется основная программа, и из нее вызывается подпрограмма 1. Если нам надо сохранить значения данных и внутренних регистров основной программы на время выполнения подпрограммы, мы перед вызовом подпрограммы сохраним их в стеке (запишем в стек), а после ее окончания извлечем (прочитаем) их из стека. Если же из подпрограммы 1 вызывается подпрограмма 2, то ту же самую операцию мы проделаем с данными и содержимым внутренних регистров подпрограммы 1. Понятно, что внутри подпрограммы 2 крайними в стеке (читаемыми в первую очередь) будут данные из подпрограммы 1, а данные из основной

программы будут глубже. При этом в случае чтения из стека автоматически будет соблюдаться нужный порядок читаемой информации. То же самое будет и в случае, когда таких уровней вложения подпрограмм гораздо больше. То есть то, что надо хранить подольше, прячется глубже, а то, что скоро может потребоваться — с краю.

В системе команд любого процессора для обмена информацией со стеком предусмотрены специальные команды записи в стек (PUSH) и чтения из стека (POP). В стеке можно прятать не только содержимое всех внутренних регистров процессоров, но и содержимое регистра признаков (слово состояния процессора, PSW). Это позволяет, например, при возвращении из подпрограммы контролировать результат последней команды, выполненной непосредственно перед вызовом этой подпрограммы. Можно также хранить в стеке и данные, для того чтобы удобнее было передавать их между программами и подпрограммами. В общем случае, чем больше область памяти, отведенная под стек, тем больше свободы у программиста и тем более сложные программы могут выполняться.

Следующая специальная область памяти — это **таблица векторов прерываний.**

Вообще, понятие прерывания довольно многозначно. Под прерыванием в общем случае понимается не только обслуживание запроса внешнего устройства, но и любое нарушение последовательной работы процессора. Например, может быть предусмотрено прерывание по факту некорректного выполнения арифметической операции типа деления на ноль. Или же прерывание может быть программным, когда в программе используется команда перехода на какую-то подпрограмму, из которой затем последует возврат в основную программу. В последнем случае общее с истинным прерыванием только то, как осуществляется переход на подпрограмму и возврат из нее.

Любое прерывание обрабатывается через таблицу векторов (указателей) прерываний. В этой таблице в простейшем случае находятся адреса начала программ обработки прерываний, которые и называются векторами. Длина таблицы может быть довольно большой (до нескольких сот элементов). Обычно таблица векторов прерываний располагается в начале пространства памяти (в ячейках памяти с малыми адресами). Адрес каждого вектора (или адрес начального элемента каждого вектора) представляет собой номер прерывания.

В случае аппаратных прерываний номер прерывания или задается устройством, запросившим прерывание (при векторных прерываниях), или же задается номером линии запроса прерываний (при радиальных прерываниях). Процессор, получив аппаратное прерывание, заканчивает выполнение текущей команды и обращается к памяти в область таблицы векторов прерываний, в ту ее строку, которая определяется номером запрошенного прерывания. Затем процессор читает содержимое этой строки (код вектора прерывания) и переходит в адрес памяти, задаваемый этим вектором. Начиная с этого адреса, в памяти должна располагаться программа обработки прерывания с данным номером. В конце программы обработки прерываний обязательно должна располагаться команда выхода из прерывания, выполнив которую, процессор возвращается к выполнению прерванной основной программы. Параметры процессора на время выполнения программы обработки прерывания сохраняются в стеке.

Пусть, например, процессор (рис. 37) выполнял основную программу и команду, находящуюся в адресе памяти 5000 (условно). В этот момент он получил запрос прерывания с номером (адресом вектора) 4. Процессор заканчивает выполнение команды из адреса 5000. Затем он сохраняет в стеке текущее значение счетчика команд (5001) и текущее значение PSW. После этого процессор читает из адреса 4 памяти код вектора прерывания. Пусть этот

код равен 6000. Процессор переходит в адрес памяти 6000 и приступает к выполнению программы обработки прерывания, начинающейся с этого адреса. Пусть эта программа заканчивается в адресе 6100. Дойдя до этого адреса, процессор возвращается к выполнению прерванной программы. Для этого он извлекает из стека значение адреса (5001), на котором его прервали, и бывшее в тот момент PSW. Затем процессор читает команду из адреса 5001 и дальше последовательно выполняет команды основной программы.

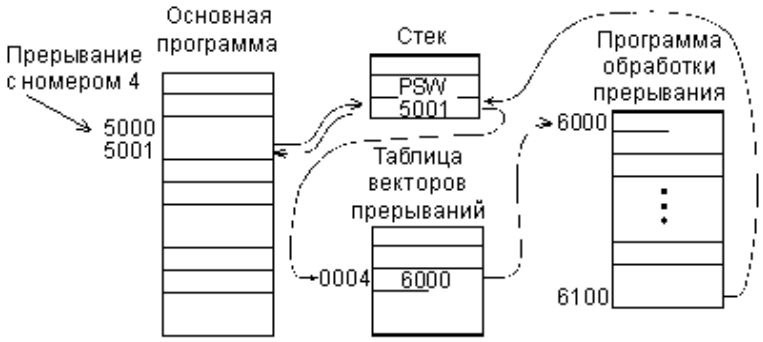


Рис. 37. Упрощенный алгоритм обработки прерывания.

Прерывание в случае аварийной ситуации обрабатывается точно так же, только адрес вектора прерывания (номер строки в таблице векторов) жестко привязан к данному типу аварийной ситуации.

Программное прерывание тоже обслуживается через таблицу векторов прерываний, но номер прерывания указывается в составе команды, вызывающей прерывание.

Такая сложная, на первый взгляд, организация прерываний позволяет программисту легко менять программы обработки прерываний, располагать их в любой области памяти, делать их любого размера и любой сложности.

Во время выполнения программы обработки прерывания может поступить новый запрос на прерывание. В этом случае он обрабатывается точно так же, как описано, но основной программой считается прерванная программа обработки предыдущего прерывания. Это называется многократным

вложением прерываний. Механизм стека позволяет без проблем обслуживать это многократное вложение, так как первым из стека извлекается тот код, который был сохранен последним, то есть возврат из обработки данного прерывания происходит в программу обработки предыдущего прерывания.

Отметим, что в более сложных случаях в таблице векторов прерываний могут находиться не адреса начала программ обработки прерываний, а так называемые дескрипторы (описатели) прерываний. Но конечным результатом обработки этого дескриптора все равно будет адрес начала программы обработки прерываний.

С точки зрения реализации механизма доступа к стековой памяти выделяют аппаратный и аппаратно-программный (внешний) стеки.

Аппаратный стек представляет собой совокупность регистров, связи между которыми организованы таким образом, что при записи и считывании данных содержимое стека автоматически сдвигается. Обычно емкость аппаратного стека ограничена диапазоном от нескольких регистров до нескольких десятков регистров, поэтому в большинстве МП такой стек используется для хранения содержимого программного счетчика и его называют стеком команд. Основное достоинство аппаратного стека - высокое быстродействие, а недостаток - ограниченная емкость.

Наиболее распространенным в настоящее время и, возможно, лучшим вариантом организации стека является использование области памяти. Для адресации стека используется указатель стека, который предварительно загружается в регистр и определяет адрес последней занятой ячейки. Помимо команд CALL и RET, по которым записывается в стек и восстанавливается содержимое программного счетчика, имеются команды PUSH и POP, которые используются для временного запоминания в стеке содержимого регистров и их восстановления, соответственно. В некоторых МП содержимое основных регистров запоминается в стеке автоматически при прерывании программ.

Содержимое регистра указателя стека при записи уменьшается, а при считывании увеличивается на 1 при выполнении команд PUSH и POP, соответственно.

Наконец, еще одна специальная область памяти микропроцессорной системы — это память устройств, подключенных к системной шине. Такое решение встречается нечасто, но иногда оно очень удобно. То есть процессор получает возможность обращаться к внутренней памяти устройств ввода/вывода или каких-то еще подключенных к системной шине устройств, как к своей собственной системной памяти. Обычно окно в пространстве памяти, выделяемое для этого, не слишком большое.

Все остальные части пространства памяти, как правило, имеют универсальное назначение. В них могут располагаться как данные, так и программы (конечно, в случае одношинной архитектуры). Иногда это пространство памяти используется как единое целое, без всяких границ. А иногда пространство памяти делится на сегменты с программно изменяемым адресом начала сегмента и с установленным размером сегмента. Оба подхода имеют свои плюсы и минусы. Например, использование сегментов позволяет защитить область программ или данных, но зато границы сегментов могут затруднять размещение больших программ и массивов данных.

В заключение остановимся на проблеме разделения адресов памяти и адресов устройств ввода/вывода. Существует два основных подхода к решению этой проблемы:

--- выделение в общем адресном пространстве системы специальной области адресов для устройств ввода/вывода;

--- полное разделение адресных пространств памяти и устройств ввода/вывода.

Первый подход хорош тем, что при обращении к устройствам ввода/вывода процессор может использовать те же команды, которые служат для взаимодействия с памятью. Но адресное пространство памяти должно быть

уменьшено на величину адресного пространства устройств ввода/вывода. Например, при 16-разрядной шине адреса всего может быть 64К адресов. Из них 56К адресов отводится под адресное пространство памяти, а 8К адресов — под адресное пространство устройств ввода/вывода.

Преимущество второго подхода состоит в том, что память занимает все адресное пространство микропроцессорной системы. Для общения с устройствами ввода/вывода применяются специальные команды и специальные стробы обмена на магистрали. Именно так сделано, например, в персональных компьютерах. Но возможности взаимодействия с устройствами ввода/вывода в данном случае существенно ограничены по сравнению с возможностями общения с памятью.

5.1. Буферная память

В микропроцессорных системах используются подсистемы с различным быстродействием, и, в частности, с различной скоростью передачи данных (рис. 38). Обычно обмен данными между такими подсистемами реализуется с использованием прерываний или канала прямого доступа к памяти. В первую очередь подсистема 1 формирует запрос на обслуживание по мере готовности данных к обмену. Однако обслуживание прерываний связано с непроизводительными потерями времени и при пакетном обмене производительность подсистемы 2 заметно уменьшается. При обмене данными с использованием канала прямого доступа к памяти подсистема 1 передает данные в память подсистемы 2. Данный способ обмена достаточно эффективен с точки зрения быстродействия, но для его реализации необходим довольно сложный контроллер прямого доступа к памяти.

Наиболее эффективно обмен данными между подсистемами с различным быстродействием реализуется при наличии между ними специальной буферной памяти. Данные от подсистемы 1 временно запоминаются в буферной памяти

Рис. 38. Применение буферной памяти.

до готовности подсистемы 2 принять их. Емкость буферной памяти должна быть достаточной для хранения тех блоков данных, которые подсистема 1 формирует между считываниями их подсистемой 2. Отличительной особенностью буферной памяти является запись данных с быстродействием и под управлением подсистемы 1, а считывание - с быстродействием и под управлением подсистемы 2 ("эластичная память"). В общем случае память должна выполнять операции записи и считывания совершенно независимо и даже одновременно, что устраняет необходимость синхронизации подсистем. Буферная память должна сохранять порядок поступления данных от подсистемы 1, т.е. работать по принципу "первое записанное слово считывается первым" (First Input First Output - FIFO). Таким образом, под буферной памятью типа FIFO понимается ЗУПВ, которое автоматически следит за порядком поступления данных и выдает их в том же порядке, допуская выполнение независимых и одновременных операций записи и считывания. На рис. 39 приведена структурная схема буферной памяти типа FIFO емкостью 64x4.

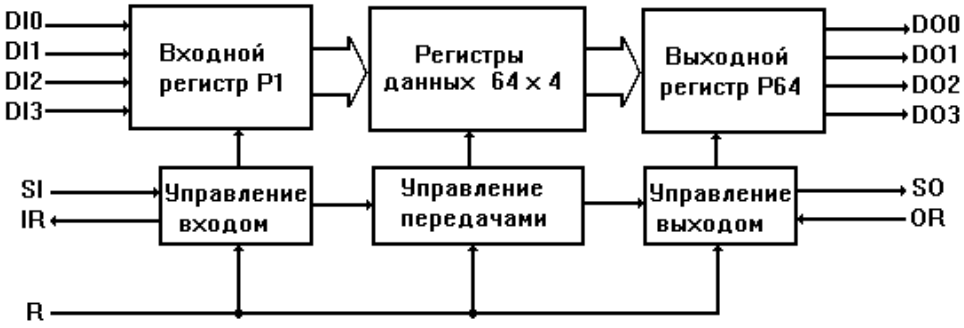


Рис. 39. Структурная схема буфера 64x4.

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

При вводе 4-битного слова под действием сигнала SI оно автоматически передвигается в ближайший к выходу свободный регистр. Состояние регистра данных отображается в соответствующем ему управляющем триггере, совокупность триггеров образует 64-битный управляющий регистр. Если регистр содержит данные, то управляющий триггер находится в состоянии 1, а если регистр не содержит данных, то триггер находится в состоянии 0. Как только управляющий бит соседнего справа регистра изменяется на 0, слово данных автоматически сдвигается к выходу. Перед началом работы в буфер подается сигнал сброса R и все управляющие триггеры переводятся в состояние 0 (все регистры буфера свободны). На выводе IR формируется логическая 1, т.е. буфер готов воспринимать входные данные. При действии сигнала ввода SI входное слово загружается в регистр P1, а управляющий триггер этого регистра устанавливается в состояние 1: на входе IR формируется логический 0. Связи между регистрами организованы таким образом, что поступившее в P1 слово "спонтанно" копируется во всех регистрах данных FIFO и появляется на выходных линиях DO0-DO3. Теперь все 64 регистра буфера содержат одинаковые слова, управляющий триггер последнего регистра P64 находится в состоянии 1, а остальные управляющие триггеры сброшены при передаче данных в соседние справа регистры. Состояние управляющего триггера P64

выведено на линию готовности выхода OR; OR принимает значение 1, когда в триггер записывается 1. Процесс ввода может продолжаться до полного заполнения буфера; в этом случае все управляющие триггеры находятся в состоянии 1 и на линии IR сохраняется логический 0.

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

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

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

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

5.2. Кэш-память.

Кэш-память (или просто кэш, от англ. Cache — склад, тайник) предназначена для промежуточного хранения информации из системной памяти с целью ускорения доступа к ней. Ускорение достигается за счет использования более

быстрой памяти и более быстрого доступа к ней. При этом в кэш-памяти хранится постоянно обновляемая копия некоторой области основной памяти.

Необходимость введения кэша связана с тем, что системная память персонального компьютера выполняется на микросхемах динамической памяти, которая характеризуется меньшей стоимостью, но и более низким быстродействием, по сравнению со статической памятью. Идея состоит в том, что благодаря введению быстрой буферной, промежуточной статической памяти можно ускорить обмен с медленной динамической памятью. По сути, кэш-память делает то же, что и применявшийся ранее конвейер команд, но на более высоком уровне. В кэш-памяти хранится копия некоторой части системной памяти, и процессор может обмениваться с этой частью памяти гораздо быстрее, чем с системной памятью. Причем в кэш-памяти могут храниться как команды, так и данные.

Выигрыш в быстродействии от применения кэша связан с тем, что процессор в большинстве случаев обращается к адресам памяти, расположенным последовательно, один за другим, или же близко друг к другу. Поэтому высока вероятность того, что информация из этих адресов памяти окажется внутри небольшой кэш-памяти. Если же процессор обращается к адресу, расположенному далеко от тех, к которым он обращался ранее, кэш оказывается бесполезным и требует перезагрузки, что может даже замедлить обмен по сравнению со структурой без кэш-памяти.

В принципе кэш-память может быть как внутренней (входить в состав процессора), так и внешней. Внутренний кэш называется кэшем первого уровня, внешний — кэшем второго уровня. Объем внутреннего кэша обычно невелик — типовое значение 32 Кбайт. Объем внешнего кэша может достигать нескольких мегабайт. Но принцип функционирования у них один и тот же.

Кэш первого уровня процессора 486 имеет четырехканальную структуру (рис. 40). Каждый канал состоит из 128 строк по 16 байт в каждой. Одноименные

строки всех четырех каналов образуют 128 наборов из четырех строк, каждый из которых обслуживает свои адреса памяти. Каждой строке соответствует 21-разрядная информация об адресе скопированного в нее блока системной памяти. Эта информация называется **тегом** (Tag) строки.

Кроме того, в состав кэша входит так называемый диспетчер, то есть область памяти с организацией 128 x 7, в которой хранятся 4-битные теги действительности (достоверности) для каждого из 128 наборов и 3-битные коды `LRU` (Least Recently Used) для каждого из 128 наборов. Тег действительности набора включает в себя 4 бита достоверности каждой из 4 строк, входящих в данный набор. Бит достоверности, установленный в

Рис. 40. Структура внутреннего кэша процессора 486.

единицу, говорит о том, что соответствующая строка заполнена; если он сброшен в нуль, то строка пуста. Биты LRU говорят о том, как давно было обращение к данному набору. Это нужно для того, чтобы обновлять наименее используемые наборы.

Адресация кэш-памяти осуществляется с помощью 28 разрядов адреса. Из них 7 младших разрядов выбирают один из 128 наборов, а 21 старший разряд сравнивается с тегами всех 4 строк выбранного набора. Если теги совпадают с разрядами адреса, то получается ситуация кэш-попадания, а если нет, то ситуация кэш-промаха.

В случае цикла чтения при кэш-попадании байт или слово читаются из кэш-памяти. При кэш-промахе происходит обновление (перезагрузка) одной из строк кэш-памяти.

В случае цикла записи при кэш-попадании производится запись как в кэш-память, так и в основную системную память. При кэш-промахе запись производится только в системную память, а обновление строки кэш-памяти не производится. Эта строка становится недостоверной (ее бит достоверности сбрасывается в нуль).

Такая политика записи называется сквозной или прямой записью (Write Through). В более поздних моделях процессоров применяется и обратная запись (Write Back), которая является более быстрой, так как требует гораздо меньшего числа обращений по внешней шине.

При использовании обратной записи в основную память записываемая информация отправляется только в том случае, когда нужной строки в кэше нет. В случае же попадания модифицируется только кэш. В основную память измененная информация попадет только при перезаписи новой строки в кэш. Прежняя строка при этом целиком переписывается в основную память, и тем самым восстанавливается идентичность содержимого кэша и основной памяти.

В случае, когда требуемая строка в кэше не представлена (ситуация кэш-промаха), запрос на запись направляется на внешнюю шину, а запрос на чтение обрабатывается несколько сложнее. Если этот запрос относится к кэшируемой области памяти, то выполняется цикл заполнения целой строки кэша (16 байт из памяти переписывается в одну из строк набора, обслуживающего данный адрес). Если затребованные данные не укладываются в одной строке, то заполняется и соседняя строка. Заполнение строки процессор старается выполнить самым быстрым способом — пакетным циклом, однако внешний контроллер памяти может потребовать использования более медленных пересылок.

Внутренний запрос процессора на данные удовлетворяется сразу, как только данные считываются из памяти, а дальнейшее заполнение строки может идти параллельно с обработкой данных. Если в наборе, который обслуживает данный адрес памяти, имеется свободная строка, заполнена будет именно она. Если же свободных строк нет, заполняется строка, к которой дольше всех не обращались. Для этого используются биты LRU, которые модифицируются при каждом обращении к строке данного набора.

Кроме того, существует возможность аннулирования строк (объявления их недостоверными) и очистки всей кэш-памяти. При сквозной записи очистка кэша проводится специальным внешним сигналом процессора, программным образом с помощью специальных команд, а также при начальном сбросе – по сигналу `RESET`. При обратной записи очистка кэша подразумевает также выгрузку всех модифицированных строк в основную память.

Отметим, что в пространстве памяти персонального компьютера имеются области, для которых кэширование принципиально недопустимо (например, разделяемая память аппаратурных адаптеров — плат расширения).

Режим пакетной передачи (Burst Mode), впервые появившийся в процессоре 486, предназначен для быстрых операций со строками кэша. Пакетный цикл обмена (Burst Cycle) отличается тем, что для пересылки всего пакета адрес по внешней шине адреса передается только один раз — в начале пакета, а затем в каждом следующем такте передаются только данные. Адрес для каждого следующего кода данных вычисляется из начального адреса по правилам, установленным как передатчиком данных, так и их приемником. Например, адрес каждого следующего слова данных вычисляется как инкрементированный адрес предыдущего. В результате время передачи одного слова данных значительно сокращается. Понятно, что обмен пакетными циклами возможен только с устройствами, изначально способными

обслуживать такой цикл. Допустимая длина пакета не слишком велика, например, при чтении размер пакета ограничен одной строкой кэша.

6. Способы обмена информацией в микропроцессорной системе

6.1. Режимы работы микропроцессорной системы

Как уже отмечалось, микропроцессорная система обеспечивает большую гибкость работы, она способна настраиваться на любую задачу. Гибкость эта обусловлена прежде всего тем, что функции, выполняемые системой, определяются программой (программным обеспечением, software), которую выполняет процессор. Аппаратура (аппаратное обеспечение, hardware) остается неизменной при любой задаче. Записывая в память системы программу, можно заставить микропроцессорную систему выполнять любую задачу, поддерживаемую данной аппаратурой. К тому же шинная организация связей микропроцессорной системы позволяет довольно легко заменять аппаратные модули, например, заменять память на новую большего объема или более высокого быстродействия, добавлять или модернизировать устройства ввода/вывода, наконец, заменять процессор на более мощный. Это также позволяет увеличить гибкость системы, продлить ее жизнь при любом изменении требований к ней. Но гибкость микропроцессорной системы определяется не только этим. Настраиваться на задачу помогает еще и выбор режима работы системы, то есть режима обмена информацией по системной магистрали (шине). Практически любая развитая микропроцессорная система (в том числе и компьютер) поддерживает три основных режима обмена по магистрали:

--- программный обмен информацией;

--- обмен с использованием прерываний (Interrupts);

--- обмен с использованием прямого доступа к памяти (ПДП, DMA — Direct Memory Access).

Программный обмен информацией является основным в любой микропроцессорной системе. Он предусмотрен всегда, без него невозможны другие режимы обмена. В этом режиме процессор является единоличным хозяином (задатчиком, Master) системной магистрали. Все операции (циклы) обмена информацией в данном случае инициируются только процессором, все они выполняются строго в порядке, предписанном исполняемой программой. Процессор читает (выбирает) из памяти коды команд и исполняет их, читая данные из памяти или из устройства ввода/вывода, обрабатывая их, записывая данные в память или передавая их в устройство ввода/вывода. Путь процессора по программе может быть линейным, циклическим, может содержать переходы (прыжки), но он всегда непрерывен и полностью находится под контролем процессора. Ни на какие внешние события, не связанные с программой, процессор не реагирует (рис. 41). Все сигналы на магистрали в данном случае контролируются процессором.

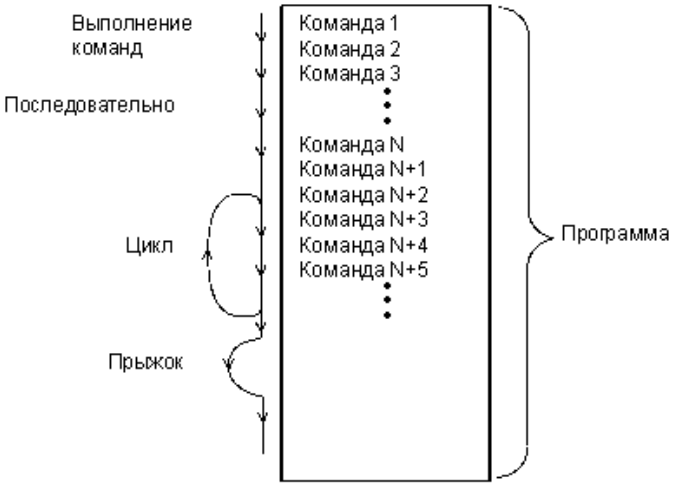


Рис. 41. Программный обмен информацией.

Обмен по прерываниям используется тогда, когда необходима реакция микропроцессорной системы на какое-то внешнее событие, на приход

внешнего сигнала. В случае компьютера, внешним событием может быть, например, нажатие на клавишу клавиатуры или приход по локальной сети пакета данных. Компьютер должен реагировать на это, соответственно, выводом символа на экран или же чтением и обработкой принятого по сети пакета.

В общем случае организовать реакцию на внешнее событие можно тремя различными путями:

--- с помощью постоянного программного контроля факта наступления события (так называемый метод опроса флага или polling);

--- с помощью прерывания, то есть насильственного перевода процессора с выполнения текущей программы на выполнение экстренно необходимой программы;

--- с помощью прямого доступа к памяти (ПДП), то есть без участия процессора при его отключении от системной магистрали.

Проиллюстрировать эти три способа можно следующим простым примером. Допустим, вы готовите себе завтрак, поставив на плиту кипятиться молоко. Естественно, на закипание молока надо реагировать, причем срочно. Как это организовать? Первый путь — постоянно следить за молоком, но тогда вы ничего другого не сможете делать. Правильнее будет регулярно поглядывать на молоко, делая одновременно что-то другое. Это программный режим с опросом флага. Второй путь — установить на кастрюлю с молоком датчик, который подаст звуковой сигнал при закипании молока, и спокойно заниматься другими делами. Услышав сигнал, вы выключите молоко. Правда, возможно, вам придется сначала закончить то, что вы начали делать, так что ваша реакция будет медленнее, чем в первом случае. Наконец, третий путь состоит в том, чтобы соединить датчик на кастрюле с управлением плитой так, чтобы при закипании молока горелка была выключена без вашего участия (правда,

аналогия с ПДП здесь не очень точная, так как в данном случае на момент выполнения действия вас не отвлекают от работы).

Первый случай с опросом флага реализуется в микропроцессорной системе постоянным чтением информации процессором из устройства ввода/вывода, связанного с тем внешним устройством, на поведение которого необходимо срочно реагировать.

Во втором случае в режиме прерывания процессор, получив запрос прерывания от внешнего устройства (часто называемый IRQ — Interrupt ReQuest), заканчивает выполнение текущей команды и переходит к программе обработки прерывания. Закончив выполнение программы обработки прерывания, он возвращается к прерванной программе с той точки, где его прервали (рис. 42).

Здесь важно то, что вся работа, как и в случае программного режима, осуществляется самим процессором, внешнее событие просто временно отвлекает его. Реакция на внешнее событие по прерыванию в общем случае медленнее, чем при программном режиме. Как и в случае программного обмена, здесь все сигналы на магистрали выставляются процессором, то есть он полностью контролирует магистраль. Для обслуживания прерываний в систему иногда вводится специальный модуль контроллера прерываний, но он в обмене информацией не участвует. Его задача состоит в том, чтобы упростить работу процессора с внешними запросами прерываний. Этот контроллер обычно программно управляется процессором по системной магистрали.

Естественно, никакого ускорения работы системы прерывание не дает. Его применение позволяет только отказаться от постоянного опроса флага внешнего события и временно, до наступления внешнего события, занять процессор выполнением каких-то других задач.

Прямой доступ к памяти (ПДП, DMA) — это режим, принципиально отличающийся от двух ранее рассмотренных режимов тем, что обмен по

системной шине идет без участия процессора. Внешнее устройство, требующее обслуживания, сигнализирует процессору, что необходим режим ПДП. В ответ

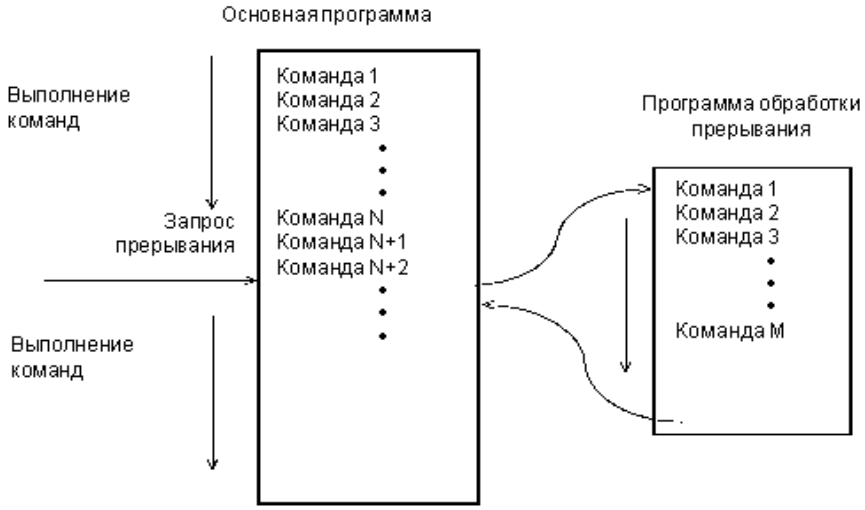


Рис. 42. Обслуживание прерывания.

на это процессор заканчивает выполнение текущей команды и отключается от всех шин, сигнализируя запросившему устройству, что обмен в режиме ПДП можно начинать.

Операция ПДП сводится к пересылке информации из устройства ввода/вывода в память или же из памяти в устройство ввода/вывода. Когда пересылка информации будет закончена, процессор вновь возвращается к прерванной программе, продолжая ее с той точки, где его прервали (рис. 43). Это похоже на режим обслуживания прерываний, но в данном случае процессор не участвует в обмене. Как и в случае прерываний, реакция на внешнее событие при ПДП существенно медленнее, чем при программном режиме.

Понятно, что в этом случае требуется введение в систему дополнительного устройства (контроллера ПДП), которое будет осуществлять полноценный обмен по системной магистрали без всякого участия процессора. Причем процессор предварительно должен сообщить этому контроллеру ПДП, откуда ему следует брать информацию и/или куда ее следует помещать. Контроллер ПДП может считаться специализированным процессором, который отличается

тем, что сам не участвует в обмене, не принимает в себя информацию и не выдает ее (рис. 44).

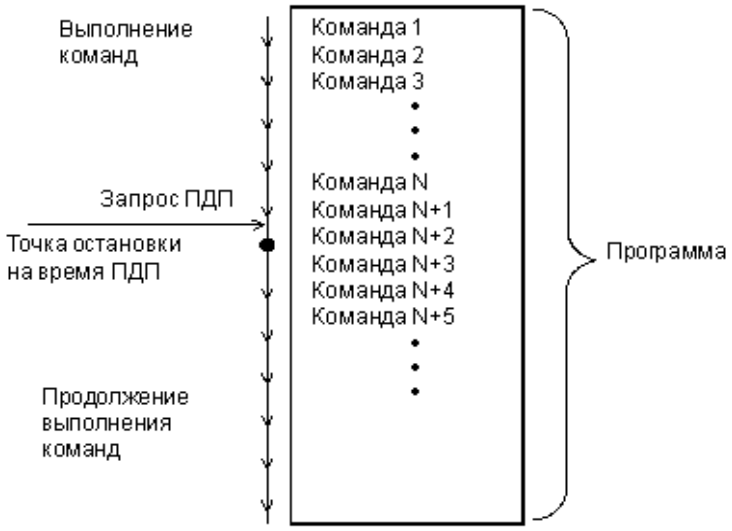


Рис. 43. Обслуживание ПДП.

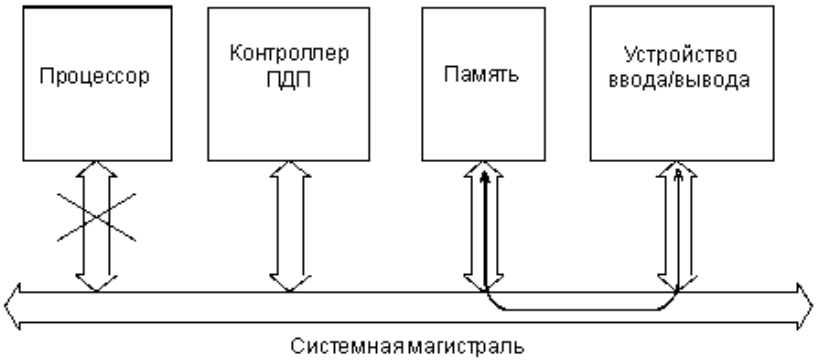


Рис. 44. Информационные потоки в режиме ПДП.

В принципе контроллер ПДП может входить в состав устройства ввода/вывода, которому необходим режим ПДП или даже в состав нескольких устройств ввода/вывода. Теоретически обмен с помощью прямого доступа к памяти может обеспечить более высокую скорость передачи информации, чем программный обмен, так как процессор передает данные медленнее, чем специализированный контроллер ПДП. Однако на практике это преимущество реализуется далеко не всегда. Скорость обмена в режиме ПДП обычно ограничена возможностями магистрали. К тому же необходимость

программного задания режимов контроллера ПДП может свести на нет выигрыш от более высокой скорости пересылки данных в режиме ПДП. Поэтому режим ПДП применяется редко.

Если в системе уже имеется самостоятельный контроллер ПДП, то это может в ряде случаев существенно упростить аппаратуру устройств ввода/вывода, работающих в режиме ПДП. В этом, пожалуй, состоит единственное бесспорное преимущество режима ПДП.

6.2. Способы обмена информацией в микропроцессорной системе

В МПС применяются три режима ввода/вывода:

--- программно-управляемый ввод/вывод (ВВ) (называемый также программным или неформированным ВВ),

--- ввод/вывод по прерываниям (форсированный ВВ)

--- прямой доступ к памяти.

Первый из них характеризуется тем, что инициирование и управление ВВ осуществляется программой, выполняемой процессором, а внешние устройства играют сравнительно пассивную роль и сигнализируют только о своем состоянии, в частности, о готовности к операциям ввода/вывода.

Во втором режиме ВВ инициируется не процессором, а внешним устройством, генерирующим специальный сигнал прерывания. Реагируя на этот сигнал готовности устройства к передаче данных, процессор передает управление подпрограмме обслуживания устройства, вызвавшего прерывание. Действия, выполняемые этой подпрограммой, определяются пользователем, а непосредственными операциями ВВ управляет процессор.

В режиме прямого доступа к памяти, который используется, когда пропускной способности процессора недостаточно, действия процессора приостанавливаются, он отключается от системной шины и не участвует в

передачах данных между основной памятью и быстродействующим ВУ. Заметим, что во всех вышеуказанных случаях основные действия, выполняемые на системной магистрали МПС, подчиняются двум основным принципам:

1. В процессе взаимодействия любых двух устройств МПС одно из них обязательно выполняет активную, управляющую роль и является задатчиком, второе оказывается управляемым, исполнителем. Чаще всего задатчиком является процессор.

2. Другим важным принципом, заложенным в структуру интерфейса, является принцип квитирования (запроса - ответа): каждый управляющий сигнал, посланный задатчиком, подтверждается сигналом исполнителя. При отсутствии ответного сигнала исполнителя в течение заданного интервала времени формируется так называемый тайм-аут, задатчик фиксирует ошибку обмена и прекращает данную операцию.

Программно-управляемый ввод/вывод характеризуется тем, что все действия по вводу/выводу реализуются командами прикладной программы. Наиболее простыми эти действия оказываются для "всегда готовых" внешних устройств, например индикатора на светодиодах. При необходимости ВВ в соответствующем месте программы используются команды IN или OUT. Такая передача данных называется синхронным или безусловным ВВ.

Однако для большинства ВУ до выполнения операций ВВ надо убедиться в их готовности к обмену, т.е. ВВ является асинхронным. Общее состояние устройства характеризуется флагом готовности READY, называемым также флагом готовности/занятости (READY/BUSY). Иногда состояния готовности и занятости идентифицируются отдельными флагами READY и BUSY, входящими в слово состояния устройства.

Процессор проверяет флаг готовности с помощью одной или нескольких команд. Если флаг установлен, то инициируются собственно ввод или вывод

одного или нескольких слов данных. Когда же флаг сброшен, процессор выполняет цикл из 2-3 команд с повторной проверкой флага READY до тех пор, пока устройство не будет готово к операциям ВВ. Данный цикл называется циклом ожидания готовности ВУ и реализуется в различных процессорах по-разному.

Основной недостаток программного ВВ связан с непроизводительными потерями времени процессора в циклах ожидания. К достоинствам следует отнести простоту его реализации, не требующей дополнительных аппаратных средств.

6.3. Организация прерываний в МПС

Одной из разновидностей программно-управляемого обмена данными с внешним устройством (ВУ) в МПС является обмен с прерыванием программы, отличающийся от асинхронного программно-управляемого обмена тем, что переход к выполнению команд, физически реализующих обмен данными, осуществляется с помощью специальных аппаратных средств. Команды обмена данными в этом случае выделяют в отдельный программный модуль - подпрограмму обработки прерывания. Задачей аппаратных средств обработки прерывания в процессоре МПС как раз и является приостановка выполнения одной программы (ее еще называют основной программой) и передача управления подпрограмме обработки прерывания. Действия, выполняемые при этом процессором, как правило, те же, что и при обращении к подпрограмме. Только при обращении к подпрограмме они инициируются командой, а при обработке прерывания - управляющим сигналом от ВУ, который называют "Запрос на прерывание" или "Требование прерывания".

Эта важная особенность обмена с прерыванием программы позволяет организовать обмен данными с ВУ в произвольные моменты времени, не

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

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

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

Формирование сигналов прерываний - запросов ВУ на обслуживание происходит в контроллерах соответствующих ВУ. В простейших случаях в качестве сигнала прерывания может использоваться сигнал "Готовность ВУ", поступающий из контроллера ВУ в системный интерфейс МПС. Однако такое простое решение обладает существенным недостатком - процессор не имеет возможности управлять прерываниями, т. е. разрешать или запрещать их для отдельных ВУ. В результате организация обмена данными в режиме прерывания с несколькими ВУ существенно усложняется.

Для решения этой проблемы регистр состояния и управления контроллера ВУ (рис. 45) дополняют еще одним разрядом - "Разрешение прерывания". Запись 1 или 0 в разряд "Разрешение прерывания" производится программным путем по одной из линий шины данных системного интерфейса. Управляющий сигнал

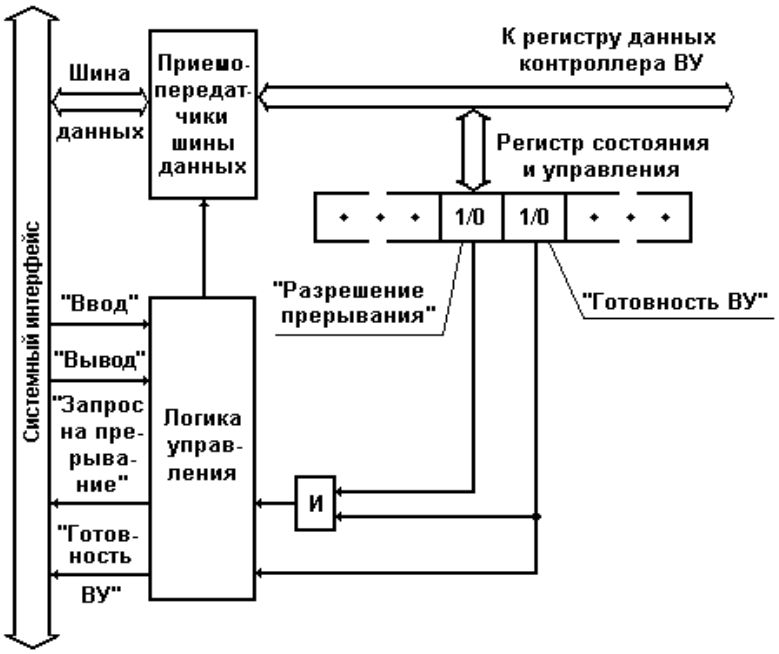


Рис. 45. Фрагмент блок-схемы контроллера ВУ с разрядом "Разрешение прерывания" в регистре состояния и управления

системного интерфейса "Запрос на прерывание" формируется с помощью схемы совпадения только при наличии единиц в разрядах "Готовность ВУ" и "Разрешение прерывания" регистра состояния и управления контроллера.

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

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

--- программный опрос регистров состояния (разряд "Готовность ВУ") контроллеров всех ВУ;

--- использование векторов прерывания.

Организация прерываний с программным опросом готовности предполагает наличие в памяти МПС единой подпрограммы обслуживания прерываний от всех внешних устройств. Структура такой подпрограммы приведена на рис. 46.

Обслуживание ВУ с помощью единой подпрограммы обработки прерываний производится следующим образом. В конце последнего машинного цикла выполнения очередной команды основной программы процессор проверяет наличие требования прерывания от ВУ. Если сигнал прерывания есть и в процессоре прерывание разрешено, то процессор переключается на выполнение подпрограммы обработки прерываний.

После сохранения содержимого регистров процессора, используемых в подпрограмме, начинается последовательный опрос регистров состояния

контроллеров всех ВУ, работающих в режиме прерывания. Как только подпрограмма обнаружит готовое к обмену ВУ, сразу выполняются действия по его обслуживанию. Завершается подпрограмма обработки прерывания после опроса готовности всех ВУ и восстановления содержимого регистров процессора.

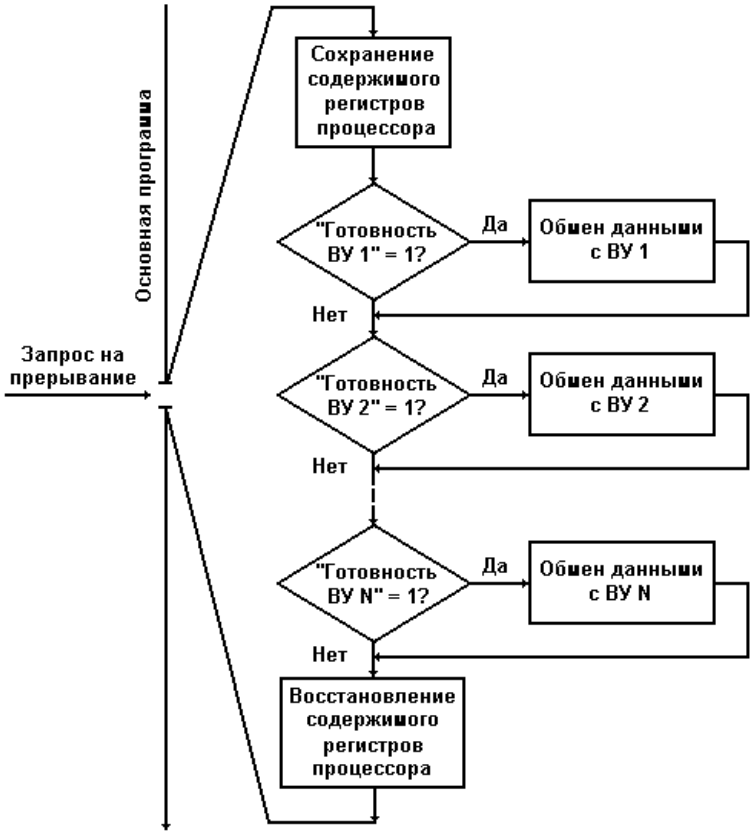


Рис. 46. Структура единой программы обработки прерываний и ее связь с основной программой.

по его обслуживанию. Завершается подпрограмма обработки прерывания после опроса готовности всех ВУ и восстановления содержимого регистров процессора.

Приоритет ВУ в МПС с программным опросом готовности внешнего устройства однозначно определяется порядком их опроса в подпрограмме обработки прерываний. Чем раньше в подпрограмме опрашивается готовность ВУ, тем меньше время реакции на его запрос и выше приоритет.

Необходимость проверки готовности всех внешних устройств существенно увеличивает время обслуживания тех ВУ, которые опрашиваются последними. Это является основным недостатком рассматриваемого способа организации прерываний. Поэтому обслуживание прерываний с опросом готовности ВУ используется только в тех случаях, когда отсутствуют жесткие требования на время обработки сигналов прерывания внешних устройств.

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

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

Рассмотрим организацию векторной системы с интерфейсным вектором. Вектор прерывания выдается контроллером не одновременно с запросом на прерывание, а только по разрешению процессора, как это реализовано в схеме на рис. 47.

Это делается для того, чтобы исключить одновременную выдачу векторов прерывания от нескольких ВУ. В ответ на сигнал контроллера ВУ "Запрос на

прерывание" процессор формирует управляющий сигнал "Предоставление прерывания (вх)", который разрешает контроллеру ВУ, запросившему обслуживание, выдачу вектора прерывания в шину адреса системного интерфейса. Для этого в контроллере используются регистр вектора прерывания и схема совпадения И3. Регистр вектора прерывания обычно

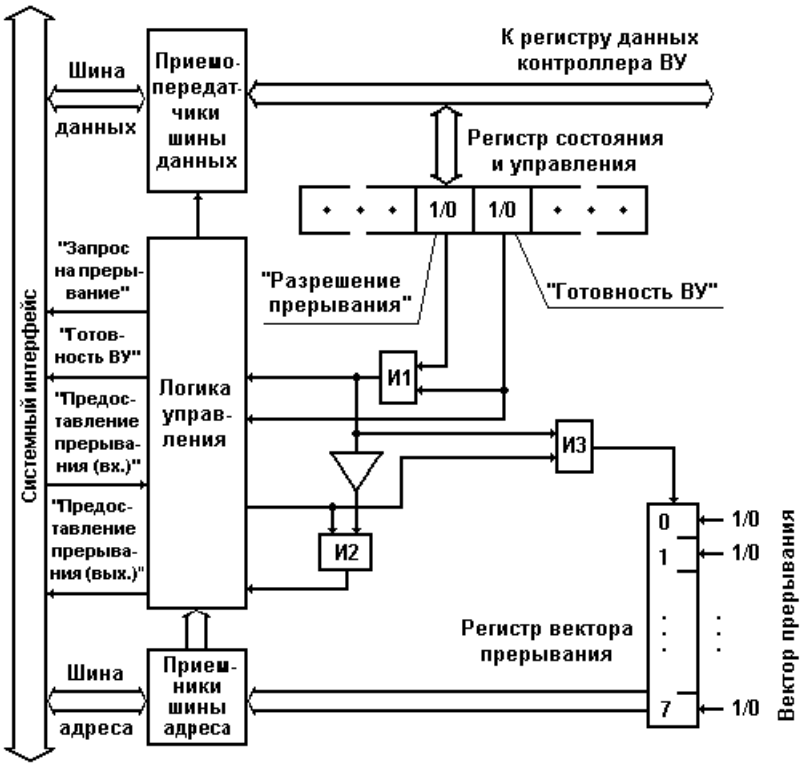


Рис. 47. Формирование векторов прерывания в контроллере ВУ

реализуется с помощью перемычек или переключателей, что позволяет пользователю устанавливать для конкретных ВУ требуемые значения векторов прерывания.

Управляющий сигнал "Предоставление прерывания (вых)" формируется в контроллере ВУ с помощью схемы совпадения И2. Этот сигнал используется для организации последовательного аппаратного опроса готовности ВУ и реализации тем самым требуемых приоритетов ВУ. Процессор при поступлении в него по общей линии системного интерфейса "Запрос на

прерывание" сигнала прерывания формирует управляющий сигнал "Предоставление прерывания (вх.)", который поступает сначала в контроллер ВУ с наивысшим приоритетом (рис.48). Если это устройство не требовало обслуживания, то его контроллер пропускает сигнал "Предоставление прерывания" на следующий контроллер, иначе дальнейшее распространение сигнала прекращается и контроллер выдает вектор прерывания на адресно-информационную шину(системную магистраль).

Рис. 48. Реализация приоритетов ВУ в МПС с векторной системой прерываний: с интерфейсным вектором ППР (вх) - "Предоставление прерывания (входной)"; ППР (вых) – “Предоставление прерывания (выходной)"

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

Векторная система с внеинтерфейсным вектором прерывания используется в IBM-совместимых персональных компьютерах. В этих компьютерах контроллеры внешних устройств не имеют регистров для хранения векторов прерывания, а для идентификации устройств, запросивших обслуживания, используется общий для всех ВУ контроллер прерываний.

6.4. Организация прямого доступа к памяти

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

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

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

Существуют две разновидности прямого доступа к памяти с "захватом цикла". Наиболее простой способ организации ПДП состоит в том, что для обмена

используются те машинные циклы процессора, в которых он не обменивается данными с памятью. В такие циклы контроллер ПДП может обмениваться данными с памятью, не мешая работе процессора. Однако возникает необходимость выделения таких циклов, чтобы не произошло временного перекрытия обмена ПДП с операциями обмена, инициируемыми процессором. В некоторых процессорах формируется специальный управляющий сигнал, указывающий циклы, в которых процессор не обращается к системному интерфейсу. При использовании других процессоров для выделения таких циклов необходимо применение в контроллерах ПДП специальных селектирующих схем, что усложняет их конструкцию. Применение рассмотренного способа организации ПДП не снижает производительности МПС, но при этом обмен в режиме ПДП возможен только в случайные моменты времени одиночными байтами или словами.

Более распространенным является ПДП с "захватом цикла" и принудительным отключением процессора от шин системного интерфейса. Для реализации такого режима ПДП системный интерфейс МПС дополняется двумя линиями для передачи управляющих сигналов "Требование прямого доступа к памяти" (ТПДП) и "Предоставление прямого доступа к памяти" (ППДП).

Управляющий сигнал ТПДП формируется контроллером прямого доступа к памяти. Процессор, получив этот сигнал, приостанавливает выполнение очередной команды, не дожидаясь ее завершения, выдает на системный интерфейс управляющий сигнал ППДП и отключается от шин системного интерфейса. С этого момента все шины системного интерфейса управляются контроллером ПДП. Контроллер ПДП, используя шины системного интерфейса, осуществляет обмен одним байтом или словом данных с памятью МПС и затем, сняв сигнал ТПДП, возвращает управление системным интерфейсом процессору. Как только контроллер ПДП будет готов к обмену следующим байтом, он вновь "захватывает" цикл процессора и т.д. В

промежутках между сигналами ТПДП процессор продолжает выполнять команды программы. Тем самым выполнение программы замедляется, но в меньшей степени, чем при обмене в режиме прерываний.

Применение в МПС обмена данными с ВУ в режиме ПДП всегда требует предварительной подготовки, а именно: для каждого ВУ необходимо выделить область памяти, используемую при обмене, и указать ее размер, т.е. количество записываемых в память или читаемых из памяти байт (слов) информации. Следовательно, контроллер ПДП должен обязательно иметь в своем составе регистр адреса и счетчик байт (слов). Перед началом обмена с ВУ в режиме ПДП процессор должен выполнить программу загрузки. Эта программа обеспечивает запись в указанные регистры контроллера ПДП начального адреса выделенной ВУ памяти и ее размера в байтах или словах в зависимости от того, какими порциями информации ведется обмен. Сказанное не относится к начальной загрузке программ в память в режиме ПДП. В этом случае содержимое регистра адреса и счетчика байт слов устанавливается переключателями или перемычками непосредственно на плате контроллера.

Блок-схема простого контроллера ПДП, обеспечивающего ввод данных в память МПС по инициативе ВУ в режиме ПДП "Захват цикла", приведена на рис. 49.

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

Байты данных из ВУ поступают в регистр данных контроллера в постоянном темпе. При этом каждый байт сопровождается управляющим сигналом из ВУ

"Ввод данных", который обеспечивает запись байта данных в регистр данных контроллера. По этому же сигналу и при ненулевом состоянии счетчика байт контроллер формирует сигнал ТПДП. По ответному сигналу процессора ППДП контроллер выставляет на шины адреса и данных системного интерфейса содержимое своих регистров адреса и данных соответственно. Формируя управляющий сигнал "Вывод", контроллер ПДП обеспечивает запись байта данных из своего регистра данных в память МПС. Сигнал ППДП используется в контроллере и для модификации счетчика байт и регистра адреса. По

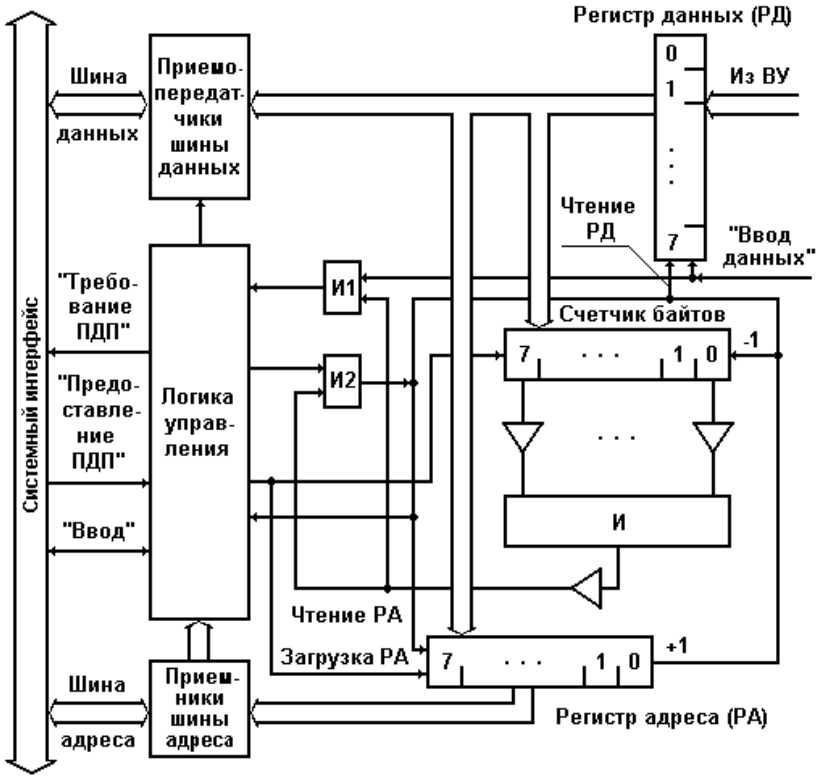


Рис. 49. Контроллер ПДП для ввода данных из ВУ в режиме "Захват цикла" и отключением процессора от шин системного интерфейса.

каждому сигналу ППДП из содержимого счетчика байт вычитается единица, и как только содержимое счетчика станет равно нулю, контроллер прекратит формирование сигналов "Требование прямого доступа к памяти".

На примере простого контроллера ПДП мы рассмотрели только процесс подготовки контроллера и непосредственно передачу данных в режиме ПДП.

На практике любой сеанс обмена данными с ВУ в режиме ПДП всегда инициируется программой, выполняемой процессором, и включает два следующих этапа.

1. На этапе подготовки ВУ к очередному сеансу обмена процессор в режиме программно-управляемого обмена опрашивает состояние ВУ (проверяет его готовность к обмену) и посылает в ВУ команды, обеспечивающие подготовку ВУ к обмену. Такая подготовка может сводиться, например, к перемещению головок на требуемую дорожку в накопителе на жестком диске. Затем выполняется загрузка регистров контроллера ПДП. На этом подготовка к обмену в режиме ПДП завершается и процессор переключается на выполнение другой программы.

2. Обмен данными в режиме ПДП начинается после завершения подготовительных операций в ВУ по инициативе либо ВУ, как это было рассмотрено выше, либо процессора. В этом случае контроллер ПДП необходимо дополнить регистром состояния и управления, содержимое которого будет определять режим работы контроллера ПДП. Один из разрядов этого регистра будет инициировать обмен данными с ВУ. Загрузка информации в регистр состояния и управления контроллера ПДП производится программным путем.

Наиболее распространенным является обмен в режиме прямого доступ к памяти с блокировкой процессора. Он отличается от ПДП с "захватом цикла" тем, что управление системным интерфейсом передается контроллеру ПДП не на время обмена одним байтом, а на время обмена блоком данных. Такой режим ПДП используется в тех случаях, когда время обмена одним байтом с ВУ сопоставимо с циклом системной шины.

В МПС можно использовать несколько ВУ, работающих в режиме ПДП. Предоставление таким ВУ шин системного интерфейса для обмена данными производится на приоритетной основе. Приоритеты ВУ реализуются так же,

как и при обмене данными в режиме прерывания, но вместо управляющих сигналов "Требование прерывания" и "Предоставление прерывания" используются сигналы "Требование прямого доступа" и "Предоставление прямого доступа", соответственно.

7. Функции устройств ввода/вывода

Устройства ввода/вывода обмениваются информацией с магистралью по тем же принципам, что и память. Наиболее существенное отличие с точки зрения организации обмена состоит в том, что модуль памяти имеет в адресном пространстве системы много адресов (до нескольких десятков миллионов), а устройство ввода/вывода обычно имеет немного адресов (обычно до десяти), а иногда и всего один адрес. Но модули памяти системы обмениваются информацией только с магистралью, с процессором, а устройства ввода/вывода взаимодействуют еще и с внешними устройствами, цифровыми или аналоговыми. Поэтому разнообразие устройств ввода/вывода неизмеримо больше, чем модулей памяти. Часто используются еще и другие названия для устройств ввода/вывода: устройства сопряжения, контроллеры, карты расширения, интерфейсные модули и т.д.

Объединяют все устройства ввода/вывода общие принципы обмена с магистралью и, соответственно, общие принципы организации узлов, которые осуществляют сопряжение с магистралью. Упрощенная структура устройства ввода/вывода (точнее, его интерфейсной части) приведена на рис. 50. Как и в случае модуля памяти, она обязательно содержит схему селектора адреса, схему управления для обработки стробов обмена и буферы данных. Самые простейшие устройства ввода/вывода выдают на внешнее устройство код данных в параллельном формате и принимают из внешнего устройства код данных в параллельном формате. Такие устройства ввода/вывода часто называют параллельными портами ввода/вывода. Они наиболее универсальны,

то есть удовлетворяют потребности сопряжения с большим числом внешних устройств, поэтому их часто вводят в состав микропроцессорной системы в качестве стандартных устройств. Параллельные порты обычно имеются в составе микроконтроллеров. Именно через параллельные порты микроконтроллер связывается с внешним миром.

Входной порт (порт ввода) в простейшем случае представляет собой параллельный регистр, в который процессор может записывать информацию. Выходной порт (порт вывода) обычно представляет собой просто однонаправленный буфер, через который процессор может читать информацию от внешнего устройства. Именно такие порты показаны для примера на рис. 50. Порт может быть и двунаправленным (входным/выходным). В этом случае процессор пишет информацию во внешнее устройство и читает информацию из внешнего устройства по одному и тому же адресу в адресном пространстве системы. Входные и выходные линии для связи с внешним устройством при этом могут быть объединены поразрядно, образуя двунаправленные линии.

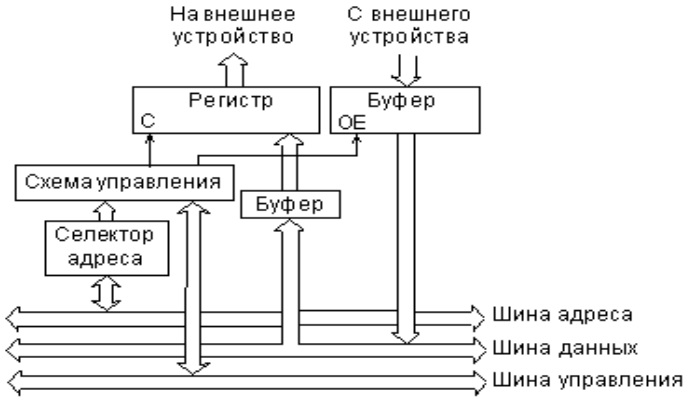


Рис. 50. Структура простейшего устройства ввода/вывода.

При обращении со стороны магистрали селектор адреса распознает адрес, приписанный данному устройству ввода/вывода. Схема управления выдает внутренние стробы обмена в ответ на магистральные стробы обмена. Входной буфер данных обеспечивает электрическое согласование шины данных с этим

устройством (буфер может и отсутствовать). Данные из шины данных записываются в регистр по сигналу С и выдаются на внешнее устройство. Выходной буфер данных передает входные данные с внешнего устройства на шину данных магистрали в цикле чтения из порта. Более сложные устройства ввода/вывода (устройства сопряжения) имеют в своем составе внутреннюю буферную оперативную память и даже могут иметь микроконтроллер, на который возложено выполнение функций обмена с внешним устройством. Каждому устройству ввода/вывода отводится свой адрес в адресном пространстве микропроцессорной системы. Дублирование адресов должно быть исключено, за этим должны следить разработчик и пользователь микропроцессорной системы. Устройства ввода/вывода помимо программного обмена могут также поддерживать режим обмена по прерываниям. В этом случае они преобразуют поступающий от внешнего устройства сигнал запроса на прерывание в сигнал запроса прерывания, необходимый для данной магистрали (или в последовательность сигналов при векторном прерывании). Если нужно использовать режим ПДП, устройство ввода/вывода должно выдать сигнал запроса ПДП на магистраль и обеспечить работу в циклах ПДП, принятых для данной магистрали.

В составе микропроцессорных систем, как правило, выделяются три специальные группы устройств ввода/вывода:

--- устройства интерфейса пользователя (ввода информации пользователем и вывода информации для пользователя);

--- устройства ввода/вывода для длительного хранения информации;

--- таймерные устройства.

К устройствам ввода для интерфейса пользователя относятся контроллеры клавиатуры, тумблеров, отдельных кнопок, мыши, трекбола, джойстика и т.д.

К устройствам вывода для интерфейса пользователя относятся контроллеры светодиодных индикаторов, табло, жидкокристаллических, плазменных и

электронно-лучевых экранов и т.д. В простейших случаях управляющих контроллеров или микроконтроллеров эти средства могут отсутствовать. В сложных микропроцессорных системах они есть обязательно. Роль внешнего устройства в данном случае играет человек.

Устройства ввода/вывода для длительного хранения информации обеспечивают сопряжение микропроцессорной системы с дисководами (компакт-дисков или магнитных дисков), а также с накопителями на магнитной ленте. Применение таких устройств существенно увеличивает возможности микропроцессорной системы в отношении хранения выполняемых программ и накопления массивов данных. В простейших контроллерах эти устройства отсутствуют.

Таймерные устройства отличаются от других устройств ввода/вывода тем, что они могут не иметь внешних выводов для подключения к внешним устройствам. Эти устройства предназначены для того, чтобы микропроцессорная система могла выдерживать заданные временные интервалы, следить за реальным временем, считать импульсы и т.д. В основе любого таймера лежит кварцевый тактовый генератор и многоразрядные двоичные счетчики, которые могут перезапускать друг друга. Процессор может записывать в таймер коэффициенты деления тактовой частоты, количество отсчитываемых импульсов, задавать режим работы счетчиков таймера, а читает процессор выходные коды счетчиков. В принципе выполнить практически все функции таймера можно и программным путем, поэтому иногда таймеры в системе отсутствуют. Но включение в систему таймера позволяет решать более сложные задачи и строить более эффективные алгоритмы.

Еще один важный класс устройств ввода/вывода — это устройства для подключения к информационным сетям (локальным и глобальным). Эти устройства распространены не так широко, как устройства трех перечисленных ранее групп, но их значение с каждым годом становится все больше. Сейчас

средства связи с информационными сетями вводятся иногда даже в простые контроллеры.

Иногда устройства ввода/вывода обеспечивают сопряжение с внешними устройствами с помощью аналоговых сигналов. Это бывает очень удобно, поэтому в состав некоторых микроконтроллеров даже вводят внутренние ЦАП и АЦП.

7.1. Организация ввода/вывода в микропроцессорной системе

Вводом/выводом (ВВ) называется передача данных между ядром компьютера, включающим в себя микропроцессор и основную память, и внешними устройствами (ВУ). Это единственное средство взаимодействия компьютера с "внешним миром", и архитектура ВВ (режимы работы, форматы команд, особенности прерываний, скорость обмена и др.) непосредственно влияет на эффективность всей системы. За время эволюции компьютеров подсистема ВВ претерпела наибольшие изменения благодаря расширению сферы применения компьютеров и появлению новых внешних устройств. Особенно важную роль средства ВВ играют в управляющих компьютерах. Разработка аппаратных средств и программного обеспечения ВВ является наиболее сложным этапом проектирования новых систем на базе компьютера, а возможности ВВ серийных машин представляют собой один из важных параметров, определяющих выбор машины для конкретного применения.

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

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

Регистр, содержащий группу бит, к которой процессор обращается в операциях ВВ, образует порт ВВ. Таким образом, наиболее общая программная модель внешнего устройства, которое может выполнять ввод и вывод, содержит четыре регистра ВВ: регистр выходных данных (выходной порт), регистр входных данных (входной порт), регистр управления и регистр состояния (рис. 51). Каждый из этих регистров должен иметь однозначный адрес, который идентифицируется дешифратором адреса. В зависимости от особенностей устройства общая модель конкретизируется, например, отдельные регистры состояния и управления объединяются в один регистр, в устройстве ввода (вывода) имеется только регистр входных (выходных) данных, для ввода и вывода используется двунаправленный порт.

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

Интерфейс с изолированными шинами характеризуется раздельной адресацией памяти и внешних устройств при обмене информацией. Изолированный ВВ предполагает наличие специальных команд ввода/вывода, общий формат которых показан на рис. 52. При выполнении команды ввода IN содержимое адресуемого входного регистра PORT передается во внутренний регистр REG процессора, а при выполнении команды OUT содержимое регистра REG

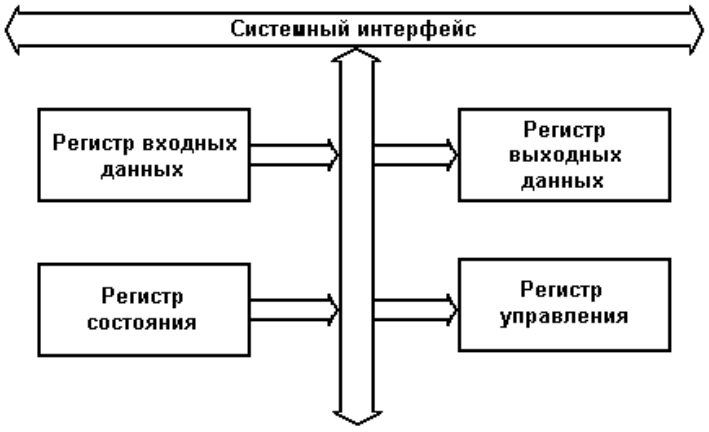


Рис. 51. Программная модель внешнего устройства.

передается в выходной порт PORT. В процессоре могут быть и другие команды, относящиеся к ВВ и связанные с проверкой и модификацией содержимого регистра управления и состояния.

КОП (IN)	REG	PORT
КОП (OUT)	PORT	REG

Рис. 52. Команды ввода/вывода (общий формат)

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

В компьютере, рассчитанном на изолированный ВВ, нетрудно перейти к ВВ, отображенному на память. Если, например, адресное пространство памяти составляет 64 Кбайт, а для программного обеспечения достаточно 32 Кбайт, то область адресов от 0 до 32 К-1 используется для памяти, от 32 К до 64 К-1 - для

ввода/вывода. При этом признаком, дифференцирующим обращения к памяти и портам ВВ, может быть старший бит адреса.

Таким образом, интерфейс с общими шинами (ввод/вывод с отображением на память) имеет организацию, при которой часть общего адресного пространства отводится для внешних устройств, регистры которых адресуются так же, как и ячейки памяти. В этом случае для адресации портов ВВ используются полные адресные сигналы: READ - чтение, WRITE - запись.

В операционных системах компьютеров имеется набор подпрограмм (драйверов ВВ), управляющих операциями ВВ стандартных внешних устройств. Благодаря им пользователь может не знать многих особенностей ВУ и интерфейсов ВВ, а применять четкие программные протоколы.

Рассмотрим некоторые общие вопросы, связанные с обменом данными между ВУ и МПС. Существуют два способа передачи слов информации по линиям данных: параллельный, когда одновременно пересылаются все биты слова, и последовательный, когда биты слова пересылаются поочередно, начиная, например, с его младшего разряда.

Так как между отдельными проводниками шины для параллельной передачи данных существует электрическая емкость, то при изменении сигнала, передаваемого по одному из проводников, возникает помеха (короткий выброс напряжения) на других проводниках. С увеличением длины шины (увеличением емкости проводников) помехи возрастают и могут восприниматься приемником как сигналы. Поэтому рабочее расстояние для шины параллельной передачи данных ограничивается длиной 1-2 м, и только за счет существенного удорожания шины или снижения скорости передачи длину шины можно увеличить до 10-20 м.

Указанное обстоятельство и желание использовать для дистанционной передачи информации телеграфные и телефонные линии обусловили широкое распространение способа последовательного обмена данными между ВУ и

МПС и между несколькими МПС. Возможны два режима последовательной передачи данных: синхронный и асинхронный.

При синхронной последовательной передаче каждый передаваемый бит данных сопровождается импульсом синхронизации информирующим приемник о наличии на линии информационного бита. Следовательно, между передатчиком и приемником должны быть протянуты минимум три провода: два для передачи импульсов синхронизации и бит данных, а также общий заземленный проводник. Если же передатчик (например, МПС) и приемник (например, дисплей) разнесены на несколько метров, то каждый из сигналов (информационный и синхронизирующий) придется посылать либо по экранированному кабелю, либо с помощью витой пары проводов, один из которых заземлен или передает сигнал, инверсный основному.

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

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

Стандартный формат асинхронной последовательной передачи данных, используемый в компьютере и ВУ, содержит n пересылаемых бит информации (при пересылке символов n равно 7 или 8 битам) и 3-4 дополнительных бита: стартовый бит, бит контроля четности (или нечетности) и 1 или 2 стоповых бита (рис. 53, а). Бит четности (или нечетности) может отсутствовать. Когда передатчик бездействует (данные не посылаются на линию), на линии сохраняется уровень сигнала, соответствующий логической 1.

Рис. 53. Формат асинхронной последовательной передачи данных.

Передатчик может начать пересылку символа в любой момент времени посредством генерирования стартового бита, т. е. перевода линии в состояние логического 0 на время, точно равное времени передачи бита. Затем происходит передача битов символа, начиная с младшего значащего бита, за которым следует дополнительный бит контроля по четности или нечетности. Далее с помощью стопового бита линия переводится в состояние логической 1 (рис. 53, б). При единичном бите контроля стоповый бит не изменяет состояния сигнала на линии. Состояние логической 1 должно поддерживаться в течение промежутка времени, равного 1 или 2 временам передачи бита.

Промежуток времени от начала стартового бита до конца стопового бита (стоповых бит) называется кадром. Сразу после стоповых бит передатчик

может посылать новый стартовый бит, если имеется другой символ для передачи; в противном случае уровень логической 1 может сохраняться на протяжение всего времени, пока бездействует передатчик. Новый стартовый бит может быть послан в любой момент времени после окончания стопового бита, например, через промежуток времени, равный 0.43 или 1.5 времени передачи бита.

В линиях последовательной передачи данных передатчик и приемник должны быть согласованы по всем параметрам формата, изображенного на рис. 53, включая номинальное время передачи бита. Для этого в приемнике устанавливается генератор синхроимпульсов, частота которого должна совпадать с частотой аналогичного генератора передатчика. Кроме того, для обеспечения оптимальной защищенности сигнала от искажения, шумов и разброса частоты синхроимпульсов приемник должен считывать принимаемый бит в середине его длительности. Рассмотрим работу приемника с того момента, когда он закончил прием символа данных и перешел в режим обнаружения стартового бита следующего слова.

Если линия перешла в состояние логического нуля и находится в этом состоянии в течение времени, не меньшего половины временного интервала передачи бита, то приемник переводится в режим считывания бит информации. В противном случае приемник остается в режиме обнаружения, так как вероятнее всего это был не стартовый бит, а шумовая помеха. В новом режиме приемник вырабатывает сигналы считывания через интервалы, равные времени передачи бита, т. е. выполняет считывание и сохранение принимаемых бит примерно на середине их передачи. Аналогичным образом будут считаны бит контроля четности и сигнал логической единицы (стоповый бит). Если оказалось, что на месте стопового бита обнаружен сигнал логического нуля, то произошла "Ошибка кадра" и символ принят неправильно. Иначе проверяется,

четно ли общее число единиц в информационных битах и бите контроля, и если оно четно, производится запись принятого символа в буфер приемника.

Передний фронт стартового бита сигнализирует о начале поступления передаваемой информации, а момент его появления служит точкой отсчета времени для считывания бит данных. Стоповый бит предоставляет время для записи принятого символа в буфер приемника и обеспечивает возможность выявления ошибки кадра. Наиболее часто ошибки кадра появляются тогда, когда приемник ошибочно синхронизирован с битом 0, который в действительности не является стартовым битом. Если передатчик бездействует (посылает сигнал логической единицы) в течение одного кадра или более, то всегда можно восстановить правильную синхронизацию. Хуже обстоит дело при рассинхронизации генераторов передатчика и приемника, когда временной интервал между сигналами считывания принимаемых битов будет меньше или больше времени передачи бита.

Например, если при считывании битов посылки, показанной на рис. 53, б, временной интервал между сигналами считывания станет на 6% меньше, чем время передачи бита, то восьмой и девятый сигналы считывания будут выработаны тогда, когда на линии находится бит контроля четности (рис.54). Следовательно, не будет обнаружен стоповый бит и будет зафиксирована ошибка кадра, несмотря на правильность принятой информации. Однако при 18%-й рассинхронизации генераторов, когда вместо кода (01110001) приемник зафиксирует код (11100001), никаких ошибок не будет обнаружено - четность соблюдена и стоповый (девятый по порядку) бит равен 1(рис. 54).

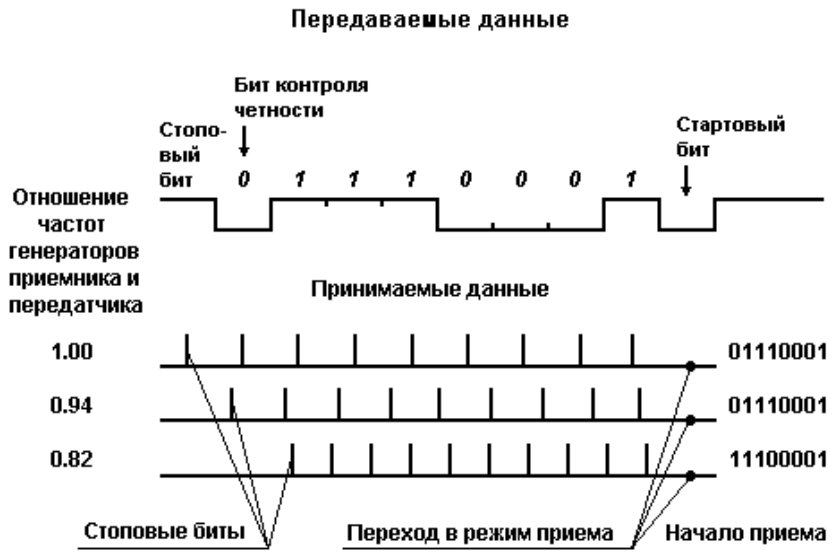


Рис. 54. Ошибка из-за рассинхронизации генераторов передатчика и приемника

7.2. Параллельная передача данных

Параллельная передача данных между контроллером и ВУ является по своей организации наиболее простым способом обмена. Для организации параллельной передачи данных помимо шины данных, количество линий в которой равно числу одновременно передаваемых битов данных, используется минимальное количество управляющих сигналов.

В простом контроллере ВУ, обеспечивающем побайтную передачу данных на внешнее устройство (рис. 55), в шине связи с ВУ используются всего два управляющих сигнала: "Выходные данные готовы" и "Данные приняты".

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

формируется управляющий сигнал "Выходные данные готовы" в шине связи с ВУ.

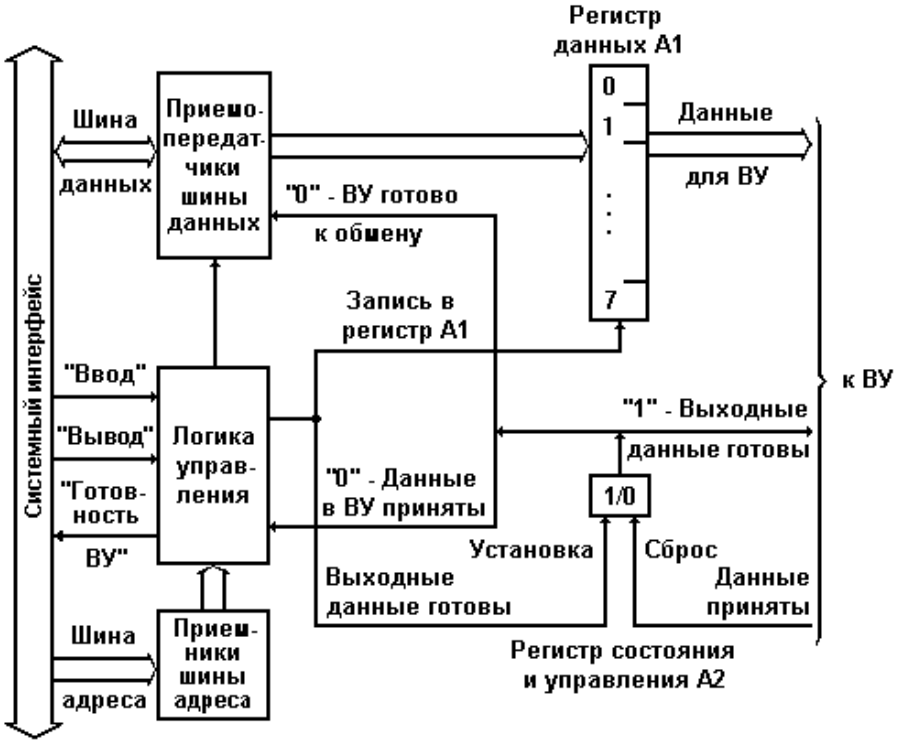


Рис. 55. Простой параллельный контроллер вывода.

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

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

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

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

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

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

Т а б л и ц а 1

Фрагмент программы передачи байта данных в асинхронном режиме

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

состояния А2 по шине данных системного интерфейса.

Команда в третьей строке приводит к следующим действиям. Процессор проверяет значение соответствующего разряда принятых данных. Нуль в этом разряде указывает на неготовность ВУ к приему данных и, следовательно, на необходимость возврата к проверке содержимого A2, т. е. процессор, выполняя три первые команды, ожидает готовности ВУ к приему данных. Единица в этом разряде подтверждает готовность ВУ и, следовательно, возможность передачи байта данных.

В седьмой строке осуществляется пересылка данных из регистра AX процессора в регистр данных контроллера A1. Процессор по шине адреса передает в контроллер адрес A1, а по шине данных - байт данных, сопровождая их сигналом "Вывод" (IOWR#). Логика управления контроллера обеспечивает запись данных с шины данных в регистр данных A1 и устанавливает в ноль бит готовности регистра состояния A2, формируя тем самым управляющий сигнал для ВУ "Выходные данные готовы". ВУ принимает байт данных и управляющим сигналом "Данные приняты" устанавливает в единицу регистр состояния A2. Далее контроллер ВУ по этому сигналу может сформировать и передать в процессор сигнал "Готовность ВУ", который в данном случае извещает процессор о приеме данных внешним устройством и разрешает процессору снять сигнал "Вывод". Тем самым завершается цикл вывода данных в команде пересылки. В IBM-совместимых персональных компьютерах с шиной ISA сигнал "Готовность ВУ" не формируется, а имеется сигнал IO CH RDY#, позволяющий продлить цикл обмена, если устройство недостаточно быстрое. В данном случае нет необходимости в сигнале "Готовность ВУ". Шина ISA является синхронной и, следовательно, все операции выполняются по тактовым импульсам.

Блок-схема простого контроллера ВУ, обеспечивающего побайтный прием данных из ВУ, приведена на рис. 56. В этом контроллере при взаимодействии с

внешним устройством также используются два управляющих сигнала: "Данные от ВУ готовы" и "Данные приняты".

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

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

При этом формируются: управляющий сигнал системного интерфейса "Готовность ВУ"; признак готовности ВУ к обмену, передаваемый в процессор по одной из линий шины данных системного интерфейса посредством операции ввода при реализации программы асинхронного обмена. Тем самым контроллер извещает процессор о готовности данных в регистре А1.

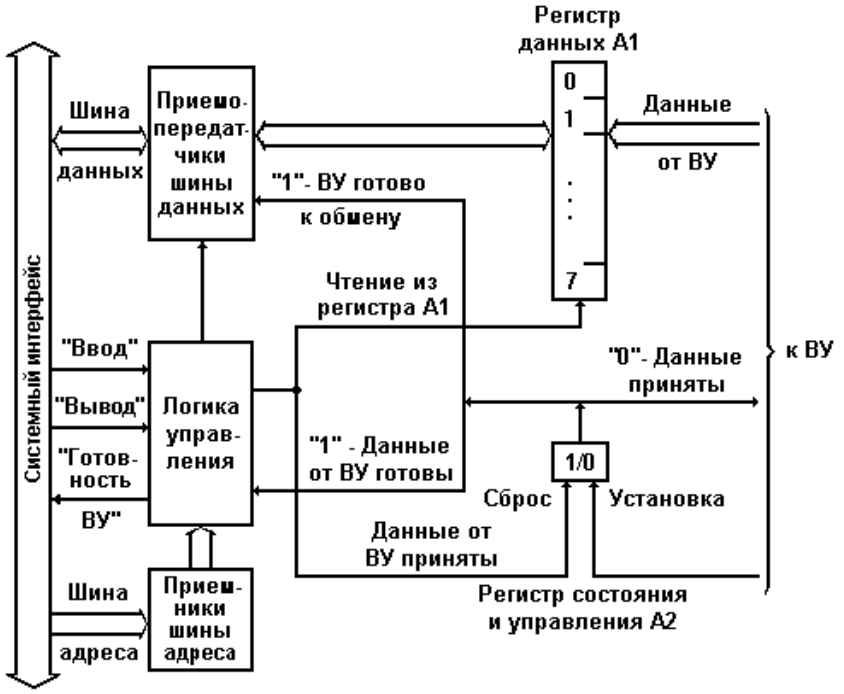


Рис. 56. Простой параллельный контроллер ввода

Процессор, выполняя программу асинхронного обмена, читает байт данных из регистра данных контроллера и обнуляет регистр состояния и управления А2.

При этом формируется управляющий сигнал "Данные приняты" в шине связи с внешним устройством.

Логика управления контроллера и приемопередатчики шин системного интерфейса выполняют те же функции, что и в контроллере вывода (рис. 55),

Рассмотрим работу параллельного интерфейса ввода при реализации программы асинхронного обмена. Алгоритм асинхронного ввода так же прост, как и асинхронного вывода.

1. Процессор проверяет наличие данных в регистре данных контроллера A1.
2. Если данные готовы (логическая 1 в регистре A2), то они передаются из регистра данных A1 на шину данных системного интерфейса и далее в регистр процессора или ячейку памяти микрокомпьютера. Иначе повторяется п. 1.

Пример 2. Фрагмент программы приема байта данных в асинхронном режиме с использованием параллельного интерфейса (контроллер ВУ):

В третьей строке выполняется проверка содержимого регистра A2, т.е. признака наличия данных в регистре данных A1. Команда выполняется точно так же, как и в примере 1. Единица в нулевом разряде (содержимое регистра A2) подтверждает, что данные от ВУ записаны в регистр данных контроллера и необходимо переслать их на шину данных. Нуль в знаковом разряде указывает на неготовность данных от ВУ и, следовательно, на необходимость вернуться к проверке готовности.

IN AL, DX - пересылка данных из регистра данных контроллера A1 в регистр процессора AL. Процессор передает в контроллер по шине адреса системного интерфейса адрес A1, сопровождая его сигналом "Ввод". Логика управления контроллера в ответ на сигнал "Ввод" (IORD#) обеспечивает передачу байта данных из регистра данных A1 на шину данных и, в общем случае, сопровождает его сигналом "Готовность ВУ". Этот сигнал подтверждает наличие данных от ВУ на шине данных и, по которому процессор считывает

Таблица 2

Фрагмент программы приема байта данных в асинхронном режиме с использованием параллельного интерфейса (контроллер ВУ)

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

байт с шины данных и помещает его в указанный регистр. В IBM-совместимом персональном компьютере с шиной ISA процессор считывает байт с шины данных по истечении определенного времени после установки сигнала IORD#. Затем логика управления обнуляет регистр состояния и управления A2, формируя тем самым управляющий сигнал для внешнего устройства "Данные приняты". Таким образом завершается цикл ввода данных.

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

Последовательная передача данных. Использование последовательных линий связи для обмена данными с внешними устройствами возлагает на контроллеры ВУ дополнительные по сравнению с контроллерами для параллельного обмена функции. Во-первых, возникает необходимость преобразования формата данных из параллельного формата, в котором они

поступают в контроллер ВУ из системного интерфейса компьютера, в последовательный при передаче в ВУ. И обратно из последовательного в параллельный при приеме данных из ВУ. Во-вторых, требуется реализовать соответствующий режиму работы внешнего устройства способ обмена данными: синхронный или асинхронный.

7.3. Синхронный последовательный интерфейс

Простой контроллер для синхронной передачи данных в ВУ по последовательной линии связи (последовательный интерфейс) представлен на рис. 57.

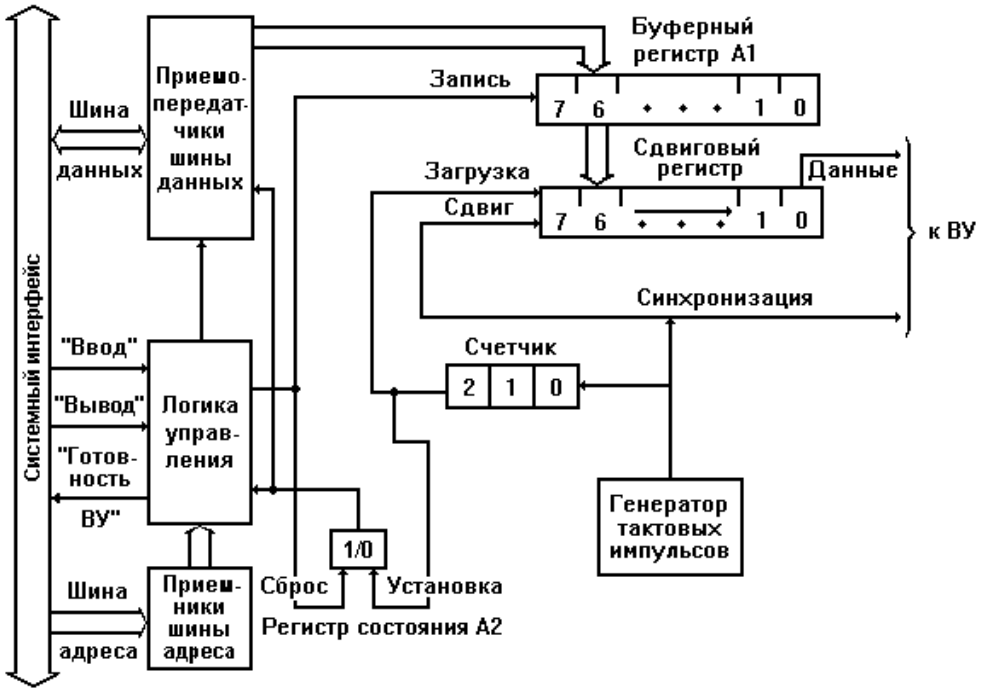


Рис. 57. Контроллер последовательной синхронной передачи

Восьмиразрядный адресуемый буферный регистр контроллера A1 служит для временного хранения байта данных до его загрузки в сдвиговый регистр. Запись байта данных в буферный регистр с шины данных системного интерфейса производится так же, как и в параллельном интерфейсе.

Параллельная передача данных (рис. 55), только при наличии единицы в одноразрядном адресуемом регистре состояния контроллера А2. Единица в регистре состояния указывает на готовность контроллера принять очередной байт в буферный регистр. Содержимое регистра А2 передается в процессор по одной из линий шины данных системного интерфейса и используется для формирования управляющего сигнала системного интерфейса "Готовность ВУ". При записи очередного байта в буферный регистр А1 обнуляется регистр состояния А2.

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

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

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

Количество переданных в линию тактовых сигналов, а следовательно, и переданных бит информации подсчитывается счетчиком тактовых импульсов.

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

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

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

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

7.4. Асинхронный последовательный интерфейс

Организация асинхронного последовательного обмена данными с внешним устройством осложняется тем, что на передающей и приемной стороне последовательной линии связи используются настроенные на одну частоту, но физически разные генераторы тактовых импульсов и, следовательно, общая синхронизация отсутствует. Рассмотрим на примерах организацию контроллеров последовательных интерфейсов для последовательных асинхронных передачи и приема данных.

Простейший контроллер для асинхронной передачи данных в ВУ по последовательной линии связи представлен на рис. 58. Он предназначен для передачи данных в формате с двумя стоповыми битами.

После передачи очередного байта данных в регистр состояния А2 записывается 1. Единичный выходной сигнал регистра А2 информирует процессор о готовности контроллера к приему следующего байта данных и передаче его по линии связи в ВУ. Этот же сигнал запрещает формирование импульсов со схемы выработки импульсов сдвига - делителя частоты сигналов тактового генератора на 16. Счетчик импульсов сдвига (счетчик по mod10) находится в нулевом состоянии и его единичный выходной сигнал поступает на вентиль И,

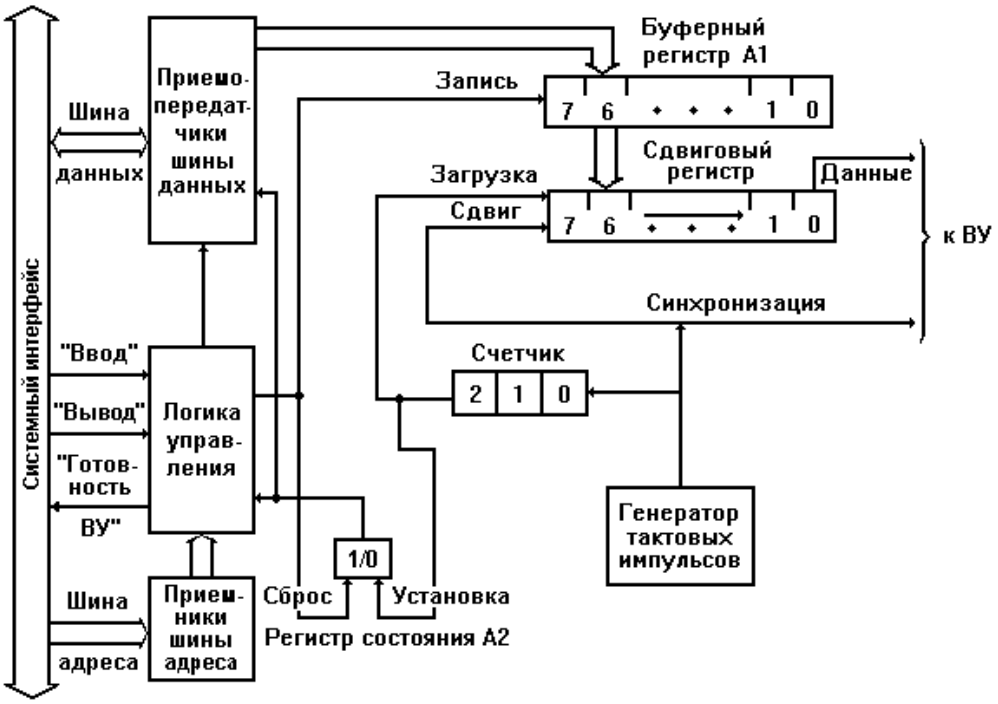


Рис. 58. Контроллер последовательной асинхронной передачи.

подготавливая цепь выработки сигнала загрузки сдвигового регистра.

Процесс передачи байта данных начинается с того, что процессор, выполняя команду "Вывод", выставляет этот байт на шине данных. Одновременно процессор формирует управляющий сигнал системного интерфейса "Вывод",

по которому производятся запись передаваемого байта в буферный регистр А1, сброс регистра состояния А2 и формирование на вентиле И сигнала "Загрузка". Передаваемый байт переписывается в разряды 1, ... , 8 сдвигового регистра, в нулевой разряд сдвигового регистра записывается 0 (стартовый бит), а в разряды 9 и 10 - 1 (стоповые биты). Кроме того, снимается сигнал "Сброс" с делителя частоты, он начинает накапливать импульсы генератора тактовой частоты и в момент приема шестнадцатого тактового импульса вырабатывает импульс сдвига.

На выходной линии контроллера "Данные" поддерживается состояние 0 (значение стартового бита) до тех пор, пока не будет выработан первый импульс сдвига. Импульс сдвига изменит состояние счетчика импульсов сдвига и перепишет в нулевой разряд сдвигового регистра первый информационный бит передаваемого байта данных. Состояние, соответствующее значению этого бита, будет поддерживаться на линии "Данные" до следующего импульса сдвига.

Аналогично будут переданы остальные информационные биты, первый стоповый бит и, наконец, второй стоповый бит, при передаче которого счетчик импульсов сдвига снова установится в нулевое состояние. Это приведет к записи 1 в регистр состояния А2. Единичный сигнал с выхода регистра А2 запретит формирование импульсов сдвига, а также информирует процессор о готовности к приему нового байта данных. После завершения передачи очередного кадра (стартового бита, информационного байта и двух стоповых бит) контроллер поддерживает в линии связи уровень логической единицы (значение второго стопового бита).

Уровень логической единицы поступает по линии "Данные" в контроллер для асинхронного приема данных (рис. 59). Этот уровень создает условия для выработки сигнала, запрещающего работу делителя частоты генератора тактовых импульсов. Действительно, после приема предыдущего байта

данных счетчик импульсов сдвига (счетчик по mod9) находится в нулевом состоянии и на вентиль «И» поступают два единичных сигнала: со счетчика сдвигов и из линии "Данные". На выходе вентиля «И» вырабатывается сигнал сброса делителя частоты сигналов тактового генератора, запрещающий формирование импульсов сдвига.

В момент смены стопового бита на стартовый бит (момент начала передачи нового кадра) на линии "Данные" появится уровень логического нуля и тем самым будет снят сигнал сброса с делителя частоты. Состояние 4-разрядного двоичного счетчика (делителя частоты) начнет изменяться. Когда на счетчике накопится значение 8, он выдаст сигнал, поступающий на входы сдвигового

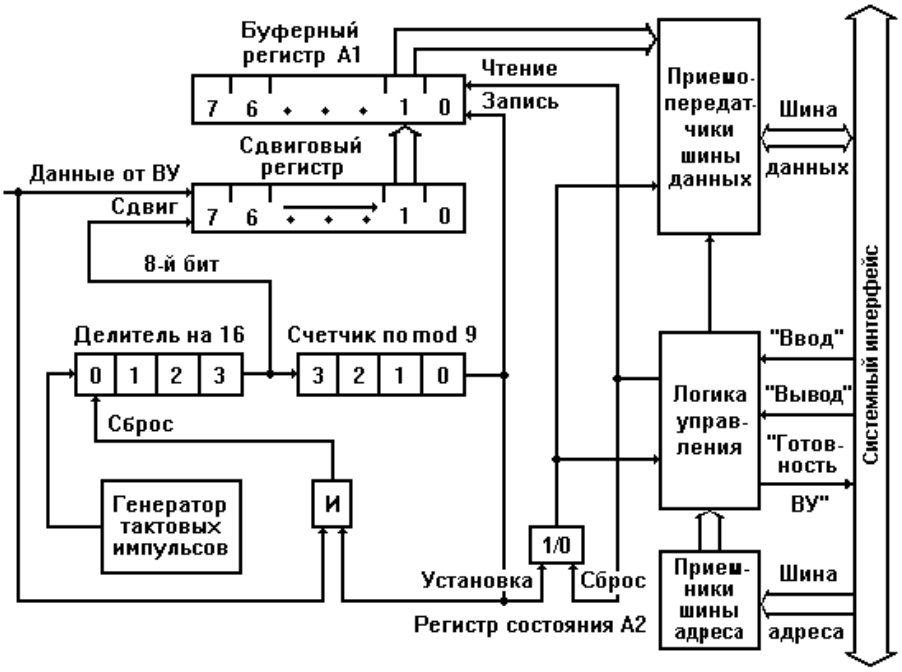


Рис. 59. Контроллер последовательного асинхронного приема

регистра и счетчика импульсов сдвига. Так как частота сигналов генератора тактовых импульсов приемника должна совпадать с частотой генератора тактовых импульсов передатчика, то сдвиг (считывание) бита произойдет примерно на середине временного интервала, отведенного на передачу бита

данных, т. е. времени, необходимого для выработки шестнадцати тактовых импульсов. Это делается для уменьшения вероятности ошибки из-за возможного различия частот генераторов передатчика и приемника, искажения формы передаваемых сигналов (переходные процессы) и т. п. Следующий сдвиг произойдет после прохождения шестнадцати тактовых импульсов, т. е. на середине временного интервала передачи первого информационного бита.

При приеме в сдвиговый регистр девятого бита кадра (восьмого информационного бита) из него "выдвинется" стартовый бит и, следовательно, в сдвиговом регистре будет размещен весь принятый байт информации. В этот момент счетчик импульсов сдвига придет в нулевое состояние и на его выходе будет выработан единичный сигнал, по которому содержимое сдвигового регистра перепишется в буферный регистр, в регистр состояния А2 запишется 1 и он будет информировать процессор об окончании приема очередного байта, вентиль И подготовится к выработке сигнала "Сброс" (этот сигнал сформируется после прихода первого стопового бита).

Получив сигнал готовности (1 в регистре А2), процессор выполнит команду "Ввод" (см. пример 2, раздела «Параллельная передача данных»). При этом вырабатывается управляющий сигнал системного интерфейса "Ввод", по которому производятся пересылка принятого байта данных из буферного регистра в процессор (сигнал "Чтение") и сброс регистра состояния А2.

Отметим, что для простоты изложения в контроллере на рис. 59 не показаны схемы контроля стоповых бит принимаемого кадра. Не показаны также схемы контроля четности или нечетности (паритета) передаваемой информации (обычно в передаваемом байте восьмому биту придается значение 0 или 1, так чтобы в этом байте было четное количество единиц). В реальных контроллерах имеются такие схемы, и если контроллер не принимает из линии связи нужного количества стоповых бит или вырабатывается сигнал ошибки паритета в схеме

контроля четности, то принятые в текущем кадре биты данных игнорируются и контроллер ожидает поступления нового стартового бита.

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

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

8. Классификация и структура микроконтроллеров

В настоящее время выпускается целый ряд типов микроконтроллеров (МК). Все эти приборы можно условно разделить на три основных класса:

--- 8-разрядные МК для встраиваемых приложений;

--- 16- и 32-разрядные МК;

--- цифровые сигнальные процессоры (DSP).

Наиболее распространенным представителем семейства МК являются 8-разрядные приборы, широко используемые в промышленности, бытовой и компьютерной технике. Они прошли в своем развитии путь от простейших приборов с относительно слаборазвитой периферией до современных многофункциональных контроллеров, обеспечивающих реализацию сложных алгоритмов управления в реальном масштабе времени. Причиной жизнеспособности 8-разрядных МК является использование их для управления

реальными объектами, где применяются, в основном, алгоритмы с преобладанием логических операций, скорость обработки которых практически не зависит от разрядности процессора.

Росту популярности 8-разрядных МК способствует постоянное расширение номенклатуры изделий, выпускаемых такими известными фирмами, как Motorola, Microchip, Intel, Zilog, Atmel и многими другими. Современные 8-разрядные МК обладают, как правило, рядом отличительных признаков. Перечислим основные из этих признаков:

--- модульная организация, при которой на базе одного процессорного ядра (центрального процессора) проектируется ряд (линейка) МК, различающихся объемом и типом памяти программ, объемом памяти данных, набором периферийных модулей, частотой синхронизации;

--- использование закрытой архитектуры МК, которая характеризуется отсутствием линий магистралей адреса и данных на выводах корпуса МК. Таким образом, МК представляет собой законченную систему обработки данных, наращивание возможностей которой с использованием параллельных магистралей адреса и данных не предполагается;

--- использование типовых функциональных периферийных модулей (таймеры, процессоры событий, контроллеры последовательных интерфейсов, аналого-цифровые преобразователи и др.), имеющих незначительные отличия в алгоритмах работы в МК различных производителей;

---расширение числа режимов работы периферийных модулей, которые задаются в процессе инициализации регистров специальных функций МК.

При модульном принципе построения все МК одного семейства содержат процессорное ядро, одинаковое для всех МК данного семейства, и изменяемый функциональный блок, который отличает МК разных моделей.

Структура модульного МК приведена на рис. 60.

Процессорное ядро включает в себя:

--- центральный процессор;

--- внутреннюю контроллерную магистраль (ВКМ) в составе шин адреса, данных и управления;

--- схему синхронизации МК;

--- схему управления режимами работы МК, включая поддержку режимов пониженного энергопотребления, начального запуска (сброса) и т.д.

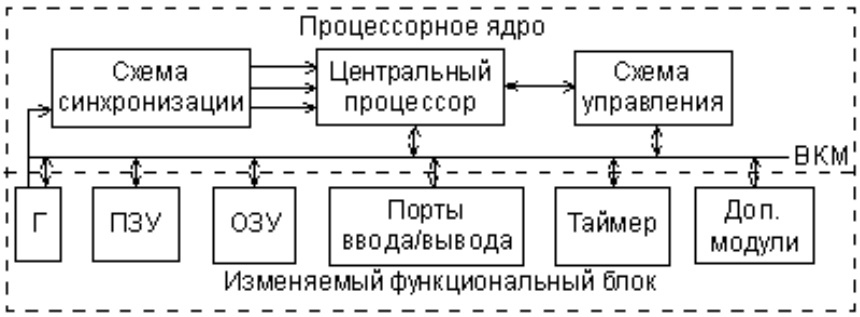


Рис. 60. Модульная организация МК.

Изменяемый функциональный блок включает в себя модули памяти различного типа и объема, порты ввода/вывода, модули тактовых генераторов (Г), таймеры. В относительно простых МК модуль обработки прерываний входит в состав процессорного ядра. В более сложных МК он представляет собой отдельный модуль с развитыми возможностями. В состав изменяемого функционального блока могут входить и такие дополнительные модули как компараторы напряжения, аналого-цифровые преобразователи (АЦП) и другие. Каждый модуль проектируется для работы в составе МК с учетом протокола ВКМ. Данный подход позволяет создавать разнообразные по структуре МК в пределах одного семейства.

8.1. Процессорное ядро микроконтроллера

8.1.1. Структура процессорного ядра МК

Основными характеристиками, определяющими производительность процессорного ядра МК, являются:

--- набор регистров для хранения промежуточных данных;

система команд процессора;

--- способы адресации операндов в пространстве памяти;

--- организация процессов выборки и исполнения команды.

С точки зрения системы команд и способов адресации операндов процессорное ядро современных 8-разрядных МК реализует один из двух принципов построения процессоров:

процессоры с CISC-архитектурой, реализующие так называемую полную систему команд (Complicated Instruction Set Computer);

процессоры с RISC-архитектурой, реализующие сокращенную систему команд (Reduced Instruction Set Computer).

CISC-процессоры выполняют большой набор команд с развитыми возможностями адресации, давая разработчику возможность выбрать наиболее подходящую команду для выполнения необходимой операции. В применении к 8-разрядным МК процессор с CISC-архитектурой может иметь однобайтовый, двухбайтовый и трехбайтовый (редко четырехбайтовый) формат команд. При этом система команд, как правило, неортогональна, то есть не все команды могут использовать любой из способов адресации применительно к любому из регистров процессора. Выборка команды на исполнение осуществляется побайтно в течение нескольких циклов работы МК. Время выполнения команды может составлять от 1 до 12 циклов. К МК с CISC-архитектурой относятся МК фирмы Intel с ядром MCS-51, которые поддерживаются в настоящее время целым рядом производителей, МК семейств HC05, HC08 и HC11 фирмы Motorola и ряд других.

В процессорах с RISC-архитектурой набор исполняемых команд сокращен до минимума. Для реализации более сложных операций приходится комбинировать команды. При этом все команды имеют формат фиксированной длины (например, 12, 14 или 16 бит), выборка команды из памяти и ее

исполнение осуществляется за один цикл (такт) синхронизации. Система команд RISC-процессора предполагает возможность равноправного использования всех регистров процессора. Это обеспечивает дополнительную гибкость при выполнении ряда операций. К МК с RISC-процессором относятся МК AVR фирмы Atmel, МК PIC16 и PIC17 фирмы Microchip и другие.

На первый взгляд, МК с RISC-процессором должны иметь более высокую производительность по сравнению с CISC МК при одной и той же тактовой частоте внутренней магистрали. Однако на практике вопрос о производительности более сложен и неоднозначен.

Во-первых, оценка производительности МК по времени выполнения команд различных систем (RISC и CISC) не совсем корректна. Обычно производительность МП и МК принято оценивать числом операций пересылки «регистр-регистр», которые могут быть выполнены в течение одной секунды. В МК с CISC-процессором время выполнения операции «регистр-регистр» составляет от 1 до 3 циклов, что, казалось бы, уступает производительности МК с RISC-процессором. Однако стремление к сокращению формата команд при сохранении ортогональности системы команд RISC-процессора приводит к вынужденному ограничению числа доступных в одной команде регистров. Так, например, системой команд МК PIC16 предусмотрена возможность пересылки результата операции только в один из двух регистров: регистр-источник операнда f или рабочий регистр W. Таким образом, операция пересылки содержимого одного из доступных регистров в другой (не источник операнда и не рабочий) потребует использования двух команд. Такая необходимость часто возникает при пересылке содержимого одного из регистров общего назначения (РОН) в один из портов МК. В то же время, в системе команд большинства CISC-процессоров присутствуют команды пересылки содержимого РОН в один из портов ввода/вывода. То есть более сложная система команд иногда позволяет реализовать более эффективный способ выполнения операции.

Во-вторых, оценка производительности МК по скорости пересылки «регистр-регистр» не учитывает особенностей конкретного реализуемого алгоритма управления. Так, при разработке быстродействующих устройств автоматизированного управления основное внимание следует уделять времени выполнения операций умножения и деления при реализации уравнений различных передаточных функций. А при реализации пульта дистанционного управления бытовой техникой следует оценивать время выполнения логических функций, которые используются при опросе клавиатуры и генерации последовательной кодовой посылки управления. Поэтому в критических ситуациях, требующих высокого быстродействия, следует оценивать производительность на множестве тех операций, которые преимущественно используются в алгоритме управления и имеют ограничения по времени выполнения.

В-третьих, необходимо еще учитывать, что указанные в справочных данных на МК частоты синхронизации обычно соответствуют частоте подключаемого кварцевого резонатора, в то время как длительность цикла центрального процессора определяется частотой обмена по ВКМ. Соотношение этих частот индивидуально для каждого МК и должно быть принято в расчет при сравнении производительности различных моделей контроллеров.

С точки зрения организации процессов выборки и исполнения команды в современных 8-разрядных МК применяется одна из двух уже упоминавшихся архитектур МПС: фон-неймановская (принстонская) или гарвардская.

Большинство производителей современных 8-разрядных МК используют гарвардскую архитектуру. Однако гарвардская архитектура является недостаточно гибкой для реализации некоторых программных процедур. Поэтому сравнение МК, выполненных по разным архитектурам, следует проводить применительно к конкретному приложению.

8.1.2. Система команд процессора МК

Так же, как и в любой микропроцессорной системе, набор команд процессора МК включает в себя четыре основные группы команд:

--- команды пересылки данных;

--- арифметические команды;

--- логические команды;

--- команды переходов.

Для реализации возможности независимого управления разрядами портов (регистров) в большинстве современных МК предусмотрена также группа команд битового управления (булевый или битовый процессор). Наличие команд битового процессора позволяет существенно сократить объем кода управляющих программ и время их выполнения.

В ряде МК выделяют также группу команд управления ресурсами контроллера, используемую для настройки режимов работы портов ввода/вывода, управления таймером и т.п. В большинстве современных МК внутренние ресурсы контроллера отображаются на память данных, поэтому для целей управления ресурсами используются команды пересылки данных.

Система команд МК по сравнению с системой команд универсального МП имеет, как правило, менее развитые группы арифметических и логических команд, зато более мощные группы команд пересылки данных и управления. Эта особенность связана со сферой применения МК, требующей, прежде всего, контроля окружающей обстановки и формирования управляющих воздействий.

8.2. Схема синхронизации МК

Схема синхронизации МК обеспечивает формирование сигналов синхронизации, необходимых для выполнения командных циклов центрального процессора, а также обмена информацией по внутренней магистрали. В

зависимости от исполнения центрального процессора командный цикл может включать в себя от одного до нескольких (4 — 6) тактов синхронизации. Схема синхронизации формирует также метки времени, необходимые для работы таймеров МК. В состав схемы синхронизации входят делители частоты, которые формируют необходимые последовательности синхросигналов.

8.3. Память программ и данных МК

В МК используется три основных вида памяти. Память программ представляет собой постоянную память (ПЗУ), предназначенную для хранения программного кода (команд) и констант. Ее содержимое в ходе выполнения программы не изменяется. Память данных предназначена для хранения переменных в процессе выполнения программы и представляет собой ОЗУ. Регистры МК — этот вид памяти включает в себя внутренние регистры процессора и регистры, которые служат для управления периферийными устройствами (регистры специальных функций).

Память программ Основным свойством памяти программ является ее энергонезависимость, то есть возможность хранения программы при отсутствии питания. С точки зрения пользователей МК следует различать следующие типы энергонезависимой памяти программ:

---ПЗУ масочного типа — mask - ROM. В ПЗУ запись информации производится в последней операции производства микросхемы - металлизации. Металлизация производится при помощи маски, поэтому такие ПЗУ получили название масочных. Содержимое ячеек ПЗУ этого типа не может быть впоследствии заменено или перепрограммировано. Поэтому МК с таким типом памяти программ следует использовать только после достаточно длительной опытной эксплуатации. Основным недостатком данной памяти является необходимость значительных затрат на создание нового комплекта

фотошаблонов и их внедрение в производство. Обычно такой процесс занимает 2-3 месяца и является экономически выгодным только при выпуске десятков тысяч приборов. ПЗУ масочного типа обеспечивают высокую надежность хранения информации по причине программирования в заводских условиях с последующим контролем результата (рис. 61).

Масочные ПЗУ изображаются на принципиальных схемах как показано на рис. 62. Адреса ячеек памяти в этой микросхеме подаются на выводы A0 ... A9. Микросхема выбирается сигналом CS. При помощи этого сигнала можно наращивать объем ПЗУ. Чтение микросхемы производится сигналом RD.

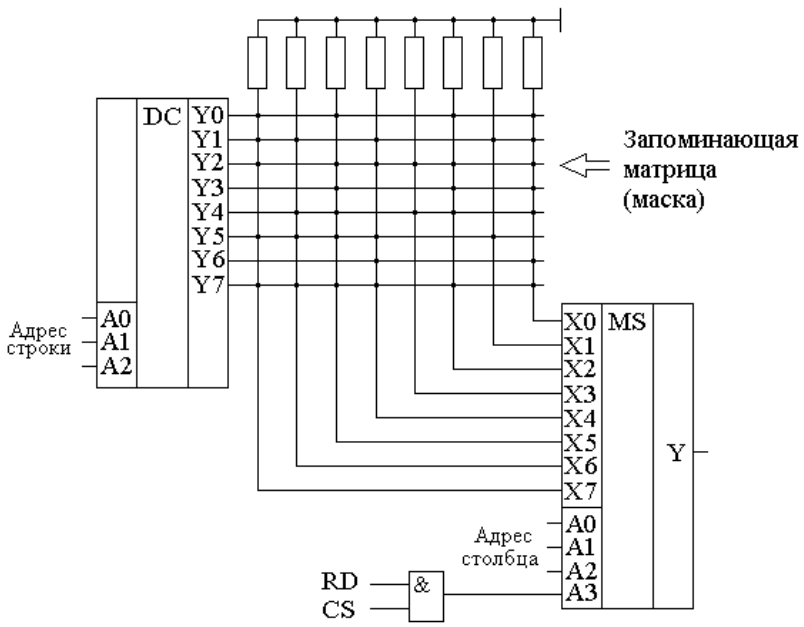


Рис. 61. Схема масочного постоянного запоминающего устройства.

Программирование масочного ПЗУ производится на заводе изготовителе, что очень неудобно для мелких и средних серий производства, не говоря уже о стадии разработки устройства. Естественно, что для крупносерийного производства масочные ПЗУ являются самым дешевым видом ПЗУ, и поэтому широко применяются в настоящее время. Для мелких и средних серий

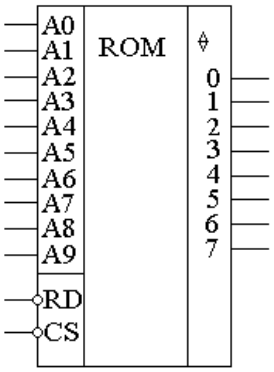


Рис. 62. Обозначение масочного постоянного запоминающего устройства на принципиальных схемах.

производства радиоаппаратуры были разработаны микросхемы, которые можно программировать в специальных устройствах - программаторах. В этих микросхемах постоянное соединение проводников в запоминающей матрице заменяется плавкими перемычками, изготовленными из поликристаллического кремния. При производстве микросхемы изготавливаются все перемычки, что эквивалентно записи во все ячейки памяти логических единиц. В процессе программирования на выводы питания и выходы микросхемы подается повышенное питание. При этом, если на выход микросхемы подается напряжение питания (логическая единица), то через перемычку ток протекать не будет и перемычка останется неповрежденной. Если же на выход микросхемы подать низкий уровень напряжения (присоединить к корпусу), то через перемычку будет протекать ток, который испарит эту перемычку. При последующем считывании информации из этой ячейки будет считываться логический ноль. Программируемые ПЗУ оказались очень удобны при мелкосерийном и среднесерийном производстве.

---ПЗУ, программируемые пользователем, с ультрафиолетовым стиранием — EPROM (Erasable Programmable ROM).

ПЗУ с ультрафиолетовым стиранием строится на основе запоминающей матрицы построенной на ячейках памяти, внутреннее устройство которой приведено на рис.63.

Ячейка представляет собой МОП транзистор, в котором затвор выполняется из поликристаллического кремния. Затем в процессе изготовления микросхемы этот затвор окисляется и в результате он будет окружен оксидом кремния -

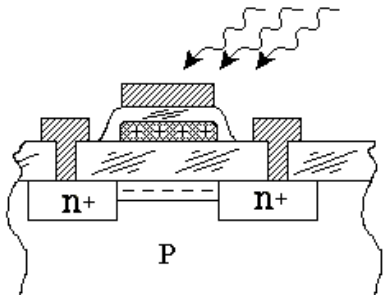


Рис. 63. Запоминающая ячейка ПЗУ с ультрафиолетовым и электрическим стиранием.

диэлектриком с прекрасными изолирующими свойствами. В описанной ячейке при полностью стертом ПЗУ заряда в плавающем затворе нет, и поэтому транзистор ток не проводит. При программировании микросхемы на второй затвор, находящийся над плавающим затвором, подается высокое напряжение и в плавающий затвор за счет туннельного эффекта индуцируются заряды. После снятия программирующего напряжения на плавающем затворе индуцированный заряд остается и, следовательно, транзистор остается в проводящем состоянии. Заряд на плавающем затворе может храниться десятки лет.

Структурная схема постоянного запоминающего устройства не отличается от описанного ранее масочного ПЗУ. Единственно вместо перемычки используется описанная выше ячейка. В репрограммируемых ПЗУ стирание ранее записанной информации осуществляется ультрафиолетовым излучением. Для того, чтобы этот свет мог беспрепятственно проходить к

полупроводниковому кристаллу, в корпус микросхемы встраивается окошко из кварцевого стекла.

При облучении микросхемы, изолирующие свойства оксида кремния теряются и накопленный заряд из плавающего затвора стекает в объем полупроводника и транзистор запоминающей ячейки переходит в закрытое состояние. Время стирания микросхемы колеблется в пределах 10 - 30 минут. Количество циклов записи - стирания микросхем находится в диапазоне от 10 до 100 раз, после чего микросхема выходит из строя. Это связано с разрушающим воздействием ультрафиолетового излучения. В качестве примера таких микросхем можно назвать микросхемы 573 серии российского производства, микросхемы серий 27cXXX зарубежного производства. В этих микросхемах чаще всего хранятся программы BIOS универсальных компьютеров. Репрограммируемые ПЗУ изображаются на принципиальных схемах как показано на рис. 64.

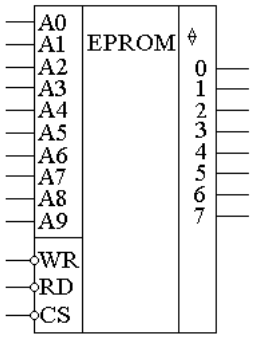


Рис. 64. Обозначение репрограммируемого постоянного запоминающего устройства на принципиальных схемах.

Такой керамический корпус с кварцевым окошком стоит довольно дорого, что значительно увеличивает стоимость МК. Для уменьшения стоимости МК с EPROM его заключают в корпус без окошка (версия EPROM с однократным программированием).

ПЗУ, однократно программируемые пользователем, — OTPROM (One - Time Programmable ROM). Они представляют собой версию EPROM, выполненную в корпусе без окошка для уменьшения стоимости МК на его основе. Сокращение стоимости при использовании таких корпусов настолько значительно, что в последнее время эти версии EPROM часто используют вместо масочных ПЗУ.

Так так корпуса с кварцевым окошком очень дороги, а также малое количество циклов записи - стирания привели к поиску способов стирания информации из ППЗУ электрическим способом. В качестве запоминающей ячейки в них используются такие же ячейки как и в РПЗУ, но они стираются электрическим потенциалом, поэтому количество циклов записи - стирания для этих микросхем достигает 1000000 раз. Время стирания ячейки памяти в таких микросхемах уменьшается до 10 мс. Схема управления для таких микросхем получилась сложная, поэтому наметилось два направления развития этих микросхем:

--- ЭСППЗУ;

--- FLASH –ПЗУ.

Электрически стираемые ППЗУ дороже и меньше по объему, но зато позволяют перезаписывать каждую ячейку памяти отдельно. В результате эти микросхемы обладают максимальным количеством циклов записи - стирания. Область применения электрически стираемых ПЗУ - хранение данных, которые не должны стираться при выключении питания. К таким микросхемам относятся отечественные микросхемы 573РР3, 558РР и зарубежные микросхемы серии 28сХХ. Электрически стираемые ПЗУ обозначаются на схемах как показано на рис. 65.

--- ПЗУ, программируемые пользователем, с электрическим стиранием — EEPROM (Electrically Erasable Programmable ROM). ПЗУ данного типа можно считать новым поколением EPROM, в которых стирание ячеек памяти производится также электрическими сигналами за счет использования

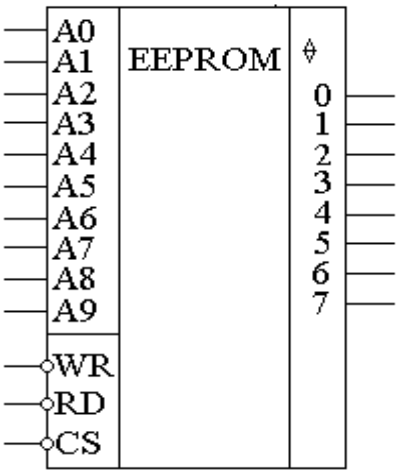


Рис. 65. Обозначение электрически стираемого постоянного запоминающего устройства на принципиальных схемах.

туннельных механизмов. Применение EEPROM позволяет стирать и программировать МК, не снимая его с платы. Таким способом можно производить отладку и модернизацию программного обеспечения. Это дает огромный выигрыш на начальных стадиях разработки микроконтроллерных систем или в процессе их изучения, когда много времени уходит на поиск причин неработоспособности системы и выполнение циклов стирания-программирования памяти программ. Технология программирования памяти EEPROM допускает побайтовое стирание и программирование ячеек. Несмотря на очевидные преимущества EEPROM, только в редких моделях МК такая память используется для хранения программ. Связано это с тем, что, во-первых, EEPROM имеют ограниченный объем памяти. Во-вторых, почти одновременно с EEPROM появились Flash-ПЗУ, которые при сходных потребительских характеристиках имеют более низкую стоимость;

--- ПЗУ с электрическим стиранием типа Flash — Flash-ROM.

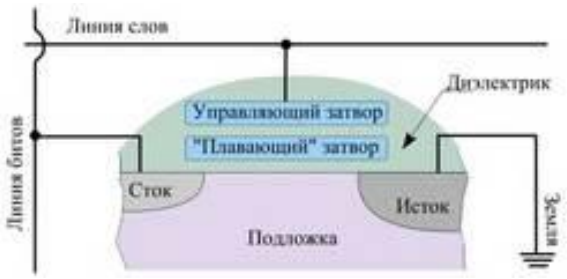


Рис. 66. Ячейка флэш-памяти на одном транзисторе.

Ячейки флэш-памяти бывают как на одном (рис.66), так и на двух транзисторах. В простейшем случае каждая ячейка хранит один бит информации и состоит из одного полевого транзистора со специальной электрически изолированной областью ("плавающим" затвором - floating gate), способной хранить заряд многие годы. Наличие или отсутствие заряда кодирует один бит информации.

При записи заряд помещается на плавающий затвор одним из двух способов (зависит от типа ячейки): методом инжекции "горячих" электронов или методом туннелирования электронов. Стирание содержимого ячейки (снятие заряда с "плавающего" затвора) производится методом туннелирования. Как правило, наличие заряда на транзисторе понимается как логический "0", а его отсутствие - как логическая "1". Современная флэш-память обычно изготавливается по 0,13- и 0,18-микронному техпроцессу.

Общий принцип работы ячейки флэш-памяти. Рассмотрим простейшую ячейку флэш-памяти на одном n – p - n транзисторе. Ячейки подобного типа чаще всего применяются в микросхемах EPROM (табл. 3).

Поведение транзистора зависит от количества электронов на "плавающем" затворе. "Плавающий" затвор играет ту же роль, что и конденсатор в динамической памяти, т. е. хранит запрограммированное значение.

Помещение заряда на "плавающий" затвор в такой ячейке производится методом инжекции "горячих" электронов (CHE - channel hot electrons), а снятие заряда осуществляется методом квантомеханического туннелирования.

Эффект туннелирования - один из эффектов, использующих волновые свойства электрона. Сам эффект заключается в преодолении электроном потенциального барьера малой "толщины". Для наглядности представим себе структуру, состоящую из двух проводящих областей, разделенных тонким слоем диэлектрика (обеднённая область). Преодолеть этот слой обычным способом электрон не может - не хватает энергии. Но при создании

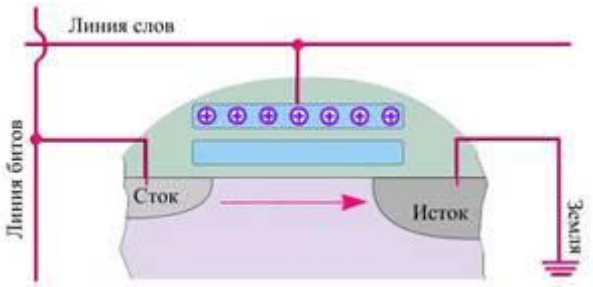
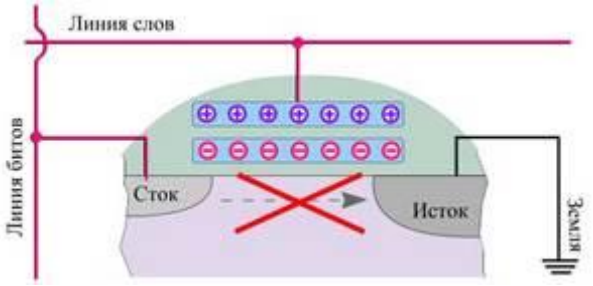
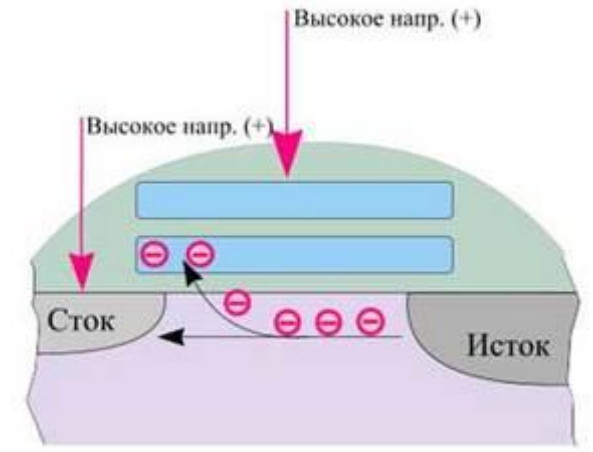
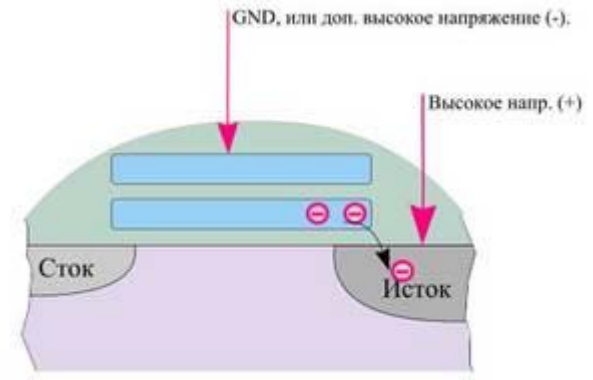
определённых условий (соответствующее напряжение и т.п.) электрон проскакивает слой диэлектрика (туннелирует сквозь него), создавая ток.
Важно отметить, что при туннелировании электрон оказывается "по другую сторону", не проходя через диэлектрик.

Функционально Flash-память (рис. 67) мало отличается от EEPROM. Основное различие состоит в способе стирания записанной информации. В памяти EEPROM стирание производится отдельно для каждой ячейки, а во Flash-памяти стирать можно только целыми блоками. Если необходимо изменить содержимое одной ячейки Flash-памяти, потребуется перепрограммировать весь блок. Упрощение декодирующих схем по сравнению с EEPROM привело к тому, что МК с Flash-памятью становятся конкурентоспособными по отношению не только к МК с однократно программируемыми ПЗУ, но и с масочными ПЗУ также.

Существует три основных типа доступа к флэш-памяти:
--- обычный (Conventional): произвольный асинхронный доступ к ячейкам памяти.
--- пакетный (Burst): синхронный, данные читаются параллельно, блоками по 16 или 32 слова. Считанные данные передаются последовательно, передача синхронизируется. Преимущество перед обычным типом доступа - быстрое последовательное чтение данных. Недостаток - медленный произвольный доступ.
--- страничный (Page): асинхронный, блоками по 4 или 8 слов. Преимущества: очень быстрый произвольный доступ в пределах текущей страницы. Недостаток: относительно медленное переключение между страницами.
Существуют микросхемы флэш-памяти, позволяющие одновременную запись и стирание (RWW - Read While Write или Simultaneous R/W) в разные банки памяти.

Т а б л и ц а 3

Общий принцип работы ячейки флэш-памяти.

	При чтении, в отсутствие заряда на "плавающем" затворе, под воздействием положительного поля на управляющем затворе, образуется n-канал в подложке между истоком и стоком, и возникает ток.
	Наличие заряда на "плавающем" затворе меняет вольт - амперные характеристики транзистора таким образом, что при обычном для чтения напряжении канал не появляется, и тока между истоком и стоком не возникает.
	При программировании, на сток и управляющий затвор подаётся высокое напряжение (на управляющий затвор напряжение подаётся приблизительно в два раза выше). "Горячие" электроны из канала инжектируются на плавающий затвор и изменяют вольт - амперные характеристики транзистора.
	При стирании высокое напряжение подаётся на исток. На управляющий затвор (опционально) подаётся высокое отрицательное напряжение. Электроны туннелируют на исток.

Электроны называют "горячими" за то, что обладают высокой энергией, достаточной для преодоления потенциального барьера, создаваемого тонкой плёнкой диэлектрика.

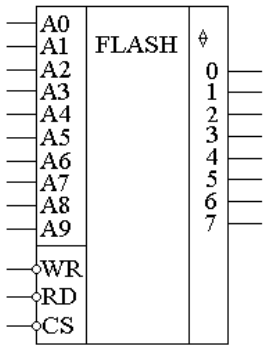


Рис. 67. Обозначение Flash-памяти на принципиальных схемах.

В общем случае при обращении к постоянному запоминающему устройству сначала необходимо выставить адрес ячейки памяти на шине адреса, а затем произвести операцию чтения из микросхемы. Эта временная диаграмма приведена на рис. 68.

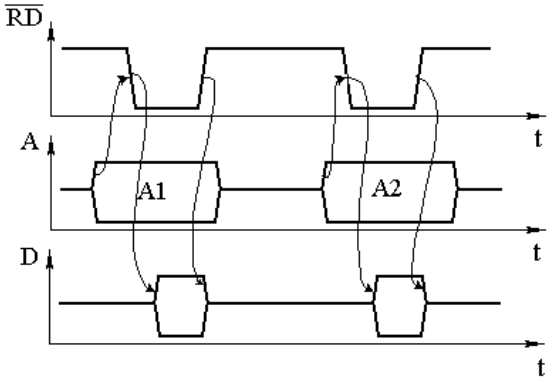


Рис. 68. Временная диаграмма чтения информации из ПЗУ.

На рис. 68 стрелками показана последовательность, в которой должны формироваться управляющие сигналы. На этом рисунке RD - это сигнал чтения, A - сигналы выбора адреса ячейки (так как отдельные биты в шине адреса могут принимать разные значения, то показаны пути перехода как в

единичное, так и в нулевое состояние), D - выходная информация, считанная из выбранной ячейки ПЗУ.

Память данных МК выполняется, как правило, на основе статического ОЗУ. Термин «статическое» означает, что содержимое ячеек ОЗУ сохраняется при снижении тактовой частоты МК до сколь угодно малых значений (с целью снижения энергопотребления). Большинство МК имеют такой параметр, как «напряжение хранения информации» — $U_{STANDBY}$. При снижении напряжения питания ниже минимально допустимого уровня U_{DDMIN}, но выше уровня $U_{STANDBY}$ работа программы МК выполняться не будет, но информация в ОЗУ сохраняется. При восстановлении напряжения питания можно будет сбросить МК и продолжить выполнение программы без потери данных. Уровень напряжения хранения составляет обычно около 1В, что позволяет в случае необходимости перевести МК на питание от автономного источника (батареи) и сохранить в этом режиме данные ОЗУ.

Объем памяти данных МК, как правило, невелик и составляет обычно десятки и сотни байт. Это обстоятельство необходимо учитывать при разработке программ для МК. Так, при программировании МК константы, если возможно, не хранятся как переменные, а заносятся в ПЗУ программ. Максимально используются аппаратные возможности МК, в частности, таймеры. Прикладные программы должны ориентироваться на работу без использования больших массивов данных.

8.4. Регистры МК

Как и все МПС, МК имеют набор регистров, которые используются для управления его ресурсами. В число этих регистров входят обычно регистры процессора (аккумулятор, регистры состояния, индексные регистры), регистры управления (регистры управления прерываниями, таймером), регистры, обеспечивающие ввод/вывод данных (регистры данных портов, регистры

управления параллельным, последовательным или аналоговым вводом/выводом). Обращение к этим регистрам может производиться по-разному.

В МК с RISC-процессором все регистры (часто и аккумулятор) располагаются по явно задаваемым адресам. Это обеспечивает более высокую гибкость при работе процессора.

Одним из важных вопросов является размещение регистров в адресном пространстве МК. В некоторых МК все регистры и память данных располагаются в одном адресном пространстве. Это означает, что память данных совмещена с регистрами. Такой подход называется «отображением ресурсов МК на память».

В других МК адресное пространство устройств ввода/вывода отделено от общего пространства памяти. Отдельное пространство ввода/вывода дает некоторое преимущество процессорам с гарвардской архитектурой, обеспечивая возможность считывать команду во время обращения к регистру ввода/вывода.

8.5. Стек МК

В микроконтроллерах ОЗУ данных используется также для организации вызова подпрограмм и обработки прерываний. При этих операциях содержимое программного счетчика и основных регистров (аккумулятор, регистр состояния и другие) сохраняется и затем восстанавливается при возврате к основной программе.

В фон-неймановской архитектуре единая область памяти используется, в том числе, и для реализации стека. При этом снижается производительность устройства, так как одновременный доступ к различным видам памяти невозможен. В частности, при выполнении команды вызова подпрограммы

следующая команда выбирается после того, как в стек будет помещено содержимое программного счетчика.

В гарвардской архитектуре стековые операции производятся в специально выделенной для этой цели памяти. Это означает, что при выполнении программы вызова подпрограмм процессор с гарвардской архитектурой производит несколько действий одновременно.

Необходимо помнить, что МК обеих архитектур имеют ограниченную емкость памяти для хранения данных. Если в процессоре имеется отдельный стек и объем записанных в него данных превышает его емкость, то происходит циклическое изменение содержимого указателя стека, и он начинает ссылаться на ранее заполненную ячейку стека. Это означает, что после слишком большого количества вызовов подпрограмм в стеке окажется неправильный адрес возврата. Если МК использует общую область памяти для размещения данных и стека, то существует опасность, что при переполнении стека произойдет запись в область данных либо будет сделана попытка записи загружаемых в стек данных в область ПЗУ.

8.6. Внешняя память

Несмотря на существующую тенденцию по переходу к закрытой архитектуре МК, в некоторых случаях возникает необходимость подключения дополнительной внешней памяти (как памяти программ, так и данных).

Если МК содержит специальные аппаратные средства для подключения внешней памяти, то эта операция производится штатным способом (как для МП).

Второй, более универсальный, способ заключается в том, чтобы использовать порты ввода/вывода для подключения внешней памяти и реализовать обращение к памяти программными средствами. Такой способ позволяет задействовать простые устройства ввода/вывода без реализации сложных

шинных интерфейсов, однако приводит к снижению быстродействия системы при обращении к внешней памяти

9. Организация связи микроконтроллера с внешней средой и временем

9.1. Порты ввода/вывода

Каждый МК имеет некоторое количество линий ввода/вывода, которые объединены в многоразрядные (чаще 8-разрядные) параллельные порты ввода/вывода. В памяти МК каждому порту ввода/вывода соответствует свой адрес регистра данных. Обращение к регистру данных порта ввода/вывода производится теми же командами, что и обращение к памяти данных. Кроме того, во многих МК отдельные разряды портов могут быть опрошены или установлены командами битового процессора.

В зависимости от реализуемых функций различают следующие типы параллельных портов*:*

--- однонаправленные порты, предназначенные только для ввода или только для вывода информации;

--- двунаправленные порты, направление передачи которых (ввод или вывод) определяется в процессе инициализации МК;

--- порты с альтернативной функцией (мультиплексированные порты). Отдельные линии этих портов используются совместно со встроенными периферийными устройствами МК, такими как таймеры, АЦП, контроллеры последовательных интерфейсов;

--- порты с программно управляемой схемотехникой входного/выходного буфера.

Порты выполняют роль устройств временного согласования функционирования МК и объекта управления, которые в общем случае

работают асинхронно. Различают три типа алгоритмов обмена информацией между МК и внешним устройством через параллельные порты ввода/вывода:

--- режим простого программного ввода/вывода;

--- режим ввода/вывода со стробированием;

--- режим ввода/вывода с полным набором сигналов подтверждения обмена.

Типовая схема двунаправленного порта ввода/вывода МК приведена на рис. 69.

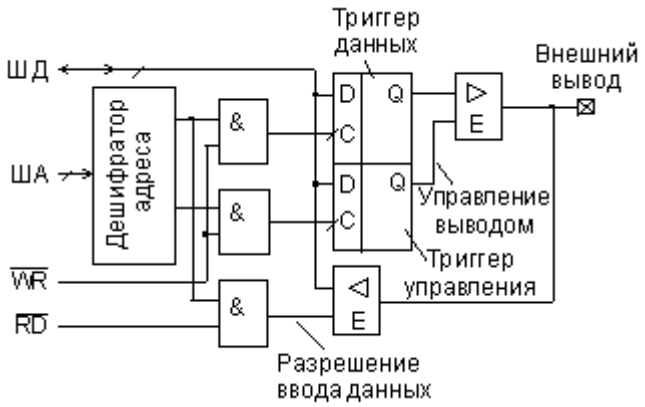


Рис.69. Типовая схема двунаправленного порта ввода/вывода МК.

Триггер управления разрешает вывод данных на внешний вывод. В современных МК, как правило, обеспечивается индивидуальный доступ к триггерам данных и управления, что позволяет использовать каждую линию независимо в режиме ввода или вывода.

Необходимо обратить особое внимание на то, что при вводе данных считывается значение сигнала, поступающее на внешний вывод, а не содержимое триггера данных. Если к внешнему выводу МК подключены выходы других устройств, то они могут установить свой уровень выходного сигнала, который и будет считан вместо ожидаемого значения триггера данных.

Другим распространенным вариантом схемотехнической организации порта ввода/вывода является вывод с "открытым истоком", называемый еще

"квазидвунаправленным". Такая организация вывода позволяет создавать шины с объединением устройств по схеме "монтажное И".

9.2. Таймеры и процессоры событий

Большинство задач управления, которые реализуются с помощью МК, требуют исполнения их в реальном времени. Под этим понимается способность системы получить информацию о состоянии управляемого объекта, выполнить необходимые расчетные процедуры и выдать управляющие воздействия в течение интервала времени, достаточного для желаемого изменения состояния объекта.

Возлагать функции формирования управления в реальном масштабе времени только на центральный процессор неэффективно, так как это занимает ресурсы, необходимые для расчетных процедур. Поэтому в большинстве современных МК используется аппаратная поддержка работы в реальном времени с использованием таймера (таймеров).

Модули таймеров служат для приема информации о времени наступления тех или иных событий от внешних датчиков событий, а также для формирования управляющих воздействий во времени.

Модуль таймера 8-разрядного МК представляет собой 8-ми или 16-разрядный счетчик со схемой управления. Схемотехникой МК обычно предусматривается возможность использования таймера в режиме счетчика внешних событий, поэтому его часто называют таймером/счетчиком. Структура типичного 16-разрядного таймера/счетчика в составе МК приведена на рис. 70.

В памяти МК 16-разрядный счетчик отображается двумя регистрами: TH — старший байт счетчика, TL — младший байт. Регистры доступны для чтения и для записи. Направление счета — только прямое, то есть при поступлении

входных импульсов содержимое счетчика инкрементируется. В зависимости от настройки счетчик может использовать один из источников входных сигналов:

--- импульсную последовательность с выхода управляемого делителя частоты f_{BUS};

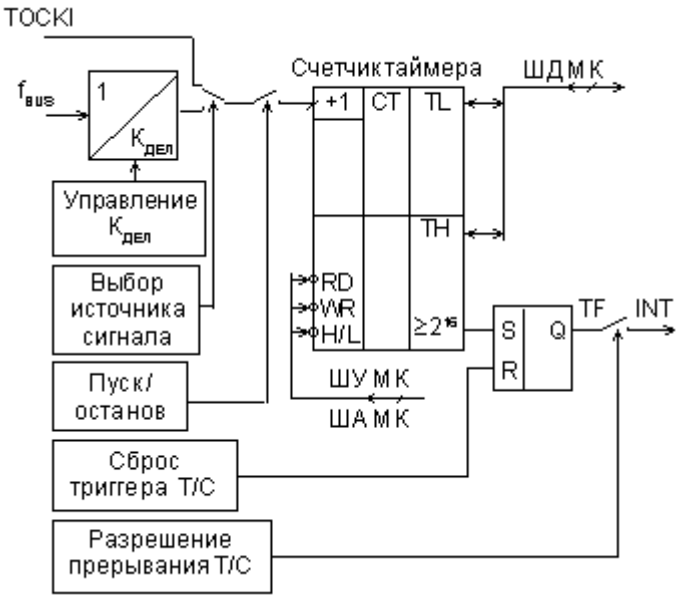


Рис. 70. Структура модуля таймера/счетчика.

--- сигналы внешних событий, поступающие на вход TOCKI контроллера.

В первом случае говорят, что счетчик работает в режиме таймера, во втором — в режиме счетчика событий. При переполнении счетчика устанавливается в "единицу" триггер переполнения TF, который генерирует запрос на прерывание, если прерывания от таймера разрешены. Пуск и останов таймера могут осуществляться только под управлением программы. Программным способом можно также установить старший и младший биты счетчика в произвольное состояние или прочитать текущий код счетчика.

Рассмотренный "классический" модуль таймера/счетчика широко применяется в различных моделях относительно простых МК. Он может использоваться для измерения временных интервалов и формирования последовательности

импульсов. Основными недостатками "классического" таймера/счетчика являются:

--- потери времени на выполнение команд пуска и останова таймера, приводящие к появлению ошибки при измерении временных интервалов и ограничивающие минимальную длительность измеряемых интервалов времени единицами мс;

--- сложности при формировании временных интервалов (меток времени), отличных от периода полного коэффициента счета, равного $(Кдел/f_{BUS}) \cdot 2^{16}$;

невозможность одновременного обслуживания (измерения или формирования импульсного сигнала) сразу нескольких каналов.

Первые из двух перечисленных недостатков были устранены в усовершенствованном модуле таймера/счетчика, используемом в МК семейства MCS-51. Дополнительная логика счетного входа позволяет тактовым импульсам поступать на вход счетчика, если уровень сигнала на одной из линий ввода равен "1". Такое решение повышает точность измерения временных интервалов, так как пуск и останов таймера производится аппаратно. Также в усовершенствованном таймере реализован режим перезагрузки счетчика произвольным кодом в момент переполнения. Это позволяет формировать временные последовательности с периодом, отличным от периода полного коэффициента счета.

Однако эти усовершенствования не устраняют главного недостатка модуля "классического" таймера — одноканального режима работы. Совершенствование подсистемы реального времени МК ведется по следующим направлениям:

--- увеличение числа модулей таймеров/счетчиков. Этот путь характерен для фирм, выпускающих МК со структурой MCS-51, а также для МК компаний Mitsubishi и Hitachi;

--- модификация структуры модуля таймера/счетчика. В этом случае увеличение числа каналов достигается не за счет увеличения числа счетчиков, а за счет введения дополнительных аппаратных средств входного захвата (input capture — IC) и выходного сравнения (output compare — OC). Такой подход используется, в частности, в МК компании Motorola. Принцип действия канала входного захвата таймера/счетчика иллюстрируется рис. 71.

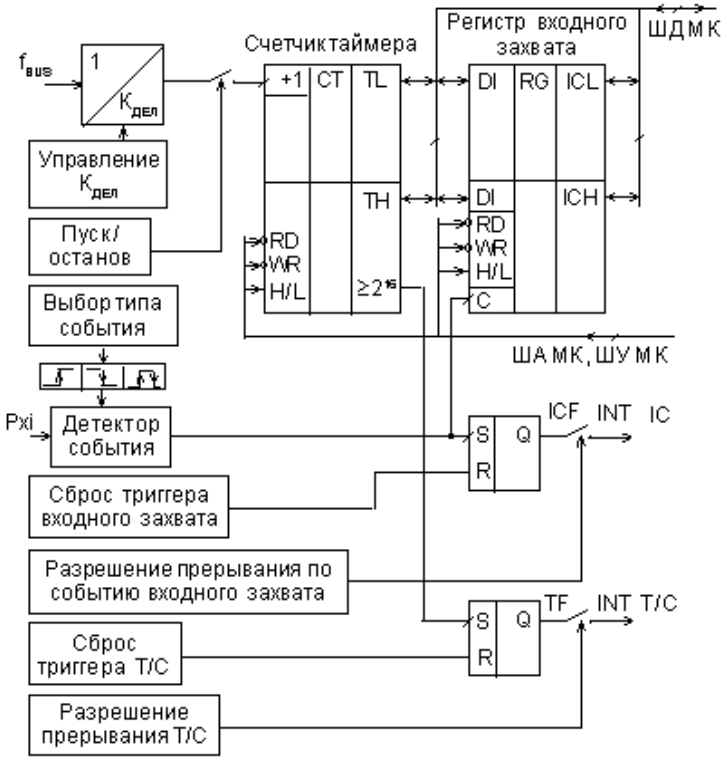


Рис. 71. Структурная схема канала входного захвата таймера.

Схема детектора события "наблюдает" за уровнем напряжения на одном из входов МК. Чаще всего это одна из линий порта ввода/вывода. При изменении уровня логического сигнала с "0" на "1" и наоборот вырабатывается строб записи, и текущее состояние счетчика таймера записывается в 16-разрядный регистр входного захвата. Описанное действие в микропроцессорной технике называют событием захвата. Предусмотрена возможность выбора типа сигнала на входе, и это воспринимается как событие:

--- положительный (передний) фронт сигнала;

--- отрицательный (задний) фронт сигнала;

--- любое изменение логического уровня сигнала.

Выбор типа события захвата устанавливается в процессе инициализации таймера и может неоднократно изменяться в ходе выполнения программы. Каждое событие захвата приводит к установке в "1" триггера входного захвата и появлению на его выходе флага (признака) входного захвата ICF. Состояние триггера входного захвата может быть считано программно, а если прерывания по событию захвата разрешены — формируется запрос на прерывание INT IC. Использование режима входного захвата позволяет исключить ошибки измерения входного интервала времени, связанные со временем перехода к подпрограмме обработки прерывания, так как копирование текущего состояния счетчика осуществляется аппаратными, а не программными средствами. Однако время перехода на подпрограмму обработки прерывания накладывает ограничение на длительность измеряемого интервала времени, так как предполагается, что второе событие захвата произойдет позже, чем код первого события будет считан МК.

Структурная схема канала выходного сравнения представлена на рис. 72. Цифровой компаратор непрерывно сравнивает текущий код счетчика таймера с кодом, который записан в 16-разрядном регистре выходного сравнения. В момент равенства кодов на одном из выходов M(Pxj) на рис. 72 устанавливается заданный уровень логического сигнала. Обычно предусмотрено три типа изменения сигнала на выходе Pxj в момент события выходного сравнения:

--- установка высокого логического уровня;

--- установка низкого логического уровня;

--- инвертирование сигнала на выходе.

При наступлении события сравнения устанавливаются в "1" триггер выходного сравнения и соответствующий ему признак выходного сравнения OCF. Аналогично режиму входного захвата состояние триггера выходного

сравнения может быть считано программно, а если прерывания по событию сравнения разрешены — формируется запрос на прерывание INT OC.

Режим выходного сравнения предназначен, прежде всего, для формирования временных интервалов заданной длительности. Длительность сформированного

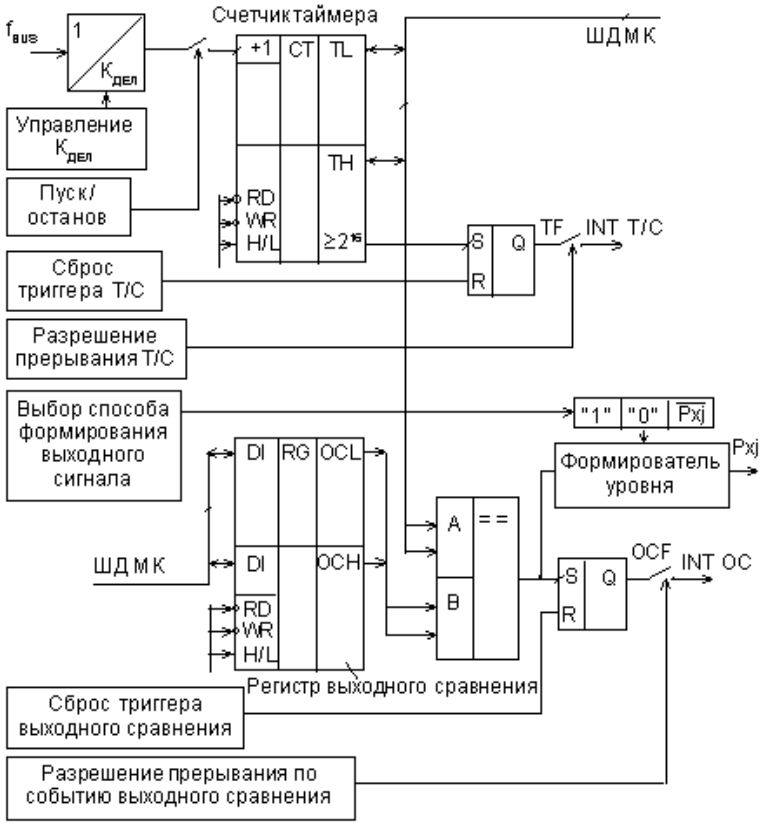


Рис. 72. Структурная схема канала выходного сравнения таймера.

временного интервала определяется только разностью кодов, последовательно загружаемых в регистр выходного сравнения, и не зависит от программного обеспечения МК. Время, необходимое для записи нового значения кода в регистр канала сравнения, ограничивает минимальную длительность формируемого временного интервала.

Модули усовершенствованного таймера используются в составе МК в различных модификациях. При этом число каналов входного захвата и выходного сравнения в модуле может быть различным. Так, в МК семейства HC05 фирмы Motorola типовыми решениями являются модули 1IC+1OC или

2IC+2OC, а модуль таймера в составе МК только один. В ряде модулей каналы могут быть произвольно настроены на функцию входного захвата или выходного сравнения посредством инициализации. Счетчик модуля усовершенствованного таймера может не иметь функции программного останова. В этом случае состояние счетчика нельзя синхронизировать с каким-либо моментом работы МК, и такой счетчик характеризуется как
свободно считающий (free counter). Аппаратные средства усовершенствованного таймера позволяют решить многие задачи управления в реальном времени. Однако по мере роста сложности алгоритмов управления отчетливо проявляются ограничения модулей усовершенствованного таймера, а именно:

--- недостаточное число каналов захвата и сравнения, принадлежащих одному счетчику временной базы. Это не позволяет сформировать синхронизированные между собой многоканальные импульсные последовательности;

--- однозначно определенная конфигурация канала (или захват или сравнение) часто не удовлетворяет потребностям решаемой задачи;

--- формирование сигналов по методу широтно-импульсной модуляции (ШИМ) требует программной поддержки, что снижает максимально достижимую частоту выходного сигнала.

Поэтому следующим этапом развития модулей подсистемы реального времени МК стали модули процессоров событий. Впервые модули процессоров событий были использованы компанией Intel в МК семейства 8xC51Fx. Этот модуль получил название программируемого счетного массива (Programmable Counter Array — PCA).

PCA обеспечивает более широкие возможности работы в реальном масштабе времени и в меньшей степени расходует ресурсы центрального процессора, чем стандартный и усовершенствованный таймеры/счетчики. К преимуществам

PCA также можно отнести более простое программирование и более высокую точность. К примеру, PCA может обеспечить лучшее временное разрешение, чем таймеры 0, 1 и 2 МК семейства MCS-51, так как счетчик PCA способен работать с тактовой частотой, втрое большей, чем у этих таймеров. PCA также может решать многие задачи, выполнение которых с использованием таймеров требует дополнительных аппаратных затрат (например, определение фазового сдвига между импульсами или генерация ШИМ - сигнала). PCA состоит из 16-битного таймера-счетчика и пяти 16-битных модулей сравнения-защелки, как показано на рис. 73.

Таймер-счетчик PCA используется в качестве базового таймера для функционирования всех пяти модулей сравнения-защелки. Вход таймера-счетчика PCA может быть запрограммирован на счет сигналов от следующих источников:

--- выход делителя на 12 тактового генератора МК;

--- выход делителя на 4 тактового генератора МК;

--- сигнал переполнения таймера 0;

--- внешний входной сигнал на выводе ECI (P1.2).

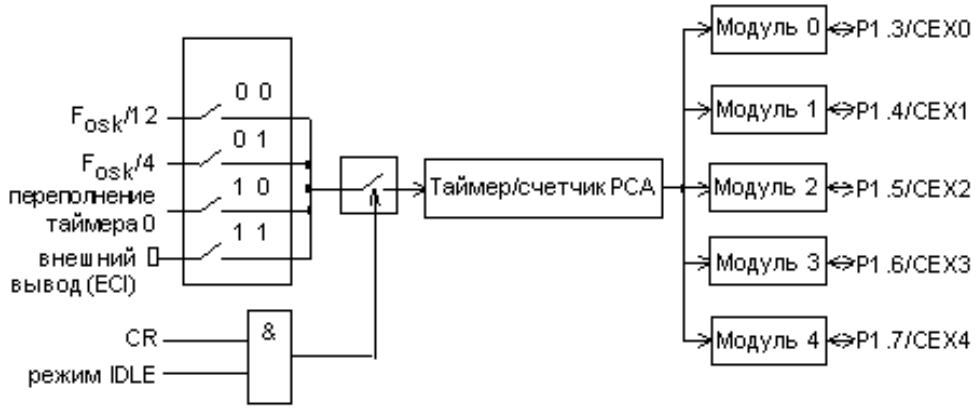


Рис. 73. Структура процессора событий МК семейства Intel 8xC51Fx.

Любой из модулей сравнения-защелки может быть запрограммирован для работы в следующих режимах:

--- защелкивания по фронту и/или спаду импульса на входе CEXi;

--- программируемого таймера;

--- высокоскоростного выхода;

--- широтно-импульсного модулятора.

Модуль 4 может быть также запрограммирован как сторожевой таймер (Watchdog Timer – WDT).

Режим защелкивания по импульсу на входе МК эквивалентен режиму входного захвата (IC) усовершенствованного таймера. Режимы программируемого таймера и высокоскоростного выхода близки по своим функциональным возможностям к режиму выходного сравнения (ОС).

В режиме ШИМ на соответствующем выводе МК формируется последовательность импульсов с периодом, равным периоду базового таймера/счетчика PCA. Значение 8-разрядного кода, записанное в младший байт регистра-защелки соответствующего модуля задает скважность формируемого сигнала. При изменении кода от 0 до 255 скважность меняется от 100 % до 0,4 %.

Режим ШИМ очень прост с точки зрения программного обслуживания. Если изменения скважности не предполагается, то достаточно один раз занести соответствующий код в регистр данных модуля, проинициализировать режим ШИМ, и импульсная последовательность будет воспроизводиться с заданными параметрами без вмешательства программы.

Назначение и особенности работы сторожевого таймера будут рассмотрены далее отдельно.

При работе модуля сравнения-защелки в режиме защелки, программируемого таймера или высокоскоростного выхода модуль может сформировать сигнал прерывания. Сигналы от всех пяти модулей сравнения-защелки и сигнал переполнения таймера PCA разделяют один вектор прерывания. Иными словами, если прерывания разрешены, то и сигнал переполнения таймера PCA

и сигнал от любого из модулей вызывают одну и ту же подпрограмму прерываний, которая должна сама идентифицировать источник, вызвавший ее.
Для работы с внешними устройствами таймер-счетчик PCA и модули сравнения-защелки используют выводы P1 порта МК. Если какой-либо вывод порта не используется при работе PCA, или PCA не задействован, порт может применяться стандартным образом.
Реализованный в 8xC51FX PCA оказался настолько удачным, что архитектура данных МК стала промышленным стандартом де-факто, а сам PCA многократно воспроизводился в различных модификациях микроконтроллеров разных фирм.
Тенденция развития подсистемы реального времени современных МК находит свое отражение в увеличении числа каналов процессоров событий и расширении их функциональных возможностей.

9.3. Модуль прерываний МК

Обработка прерываний в МК происходит в соответствии с общими принципами обработки прерываний в МПС. Модуль прерываний принимает запросы прерывания и организует переход к выполнению определенной прерывающей программы. Запросы прерывания могут поступать как от внешних источников, так и от источников, расположенных в различных внутренних модулях МК. В качестве входов для приема запросов от внешних источников чаще всего используются выводы параллельных портов ввода/вывода, для которых эта функция является альтернативной. Источниками запросов внешних прерываний также могут быть любые изменения внешних сигналов на некоторых специально выделенных линиях портов ввода/вывода.
Источниками внутренних запросов прерываний могут служить следующие события:

--- переполнение таймеров/счетчиков;

--- сигналы от каналов входного захвата и выходного сравнения --- таймеров/счетчиков или от процессора событий;

--- готовность памяти EEPROM;

--- сигналы прерывания от дополнительных модулей МК, включая завершение передачи или приема информации по одному из последовательных портов и другие.

Любой запрос прерывания поступает на обработку, если прерывания в МК разрешены и разрешено прерывание по данному запросу. Адрес, который загружается в программный счетчик при переходе к обработке прерывания, называется "вектор прерывания". В зависимости от организации модуля прерываний конкретного МК различные источники прерываний могут иметь разные векторы или использовать некоторые из них совместно. Использование различными прерываниями одного вектора обычно не вызывает проблем при разработке программного обеспечения, так как аппаратная часть МК фиксирована, а контроллер чаще всего выполняет одну-единственную программу.

Вопрос о приоритетах при одновременном поступлении нескольких запросов на прерывание решается в различных МК по-разному. Есть МК с одноуровневой системой приоритетов (все запросы равноценны), многоуровневой системой с фиксированными приоритетами и многоуровневой программируемой системой приоритетов.

Отдельно необходимо описать аппаратные прерывания, связанные с включением питания, подачей сигнала "сброс" и переполнением сторожевого таймера. Они имеют немаскируемый характер и чаще всего разделяют один общий вектор прерывания. Это вполне логично, поскольку результатом каждого из событий является начальный сброс МК.

10. Вспомогательные аппаратные средства микроконтроллера

10.1. Минимизация энергопотребления в системах на основе МК

Малый уровень энергопотребления является зачастую определяющим фактором при выборе способа реализации цифровой управляющей системы. Современные МК предоставляют пользователю большие возможности в плане экономии энергопотребления и имеют, как правило, следующие основные режимы работы:

--- активный режим (Run mode) — основной режим работы МК. В этом режиме МК исполняет рабочую программу и все его ресурсы доступны. Потребляемая мощность имеет максимальное значение P_{RUN}. Большинство современных МК выполнено по КМОП - технологии, поэтому мощность потребления в активном режиме сильно зависит от тактовой частоты;

--- режим ожидания (Wait mode, Idle mode или Halt mode). В этом режиме прекращает работу центральный процессор, но продолжают функционировать периферийные модули, которые контролируют состояние объекта управления. При необходимости сигналы от периферийных модулей переводят МК в активный режим, и рабочая программа формирует необходимые управляющие воздействия. Перевод МК из режима ожидания в рабочий режим осуществляется по прерываниям от внешних источников или периферийных модулей, либо при сбросе МК. В режиме ожидания мощность потребления МК PWAIT снижается по сравнению с активным режимом в 5…10 раз;

--- режим останова (Stop mode, Sleep mode или Power Down mode). В этом режиме прекращает работу, как центральный процессор, так и большинство периферийных модулей. Переход МК из состояния останова в рабочий режим возможен, как правило, только по прерываниям от внешних источников или после подачи сигнала сброса. В режиме останова мощность потребления МК

P_{STOP} снижается по сравнению с активным режимом примерно на три порядка и составляет единицы микроватт.

Два последних режима называют режимами пониженного энергопотребления. Минимизация энергопотребления системы на МК достигается за счет оптимизации мощности потребления МК в активном режиме, а также использования режимов пониженного энергопотребления. При этом необходимо иметь в виду, что режимы ожидания и останова существенно отличаются временем перехода из режима пониженного энергопотребления в активный режим. Выход из режима ожидания обычно происходит в течение 3…5 периодов синхронизации МК, в то время как задержка выхода из режима останова составляет несколько тысяч периодов синхронизации. Кроме снижения динамики работы системы значительное время перехода в активный режим является причиной дополнительного расхода энергии.

Мощность потребления МК в активном режиме является одной из важнейших характеристик контроллера. Она в значительной степени зависит от напряжения питания МК и частоты тактирования. В зависимости от диапазона питающих напряжений все МК можно разделить на три основные группы:

--- МК с напряжением питания 5,0В±10%. Эти МК предназначены, как правило, для работы в составе устройств с питанием от промышленной или бытовой сети, имеют развитые функциональные возможности и высокий уровень энергопотребления.

--- МК с расширенным диапазоном напряжений питания: от 2,0…3,0В до 5,0-7,0В. МК данной группы могут работать в составе устройств как с сетевым, так и с автономным питанием.

--- МК с пониженным напряжением питания: от 1,8 до 3В. Эти МК предназначены для работы в устройствах с автономным питанием и обеспечивают экономный расход энергии элементов питания.

Зависимость тока потребления от напряжения питания МК почти прямо пропорциональная. Поэтому снижение напряжения питания весьма существенно понижает мощность потребления МК. Необходимо, однако, иметь в виду, что для многих типов МК с понижением напряжения питания уменьшается максимально допустимая частота тактирования, то есть выигрыш в потребляемой мощности сопровождается снижением производительности системы.

В современных МК выполненных по технологии КМОП мощность потребления в активном режиме P_{RUN} практически прямо пропорциональна тактовой частоте. Поэтому, выбирая частоту тактового генератора, не следует стремиться к предельно высокому быстродействию МК в задачах, которые этого не требуют. Часто определяющим фактором оказывается разрешающая способность измерителей или формирователей временных интервалов на основе таймера или скорость передачи данных по последовательному каналу. Они способны работать при сколь угодно низких тактовых частотах, вплоть до нулевых. В справочных данных при этом указывается, что минимальная частота тактирования равна dc (direct current). Это означает, что возможно использование МК в пошаговом режиме, например, для отладки. Мощность потребления МК при низких частотах тактирования обычно отражает значение тока потребления при f_{OSC} = 32768 Гц (часовой кварцевый резонатор).

10.2. Тактовые генераторы МК

Современные МК содержат встроенные тактовые генераторы, которые требуют минимального числа внешних времязадающих элементов. На практике используются три основных способа определения тактовой частоты генератора: с помощью кварцевого резонатора, керамического резонатора и внешней RC-цепи.

Типовая схема подключения кварцевого или керамического резонатора приведена на рис. 74, а.

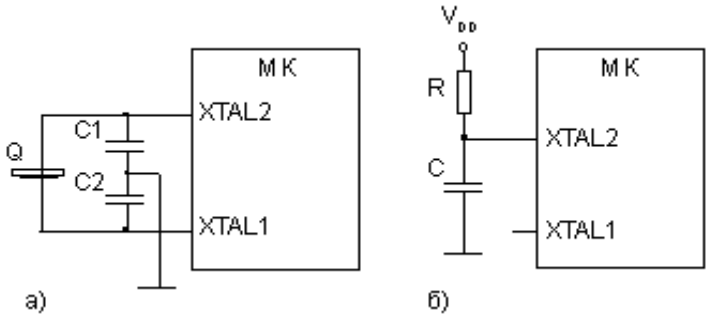


Рис. 74. Тактирование с использованием кварцевого или керамического резонатора (а) и с использованием RC-цепи (б).

Кварцевый или керамический резонатор Q подключается к выводам XTAL1 и XTAL2, которые обычно представляют собой вход и выход инвертирующего усилителя. Номиналы конденсаторов C1 и C2 определяются производителем МК для конкретной частоты резонатора. Иногда требуется включить резистор порядка нескольких мегаом между выводами XTAL1 и XTAL2 для стабильной работы генератора.

Использование кварцевого резонатора позволяет обеспечить высокую точность и стабильность тактовой частоты (разброс частот кварцевого резонатора обычно составляет менее 0,01%). Такой уровень точности требуется для обеспечения точного хода часов реального времени или организации интерфейса с другими устройствами. Основными недостатками кварцевого резонатора являются его низкая механическая прочность (высокая хрупкость) и относительно высокая стоимость.

При менее жестких требованиях к стабильности тактовой частоты возможно использование более стойких к ударной нагрузке керамических резонаторов. Многие керамические резонаторы имеют встроенные конденсаторы, что позволяет уменьшить количество внешних подключаемых элементов с трех до

одного. Керамические резонаторы имеют разброс частот порядка нескольких десятых долей процента (обычно около 0,5 %).

Самым дешевым способом задания тактовой частоты МК является использование внешней RC - цепи, как показано на рис. 74, б. Внешняя RC-цепь не обеспечивает высокой точности задания тактовой частоты (разброс частот может доходить до десятков процентов). Это неприемлемо для многих приложений, где требуется точный подсчет времени. Однако имеется масса практических задач, где точность задания тактовой частоты не имеет большого значения.

Зависимость тактовой частоты МК от номиналов RC-цепи зависит от конкретной реализации внутреннего генератора и приводится в руководстве по применению контроллера.

Практически все МК допускают работу от внешнего источника тактового сигнала, который подключается к входу XTAL1 внутреннего усилителя. При помощи внешнего тактового генератора можно задать любую тактовую частоту МК (в пределах рабочего диапазона) и обеспечить синхронную работу нескольких устройств.

Некоторые современные МК содержат встроенные RC или кольцевые генераторы, которые позволяют контроллеру работать без внешних цепей синхронизации. Работа внутреннего генератора обычно разрешается путем программирования соответствующего бита регистра конфигурации МК.

В большинстве моделей МК частота времязадающего элемента (резонатора или RC-цепи) и частота тактирования f_{BUS} жестко связаны коэффициентом деления встроенного делителя частоты. Поэтому изменение частоты программным путем не представляется возможным. Однако ряд последних семейств МК (например, HC08 фирмы Motorola) имеют в своем составе схему тактирования, основанную на принципе синтезатора частоты с контуром фазовой автоподстройки (PLL — phase loop lock). Такая схема работает как умножитель

частоты и позволяет задавать тактовую частоту с помощью низкочастотного кварцевого резонатора, что снижает уровень электромагнитного излучения МК. Коэффициенты деления контура PLL могут быть изменены программным путем, что позволяет снизить тактовую частоту (и, соответственно, потребляемую мощность) в промежутки времени, когда высокое быстродействие не требуется.

Например, в некоторых МК семейства AVR фирмы Atmel тактовая частота контроллера, задаваемая внутренней RC-цепью, также может изменяться программными средствами.

10.3. Аппаратные средства обеспечения надежной работы МК

Прикладная программа, записанная в память программ МК, должна обеспечивать его надежную работу при любых комбинациях входных сигналов. Однако в результате электромагнитных помех, колебаний напряжения питания и других внешних факторов предусмотренный разработчиком ход выполнения программы может быть нарушен. С целью обеспечения надежного запуска, контроля работы МК и восстановления работоспособности системы в отсутствие оператора все современные МК снабжаются аппаратными средствами обеспечения надежной работы. К ним относятся:

--- схема формирования сигнала сброса МК;

--- модуль мониторинга напряжения питания;

--- сторожевой таймер.

10.3.1. Схема формирования сигнала сброса МК

При включении напряжения питания МК должен начать выполнять записанную в памяти программу работы. На этапе нарастания напряжения питания МК

принудительно переводится в начальное состояние, которое называют состоянием сброса. При этом устанавливаются в исходное состояние внутренние магистрали МК, сигналы управления и регистры специальных функций. Последние определяют начальное состояние периферийных модулей МК, которое чаще всего по умолчанию неактивно.

С целью обеспечения надежного запуска от любых источников питания с различной динамикой нарастания напряжения большинство современных МК содержат встроенный детектор напряжения питания (схема Power - On-Reset — POR), который формирует сигнал сброса при нарастании напряжения питания. В частности, входящий в состав МК семейства PIC16 таймер установления питания (PWRT) начинает отсчет времени после того, как напряжение питания превысило уровень около 1,2…1,8В. По истечении выдержки около 72 мс, считается, что напряжение достигло номинала.

Сразу после выхода из состояния сброса МК выполняет следующие действия:

--- запускает генератор синхронизации МК. Для стабилизации частоты тактирования внутренними средствами формируется задержка времени;

--- считывает энергонезависимые регистры конфигурации в соответствующие регистры ОЗУ (если необходимо);

--- загружает в счетчик команд адрес начала рабочей программы;

--- производит выборку первой программы из памяти программ и приступает к выполнению программы.

Адрес ячейки памяти, в которой хранится код первой исполняемой команды, называют вектором начального запуска или вектором сброса. В некоторых МК этот адрес однозначно определен и приведен в техническом описании. Про такие МК говорят, что они имеют фиксированный вектор сброса. В других МК вектор сброса может быть произвольно определен пользователем. На этапе программирования МК необходимый вектор начального запуска записывается в ячейки с фиксированными адресами, и при выходе МК из сброса

автоматически загружается в счетчик команд. О таких МК говорят, что они имеют загружаемый вектор сброса. Загружаемый вектор сброса имеют все 8-разрядные МК фирмы Motorola, выполненные по структуре с единым адресным пространством команд и данных.

Для перевода МК в состояние сброса при установившемся напряжении питания достаточно подать сигнал высокого или низкого уровня (в соответствии со спецификацией МК) на вход сброса (RESET). Обычно для формирования сигнала сброса при включении напряжения питания и нажатии кнопки сброса используют RC-цепь. Типовые схемы формирования сигнала сброса представлены на рис. 75.

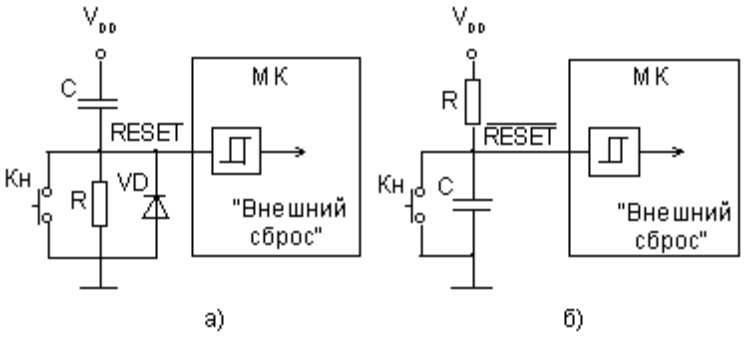


Рис. 75. Типовые схемы формирования сигнала внешнего сброса для МК с высоким активным уровнем сигнала сброса (а) и низким активным уровнем сигнала сброса (б).

Кнопка Кн предназначена для «ручного» сброса МК при отладке. Диод VD препятствует попаданию на вход RESET отрицательного напряжения при выключении питания. Номиналы R и C определяют задержку времени, необходимую для завершения всех переходных процессов при сбросе (указываются в техническом описании на МК). Триггер Шмита на входе допускает подачу сигнала сброса с ненормированной длительностью фронта. При отсутствии триггера Шмита на входе приходится использовать специальную внешнюю схему формирователя.

В современных МК линия RESET обычно является двунаправленной и имеет низкий активный уровень. При нажатии кнопки «сброс» или включении питания буфер линии устанавливается в режим ввода и реализует так называемый внешний сброс. МК может перейти в состояние сброса также по сигналам устройств контроля состояния, которые имеются в составе контроллера. В этом случае говорят, что МК находится в состоянии внутреннего сброса. При этом буфер линии RESET устанавливается в состояние вывода с низким логическим уровнем на выходе. Данный сигнал может быть использован для установки в начальное состояние периферийных ИС.

Порядок выхода МК из состояний внешнего и внутреннего сброса в целом одинаков.

10.3.2. Блок детектирования пониженного напряжения питания

В реальных условиях эксплуатации может сложиться такая ситуация, при которой напряжение питания МК опустится ниже минимально допустимого, но не достигнет порога отпускания схемы POR. В этих условиях МК может «зависнуть». При восстановлении напряжения питания до номинального значения МК останется неработоспособным.

Для восстановления работоспособности системы после «просадки» напряжения питания МК необходимо снова сбросить. Для этой цели в современных МК реализован дополнительный блок детектирования пониженного напряжения питания. Такой модуль используется в МК семейства HC08 фирмы Motorola, аналогичный модуль имеется в составе семейства PIC17 фирмы Microchip. Рассматриваемый модуль генерирует сигнал внутреннего сброса при снижении напряжения питания до уровня чуть ниже минимально допустимого. Уровень, срабатывания блока детектирования пониженного напряжения питания,

значительно превышает напряжение сохранения данных в ОЗУ МК. Событие сброса по сигналу блока пониженного напряжения питания отмечается специальным битом в одном из регистров МК. Следовательно, программно анализируя этот бит после сброса МК, можно установить, что данные целы, и продолжить выполнение программы.

10.3.3. Сторожевой таймер

Если, несмотря на все принятые меры, МК все же «завис», то на случай выхода из этого состояния все современные контроллеры имеют встроенный модуль сторожевого таймера. Принцип действия сторожевого таймера показан на рис. 76.

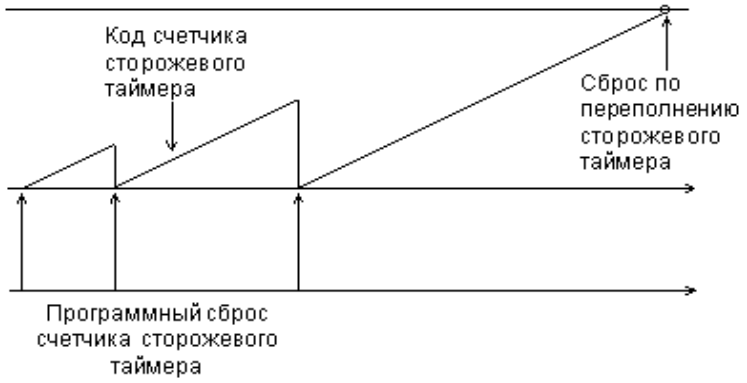


Рис. 76. Принцип действия сторожевого таймера.

Основу сторожевого таймера составляет многоразрядный счетчик. При сбросе МК счетчик обнуляется. После перехода МК в активный режим работы значение счетчика начинает увеличиваться независимо от выполняемой программы. При достижении счетчиком максимального кода генерируется сигнал внутреннего сброса, и МК начинает выполнять рабочую программу сначала.

Для исключения сброса по переполнению сторожевого таймера рабочая программа МК должна периодически сбрасывать счетчик. Сброс счетчика

сторожевого таймера осуществляется путем исполнения специальной команды (например, CLRWDT) или посредством записи некоторого указанного кода в один из регистров специальных функций. Тогда при нормальном, предусмотренном разработчиком, порядке исполнения рабочей программы, переполнения счетчика сторожевого таймера не происходит, и он не оказывает влияния на работу МК. Однако, если исполнение рабочей программы было нарушено, например, вследствие «зависания», то велика вероятность того, что счетчик не будет сброшен вовремя. Тогда произойдет сброс по переполнению сторожевого таймера, и нормальный ход выполнения рабочей программы будет восстановлен.

Модули сторожевых таймеров конкретных МК могут иметь различные особенности:

--- в ряде МК векторы внешнего сброса и сброса по переполнению сторожевого таймера совпадают. Это не позволяет выявить причину сброса программным путем и затрудняет написание рабочей программы. Более высокоуровневые МК имеют либо различные векторы сброса, либо отмечают событие сброса по переполнению сторожевого таймера установкой специального бита в одном из регистров специальных функций;

--- в некоторых МК при переходе в один из режимов пониженного энергопотребления, когда рабочая программа не выполняется, автоматически приостанавливается работа сторожевого таймера. В других МК сторожевой таймер имеет независимый тактовый генератор, который продолжает функционировать и в режиме ожидания. В этом случае необходимо периодически выводить МК из состояния ожидания для сброса сторожевого таймера. В PIC-контроллерах фирмы Microchip выработка таких сбросов может быть запрещена путем записи нуля в специальный бит конфигурации WDTE.

Использование сторожевого таймера существенно повышает способность к самовосстановлению системы на основе МК.

Описанные выше модули составляют так называемый базовый комплект МК и входят в состав любого современного контроллера. Очевидна необходимость включения в состав МК дополнительных модулей, состав и возможности которых определяются конкретной решаемой задачей. Среди таких дополнительных модулей следует, прежде всего, отметить:

--- модули последовательного ввода/вывода данных;

--- модули аналогового ввода/вывода.

10.4. Модули последовательного ввода/вывода

Наличие в составе 8-разрядного МК модуля контроллера последовательного ввода/вывода стало в последнее время обычным явлением. Задачи, которые решаются средствами модуля контроллера последовательного ввода/вывода, можно разделить на три основные группы:

--- связь встроенной микроконтроллерной системы с системой управления верхнего уровня, например, с персональным компьютером. Чаще всего для этой цели используются интерфейсы RS-232C и RS-485;

--- связь с внешними по отношению к МК периферийными ИС, с датчиками физических величин с последовательным выходом. Для этих целей используются интерфейсы I^2C, SPI, а также нестандартные протоколы обмена;

--- интерфейс связи с локальной сетью в мультимикроконтроллерных системах. В системах с числом МК до пяти обычно используются сети на основе интерфейсов I^2C, RS-232C и RS-485 с собственными сетевыми протоколами высокого уровня. В более сложных системах все более популярным становится протокол CAN.

С точки зрения организации обмена информацией упомянутые типы интерфейсов последовательной связи отличаются режимом передачи данных (синхронный или асинхронный), форматом кадра (число бит в посылке при

передаче байта полезной информации) и временными диаграммами сигналов на линиях (уровни сигналов и положение фронтов при переключениях).

Число линий, по которым происходит передача в последовательном коде, обычно равно двум (I^2C, RS-232C, RS-485) или трем (SPI, некоторые нестандартные протоколы). Данное обстоятельство позволяет спроектировать модули контроллеров последовательного обмена таким образом, чтобы с их помощью на аппаратном уровне можно было реализовать несколько типов последовательных интерфейсов. При этом режим передачи (синхронный или асинхронный) и формат кадра поддерживаются на уровне логических сигналов, а реальные физические уровни сигналов для каждого интерфейса получают с помощью специальных ИС, которые называют приемопередатчиками, конверторами, трансиверами.

Среди различных типов встроенных контроллеров последовательного обмена, которые входят в состав тех или иных 8-разрядных МК, является — модуль UART (Universal Asynchronous Receiver and Transmitter). UART — это универсальный асинхронный приемопередатчик. Большинство модулей UART, кроме асинхронного режима обмена, способны также реализовать режим синхронной передачи данных.

Не все производители МК используют термин UART для обозначения типа модуля контроллера последовательного обмена. Так, в МК фирмы Motorola модуль асинхронной приемопередачи, который поддерживает те же режимы асинхронного обмена, что и UART, принято называть SCI (Serial Communication Interface). Следует отметить, что модуль типа SCI обычно реализует только режим асинхронного обмена, то есть его функциональные возможности уже по сравнению с модулями типа UART. Однако бывают и исключения: под тем же именем SCI в МК MC68HC705B16 скрывается модуль синхронно-асинхронной передачи данных.

Модули типа UART в асинхронном режиме работы позволяют реализовать протокол обмена для интерфейсов RS-232C, RS-422A, RS-485, в синхронном режиме — нестандартные синхронные протоколы обмена, и в некоторых моделях — SPI. В МК фирмы Motorola традиционно предусмотрены два модуля последовательного обмена: модуль SCI с возможностью реализации только протоколов асинхронной приемопередачи для интерфейсов RS-232C, RS-422A, RS-485 и модуль контроллера синхронного интерфейса в стандарте SPI.

Протоколы интерфейсов локальных сетей на основе МК (I^2C и CAN) отличает более сложная логика работы. Поэтому контроллеры CAN интерфейса всегда выполняются в виде самостоятельного модуля. Интерфейс I^2C с возможностью работы как в ведущем, так и ведомом режиме, также обычно поддерживается специальным модулем (модуль последовательного порта в МК 89C52 фирмы Philips). Но если реализуется только ведомый режим I^2C, то в МК PIC16 фирмы Microchip он успешно сочетается с SPI: настройка одного и того же модуля на один из протоколов осуществляется путем инициализации.

В последнее время появилось большое количество МК со встроенными модулями контроллеров CAN и модулями универсального последовательного интерфейса периферийных устройств USB (Universal Serial Bus). Каждый из этих интерфейсов имеет достаточно сложные протоколы обмена.

10.5. Модули аналогового ввода/вывода

Необходимость приема и формирования аналоговых сигналов требует наличия в МК модулей аналогового ввода/вывода.

Простейшим устройством аналогового ввода в МК является встроенный компаратор напряжения. Компаратор сравнивает входное аналоговое напряжение с опорным потенциалом V_{REF} и устанавливает на выходе логическую «1», если входное напряжение больше опорного. Компараторы

удобнее всего использовать для контроля определенного значения входного напряжения, например, в термостатах. В комбинации с внешним генератором линейно изменяющегося напряжения встроенный компаратор позволяет реализовать на МК интегрирующий аналого-цифровой преобразователь (АЦП).

Более широкие возможности для работы с аналоговыми сигналами дает АЦП, встроенный в МК. Чаще всего он реализуется в виде модуля многоканального АЦП, предназначенного для ввода в МК аналоговых сигналов с датчиков физических величин и преобразования этих сигналов в двоичный код. Структурная схема типового модуля АЦП представлена на рис. 77.

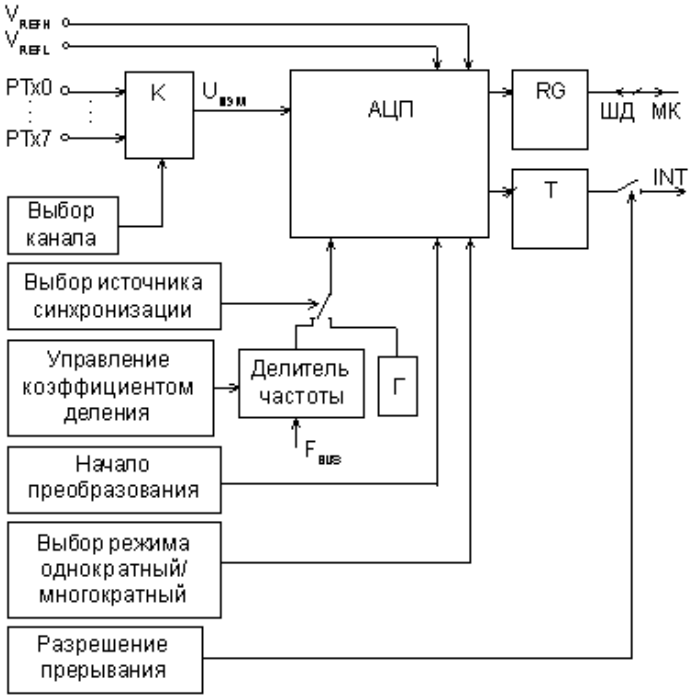


Рис. 77. Структура модуля АЦП.

Многоканальный аналоговый коммутатор К служит для подключения одного из источников аналоговых сигналов (PTx0...PTx7) к входу АЦП. Выбор источника сигнала для преобразования осуществляется посредством записи номера канала коммутатора в соответствующие разряды регистра управления АЦП.

Два вывода модуля АЦП используются для задания опорного напряжения Uоп: VREFH — верхний предел Uоп, VREFL — нижний предел. Разность

потенциалов на входах VREFH и VREFL и составляет Uоп. Разрешающая способность АЦП составляет Uоп/2n, где n — число двоичных разрядов в слове результата. Максимальное значение опорного напряжения, как правило, равно напряжению питания МК. Если измеряемое напряжение Uизм > VREFH, то результат преобразования будет равен FF, код 00 соответствует напряжениям Uизм < VREFL. Для достижения максимальной точности измерения следует выбрать максимально допустимое значение Uоп. В этом случае напряжение смещения нуля входного буфера и нелинейность передаточной характеристики АЦП будут вносить относительно малые погрешности.

Собственно аналого-цифровой преобразователь выполнен по методу последовательного приближения. Практически во всех моделях 8-разрядных МК разрядность АЦП также составляет 8 разрядов. Соответственно, формат представления результатов измерения АЦП — однобайтовый. Исключение составляют лишь модули АЦП микроконтроллеров для управления преобразователями частоты для электроприводов, разрешающая способность которых равна 10 разрядам. Два младших разряда результата получают с помощью дополнительного емкостного делителя, не связанного с регистром последовательного приближения.

Длительность такта преобразования задает генератор синхронизации: один цикл равен двум периодам частоты генератора t_{ADC}. Время преобразования для типовых модулей АЦП микроконтроллеров составляет от единиц до десятков микросекунд.

Источником синхронизации модуля АЦП может служить встроенный RC-генератор (Г) или импульсная последовательность тактирования межмодульных магистралей МК. В первом случае частота синхронизации АЦП обязательно окажется оптимальной, то есть той, которая рекомендуется в техническом описании. Во втором случае выбранная по другим соображениям

f_{BUS} может оказаться неподходящей для модуля АЦП. На этот случай в составе некоторых модулей предусмотрен программируемый делитель частоты f_{BUS}.

Момент завершения каждого цикла преобразования отмечается установкой триггера готовности данных. Если прерывания от модуля АЦП разрешены, то генерируется запрос на прерывания. Как правило, чтение регистра результата сбрасывает триггер готовности.

Большинство модулей АЦП имеют только режим программного запуска: установка одного из битов регистра режима запускает очередное измерение. Наиболее универсальные модули АЦП имеют также режим автоматического запуска, при котором после завершения одного цикла преобразования немедленно начинается следующий. Однако данные измерения каждого цикла должны быть считаны программным способом.

Цифро-аналоговые преобразователи в составе МК являются редкостью. Функция цифро-аналогового преобразователя реализуется средствами модуля программируемого таймера в режиме ШИМ. На одном из выводов МК формируется высокочастотная импульсная последовательность с регулируемой длительностью импульса. Полученный сигнал сглаживается фильтром нижних частот на операционном усилителе. Разрешающая способность такого ЦАП определяется дискретностью регулирования коэффициента заполнения в режиме ШИМ.

11. Разработка микропроцессорной системы на основе микроконтроллера

11.1. Основные этапы разработки

МПС на основе МК используются чаще всего в качестве встроенных систем для решения задач управления некоторым объектом. Важной особенностью данного применения является работа в реальном времени, т.е. обеспечение

реакции на внешние события в течение определенного временного интервала. Такие устройства получили название контроллеров.

Технология проектирования контроллеров на базе МК полностью соответствует принципу неразрывного проектирования и отладки аппаратных и программных средств, принятому в микропроцессорной технике. Это означает, что перед разработчиком такого рода МПС стоит задача реализации полного цикла проектирования, начиная от разработки алгоритма функционирования и заканчивая комплексными испытаниями в составе изделия, а, возможно, и сопровождением при производстве. Сложившаяся к настоящему времени методология проектирования контроллеров может быть представлена так, как показано на рис. 78.

В техническом задании формулируются требования к контроллеру с точки зрения реализации определенной функции управления. Техническое задание включает в себя набор требований, который определяет, что пользователь хочет от контроллера и что разрабатываемый прибор должен делать. Техническое задание может иметь вид текстового описания, не свободного в общем случае от внутренних противоречий.

На основании требований пользователя составляется функциональная спецификация, которая определяет функции, выполняемые контроллером для пользователя после завершения проектирования, уточняя тем самым, насколько устройство соответствует предъявляемым требованиям. Она включает в себя описания форматов данных, как на входе, так и на выходе, а также внешние условия, управляющие действиями контроллера.

Функциональная спецификация и требования пользователя являются критериями оценки функционирования контролера после завершения проектирования. Может потребоваться проведение нескольких итераций, включающих обсуждение требований и функциональной спецификации с потенциальными пользователями контроллера, и соответствующую коррекцию

требований и спецификации. Требования к типу используемого МК формулируются на данном этапе чаще всего в неявном виде.

Этап разработки алгоритма управления является наиболее ответственным, поскольку ошибки данного этапа обычно обнаруживаются только при испытаниях законченного изделия и приводят к необходимости дорогостоящей переработки всего устройства. Разработка алгоритма обычно сводится к выбору одного из нескольких возможных вариантов алгоритмов,

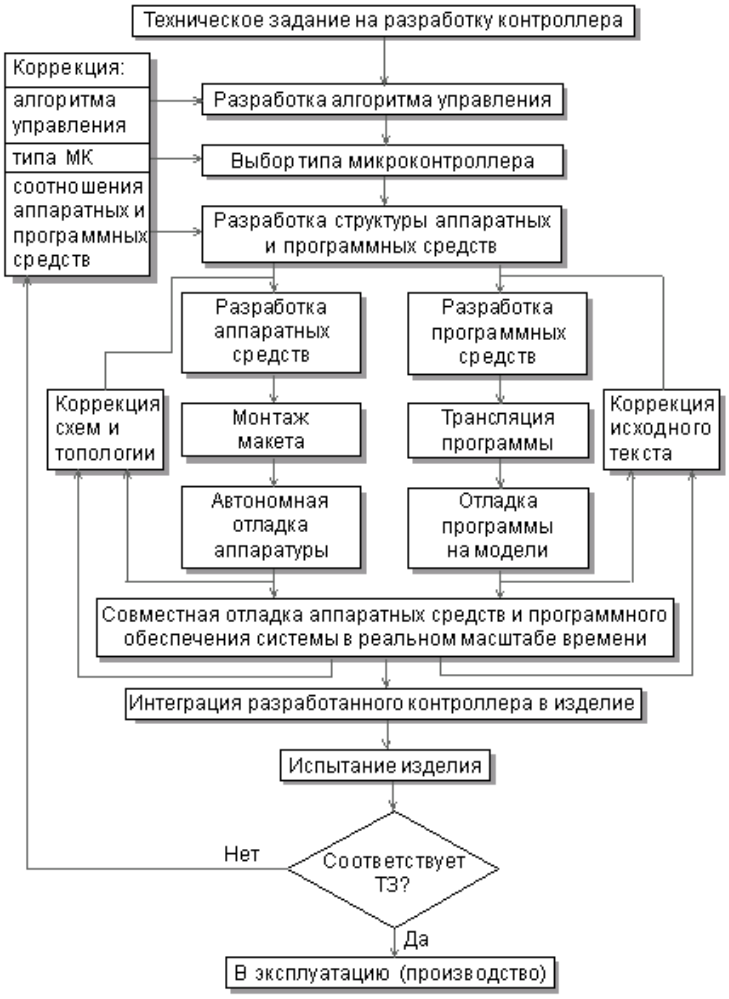


Рис. 78. Основные этапы разработки контроллера.

отличающихся соотношением объема программного обеспечения и аппаратных средств.

При этом необходимо исходить из того, что максимальное использование аппаратных средств упрощает разработку и обеспечивает высокое быстродействие контроллера в целом, но сопровождается, как правило, увеличением стоимости и потребляемой мощности. Связано это с тем, что увеличение доли аппаратных средств достигается либо путем выбора более сложного МК, либо путем использования специализированных интерфейсных схем. И то, и другое приводит к росту стоимости и энергопотребления. Увеличение удельного веса программного обеспечения позволяет сократить число элементов контроллера и стоимость аппаратных средств, но это приводит к снижению быстродействия, увеличению необходимого объема внутренней памяти МК, увеличению сроков разработки и отладки программного обеспечения. Критерием выбора здесь и далее является возможность максимальной реализации заданных функций программными средствами при минимальных аппаратных затратах и при условии обеспечения заданных показателей быстродействия и надежности в полном диапазоне условий эксплуатации. Часто определяющими требованиями являются возможность защиты информации (программного кода) контроллера, необходимость обеспечения максимальной продолжительности работы в автономном режиме и другие. В результате выполнения этого этапа окончательно формулируются требования к параметрам используемого МК.

При выборе типа МК учитываются следующие основные характеристики:

--- разрядность;

--- быстродействие;

--- набор команд и способов адресации;

--- требования к источнику питания и потребляемая мощность в различных режимах;

--- объем ПЗУ программ и ОЗУ данных;

--- возможности расширения памяти программ и данных;

--- наличие и возможности периферийных устройств, включая средства поддержки работы в реальном времени (таймеры, процессоры событий и т.п.);
--- возможность перепрограммирования в составе устройства;
--- наличие и надежность средств защиты внутренней информации;
--- возможность поставки в различных вариантах конструктивного исполнения;
--- стоимость в различных вариантах исполнения;
--- наличие полной документации;
--- наличие и доступность эффективных средств программирования и отладки МК;
--- количество и доступность каналов поставки, возможность замены изделиями других фирм.

Список этот не является исчерпывающим, поскольку специфика проектируемого устройства может перенести акцент требований на другие параметры МК. Определяющими могут оказаться, например, требования к точности внутреннего компаратора напряжений или наличие большого числа выходных каналов ШИМ.

Номенклатура выпускаемых в настоящее время МК исчисляется тысячами типов изделий различных фирм. Современная стратегия модульного проектирования обеспечивает потребителя разнообразием моделей МК с одним и тем же процессорным ядром. Такое структурное разнообразие открывает перед разработчиком возможность выбора оптимального МК, не имеющего функциональной избыточности, что минимизирует стоимость комплектующих элементов.

Однако для реализации на практике возможности выбора оптимального МК необходима достаточно глубокая проработка алгоритма управления, оценка объема исполняемой программы и числа линий сопряжения с объектом на этапе выбора МК. Допущенные на данном этапе просчеты могут впоследствии привести к необходимости смены модели МК и повторной разводки печатной

платы макета контроллера. В таких условиях целесообразно выполнять предварительное моделирование основных элементов прикладной программы с использованием программно-логической модели выбранного МК.

При отсутствии МК, обеспечивающего требуемые по ТЗ характеристики проектируемого контроллера, необходим возврат к этапу разработки алгоритма управления и пересмотр выбранного соотношения между объемом программного обеспечения и аппаратных средств. Отсутствие подходящего МК чаще всего означает, что для реализации необходимого объема вычислений (алгоритмов управления) за отведенное время нужна дополнительная аппаратная поддержка. Отрицательный результат поиска МК с требуемыми характеристиками может быть связан также с необходимостью обслуживания большого числа объектов управления. В этом случае возможно использование внешних схем обрамления МК.

На этапе разработки структуры контроллера окончательно определяется состав имеющихся и подлежащих разработке аппаратных модулей, протоколы обмена между модулями, типы разъемов. Выполняется предварительная проработка конструкции контроллера. В части программного обеспечения определяются состав и связи программных модулей, язык программирования. На этом же этапе осуществляется выбор средств проектирования и отладки.

Возможность перераспределения функций между аппаратными и программными средствами на данном этапе существует, но она ограничена характеристиками уже выбранного МК. При этом необходимо иметь в виду, что современные МК выпускаются, как правило, сериями (семействами) контроллеров, совместимых программно и конструктивно, но различающихся по своим возможностям (объем памяти, набор периферийных устройств и т.д.). Это дает возможность выбора структуры контроллера с целью поиска наиболее оптимального варианта реализации.

Нельзя не упомянуть здесь о новой идеологии разработки устройств на базе МК, предложенной фирмой «Scenix». Она основана на использовании высокоскоростных RISC-микроконтроллеров серии SX с тактовой частотой до 100 МГц. Эти МК имеют минимальный набор встроенной периферии, а все более сложные периферийные модули эмулируются программными средствами. Такие модули программного обеспечения называются «виртуальными периферийными устройствами», они обеспечивают уменьшение числа элементов контроллера, времени разработки, увеличивают гибкость исполнения. К настоящему времени разработаны целые библиотеки виртуальных устройств, содержащие отлаженные программные модули таких устройств как модули ШИМ и ФАПЧ, последовательные интерфейсы, генераторы и измерители частоты, контроллеры прерываний и многие другие.

11.2. Средства проектирования микропроцессорных контроллеров

Комплекс инструментальных средств, необходимых для проектирования и отладки микропроцессорных контроллеров, включает:

средства разработки программного обеспечения;

--- средства ввода принципиальной схемы и проектирования топологии печатной платы;

--- средства автономной отладки аппаратуры;

--- средства совместной отладки аппаратуры и программного обеспечения в реальном масштабе времени;

--- средства программирования БИС памяти программ и программируемой логики.

Общепринятым подходом в настоящее время является использование в качестве ведущего процессора (Host - processor) инструментальных средств компьютера типа IBM PC (рис. 79). Это позволяет разработчику инст-

рументария переложить на нее общесистемные задачи и сосредоточиться в каждом случае на реализации специфических функций проектирования и отладки. Одновременно компьютер служит в качестве объединяющего ядра, в том числе конструктивного, и средством коммуникации в вычислительных сетях. Пользователь при этом получает возможность, помимо средств проектирования, пользоваться широким набором прикладного программного обеспечения для MS DOS и Windows.

Перечисленные выше группы средств проектирования и отладки включают приборы и системы различной категории сложности и стоимости. Далее мы рассмотрим профессиональные средства, обеспечивающие максимальную поддержку разработчику и позволяющие создать изделие с микропроцесс-

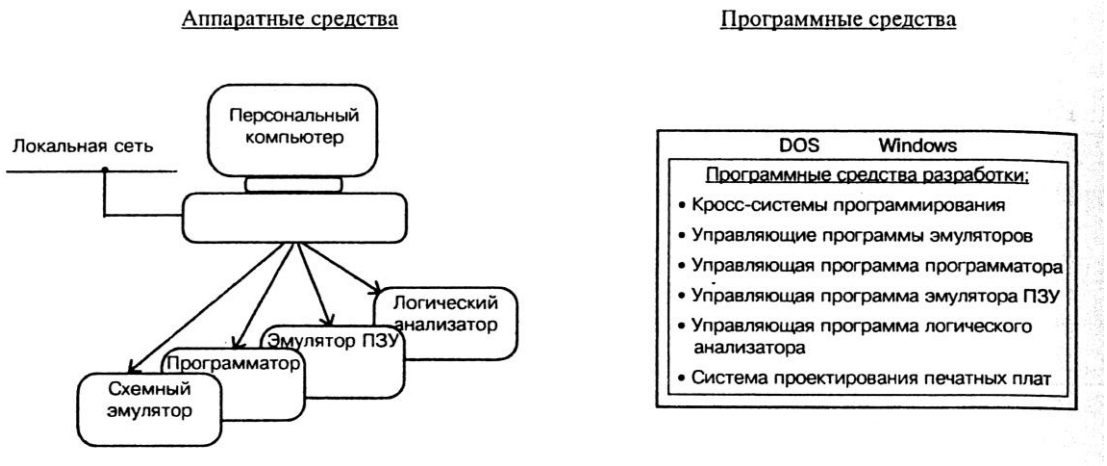


Рис. 79. Структура комплекса средств для разработки и отладки микропроцессорных контроллеров.

сорной системой управления в запланированные сроки и в пределах утвержденной сметы затрат.

11.3. Языки программирования для микроконтроллеров.

Программирование для микроконтроллеров как и программирование для универсальных компьютеров прошло большой путь развития от

программирования в машинных кодах до применения современных интегрированных систем написания программ, отладки и программирования микроконтроллеров. В настоящее время исходный текст программы пишется на одном из языков программирования.

Процесс преобразования операторов исходного языка программирования в машинные коды микропроцессора называется трансляцией исходного текста. В настоящее время ручная трансляция программ практически не используется. Трансляция производится специальными программами- трансляторами.

Существует два больших класса программ-трансляторов: компиляторы и интерпретаторы (рис. 80).

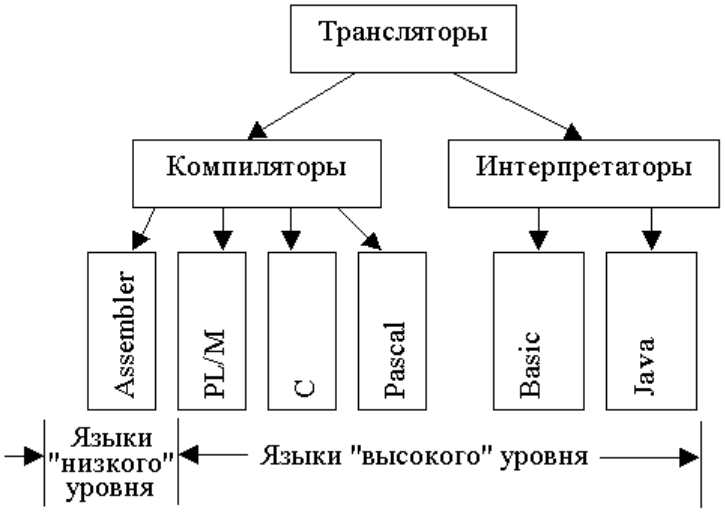


Рис. 80. Классификация программ-трансляторов языков программирования.

При использовании компиляторов весь исходный текст программы преобразуется в машинные коды, и именно эти коды записываются в память микропроцессора. При использовании интерпретатора в память микропроцессора записывается исходный текст программы, а трансляция производится при считывании очередного оператора. Естественно, что быстродействие интерпретаторов намного ниже по сравнению с

компиляторами, т.к. при использовании оператора в цикле он транслируется многократно. Однако при программировании на языке высокого уровня объем кода, который нужно хранить во внутренней памяти может быть значительно меньше по сравнению с исполняемым кодом. Еще одним преимуществом применения интерпретаторов является легкая переносимость программ с одного процессора на другой.

Сами языки программирования в свою очередь делятся на две группы:

--- языки программирования "высокого" уровня

--- языки программирования "низкого" уровня.

К языкам программирования "низкого" уровня относятся языки программирования в которых каждому оператору соответствует не более одной машинной команды. Набор машинных команд каждого конкретного процессора обязательно входит в состав такого языка программирования. Языки программирования низкого уровня в настоящее время называются ассемблерами (старое название автокоды). Для каждого процессора существует своя группа ассемблеров. Ассемблеры для одного и того же процессора различаются между собой дополнительными возможностями, облегчающими программирование.

Языки программирования "высокого" уровня позволяют заменять один оператор несколькими машинными командами. Это позволяет увеличивать производительность труда программистов. Кроме того, языки "высокого" уровня позволяют писать программы, которые могут выполняться на различных микропроцессорах. (Естественно, что при этом необходимо использовать программы - трансляторы для соответствующего процессора.)

О преимуществах и недостатках языков высокого и низкого уровней говорилось достаточно много. Выбор языка программирования зависит от состава аппаратуры, для которой пишется программа, а также от требующегося быстродействия всего программно - аппаратного комплекса в целом.

В тех случаях, когда объем ОЗУ и ПЗУ невелик (в районе нескольких килобайт) альтернативы ассемблеру нет. Именно эти языки программирования позволяют получать самый короткий и самый быстродействующий код программы (при прочих равных условиях, т.к. испортить можно все!).

Языки программирования высокого уровня позволяют значительно сократить время создания программы, но при этом увеличивается размер программы, поэтому для выбора такого языка программирования для микропроцессорных систем необходимо иметь достаточно большой объем памяти программ (несколько десятков килобайт). Увеличение объема программы связано с несколькими факторами:

--- язык программирования рассчитывается на все случаи жизни, поэтому в большинстве случаев человек мог бы написать программу короче (исключив не нужные в данном конкретном случае проверки или защиты).

--- программист не видит, к чему приводит использование тех или других операторов языка программирования, поэтому может выбирать операторы, не оптимальные как с точки зрения длины машинного кода программы, так и с точки зрения быстродействия программы.

--- программист не использует подпрограммы там, где они могли бы сократить объем программы, так как на языке программирования высокого уровня это всего один или несколько операторов.

Первый из этих пунктов постепенно утрачивает свое значение с появлением все более совершенных трансляторов. Третий пункт тоже решается тем же путем при применении различных видов оптимизаторов, входящих в состав компилятора. Однако в большинстве случаев оптимизатор не может определить одинаковые действия, если они отличаются хотя бы одной командой. Кроме того, оптимизатор работает только в пределах одного модуля!

Для программирования микроконтроллеров используются только компиляторы, поэтому рассмотрим подробнее виды этих трансляторов.

Компиляторы бывают оценочные и профессиональные.

Оценочные или учебные компиляторы позволяют написать простейшие программы для конкретного процессора и определить, подходит ли процессор для тех задач, которые предстоит решать в процессе разработки устройства. Конечно, если программа очень проста, то можно весь программный продукт написать на оценочном компиляторе. Оценочные компиляторы позволяют транслировать одиночный файл исходного текста программы. Иногда такие компиляторы позволяют включать в процесс трансляции содержимое отдельных файлов специальной директивой. В результате работы оценочного компилятора сразу получается исполняемый или загрузочный модуль программы, поэтому такие компиляторы называются компиляторы с единой трансляцией.

Профессиональные трансляторы позволяют производить трансляцию исходного текста программы по частям. Это позволяет значительно сократить время трансляции исходного текста программы, так как не нужно транслировать весь текст программы, а можно транслировать только ту часть программы, которая менялась после предыдущей трансляции. Кроме того, каждый программный модуль может писать отдельный программист. Это позволяет сократить время написания программы. Даже в том случае, если программу пишет один человек, время написания программы сокращается за счет использования готовых отлаженных и оттранслированных программных модулей. В таких компиляторах процесс трансляции программы разбивается на два этапа: трансляция программного модуля и связывание программных модулей в единую программу. Поэтому такие компиляторы называются компиляторами с раздельной трансляцией.

Оценочные компиляторы обычно предлагаются бесплатно фирмами - производителями микроконтроллеров. Только фирма Intel предложила в свое время профессиональный пакет разработки программ - язык программирования

PL/M-51 в состав которого входит профессиональный язык программирования ASM-51.

Профессиональные компиляторы разрабатываются и продаются отдельными фирмами. Для микроконтроллеров семейства MCS-51 получили известность продукты таких фирм как FRANCLIN, IAR, KEIL. В состав современных средств написания и отладки программ для микроконтроллеров обычно входят эмуляторы процессоров или отладочные платы, текстовый редактор, компиляторы языка высокого уровня (чаще всего "С") и ассемблера, редактор связей и загрузчик программы в отладочную плату. Все программы обычно объединены интегрированной средой разработки программного проекта, позволяющую поддерживать один или несколько программных проектов.

11.4. Разработка и отладка аппаратных средств

После разработки структуры аппаратных и программных средств дальнейшая работа над контроллером может быть распараллелена. Разработка аппаратных средств включает в себя разработку общей принципиальной схемы, разводку топологии плат, монтаж макета и его автономную отладку. Время выполнения этих этапов зависит от имеющегося набора апробированных функционально-топологических модулей, опыта и квалификации разработчика. На этапе ввода принципиальной схемы и разработки топологии используются, как правило, распространенные системы проектирования типа «ACCEL EDA» или «OrCad».

Автономная отладка аппаратуры на основе МК с открытой архитектурой предполагает контроль состояния многоразрядных магистралей адреса и данных с целью проверки правильности обращения к внешним ресурсам памяти и периферийным устройствам. Закрытая архитектура МК предполагает реализацию большинства функций разрабатываемого устройства внутренними средствами микроконтроллера. Поэтому разрабатываемый контроллер будет

иметь малое число периферийных ИС, а обмен с ними будет идти преимущественно по последовательным интерфейсам. Здесь на первый план выйдут вопросы согласования по нагрузочной способности параллельных портов МК и отладка алгоритмов обмена по последовательным каналам.

Для автономной отладки аппаратуры используются традиционные приборы: осциллограф, логический пробник и т.д.

11.5. Разработка и отладка программного обеспечения

Содержание этапов разработки программного обеспечения, его трансляции и отладки на моделях существенно зависит от используемых системных средств. В настоящее время ресурсы 8-разрядных МК достаточны для поддержки программирования на языках высокого уровня. Это позволяет использовать все преимущества структурного программирования, разрабатывать программное обеспечение с использованием раздельно транслируемых модулей. Одновременно продолжают широко использоваться языки низкого уровня типа ассемблера, особенно при необходимости обеспечения контролируемых интервалов времени. Задачи предварительной обработки данных часто требуют использования вычислений с плавающей точкой, трансцендентных функций.
В настоящее время проектировщику доступны средства разработки программного обеспечения, поддерживающие все этапы процесса.

Средства программирования микропроцессорных систем делятся на резидентные и кросс-средства. Резидентные средства формируют исполняемый код, который может быть выполнен инструментального компьютера. Это возможно в тех случаях, когда контроллер выполнен на микропроцессоре, причем той же серии, что и микропроцессор инструментального компьютера. Кросс-средства могут генерировать и отлаживать код программы любого типа микроконтроллера и микропроцессора. Это подход более общий, но для

эффективной работы кросс - системы необходимо, чтобы производительность инструментального компьютера существенно превосходила производительность целевого процессора (на плате контроллера).

Пакеты инструментальных программ для микропроцессоров и микроконтроллеров представляют собой наборы кросс - программ, обеспечивающих трансляцию исходных текстов рабочих программ с языков Паскаль, Си, PL/M, Ассемблер в перемещаемый объектный код, последующую компоновку отдельных модулей в единую программу и размещение ее по абсолютным адресам памяти программ.

В настоящее время самым мощным средством разработки программного обеспечения для МК являются интегрированные среды разработки, имеющие в своем составе менеджер проектов, текстовый редактор и симулятор, а также допускающие подключение компиляторов языков высокого уровня типа Паскаль или Си. При этом необходимо иметь в виду, что архитектура многих 8-разрядных МК вследствие малого количества ресурсов, страничного распределения памяти, неудобной индексной адресации и некоторых других архитектурных ограничений не обеспечивает компилятору возможности генерировать эффективный код. Для обхода этих ограничений разработчики ряда компиляторов вынуждены были перекладывать на пользователя заботу об оптимизации кода программы.

Для проверки и отладки программного обеспечения используются так называемые программные симуляторы, предоставляющие пользователю возможность выполнять разработанную программу на программно-логической модели МК. Программные симуляторы распространяются, как правило, бесплатно и сконфигурированы сразу на несколько МК одного семейства. Выбор конкретного типа МК среди моделей семейства обеспечивает соответствующая опция меню конфигурации симулятора. При этом моделируется работа ЦП, всех портов ввода/вывода, прерываний и другой

периферии. Карта памяти моделируемого МК загружается в симулятор автоматически, отладка ведется в символьных обозначениях регистров.
Загрузив программу в симулятор, пользователь имеет возможность запускать ее в пошаговом или непрерывном режимах, задавать условные или безусловные точки останова, контролировать и свободно модифицировать содержимое ячеек памяти и регистров симулируемого МК.

11.6. Методы и средства совместной отладки аппаратных и программных средств

Этап совместной отладки аппаратных и программных средств в реальном масштабе времени является самым трудоемким и требует использования инструментальных средств отладки. К числу основных инструментальных средств отладки относятся:

--- внутрисхемные эмуляторы;

--- платы развития (оценочные платы);

--- мониторы отладки;

--- эмуляторы ПЗУ.

Эмуляция как метод подразумевает замещение некоторого модуля системы (процессора или памяти программ) функциональным аналогом (эмулятором), который позволяет сделать процесс управления контролируемым и наблюдаемым. Особенностью эмуляции микропроцессорных систем является то, что функции управления и наблюдения также выполняются эмулятором, а не отдельными блоками отладочного стенда. Это связано с полной интеграцией аппаратного и программного обеспечения и сложностью логико-временных диаграмм функционирования этих систем. Отдельные приборы наблюдения могут подключаться как дополнительные. Таким образом, можно заключить, что основными функциями эмуляции как метода отладки микропроцессорных

систем являются: замещение микропроцессорной БИС или модуля памяти программ, управление формированием последовательности функциональных состояний на магистрали и наблюдение за откликами модулей системы, а также состоянием центрального процессора. Требования режима реального времени приводят к необходимости производить замещение не только функциональное, но и с точки зрения важнейших электрофизических параметров.

Внутрисхемный эмулятор – программно-аппаратное средство, способное заменить эмулируемый МК в реальной схеме. Стыковка внутрисхемного эмулятора с отлаживаемой системой производится при помощи кабеля со специальной эмуляционной головкой, которая вставляется вместо МК в отлаживаемую систему. Если МК нельзя удалить из отлаживаемой системы, то использование эмулятора возможно, только если этот микроконтроллер имеет отладочный режим, при котором все его выводы находятся в третьем состоянии. В этом случае для подключения эмулятора используют специальный адаптер - клипсу, который подключается непосредственно к выводам эмулируемого МК.

Внутрисхемный эмулятор – это наиболее мощное и универсальное отладочное средство, которое делает процесс функционирования отлаживаемого контроллера прозрачным, т.е. легко контролируемым, произвольно управляемым и модифицируемым.

Схемный эмулятор микропроцессора или микроконтроллера строится на основе имитирующего процессора, замещающего целевую микропроцессорную БИС. Кроме ресурсов ЦПУ этой БИС традиционно эмулируется память программ и данных. Эмулятор подключается к магистрали отлаживаемой системы эмуляционной вилкой через розетку вместо целевой микропроцессорной БИС (сечение 1 на рис. 81).

Непосредственное управление магистралью осуществляет имитирующий процессор, рабочая программа и функциональные тесты вначале размещаются в памяти программ эмулятора. Поскольку изменять набор операций и диаграммы обмена эмулятора трудно, этот метод применяется только для отладки систем на основе однокристальных микропроцессоров и микроконтроллеров с фиксированной системой команд. Каждому микропроцессорному семейству соответствует свой схемный эмулятор.

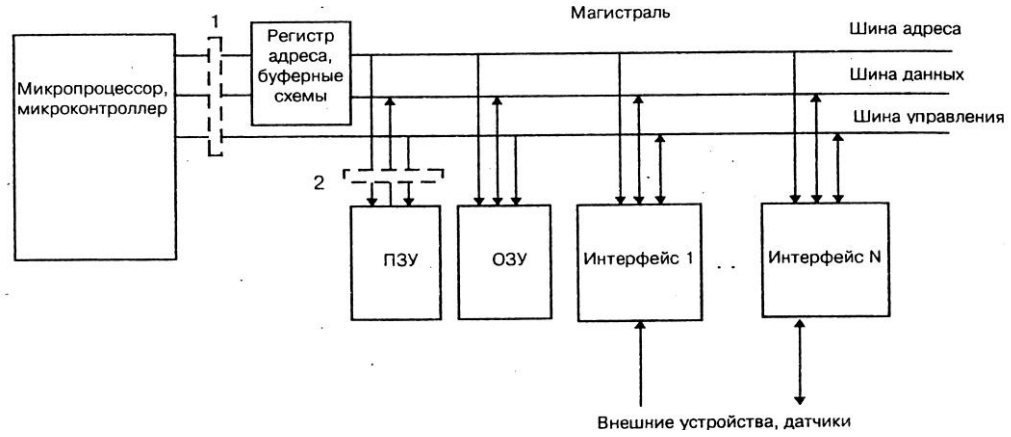


Рис. 81. Места сечений на микропроцессорной системе.

Платы развития, или, как принято их называть в зарубежной литературе, оценочные платы (Evaluation Boards), являются своего рода конструкторами для макетирования электронных устройств. Обычно это печатная плата с установленным на ней МК и всей необходимой ему стандартной периферией. На этой плате также устанавливают схемы связи с внешним компьютером. Как правило, там же имеется свободное поле для монтажа прикладных схем пользователя. Иногда предусмотрена уже готовая разводка для установки дополнительных устройств, рекомендуемых фирмой. Например, ПЗУ, ОЗУ, ЖКИ - дисплей, клавиатура, АЦП и др. Кроме учебных или макетных целей, такие доработанные пользователем платы можно использовать в качестве одноплатных контроллеров, встраиваемых в малосерийную продукцию.

Для большего удобства платы развития комплектуются еще и простейшим средством отладки на базе монитора отладки. Используются два типа мониторов отладки*:* один для МК, имеющих внешнюю шину, а второй – для МК, не имеющих внешней шины.

В первом случае отладочный монитор поставляется в виде микросхемы ПЗУ, которая вставляется в специальную розетку на плате развития. Плата также имеет ОЗУ для программ пользователя и канал связи с внешним компьютером или терминалом. Во втором случае плата развития имеет встроенные схемы программирования внутреннего ПЗУ МК, которые управляются от внешнего компьютера. При этом программа монитора просто заносится в ПЗУ МК совместно с прикладными кодами пользователя. Прикладная программа должна быть специально подготовлена: в нужные места необходимо вставить вызовы отладочных подпрограмм монитора. Затем осуществляется пробный прогон. Чтобы внести в программу исправления, пользователю надо стереть ПЗУ и произвести повторную запись. Готовую прикладную программу получают из отлаженной путем удаления всех вызовов мониторных функций и самого монитора отладки. Возможности отладки, предоставляемые комплектом «плата развития плюс монитор», не столь универсальны, как возможности внутрисхемного эмулятора, да и некоторая часть ресурсов МК в процессе отладки отбирается для работы монитора. Тем не менее, наличие набора готовых программно-аппаратных средств, позволяющих без потери времени приступить к монтажу и отладке проектируемой системы, во многих случаях является решающим фактором. Особенно если учесть, что стоимость такого комплекта несколько меньше, чем стоимость более универсального эмулятора.

Эмулятор ПЗУ – программно-аппаратное средство, позволяющее замещать ПЗУ на отлаживаемой плате, и подставляющее вместо него ОЗУ, в которое может быть загружена программа с компьютера через один из стандартных каналов связи. Это устройство позволяет пользователю избежать многократных

циклов перепрограммирования ПЗУ. Эмулятор ПЗУ нужен только для МК, которые могут обращаться к внешней памяти программ. Это устройство сравнимо по сложности и по стоимости с платами развития и имеет одно большое достоинство: универсальность. Эмулятор ПЗУ может работать с любыми типами МК.

Эмулятор ПЗУ строится с использованием матрицы памяти, замещающей память программ. Он подключается к магистрали отлаживаемой системы эмуляционной вилкой через розетку вместо микросхемы памяти (сечение 2 на рис. 81).

Эмулируемая память доступна для просмотра и модификации, но контроль над внутренними управляющими регистрами МК был до недавнего времени невозможен.

В последнее время появились модели интеллектуальных эмуляторов ПЗУ, которые позволяют «заглядывать» внутрь МК на плате пользователя. Интеллектуальные эмуляторы представляют собой гибрид из обычного эмулятора ПЗУ, монитора отладки и схем быстрого переключения шины с одного на другой. Это создает эффект, как если бы монитор отладки был установлен на плате пользователя и при этом он не занимает у МК никаких аппаратных ресурсов, кроме небольшой зоны программных шагов, примерно 4К.

Непосредственное управление магистралью осуществляет микропроцессорная БИС, рабочая программа и функциональные тесты размещаются в памяти программ эмулятора ПЗУ.

Этап совместной отладки аппаратных и программных средств в реальном масштабе времени завершается, когда аппаратура и программное обеспечение совместно обеспечивают выполнение всех шагов алгоритма работы системы. В конце этапа отлаженная программа заносится с помощью программатора в энергонезависимую память МК, и проверяется работа контроллера без

эмулятора. При этом используются лабораторные источники питания. Часть внешних источников сигналов может моделироваться.

Этап интеграции разработанного контроллера в изделие заключается в повторении работ по совместной отладке аппаратуры и управляющей программы, но при работе в составе изделия, питании от штатного источника и с информацией от штатных источников сигналов и датчиков.

Состав и объем испытаний разработанного и изготовленного контроллера зависит от условий его эксплуатации и определяется соответствующими нормативными документами. Проведение испытаний таких функционально сложных изделий, как современные контроллеры, может потребовать разработки специализированных средств контроля состояния изделия во время испытаний.

Испытания изделия с микропроцессорным контроллером можно разделить на комплексные и специальные. Особенностью комплексных испытаний является то, что для наблюдения за микропроцессорным контроллером в реальных условиях не всегда применимы лабораторные средства отладки. Автономные отладочные средства менее развиты и при этом существенно дороже. Специальные испытания (на электромагнитную совместимость, климатические и т.п.) проводятся по обычным методикам. После успешного проведения испытаний появляется файл с окончательной версией кода управляющей программы для программатора или для завода-изготовителя микроконтроллеров, который осуществляет масочное программирование внутренней памяти программ.

Библиографический список

1. Схемотехника электронных систем. Микропроцессоры и микроконтроллеры / Бойко В.И. [и др.]. - СПб.: БХВ - Петербург, 2004 - 464с.

2. *Бродин В. Б.* Микроконтроллеры. Архитектура, программирование, интерфейс / Бродин В.Б., Шагурин И. И. - М.: ЭКОМ, 1999. - 400с.

3. *Бродин В. Б.* Системы на микроконтроллерах и БИС программируемой логики / Бродин В.Б., Калинин А. В. - М. Эконом, 2002. - 398с.

4. *Бунтов В. Д.* Цифровые и микропроцессорные радиотехнические устройства: учеб. пособие / Бунтов В.Д., Макаров С. Б. СПб.: Изд-во Политехн, ун-та, 2005 - 399с.

5. Микропроцессорные системы: учеб. пособие / Александров Е. К. [и др.]. Под общ. ред. Пузанкова Д.В. - СПб.: Политехника, 2002. - 935с.

6. *Пухальский Г.И.* Проектирование микропроцессорных систем: учеб. пособие / Пухальский Г.И. - СПб.: Политехника, 2001. - 544с.

7. *Корнеев В.В.* Современные микропроцессоры.-3-изд., перераб. и доп. / Корнеев В. В., Киселев А. В. - СПб.: БХВ - Петербург, 2003.- 448 с.

8. *Нарышкин А.К.* Цифровые устройства и микропроцессоры: учеб. пособие/ Нарышкин А. К. - М.: Издательский центр «Академия», 2006.- 320 с.

9. *Ремизевич Т.В.* Микроконтроллеры для встраиваемых приложений / Ремизевич Т. В. - М., Додека, 2000. -

10. *Сташин В.В.* Проектирование цифровых устройств на однокристальных микроконтроллерах / Сташин В. В., Урусов Ф. В. Мологонцева О. Ф. – М.: Энергоатомиздат, 1990. – 224с.

11. *Щелкунов Н.Н.* Микропроцессорные средства и системы / Щелкунов Н.Н., Дианов А.П. М.: Радио и связь, 1989. - 288с.

12. *Угрюмов Е.П.* Цифровая схемотехника: учеб. пособие / Угрюмов Е. П. 2-е изд. - СПб.: БХВ-Петербург, 2005 - 800с.

13. *Микушин А. В.* Занимательно о микроконтроллерах / Микушин А. В. – СПб. : БХВ-Петербург, 2006. - 432с.

14. *Новиков Ю. В.* Основы микропроцессорной техники: учеб. пособие / Новиков Ю. В., Скоробогатов П.К. – М.: ИНТУИТ.РУ, 2004. – 440с.

15. Однокристальные микро ЭВМ: учеб. пособие / Боборыкин А.В. [и др.]. М.: МИКАП, 1994. – 400с.