ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Государственное образовательное учреждение высшего профессионального образования
«ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Ю.В. Алхимов

МИКРОПРОЦЕССОРЫ И ЦИФРОВЫЕ СИСТЕМЫ В НЕРАЗРУШАЮЩЕМ КОНТРОЛЕ

Учебное пособие

Издательство
Томского политехнического университета
2008

УДК 620.179.1:681.325.5-181.48(075.8)
ББК 30.607:32.973.26-042я73
А54

Алхимов Ю.В.

А54 Микропроцессоры и цифровые системы в неразрушающем контроле: учебное пособие / Ю.В. Алхимов. – Томск: Изд-во Томского политехнического университета, 2008. – 245 с.

ISBN 5-98298-201-6

В учебном пособии представлены основные вопросы, касающиеся архитектуры и программирования микропроцессоров и микроконтроллеров. Рассмотрены вопросы построения вычислительных систем на основе микропроцессоров, систем памяти и ввода/вывода, методика применения этих средств при проектировании аппаратуры неразрушающего контроля.

Подготовлено на кафедре физических методов и приборов контроля качества в рамках реализации Инновационной образовательной программы ТПУ по направлению «Неразрушающий контроль» и будет полезно для студентов, аспирантов и специалистов в области НК.

УДК 620.179.1:681.325.5-181.48(075.8)
ББК 30.607:32.973.26-042я73

Рекомендовано к печати Редакционно-издательским советом
Томского политехнического университета

Рецензенты

Доктор технических наук, профессор ТУСУРа
Б.А. Люкшин

Доктор технических наук, профессор ТГАСУ
О.И. Недавний

ISBN 5-98298-201-6

ВВЕДЕНИЕ

Есть вещи, без которых трудно представить современную жизнь. Всего 60 лет назад искусственные спутники Земли и космические лаборатории казались фантастикой. В наши дни без них немыслимы точные прогнозы погоды, связь между континентами, осуществление ряда тонких технологических процессов, получение материалов, обладающих особыми свойствами.

Нелегко поверить, что до 1971 г. человечество не знало микропроцессора – самого выдающегося достижения электроники после изобретения транзистора. А сегодня микропроцессоры прочно вошли в нашу жизнь, и буквально с каждым днем открываются все новые и новые возможности их применения. Сейчас уже трудно назвать такие области техники или научных исследований, в которых бы не использовались микропроцессоры. Все нарастающее распространение микропроцессоры получают в системах управления, технике связи, радиотехнике, электронике, медицинской диагностической и лечебной аппаратуре, сфере обслуживания, и даже в детских игрушках. Микропроцессоры служат основой создания новых поколений электронных вычислительных машин (микроЭВМ). По широте и эффективности применения микропроцессоров одно из первых мест занимает контрольно-измерительная техника, и в том числе приборы неразрушающего контроля и диагностики.

Широкое внедрение микропроцессорной техники в науку и производство – объективный и необходимый процесс. Для современной цивилизации характерна тенденция роста количества информации. Человек не способен уже воспринимать и обрабатывать эту информацию в реальном масштабе времени. В измерительной аппаратуре прием и обработку информации берут на себя электронные схемы и микропроцессоры.

В научно-технической литературе, в проспектах известных фирм, на страницах газет приводятся утверждения авторитетных ученых и инженеров, что микропроцессоры определяют и будут определять на ближайшие десятилетия передовые рубежи техники, ускорение научно-технического прогресса. Поэтому для современного инженера в области

приборостроения, неразрушающего контроля и диагностики знание микропроцессорной техники является совершенно необходимым. Микропроцессорные системы стали неотъемлемой частью электронных измерительных и контрольных приборов, применяемых для измерения параметров электрических сигналов, а также характеристик неэлектрических физических величин.

Проникновение микропроцессоров в измерительную технику во много раз повысило точность приборов, значительно расширило их функциональные возможности, упростило управление работой, повысило надежность, быстродействие, открыло пути решения задач, которые ранее вообще не решались. Трудно переоценить значение микропроцессоров для создания измерительно-вычислительных комплексов – автоматизированных средств измерений, предназначенных для исследования, контроля и испытаний сложных объектов. Малые габариты, масса, невысокая стоимость позволили встраивать микропроцессоры непосредственно в приборы и использовать для автоматизации процесса измерений.

В настоящее время можно выделить два **направления применения вычислительных средств в приборостроении:**

1) использование микропроцессоров и микроЭВМ для создания автоматизированных переносных приборов;
2) использование микропроцессорных контроллеров для создания автоматизированных и полностью автоматических измерительных систем и установок.

Применение в измерительных средствах микропроцессоров и микроЭВМ позволяет успешно решать следующие **задачи**:

- Расширение функциональных возможностей измерительных приборов и систем. Это достигается простым переключением режимов и алгоритмов измерений с помощью микропроцессорной системы по программам, записанным в памяти системы.
- Сокращение времени на настройку и калибровку приборов. Настройка и калибровка выполняется автоматически перед началом работы. Возможна автоматическая реализация режима «самообучения» по серии образцов и выбор оптимального алгоритма измерения и режима работы прибора.
- Повышение достоверности измерений благодаря самодиагностике и исключению ошибок оператора при настройке и калибровке прибора.

- Повышение точности измерений благодаря использованию сложных алгоритмов обработки информации, трудно реализуемых в обычных приборах.
- Повышение производительности измерений. Отдельные отсчеты выполняются прибором автоматически, запоминаются и обрабатываются. Оператор получает результаты измерений в обобщенном виде (например, в виде среднего значения за некоторый промежуток времени) или индицируются параметры, выходящие за допустимые пределы. Это позволяет освободить оператора от рутинных операций по регистрации и анализу результатов.
- Возможность работы с прибором персонала с невысокой квалификацией, поскольку управление работой прибора и анализ результатов автоматизированы.
- Сокращение затрат на разработку новых приборов. Электрические схемы приборов разного назначения унифицируются, а различие в алгоритмах функционирования обеспечивается программными средствами.

Функции, выполняемые вычислительными средствами в измерительных приборах и системах, весьма разнообразны. Их можно условно разделить на две группы:

1) функции обработки информации от первичных преобразователей;
2) функции управления.

К первой группе относятся такие функции, как реализация основных алгоритмов (решение систем уравнений, дифференцирование и интегрирование, фильтрация, свертка, обработка изображений и т. д.), статистическая обработка данных (вычисление средних значений, построение гистограмм).

Во вторую группу входят функции:

- управление режимом работы прибора (установка чувствительности, перестройка полосы пропускания фильтров и т. д.);
- управление калибровкой (установка нуля, проверка градуировки по электрическим сигналам или образцам);
- самодиагностика (проверка работоспособности прибора, определение неисправности);
- управление внешними устройствами (индикаторами, системами сканирования).

Не все эти функции могут быть реализованы на микропроцессорах и микроЭВМ. Часть из них иногда более целесообразно реализовать аппаратно, то есть используя электронные схемы на дискретных компонентах и микросхемах.

Данное учебное пособие разработано для студентов, обучающихся по направлению «Приборостроение», и имеет целью дать основные понятия по архитектуре современных микропроцессоров и однокристальных микроЭВМ (микроконтроллеров), структуре микропроцессорных вычислительных систем, по схемотехническим решениям ввода/вывода информации в микропроцессорных вычислительных системах. Дополнением к данному пособию является учебное пособие, которое содержит вопросы, примеры и упражнения, а также лабораторный практикум по курсу.

При изложении материала в пособии реализованы принципы «от общего к частному» и «от простого к сложному». В первой главе рассмотрены общие принципы архитектуры микропроцессоров и микроконтроллеров, а также вычислительных систем на их основе, даны основные понятия и определения. Во второй главе более подробно рассмотрена архитектура конкретного 16-разрядного микропроцессора. Вопросы построения систем на его основе рассмотрены в третьей главе. Четвертая глава посвящена вопросам построения запоминающих устройств для микропроцессорных систем. Ввод/вывод информации в микропроцессорных системах – тема пятой главы. В шестой главе изложен материал об однокристальных микроЭВМ (микроконтроллерах), нашедших широкое распространение в системах, встраиваемых в приборы, машины. В седьмой главе рассмотрены общие методологические вопросы применения микропроцессоров и микроЭВМ в информационно-измерительных системах. В приложении рассмотрены вопросы представления информации в цифровых системах.

Изложение материала в каждой главе начинается с небольшого введения, в котором дается связь между рассматриваемыми понятиями и подчеркиваются вопросы, на которые нужно обратить внимание при изучении материала. В конце каждой главы приведены вопросы, которые могут помочь при работе с материалом главы.

Чтобы облегчить изучение и понимание материала, мы будем использовать в тексте следующие обозначения:

полужирным шрифтом обозначаются термины и ключевые слова, на которые следует обратить внимание;

 – *так обозначены наиболее важные определения и правила;*

 – этим знаком сопровождаются примеры.

Глава 1
ИНФОРМАЦИЯ И ЕЕ ПРЕДСТАВЛЕНИЕ В КОМПЬЮТЕРНЫХ СИСТЕМАХ

Микропроцессоры и другие цифровые приборы используется в технике для построения систем, предназначенных для обработки и преобразования **информации**. Информация в цифровых системах представлена в различных видах: **данные, адреса, команды**. Понятие «информация», а также способы ее количественного измерения рассмотрены в данной главе. Информация кодируется в **цифровых** системах в виде **чисел** в **двоичной системе счисления**. Для более удобного написания чисел используется **шестнадцатеричная система счисления**. Здесь мы также рассмотрим эти вопросы.

1.1. Что такое информация

Любая наука начинается со строгих определений используемых ею понятий и терминов. Определить какое-либо понятие – значит выразить его через другие понятия, уже определенные ранее. Сложность ситуации, однако, в том, что информация является одной из исходных категорий мироздания, и следовательно, определение «информации вообще» невозможно свести к каким-то более простым, более «исходным» терминам. Что касается частных трактовок понятия «информация», то следует отметить значительное их расхождение в различных научных дисциплинах, в технике и на бытовом уровне. На бытовом уровне и во многих научных дисциплинах оно ассоциируется с понятиями «сведения», «знания», «данные», «известие», «сообщение». Общим является то, что существенным и значимым для использования является содержательная сторона информации – с позиций «здравого смысла» это представляется вполне естественным. Однако оценка смысла и ценности одной и той же информации различными людьми будет различной. Объективная количественная мера смысловой стороны информации отсутствует.

Отделив информацию от ее семантическо-содержательной основы, мы получаем возможность построить определение информации и па-

раллельно ввести ее объективную количественную меру. Открытие такого способа определения информации является одной из главных заслуг теории информации.

Отличительная особенность, которой обладает любая информация, – это то, что информация – категория нематериальная. Следовательно, для существования и распространения в нашем материальном мире она должна быть обязательно связана с какой-либо материальной основой – без нее информация не может проявиться, передаваться и сохраняться.

Материальный объект или среда, которые служат для представления или передачи информации, называются ее материальным носителем.

Материальным носителем информации может быть бумага, воздух, лазерный диск, электромагнитное поле. При этом хранение информации связано с некоторой характеристикой носителя, которая не меняется с течением времени, например намагниченные области поверхности диска или буква на бумаге, а передача информации – наоборот, с характеристикой, которая изменяется с течением времени, например амплитуда колебаний звуковой волны или напряжение в проводах. Другими словами, хранение информации связано с фиксацией состояния носителя, а распространение – с процессом, который протекает в носителе. Состояния и процессы могут иметь физическую, химическую, биологическую или иную основу – главное, что они материальны.

Однако не с любым процессом можно связать информацию. В частности, стационарный процесс, т. е. процесс с неизменными в течение времени характеристиками, информацию не переносит. Примером может служить постоянный электрический ток, ровное горение лампы или равномерный гул – они содержат лишь ту информацию, что процесс идет, то есть что-то функционирует. Если же мы будем лампу включать и выключать, то есть изменять ее яркость, то чередованием вспышек и пауз можно представить и передать информацию (например, посредством азбуки Морзе). Таким образом, для передачи необходим нестационарный процесс, т. е. процесс, характеристики которого могут изменяться; при этом информация связывается не с существованием процесса, а именно с изменением какой-либо его характеристики.

Изменение характеристики носителя, которое используется для представления информации, называется сигналом, а значение этой характеристики, отнесенное к некоторой шкале измерений, называется параметром сигнала. Последовательность сигналов называется сообщением.

Таким образом, от источника к приемнику информация передается в виде сообщений. Можно сказать, что сообщение выступает в качестве материальной оболочки для представления информации при передаче.

Следовательно, сообщение служит переносчиком информации, а информация является содержанием сообщения.

Информационный процесс – это изменение с течением времени содержания информации или представляющего его сообщения.

Существуют следующие виды информационных процессов:

- порождение (создание) новой информации;
- преобразование информации (т. е. порождение новой информации в результате обработки имеющейся);
- уничтожение информации;
- передача информации (распространение в пространстве).

С передачей информации связана еще одна пара исходных сопряженных понятий – источник и приемник информации.

Источник информации – это субъект или объект, порождающий информацию и представляющий ее в виде сообщения.

Приемник информации – это субъект или объект, принимающий сообщение и способный правильно его интерпретировать.

В определении приемника информации важным представляется то, что факт приема сообщения еще не означает получение информации; информация может считаться полученной только в том случае, если приемнику известно правило интерпретации сообщения. Другими словами, понятия «приемник сообщения» и «приемник информации» не тождественны. Например, слыша речь на незнакомом языке, человек оказывается приемником сообщения, но не приемником информации.

Таким образом, понятие «информация» является одним из наиболее общих в науке и технике и обозначает совокупность некоторых сведений, данных, знаний и т. п.

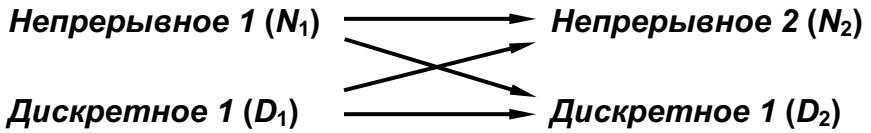
В технике информация – это свойство объектов и явлений материального мира порождать многообразие состояний, которые посредством отражения передаются от одного объекта к другому и запечатлеваются в его структуре (возможно, в измененном виде). При этом под информацией понимают не сами объекты и явления, а их отражения в виде чисел, формул, описаний, чертежей, символов и других абстрактных характеристик. Сама по себе информация может быть отнесена к области абстрактных категорий, подобных, например, математическим формулам.

Источники информации разделяют на аналоговые и дискретные. Аналоговые источники создают сообщения, которые отображаются сигналами, представляющими собой какие-либо физические величины, изменяющиеся непрерывно и принимающие бесконечное число значений в некотором диапазоне. Такое представление информации используется в аналоговых электронных схемах. Дискретные источники информации создают сообщения, состоящие из конечного множества элементов, на-

зываемых символами. Символы отображаются сигналами, принимающими конечное число значений. Если элементам дискретного сообщения поставлены в соответствие цифры или некоторые их совокупности, то такое представление информации называют цифровым. В цифровом виде передается и обрабатывается информация в цифровых электронных схемах, к числу которых относятся и вычислительные средства.

Рассмотрим информационные процессы, связанные с преобразованием одних сигналов в другие.

Поскольку имеются два типа сообщений, между ними, очевидно, возможны четыре варианта преобразований:



Осуществимы и применяются на практике все четыре вида преобразований. Примерами устройств, в которых осуществляется преобразование типа $N_1 \rightarrow N_2$, являются микрофон (звук преобразуется в электрические сигналы); магнитофон и видеомагнитофон (чередование областей намагничивания ленты превращается в электрические сигналы, которые затем преобразуются в звук и изображение); телекамера (изображение и звук превращаются в электрические сигналы); радио- и телевизионный приемник (радиоволны преобразуются в электрические сигналы, а затем в звук и изображение); аналоговая вычислительная машина (одни электрические сигналы преобразуются в другие). Особенностью данного варианта преобразования является то, что оно всегда сопровождается частичной потерей информации. Потери связаны с помехами (шумами), которые порождает само информационное техническое устройство и которые воздействуют извне. Эти помехи примешиваются к основному сигналу и искажают его. Поскольку параметр сигнала может иметь любые значения (из некоторого интервала), то невозможно разделить ситуации: был ли сигнал искажен или он изначально имел такую величину. В ряде устройств искажение происходит в силу особенностей преобразования в них сообщения, например в черно-белом телевидении теряется цвет изображения; телефон пропускает звук в более узком частотном интервале, чем интервал человеческого голоса; кино- и видеоизображение оказываются плоскими, они утрачивают объемность.

Теперь обсудим общий подход к преобразованию типа N→D. С математической точки зрения, перевод сигнала из аналоговой формы в дискретную означает замену описывающей его непрерывной функции времени Z(t) на некотором отрезке $[t_1, t_2]$ конечным множеством (массивом) $\{Z_i, t_i\}$ (i = 0…n, где n – количество точек разбиения временного

интервала). Подобное преобразование называется дискретизацией непрерывного сигнала и осуществляется посредством двух операций: развертки по времени и квантования по величине сигнала.

Развертка по времени состоит в том, что наблюдение за значением величины Z производится не непрерывно, а лишь в определенные моменты времени с интервалом Δt:

$$\Delta t = \frac{t_n - t_0}{n}.$$

Квантование по величине – это отображение вещественных значений параметра сигнала в конечное множество чисел, кратных некоторой постоянной величине – шагу квантования(ΔZ).

Совместное выполнение обоих операций эквивалентно нанесению масштабной сетки на график Z(t), как показано на рис. 1.1.

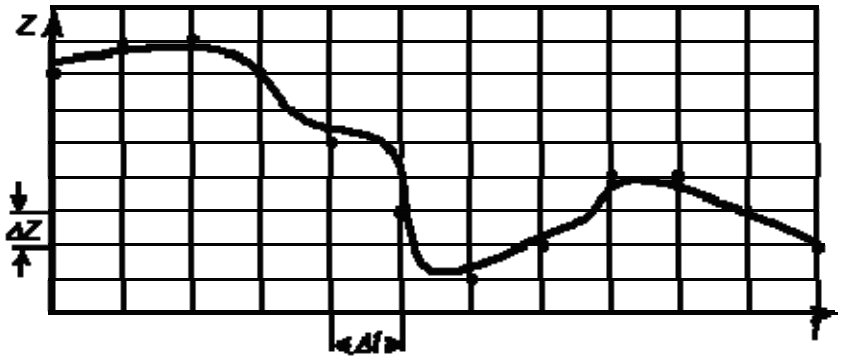


Рис. 1.1. Дискретизация аналогового сигнала за счет операций развертки по времени и квантования по величине

При такой замене довольно очевидно, что чем меньше n (больше Δt), тем меньше число узлов, но и точность замены Z(t) значениями Z_i будет меньшей. Другими словами, при дискретизации может происходить потеря части информации, связанной с особенностями функции Z(t). Увеличением количества точек n можно улучшить соответствие между получаемым массивом и исходной функцией, однако полностью избежать потерь информации все равно не удастся, поскольку n – величина конечная. Теорема отсчетов, доказанная в 1933 г. В.А. Котельниковым, дает критерий для выбора n. Теорема гласит:

Непрерывный сигнал можно полностью отобразить и точно воссоздать по последовательности измерений или отсчетов величины этого сигнала через одинаковые интервалы времени, меньшие или равные половине периода максимальной частоты, имеющейся в сигнале.

Согласно теореме отсчетов, определяющим является значение верхней границы частоты. Обозначим его ν_m.

Смысл теоремы в том, что дискретизация не приведет к потере информации и по дискретным сигналам можно будет полностью восстановить исходный аналоговый сигнал, если развертка по времени выполнена в соответствии со следующим соотношением:

$$\Delta t = \frac{1}{\nu_m}.$$

Например, для точной передачи речевого сигнала с частотой до ν_m = 4000 Гц при дискретной записи должно производиться не менее 8 000 отсчетов в секунду.

Однако, помимо временной развертки, дискретизация имеет и другую составляющую – квантование. Шаг квантования определяется чувствительностью приемного устройства. Любой получатель сообщения всегда имеет конечную предельную точность распознавания величины сигнала. Например, человеческий глаз в состоянии различить около 16 миллионов цветовых оттенков. Это означает, что при квантовании цвета нет смысла делать большее число градаций. При передаче речи достаточной оказывается гораздо меньшая точность – около 1 %. Следовательно, для амплитуды звуковых колебаний $\Delta Z = 0{,}01Z_{max}$.

Указанные соображения по выбору шага развертки по времени и квантования по величине сигнала лежат в основе оцифровки звука и изображения. Примерами устройств, в которых происходят такие преобразования, являются сканер, модем, устройства для цифровой записи звука и изображения, лазерный проигрыватель, графопостроитель.

Таким образом, преобразование сигналов типа N→D, как и обратное D→N, может осуществляться без потери содержащейся в них информации.

Преобразование типа $D_1 \rightarrow D_2$ состоит в переходе при представлении сигналов от одного алфавита к другому – такая операция носит название перекодировка и может осуществляться без потерь. Примерами ситуаций, в которых осуществляются подобные преобразования, могут быть: запись-считывание с компьютерных носителей информации; шифровка и дешифровка текста.

Таким образом, за исключением $N_1 \rightarrow N_2$ в остальных случаях оказывается возможным преобразование сообщений без потерь содержащейся в них информации. При этом на первый взгляд непрерывные и дискретные сообщения оказываются равноправными. Однако на самом деле это не так. Сохранение информации в преобразованиях N→D и D→N обеспечивается именно благодаря участию в них дискретного представления. Другими словами, преобразование сообщений без потерь информации возможно только в том случае, если хотя бы одно из

них является дискретным. В этом проявляется несимметричность видов сообщений и **преимущество дискретной формы.** К другим ее достоинствам следует отнести:

- высокая помехоустойчивость;
- простота и, как следствие, надежность, а также относительная дешевизна устройств по обработке информации;
- точность обработки информации, которая определяется количеством обрабатывающих элементов и не зависит от точности их изготовления;
- универсальность устройств.

1.2. Измерение информации

Случайные события могут быть описаны с использованием понятия «вероятность». Соотношения теории вероятностей позволяют найти (вычислить) вероятности как одиночных случайных событий, так и сложных опытов, объединяющих несколько независимых или связанных между собой событий. Однако описать случайные события можно не только в терминах вероятностей.

То, что событие случайно, означает отсутствие полной уверенности в его наступлении, что создает неопределенность в исходах опытов, связанных с данным событием. Степень неопределенности различна для разных ситуаций. Для практики важно иметь возможность произвести численную оценку неопределенности разных опытов. Попробуем ввести такую количественную меру неопределенности.

Начнем с простой ситуации, когда опыт имеет n равновероятных исходов. Очевидно, что неопределенность каждого из них зависит от n, т. е. мера неопределенности является функцией числа исходов $f(n)$.

Можно указать некоторые свойства этой функции:

1) $f(1) = 0$, поскольку при $n = 1$ исход опыта не является случайным и неопределенность отсутствует;
2) $f(n)$ возрастает с ростом n, поскольку чем больше число возможных исходов, тем более затруднительным становится предсказание результата опыта.

Для определения явного вида функции $f(n)$ рассмотрим два независимых опыта α и β с количествами равновероятных исходов, соответственно, n_α и n_β. Пусть имеет место сложный опыт, который состоит в одновременном выполнении опытов α и β. Число возможных его исходов равно $n_\alpha \cdot n_\beta$, причем все они равновероятны. Очевидно, что неопределенность исхода такого сложного опыта будет больше неопределенности опыта α. Мера неопределенности сложного опыта равна $f(n_\alpha \cdot n_\beta)$. С другой стороны, меры неопределенности отдельных опытов состав-

ляют $f(n_\alpha)$ и $f(n_\beta)$, соответственно. В первом случае (сложный опыт) проявляется общая (суммарная) неопределенность совместных событий, во втором – неопределенность каждого из событий в отдельности. Однако из независимости α и β следует, что в сложном опыте они никак не могут повлиять друг на друга. Следовательно, мера суммарной неопределенности должна быть равна сумме мер неопределенности каждого из опытов, т. е. мера неопределенности аддитивна:

$$f(n_\alpha \cdot n_\beta) = f(n_\alpha) + f(n_\beta). \qquad (1.1)$$

Задумаемся, каким может быть явный вид функции $f(n)$, чтобы он удовлетворял свойствам (1) и (2) и соотношению (1.1). Легко увидеть, что такому набору свойств удовлетворяет функция $\log(n)$, причем можно доказать, что она – единственная из всех существующих классов функций. Таким образом, за меру неопределенности опыта с n равновероятными исходами можно принять число $\log(n)$.

Нами установлен явный вид функции, описывающей меру неопределенности опыта, имеющего n равновероятных исходов. Эта величина получила название *энтропия*. В дальнейшем будем обозначать ее H.

Вновь рассмотрим опыт с n равновероятными исходами. Поскольку каждый исход случаен, он вносит свой вклад в неопределенность всего опыта, но так как все n исходов равнозначны, разумно допустить, что и их неопределенности одинаковы. Из свойства аддитивности неопределенности, а также из того, что общая неопределенность равна $\log n$, следует, что неопределенность, вносимая одним исходом, составляет

$$\frac{1}{n}\log n = -\frac{1}{n}\log\frac{1}{n} = -p\log p\,,$$

где $p = 1/n$ – вероятность любого из отдельных исходов.

Таким образом, неопределенность, вносимая каждым из равновероятных исходов, равна:

$$H = -p\ log\ p.$$

Обобщая это выражение на ситуацию, когда опыт имеет n неравновероятных исходов A_1, $A_2 \ldots A_n$, получим:

$$H(\alpha) = -\sum_{i=1}^{n} p(A_i)\log p(A_i)\,. \qquad (1.2)$$

Введенная величина, как уже было сказано, называется энтропией опыта. Энтропия является мерой неопределенности опыта, в котором проявляются случайные события, и равна средней неопределенности всех возможных его исходов.

Как следует из (1.2), $H = 0$ только в двух случаях:

- какая-либо из $p(A_i) = 1$ (тогда вероятности всех остальных исходов равны 0, т. е. реализуется ситуация, когда один из исходов является достоверным и общий итог опыта перестает быть случайным);
- все $p(A_i) = 0$ (никакие из рассматриваемых исходов опыта невозможны).

Во всех остальных случаях, очевидно, что $H > 0$. Очевидным следствием будет утверждение, что для двух независимых опытов энтропия сложного опыта равна сумме энтропий отдельных опытов.

Разность $H(\beta)$ и $H_\alpha(\beta)$, очевидно, показывает, какие новые сведения относительно β мы получаем, произведя опыт α. Эта величина называется информацией относительно опыта β, содержащейся в опыте α.

$$I(\alpha, \beta) = H(\beta) - H_\alpha(\beta). \qquad (1.3)$$

Данное выражение открывает возможность численного измерения количества информации, поскольку оценивать энтропию мы уже умеем. Из него легко получить ряд следствий.

Пусть просто произведен опыт β. Поскольку он несет полную информацию о себе самом, неопределенность его исхода полностью снимается, т. е. $H_\beta(\beta) = 0$. Тогда $I(\beta, \beta) = H(\beta)$, т. е. можно считать, что энтропия опыта равна той информации, которую мы получаем в результате его осуществления.

Количество информации, передаваемой от источника к приемнику, связано с вероятностью пребывания источника в том или ином состоянии. Если состояние источника известно заранее (до передачи информации), то количество информации, получаемой приемником при передаче, равно нулю. Если же состояние источника заранее неизвестно, то количество получаемой информации определяется формулой:

$$I = -\sum_{i=1}^{N} P_i \cdot \log_k P_i ,$$

где N – число состояний, в которых может находиться источник информации, P_i – вероятность появления i-го состояния (i = 1, 2, ..., N–1, N). При равновероятных состояниях источника (т. е. при $P_1 = P_2 = \ldots = P_N = 1/N$)

$$I = \log_k N.$$

Основание логарифма k в формулах определяет единицу количества информации. При k = 2 соответствующая единица называется **бит** (*bit* – от слов *binary digit*). Такая единица чаще всего встречается в технике, что обусловлено наиболее частым использованием двузначного алфавита для представления дискретной информации. Один бит равен количеству информации, получаемому от источника с двумя равновероятными состояниями.

Современные ЭВМ могут обрабатывать не только числовую информацию, но и информацию, заданную любыми другими символами. Обычно для представления одного символа служит слово длиной в $2^3 = 8$ бит, получившее название **байта.** Посредством слов такой длины можно закодировать $2^8 = 256$ различных символов, чего вполне достаточно при решении многих задач, связанных с обработкой символьной информации. Количество информации в этом случае удобно измерять также в байтах.

Для измерения больших объемов информации применяют специальные единицы, которые обозначаются к и М и читаются, соответственно, «кило» и «мега». При этом 1 к = 1024 = 2^{10}, 1 М = 1048576 = 2^{20}. Например, 1 Мб = 2^{10} кб = 2^{20} байт = 2^{23} бит, 1 Мбит = 2^{10} кбит = 2^{20} бит, 1 кб = 2^{10} байт = 2^{13} бит, 1 кбит = 2^{10} бит. Иногда в приближенных расчетах полагают, что к ≈ 10^3, М ≈ 10^6.

Единицей измерения скорости передачи информации по каналам связи служит **бод**, равный 1 бит/с.

Кроме перечисленных единиц для измерения количества информации, обрабатываемой и хранимой в компьютере, используют также единицы, не имеющие постоянного количественного эквивалента. К таким единицам относятся: **поле, слово, массив** и др. **Поле** представляет собой группу бит, имеющую определенное значение (например, поле, в котором указывается в кодированном виде операция, выполняемая микропроцессором). **Словом** называют число, которое может быть обработано микропроцессором за одну команду (чаще всего под словом подразумевают два байта).

Совокупность бит, байтов, полей, слов, объединяемых некоторым общим признаком (например, исходные данные для решения задачи), называют массивом.

Информация *– это содержание сообщения, понижающего неопределенность некоторого опыта с неоднозначным исходом; убыль связанной с ним энтропии является количественной мерой информации. В случае равновероятных исходов информация равна логарифму отношения числа возможных исходов до и после (получения сообщения).*

В качестве примера определим количество информации, получаемое приемником информации при предаче от источника картинки, представленной на рис. 1.2.

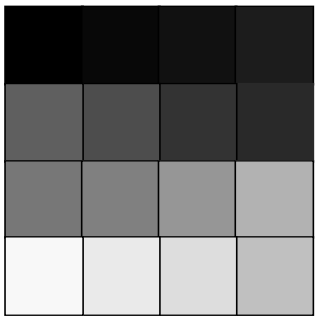


Рис. 1.2. Тестовое полутоновое изображение

Изображение представляет собой квадрат, разбитый на $N = 16 = 2^4$ полей. Каждое поле имеет неповторяющуюся заливку. Вероятности появления каждого поля одинаковы и равны $P = 1/N = 1/16$. Тогда искомое количество информации

$$I = \mathrm{N}\cdot\log_2 N = 16\cdot\log_2 2^4 = 16\cdot 4 = 64 \text{ бит} = 8 \text{ байт}.$$

1.3. Системы счисления, применяемые в цифровых системах

Информация, обрабатываемая ВС, представляет собой числа. Одно и то же число можно выразить различными комбинациями цифр и буквенных символов. Конкретный вид такой комбинации зависит от выбранной системы счисления.

Системой счисления *называют совокупность приемов и правил для обозначения и наименования чисел. В любой системе счисления число представляют некоторой совокупностью символов, которые называются цифрами*.

Каждой цифре однозначно ставится в соответствие определенное количество, выражаемое данной цифрой (количественный эквивалент).

Различают непозиционные и позиционные системы счисления. В непозиционной системе каждой цифре, независимо от места расположения, ставится в соответствие один и тот же количественный эквивалент. К числу таких систем относится, например, римская нумерация. Непозиционные системы счисления крайне неудобны для выполнения вычислений и в науке и технике не используются.

Систему счисления называют позиционной, если одной и той же цифре соответствуют разные количественные эквиваленты в зависимости от местоположения (разряда) цифры в записи числа.

Пример позиционной системы – общеизвестная десятичная система.

Для определения полного количественного эквивалента записи числа используется некоторая функция от количественных эквивалентов совокупности цифр в этой записи. Для большинства существующих систем счисления эта функция является функцией десятичного сложения.

Если в позиционной системе счисления каждая цифра имеет свой определенный символ, то такая система является системой с непосредственным представлением цифр. Существуют также системы с кодированным представлением цифр (например, двоично-десятичная система, упоминаемая далее).

При записи чисел преимущественно используются однородные системы, у которых во всех разрядах записи числа используются цифры из одного множества.

Любое число N можно представить в такой системе в виде

$$N = n_k n_{k-1} \ldots n_1 n_0, n_{-1} n_{-2} \ldots n_{-m}.$$

Количественный эквивалент числа, представленного этой записью,

$$N = r^k \cdot n_k + r^{k-1} \cdot n_{k-1} + \ldots + r^1 \cdot n_1 + r^0 \cdot n_0 + r^{-1} \cdot n_{-1} + r^{-m} \cdot n_{-m} = \sum_{i=-m}^{k} r^i \cdot n_i , \quad (1.4)$$

где число r – основание системы счисления (оно равно количеству цифр, используемых в данной системе); n_i – цифра i-го разряда (выбирается из допустимого множества значений для данной системы).

В наиболее привычной для нас десятичной системе счисления r = 10, т. е. основанием системы служит число 10, и множество цифр системы состоит из десяти цифр (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Например число $7314{,}56_{10}$ (индекс показывает основание системы счисления) представляется суммой $7314{,}56 = 7 \cdot 10^3 + 3 \cdot 10^2 + 1 \cdot 10^1 + 4 \cdot 10^0 + 5 \cdot 10^{-1} + 6 \cdot 10^{-2}$.

В цифровой технике широкое распространение получила двоичная система счисления. Это объясняет тем, что цифровые схемы (и в том числе микропроцессорные) строятся на основе логических элементов, имеющих два возможных состояния 0 и 1. Для двоичной системы счисления r = 2 и множество цифр образуют две цифры: 0 и 1. Следовательно, в этой системе счисления k-разрядное число N имеет вид $N = 2^{k-1} \cdot n_{k-1} + 2^{k-2} \cdot n_{k-2} + \ldots + 2^1 \cdot n_1 + 2^0 \cdot n_0$, где n может принимать только два значения: либо 0, либо 1. Например, число 21_{10} в двоичной системе запишется так: 10101_2.

Помимо основной двоичной системы счисления, в вычислительной технике применяют также восьмеричную и шестнадцатеричную системы. Они используются для сокращения записи двоичных чисел, так как числа из двоичной системы легко переводятся в восьмеричную и шестнадцатеричную.

Множество цифр восьмеричной системы содержит цифры: 0, 1, 2, 3, 4, 5, 6, 7. Основание системы r = 8. Так, числу $542_8 = 5 \cdot 8^2 + 4 \cdot 8^1 + 2 \cdot 8^0$ в десятичной системе соответствует число 354_{10}, а в двоичной системе – число 101100010_2.

Шестнадцатеричная позиционная система наиболее удобна при представлении больших чисел, так как записи получаются короткими. Основание системы r = 16. Для записи чисел используются десять цифр от 0 до 9 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) и шесть латинских букв: *A*, *B*, *C*, *D*, *E*, *F* (буквы *A*, *B*, *C*, *D*, *E*, *F* соответствуют числам 10, 11, 12, 13, 14, 15 десятичной системы счисления). Например, число 1101010110110010_2, которому в десятичной системе соответствует число 54706_{10}, в шестнадцатеричной системе запишется в виде $D5B2_{16}$.

При записи чисел в различных системах счисления в цифровой технике используется ряд специальных соглашений. Чтобы отличить шестнадцатеричное число от идентификаторов (названий переменных), в них, если это число начинается с буквы, при записи добавляется слева незначащий нуль (0). Для обозначения системы счисления, в которой записано число, принято использовать не индексы, а буквы после числа. Если число заканчивается запятой или после числа нет обозначения, оно считается десятичным (иногда для обозначения десятичного числа в конце его ставят букву *D*). В конце двоичного числа ставят букву *B*, шестнадцатеричного – *H*. Числа из предыдущего примера тогда будут выглядеть следующим образом: 1101010110110010B, 54706, 0D5B2H.

При вычислениях в цифровой технике иногда возникает необходимость использования десятичных чисел. Для представления таких чисел используется двоично-кодированная десятичная система счисления (в литературе на английском языке BCD – аббревиатура слов *Binary Coded Decimal*). В этом коде каждая цифра десятичной системы счисления заменяется эквивалентным четырехразрядным двоичным числом (тетрадой). Старший разряд тетрады имеет вес $2^3 = 8$, следующий – 4, второй – 2 и младший – 1. Например, цифре 3 соответствует в двоично-десятичной системе число 0011, цифре 7 – число 0111 и т. д.

Выполнение арифметических операций в различных системах счисления проводятся по правилам, аналогичным известным нам для десятичной системы. Правила эти следующие:

- арифметические операции сложения и вычитания проводятся поразрядно, то есть каждый разряд одного числа складывается (вычитается) с разрядом того же веса другого числа;
- если при выполнении операции над разрядами получается число, большее чем основание системы счисления, то говорят, что возникает перенос, который учитывается при выполнении операции с более старшим разрядом;

- если при выполнении операции вычитания уменьшаемый разряд меньше вычитаемого, то возникает заем, который уменьшает на 1 более старший разряд уменьшаемого;
- умножение проводится поразрядным умножением со сложением получаемых результатов;
- в каждой системе счисления существует своя таблица сложения и умножения.

Таблицы умножения и сложения для двоичной и шестнадцатеричной систем счисления приведены в табл. 1.1 и 1.2. Наиболее просты эти таблицы для двоичной системы и, соответственно, выполнение арифметических операций в этой системе проводится легко. Недостатком является громоздкое представление чисел в этой системе, поэтому наверное она и не нашла применения в жизни. Примеры выполнения арифметических операций будут приведены.

Числа можно переводить из одной системы счисления в любую другую позиционную систему. Для перевода числа из любой позиционной системы счисления в десятичную пользуются формулой (1.4). При выполнении вычислений пользуются правилами десятичной системы.

Например, переведем в десятичную систему числа 100110B, 0F2AH:

$$100110_2 = 0\cdot2^0+1\cdot2^1+1\cdot2^2+0\cdot2^3+0\cdot2^4+1\cdot2^5 = 2+4+32 = 38_{10};$$

$$0F2A_{16} = 10\cdot16^0+2\cdot16^1+15\cdot16^2 = 10+32+3840 = 3882_{10}.$$

Перевод целых чисел из десятичной системы в любую другую может быть сделан по следующему правилу:

Переводимое число делится нацело на основание новой системы счисления, остаток от деления запоминается. Полученное частное снова делится нацело на основание новой системы и т. д. до тех пор, пока частное не станет меньше основания новой системы. Тогда запомненные остатки будут цифрами нового представления числа, причем последнее частное будет старшим разрядом, а первый остаток младшим.

Таблица 1.1

Двоичная система счисления

Таблица сложения

	0	1
0	0	1
1	1	10

Таблица умножения

	0	1
0	0	0
1	0	1

Таблица 1.2

Шестнадцатеричная система счисления

Таблица сложения

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	20
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

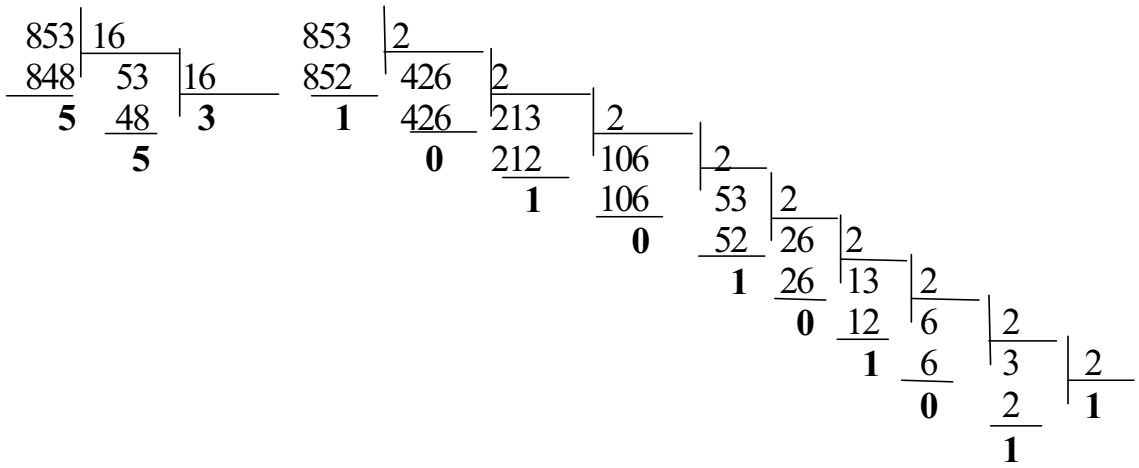
Таблица умножения

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

Сложнее дело обстоит с переводом дробных чисел. Для правильных дробей можно использовать следующий алгоритм перевода:

Переводимое дробное число умножается на основание новой системы счисления по правилам умножения десятичных чисел. В полученном произведении отделяется и запоминается целая часть, а оставшаяся дробь снова умножается с отделением дробной части. Операция повторяется до получения нулевой дробной части или при достижении нужного количества разрядов. Старшая цифра результата совпадает с первой отделенной целой частью, вторая цифра – со второй отделенной частью и т. д.

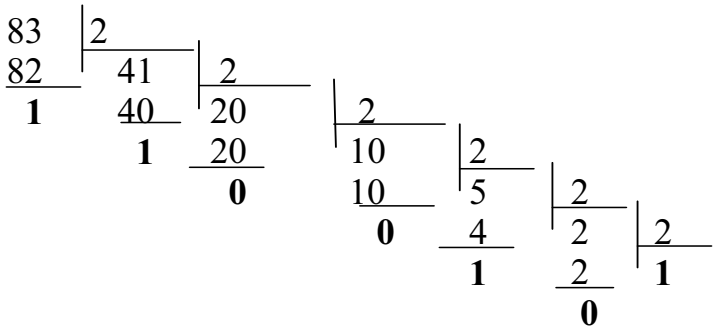
Для примера переведем в двоичную и шестнадцатеричную системы десятичное число 853:



$853_{10} = 355_{16} = 1101010101_2$.

Для смешанных дробных чисел отдельно переводится целая и дробная части.

Например, переведем в двоичную систему десятичное число 83,75.



$83_{10} = 1010011_2$;

$0{,}75 \cdot 2 = \mathbf{1}{,}5$; $0{,}5 \cdot 2 = \mathbf{1}{,}0$; $0{,}0 \cdot 2 = 0$; $0{,}75_{10} = 0{,}11_2$;

$83{,}75_{10} = 1010011{,}11_2$.

Для проверки сделаем обратный перевод в десятичную систему, воспользовавшись формулой (1.4):

$$1010011{,}11_2 = 1 \cdot 2^{-2} + 1 \cdot 2^{-1} + 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 + 0 \cdot 2^5 + 1 \cdot 2^6 =$$
$$= 0{,}25 + 0{,}5 + 1 + 2 + 16 + 64 = 83{,}75_{10}.$$

В вычислительной технике очень часто требуется переводить числа из двоичной системы счисления в восьмеричную или шестнадцатеричную и наоборот.

Таблица 1.3

Представление восьмеричных и шестнадцатеричных цифр в двоичной системе счисления

Шестнадцатеричная (**восьмеричная**) цифра	Двоичное число	Шестнадцатеричная цифра	Двоичное число
0	0**000**	8	1000
1	0**001**	9	1001
2	0**010**	A	1010
3	0**011**	B	1011
4	0**100**	C	1100
5	0**101**	D	1101
6	0**110**	E	1110
7	0**111**	F	1111

Для перевода числа из двоичной системы в восьмеричную его разбивают на триады (трехразрядные двоичные числа) справа налево и заменяют каждую триаду эквивалентной восьмеричной цифрой (если при образовании последней триады не хватает двоичных цифр, то слева добавляют необходимое количество нулей).

Аналогичным путем переводится двоичное число в число шестнадцатеричной системы, но с тем отличием, что переводимое число разбивают на тетрады (четырехразрядные двоичные числа) и заменяют каждую тетраду эквивалентной шестнадцатеричной цифрой (если при образовании последней тетрады не хватает двоичных цифр, то слева добавляют необходимое количество нулей).

И наоборот, при переводе восьмеричных чисел в двоичные каждая восьмеричная цифра заменяется тремя двоичными разрядами, а при переводе шестнадцатеричных – четырьмя двоичными разрядами. При этом удобно пользоваться табл. 1.3.

Примеры:

$1101011_2 = 001'101'011 = 153_8 = 0110'1011 = 6B_{16}$;

$0D8F4_{16} = 1101\ 1000\ 1111\ 0100_2 = 001'101'100'011'110'100 = 154364_8$.

1.4. Кодирование информации в цифровых системах

На рис. 1.3 представлена схема типов информации, используемых в компьютерных системах. Можно выделить три основных типа информации: адреса, данные и команды. Адреса представляют собой

многоразрядные числа в двоичной системе счисления, которые используются как номера ячеек памяти, портов ввода/вывода и т. д. Команда – двоичное число, являющееся инструкцией микропроцессору. Подробнее об адресах и командах идет речь в последующих главах.

Данные – это информация, обрабатываемая вычислительной системой. Данные могут быть числовыми и нечисловыми.

Нечисловые данные представляют собой некую символьную информацию, например, какой-либо текст. Так как вычислительная система (ВС) – это цифровая система, то есть любая информация должна представлять число, то возникает необходимость кодировки символов. Каждому используемому символу должно быть поставлено в соответствие двоичное число. Чтобы не возникало проблем переноса символьной информации с одной ВС на другую, используют стандартные таблицы кодировок. В табл. 1.4 приведен пример такой стандартной кодовой таблицы (таблица кодов ASCII).

Основной тип данных, используемых в ВС, – числа. Наиболее просто кодируются **целые числа без знака**. В ВС они представлены в двоичной системе счисления 8-разрядными (**байт**) или 16-разрядными (**слово**) числами.

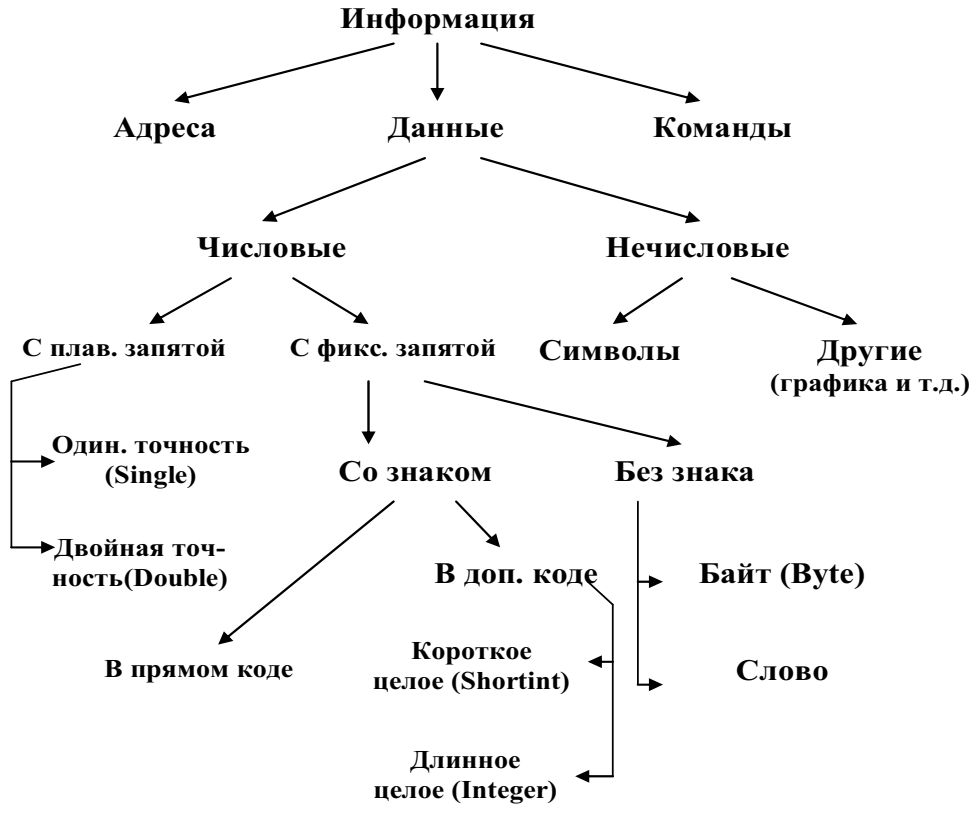


Рис. 1.3. Типы информации

Таблица 1.4

Таблица кодов ASCII

	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	'	p
0001	SOH	DCI	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	‘	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	X
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	1	|
1101	CR	GS	–	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Отрицательные числа также должны быть закодированы определенным образом. Знак числа кодируется значением старшего разряда. Общепринято кодировать положительные числа нулевым старшим разрядом, а отрицательные – значением разряда, равным r–1 (r – основание системы счисления). Тогда в двоичной системе счисления отрицательное число кодируется 1, а в десятичной системе – 9.

Для правильного выполнения арифметических операций над числами, у которых знаки закодированы старшими разрядами числа, в вычислительном устройстве используют специальные способы представления чисел со знаком: прямой и дополнительный коды.

Представление числа в **прямом коде** определяется формулой

$$N = \begin{cases} \underline{0}N \ , N \geq 0; \\ \underline{(r-1)}|N|, N < 0. \end{cases}$$

Например, $+100_{10} = \underline{0}1100100_2$; $-100_{10} = \underline{1}1100100_2$. (Подчеркнут знаковый разряд.)

Использование прямого кода не всегда удобно, так как при сложении положительного и отрицательного числа в вычислительном средстве должно быть дополнительное устройство, реализующее операцию вычитания в дополнение к операции сложения. От этого недостатка свободны числа, представленные в дополнительном коде.

Дополнительный код числа образуется по правилу:

$$N = \begin{cases} \underline{0}N\ , N \geq 0; \\ \underline{(r-1)}(r^{m+1} - |N|), N < 0, \end{cases}$$

где m – разрядность числа N.

Предыдущий числовой пример в дополнительном коде будет выглядеть следующим образом: $+100_{10} = 01100100_2$; $-100_{10} = 10011100_2$.

При сложении двух одинаковых по модулю, но противоположных по знаку чисел в дополнительном коде сумма равна 1 более старшего разряда, чем разрядность исходного числа, поэтому код и называется дополнительным (в смысле дополнения до единицы более старшего разряда). Для нашего примера:

$$\begin{array}{r} 01100100 \\ +\ 10011100 \\ \hline 100000000. \end{array}$$

При выполнении арифметических действий в вычислительном средстве перенос в старший разряд теряется и результат имеет ту же разрядность, что и исходные числа, и представлен в дополнительном коде.

Вычисление примеров 51+(–72) = (–21); (–51)+72 = 21 и (–51)+(–72) = (–123) для чисел в двоичной системе счисления в дополнительном коде будет иметь вид:

$$\begin{array}{r} 00110011\ \ (51) \\ +\ \underline{10111000\ (-72)} \\ 11101011\ (-21) \end{array} \qquad \begin{array}{r} 11001101\ (-51) \\ +\ \underline{01001000\ \ (72)} \\ \smile 00010101\ \ (21) \end{array} \qquad \begin{array}{r} 11001101\ \ (-51) \\ +\ \underline{10111000\ \ (-72)} \\ \smile 10000101\ (-123), \end{array}$$

символ $\smile$ обозначает перенос из знакового разряда.

Подчеркнем, что дополнительный код применяется для чисел **одинаковой фиксированной разрядности**. В случае двоичных чисел это 8- или 16-разрядные числа (однобайтовые короткие целые и двухбайтовые целые).

В цифровых системах используются две формы представления чисел: естественная и нормальная. При естественной форме, иначе называемой формой с фиксированной запятой, числа вводятся в виде целой и дробной частей, разделенных запятой (точкой). Положение последней строго фиксировано: запятая находится либо перед цифрой старшего разряда, либо после цифры младшего разряда. Первый вариант относится к представлению чисел, которые по модулю (без учета знака) меньше единицы, второй вариант представления распространяется только на целые числа. Порядковые номера разрядов идут слева направо, начиная с нулевого. Его называют знаковым разрядом, и в этом разряде 0 соответствует знаку плюс, а 1 – знаку минус.

Научный и ряд других приложений требуют очень широких диапазонов числовых величин. В этих случаях используются числа с плавающей запятой. Нормальная или полулогарифмическая форма, иначе называемая формой с плавающей запятой, предполагает ввод чисел в полулогарифмическом виде – числа состоят из двух частей: мантиссы числа, обозначаемой буквой m, и порядка числа, который обозначается буквой p, причем $|m| < 1$, а p – всегда целое. Положение запятой в числе зависит от порядка p (отсюда и название формы – с плавающей запятой).

Когда в мантиссе перед запятой стоит нуль, а после запятой – цифра, отличная от нуля, то такую форму называют нормализованной. Действия над числами, представленными в нормализованной форме, сложнее, чем над числами с фиксированной запятой. Но зато форма с плавающей запятой позволяет охватить очень широкий диапазон чисел.

Число с плавающей запятой записывается как $\pm m \times r^p$, где m – мантисса, r – основание системы счисления, p – порядок числа.

Числа с плавающей запятой представляются в компьютерной системе упакованными в одно двоичное число, p и m составляют части этого числа. Код для основания системы счисления r можно не сохранять, так как он всегда известен. Наиболее часто встречается величина основания системы счисления $r = 2$, соответствующая двоичным числам.

До середины 1980-х гг. каждая компьютерная фирма-изготовитель разрабатывала собственную схему кодирования чисел с плавающей запятой. Это приводило к трудностям при передаче данных между различными компьютерами. В 1985 году ANSI (Американский национальный институт стандартов) и ИИЭР разработали стандарт для двоичной арифметики с плавающей запятой. Стандарт определяет двоичные числа в формате с плавающей запятой с одинарной точностью (32 двоичных разряда) и двойной точностью (64 двоичных разряда).

Мантисса сохраняется в нормализованном виде как число со знаком. Знаковый разряд помещается в крайний левый бит 32-разрядного кода числа, позволяя легко идентифицировать число как положительное или отрицательное. Величина мантиссы имеет форму $\underline{1}$.F, где F – 23-разрядная двоичная дробь в битах 22…0 для формата с одинарной точностью и 53-разрядная дробь, сохраняемая в битах 52...0 для формата двойной точности. Старший разряд нормализованной мантиссы всегда $\underline{1}$ и этот разряд не запоминают, хотя подразумевается, что он есть.

На рис. 1.4 приведен формат двоичного числа с плавающей запятой одинарной точности. Число имеет размер 4 байта (32 бита). 23 бита отводится под мантиссу, 8 бит – под порядок и 1 бит – под знак числа.

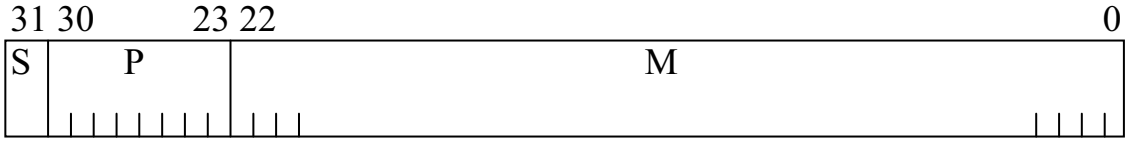


Рис. 1.4. Представление числа с плавающей запятой:
S – знак числа, P – порядок числа, M – мантисса числа

8-разрядный порядок сохраняется в смещенном формате. В числовом формате со смещением величина смещения N добавляется к каждой величине, чтобы числа в диапазоне [–N...+(N–1)] представлялись линейно увеличивающейся последовательностью номеров [0...(N–1)]. В рассматриваемом стандарте величина смещения, равная 127, добавляется к каждому значению порядка и результат сохраняется в битах 30...23. Диапазон изменения порядка составляет [–127...+126], следовательно двоичные коды порядка увеличиваются в диапазоне [00000001... 11111110]. Порядки, закодированные двоичными числами 00000000 и 11111111, зарезервированы, чтобы указать особые условия. Например, константа нуль представляется всеми нулями, а бесконечность – всеми единицами в порядке и всеми нулями в мантиссе.

Нормализованные формы характеризуются их старшими значащими разрядами, отличными от нуля, и используются, чтобы гарантировать единственное представление для каждого числа с плавающей запятой. Например, следующие записи – это представления одного и того же числа:

10.11×2^{0};
1.011×2^{1};
0.1011×2^{10};
0.01011×2^{11}.

Требование, чтобы мантисса была нормализована к форме 1.F, приводит к тому, что только число 1.011×2^{1} можно использовать для кодирования.

Примеры решения задач

1. Вычислите количество информации, которое содержит тестовый рисунок (рис. 1.5). Для заливки полей на рисунке использованы 8 полутонов серого.

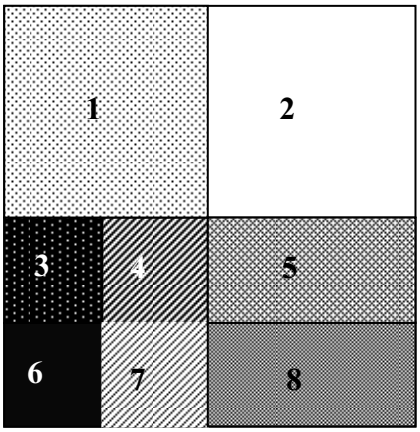


Рис. 1.5. Тестовый рисунок (пример 1)

Решение

Информация о каждом поле рисунка состоит из двух частей: информация о цвете и информация о размере. Полная информация равна сумме этих информаций $I = I_c + I_a$. Первая составляющая I_c (цвет) может быть вычислена по формуле 1.1: $I_c = \log_2 N = \log_2 8 = 3$ бита (в данном случае имеется 8 равновероятных состояний). Так как площади полей неодинаковы, для вычисления второй составляющей I_a (размер) должна быть использована более общая формула: $I = -\sum_{i=1}^{N} P_i \cdot \log_k P_i$, где $N = 8$ (количество полей) и P_i (вероятность появления каждого поля) может быть вычислена по формуле $P_i = S_i / S$, где S – полная площадь квадрата и S_i – площадь i-го поля. Нумерация полей показана на рис. 1.1. Таким образом, $I_a = 2 \cdot \frac{1}{4} \cdot \log_2 4 + 2 \cdot \frac{1}{8} \cdot \log_2 8 + 4 \cdot \frac{1}{16} \cdot \log_2 16 = 2\frac{6}{8}$ бит.

Полное количество информации тогда составит: $\mathrm{I} = 8 \cdot \left(3 + 2\frac{6}{8} \right) = 46$ бит.

Обратите внимание, что в случае, если бы мы разбили квадрат на 16 равных клеток, закрашенных в 8 тонов серого, общее количество информации было бы равно 48 битам, то есть такое кодирование рисунка не было бы оптимальным. Подходы, подобные приведенному в задаче, используются для сокращения количества информации в файлах (архивирования).

2. Что определяет точность числа в позиционной системе счисления? Сколько значащих цифр m мы должны сохранять в разных системах счисления для обеспечения заданной точности?

Решение

Самая младшая значащая цифра числа определяет точность числа. Таким образом, абсолютная ошибка не превышает единицы младшего значащего разряда числа, следовательно должно выполняться неравенство:

$$1/r^m \le \Delta,$$

где Δ – абсолютная ошибка, r – основание системы счисления, в котором записано число, m – номер младшей значащей цифры в числе. Таким образом, количество значащих цифр можно записать в виде

$$m \ge \frac{\log 1/\Delta}{\log r}.$$

3. 16-битовая (2-байтовая) ячейка памяти может быть использована для записи числа с фиксированной точкой и числа с плавающей точкой. Как мы должны распределить биты этой ячейки для мантиссы и показателя числа, чтобы в случае записи числа с плавающей точкой диапазон представления чисел был больше, чем в случае записи с фиксированной точкой?

Решение

Максимальный диапазон представления числа с фиксированной точкой в нашем случае $2^{16}-1 = 65535 \approx 2^{16}$. Если мы будем использовать для записи формат с плавающей запятой $\pm m \times 2^p$, где m и p кодируются в виде полей двухбайтового слова, мы получим максимальный размер представляемого числа $2\cdot\left(2-2^{-(m-1)}\right)\cdot 2^{2^{p-1}-1} \approx 2^{2^{p-1}+1}$, где m – количество разрядов мантиссы и p – количество разрядов порядка ($m > > 1$). Тогда мы получим: $2^{p-1}+1 > 16$ и $p > 1 + \log_2 15 \ge 5$. Если $p = 5$, то $m = n-p = 11$.

Задачи и упражнения

Измерение информации

1.1. Что такое бит? Почему эта единица измерения количества информации так широко распространена?

1.2. Для вычисления количества информации приведена общая формула

$$I = -\sum_{i=1}^{N} P_i \cdot \log_k P_i,$$

где P_i – вероятность i-го состояния источника информации ($i = 1, 2 \ldots N-1, N$), N – полное количество состояний.

Покажите, что в случае, если все состояния источника информации равновероятны, данная формула переходит в формулу 1.1.

1.3. Некоторое сообщение передается по линии связи как текст на английском языке. В сообщении используются заглавные буквы, пять специальных знаков (.,!?–) и пробел. Сколько бит необходимо для кодирования каждого символа сообщения?

1.4. Повторите вычисления для предыдущей задачи, если текст передается на русском языке.

1.5. Вычислите количество информации, содержащейся в книге, если в ней 420 страниц, 56 строк на каждой странице и, в среднем, 64 символа в каждой строке. Для кодирования символов в книге использована расширенная таблица ASCII.

1.6. Рисунок размером 10×10 квадратных сантиметров распечатывается на принтере. Какой объем памяти занимает рисунок в памяти компьютера, если разрешение принтера составляет 120 точек на дюйм?

1.7. Черно-белая фотография размером 9 × 6 квадратных сантиметров состоит из отдельных точек размером 1/3 мм. Яркость каждой точки может принимать значение из 32 возможных оттенков серого. Определите количество информации, содержащейся в рисунке.

1.8. На дискету объемом 1,44 Мб нужно записать 2 сканированных изображения в виде файлов. Сжатие файлов не используется. Размеры изображений – 6 × 9 и 9 × 12 сантиметров. Цвет изображений кодируется тремя основными цветами (красным, синим, зеленым), и яркость каждого цвета выбирается из 256 возможных значений. Определите максимальное разрешение, с которым можно сканировать данные изображения.

Системы счисления и кодирование чисел

1.9. Сколько различных чисел можно записать при помощи n знаков в системе счисления с основанием q. Вычислите это для n, равного 4, 8, 16, и q, равного 10, 2, 8, 16.

1.10. Для перевода числа из десятичной системы счисления в любую другую может быть использован метод деления (см., например, учебник), основанный на делении переводимого числа на основание новой системы счисления по правилам десятичной арифметики. Докажите корректность такого алгоритма.

1.11. Для перевода правильных дробных чисел из десятичной системы в любую другую используется алгоритм, основанный на умножении переводимого числа на основание новой системы счисления. Докажите корректность этого алгоритма.

1.12. Дробное число, содержащее n цифр после дробной точки, записанное в позиционной системе счисления с основанием q, переводится в новую систему с основанием p. Сколько значащих цифр m мы должны сохранить в новом представлении числа, чтобы сохранить исходную точность? Проделайте вычисления для n, равного 4, 8, 16, и q, равного 10, 2, 8, 16.

1.13. Переведите следующие числа из десятичной системы счисления в двоичную и шестнадцатеричную:
a) 321_{10}; b) 127_{10}; c) 74_{10}.

1.14. Переведите следующие числа из шестнадцатеричной системы в двоичную, а затем в десятичную, выполните также перевод чисел непосредственно из шестнадцатеричной системы в десятичную:
a) $0D5_{16}$; b) 127_{16}; c) $2E1_{16}$; d) $0CAB_{16}$; e) $11C2_{16}$.

1.15. Переведите следующие числа из двоичной системы в восьмеричную, а затем в десятичную:
a) 10101_2; b) 10111101_2; c) 110101111_2; d) 11110011100_2.

1.16. Переведите следующие числа из десятичной системы в шестнадцатеричную напрямую (необходимо сохранить 3 цифры после запятой в дробном числе):
a) 1125_{10}; b) 53127_{10}; c) $0{,}216_{10}$.

1.17. Переведите числа из двоичной системы в десятичную:
a) $1{,}001_2$; b) $0{,}1011_2$; c) 101,1001.

1.18. Переведите числа из десятичной в двоичную систему счисления (оставить 4 значащих цифры после запятой):
a) $0{,}65_{10}$; b) $23{,}625_{10}$; c) $1{,}217_{10}$.

1.19. Запишите числа в коде BCD:
a) 27_{10}; b) 316_{10}; c) 4571_{10}.

1.20. Каковы преимущества и недостатки представления чисел в фиксированном формате и в формате с плавающей запятой?

1.21. Запишите числа в формате с плавающей запятой как 16-разрядные двоичные числа (для представления мантиссы использовать 8 бит, для порядка – также 8 бит):
a) $0{,}1010010_2$; b) $-11011{,}1010111_2$; c) $0{,}0001011011101_2$.

1.22. Вычислите диапазон представимых чисел, закодированных как в задаче **1.21**.

1.23. Запишите числа в дополнительном коде:
a) 1011; b) –1011; c) 1101; d) –1001; e) –1000; f) –0001; g) –111111; h) 1101; i) –1011101; j) 1011101.

1.24. Вычислите S = A+B, используя дополнительный код для отрицательных чисел, проверьте результат непосредственными вычислениями:
a) A = 11010, B = –10001; b) A = –11010, B = 10001;
c) A = 1010, B = –1101; d) A = –1010, B = 1000;
e) A = 1011010, B = –1001001.

1.25. Вычислите P = A · B:
a) A = –1011, B = 1101; b) A = 1011, B = –10111; c) A = –1010111, B = –101.

1.26. Переведите пары десятичных чисел в BCD формат, сложите эти числа, а затем переведите результат снова в десятичное число:
a) A = 597, *B* = 346; b) A = 2098, *B* = 3729; c) A = 3721, B = 5683.

Глава 2
ВЫЧИСЛИТЕЛЬНАЯ СИСТЕМА И АЛГОРИТМ ЕЕ РАБОТЫ

В главе рассматриваются основополагающие понятия: **архитектура цифровой вычислительной системы** (ВС), **микропроцессор**, **микроконтроллер.** Показана структура ВС (**процессор**, **память, устройства ввода/вывода**) и определено назначение составных частей этой системы. В качестве примера рассмотрен простейший микропроцессор и его архитектура. Особое внимание нужно обратить на то, что ВС включает в себя не только **аппаратную**, но и **программную часть** (это называют **дуализмом** ВС). Аппаратная и программная часть составляют взаимодополняющие и конкурирующие между собой части архитектуры ВС.

Цифровая вычислительная система является устройством, способным решать задачи и обрабатывать информацию под управлением программы, состоящей из команд. Аппаратные средства цифровой вычислительной системы – это комплекс цифровых логических схем, которые получают информацию из одного или более источников, обрабатывают ее и посылают результаты в одно или несколько мест назначения. Микропроцессорные цифровые ВС предоставляют недорогой способ решать сложные задачи по сохранению, передаче и обработке информации, они могут быть встроены в приборы и быть основой распределенных систем управления производственными процессами.

2.1. Структура вычислительной системы и назначение ее основных частей

Выполнение сложных математических расчетов всегда сталкивалось с многочисленными трудностями. Многие задачи вообще невозможно решить, выражая результат в элементарных функциях, поэтому разрабатывались методы численного решения математических задач. Применение этих методов также требует проведения огромного количества вычислений. Поэтому перед человечеством всегда стояла задача разработки мощных средств для автоматизации вычислений. С середины 40-х годов начала развиваться ветвь цифровых вычислительных машин. Используемые вначале как машины для вычислений, по мере развития элементной базы, на которой они строились, эти приборы превра-

тились в мощные вычислительные системы, способные решать не только вычислительные задачи, но и задачи по управлению и контролю.

Основой **цифровых вычислительных систем** являются **логические** цифровые **схемы**, основанные на элементах, принимающих два возможных фиксированных значения «0» и «1». Информация в таких схемах представлена в виде импульсных электрических сигналов, имеющих амплитуду выше некоторого уровня (логический 0) или ниже определенного уровня (логическая 1).

Любая компьютерная система является цифровой системой. Это означает, что она работает с цифровыми сигналами, то есть с сигналами, принимающими два возможных состояния. Для описания цифровых сигналов и действий с ними используется математический аппарат булевой алгебры. В этой алгебре используются логические переменные, которые могут принимать значения 0 и 1 («истина» и «ложь»). На множестве логических переменных определены три операции: «и» (логическое умножение), «или» (логическое сложение), «не» (инверсия).

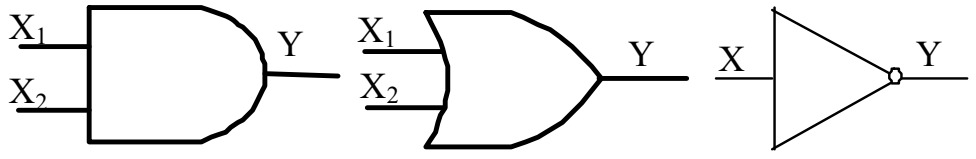
Инверсия обозначается $\bar{x}$ и определяется как $\bar{x} = 1$, если $x = 0$, и $\bar{x} = 0$, если $x = 1$. Логическое сложение определяется как $x_1 + x_2 = 0$, если $x_1 = 0$ и $x_2 = 0$ одновременно. Логическое умножение определяется как $x_1 \cdot x_2 = 1$, если $x_1 = 1$ и $x_2 = 1$ одновременно. Основные законы и соотношения алгебры логики приведены в табл. 2.1.

Таблица 2.1

Основные законы и соотношения булевой алгебры

<table>
<tr><td colspan="3">Фундаментальные законы</td></tr>
<tr><td>OR</td><td>AND</td><td>NOT</td></tr>
<tr><td>$X+0 = X$
$X+1 = 1$
$X+X = X$
$X+\overline{X} = 1$</td><td>$X\cdot 0 = 0$
$X\cdot 1 = X$
$X\cdot X = X$
$X\cdot \overline{X} = 0$</td><td>$X+\overline{X} = 1$
$X\cdot \overline{X} = 0$
$\overline{\overline{X}} = X$</td></tr>
<tr><td colspan="3">Ассоциативные законы
$(X_1+X_2)+X_3 = X_1+(X_2+X_3)$ $\quad (X_1X_2)\cdot X_3 = X_1\cdot (X_2\cdot X_3)$
Коммутативные законы
$X_1+X_2 = X_2+X_1$ $\quad X_1\cdot X_2 = X_2\cdot X_1$
Дистрибутивные законы
$X_1\cdot(X_2 + X_3) = X_1\cdot X_2+ X_1\cdot X_3$</td></tr>
<tr><td colspan="3">Законы Де Моргана
$\overline{X_1 \cdot X_2} = \overline{X_1} + \overline{X_2}$
$\overline{X_1 + X_2} = \overline{X_1} \cdot \overline{X_2}$</td></tr>
<tr><td colspan="3">Дополнительные соотношения
$X_1 + X_1 \cdot X_2 = X_1$ $\quad X_1 + \overline{X_1} \cdot X_2 = X_1 + X_2$
$(X_1+X_2)\cdot(X_1+X_3) = X_1+X_2\cdot X_3$</td></tr>
</table>

В цифровых системах используются логические элементы: OR, AND, NOT (инвертор) и др. Эти элементы используются для реализации логических схем. Для описания этих схем используются логические функции. Используя логические функции, можно проектировать логические схемы. Для представления логических функций используют алгебраические выражения, таблицы истинности и карты Карно. Примеры показаны на рис. 2.1.



$X_1 \cdot X_2 = Y$ $\quad$ $X_1 + X_2 = Y$ $\quad$ $\overline{X} = Y$

Вход		Выход
X_2	X_1	Y
0	0	0
0	1	0
1	0	0
1	1	1

a

Вход		Выход
X_2	X_1	Y
0	0	0
0	1	1
1	0	1
1	1	1

b

Вход	Выход
X	Y
0	1
1	0

c

Рис. 2.1. Символические обозначения, алгебраические выражения и таблицы истинности для логических элементов AND (a), OR (b) и инвертора (c)

Существует два типа логических схем: комбинационные и последовательностные. В комбинационных схемах выходной сигнал в любой момент времени однозначно определяется состоянием входных сигналов в этот момент времени. Для последовательностных логических схем выходные сигналы в данный момент времени также зависят от выходных сигналов в предыдущий момент, то есть обладают памятью предыдущих состояний. Для использования в электронных схемах выпускается широкий набор различных комбинационных схем в виде микросхем: различные логические элементы, шифраторы и дешифраторы, мультиплексоры и другие схемы, а также последовательностные схемы: триггеры, счетчики, регистры.

При построении цифровой ВС **реализован принцип программного управления**. Суть этого принципа заключается в следующем: цифровая схема построена таким образом, что может решать некоторый оп-

ределенный набор простых задач или выполнять определенные действия (**команды**), комбинируя эти действия в соответствии с заданным алгоритмом решения сложной задачи (**программа**), можно получить решение для очень широкого круга задач. Таким образом, цифровая ВС состоит из **аппаратных** и **программных средств**, которые выступают как неразрывное единство.

Аппаратными средствами называется относительно стабильная, неизменяемая часть вычислительной системы.

К аппаратным средствам относятся электронные схемы, из которых построена система, и схемы, обеспечивающие их работоспособность.

Программными средствами называется относительно изменчивая, непостоянная часть вычислительной системы.

К программным средствам относятся последовательности **команд,** реализующие решение задач и функции по обработке информации.

Хотя в конкретной вычислительной системе существует равновесие между аппаратными и программными средствами, в целом они являются проявлениями противоположных тенденций. Дело в том, что одна и та же задача может быть решена различными способами. Основная часть может быть перенесена на схемное решение (в принципе, и без применения программ), то есть на аппаратные средства, или, наоборот, на программные средства при минимальном использовании электронных схем. Это противоречие отражается в термине **дуализм ВС**. В большинстве случаев использование, в основном, аппаратных средств приводит к более быстродействующим системам, а использование программных средств – к более гибким и дешевым системам.

Вопрос о соотношении аппаратных и программных средств в ВС очень сложен. В универсальных ВС основная часть действий переносится на программные средства, аппаратные средства предоставляют при этом программисту широкие возможности для создания программ: большой набор команд, типов данных, с которыми может работать система, и **способов** их **адресации**. В последнее время для повышения быстродействия появилась тенденция к переносу на аппаратные средства большей доли работы. Это так называемые системы с сокращенной системой команд (RISC-системы). В них за счет повышения сложности электронных схем (и, соответственно, стоимости) увеличивается производительность.

Таким образом, ВС должна содержать электронную цифровую схему обработки информации (**процессор**), а также программу и данные, сохраняемые в устройстве хранения информации (**памяти**). К системе могут подключаться различные дополнительные приборы, обеспечивающие ввод или вывод информации (**внешние устройства**). Внеш-

ние устройства подключаются к системе через специальные электронные схемы, которые мы будем называть **интерфейсом**.

Типовая структурная схема цифровой ВС представлена на рис. 2.2.

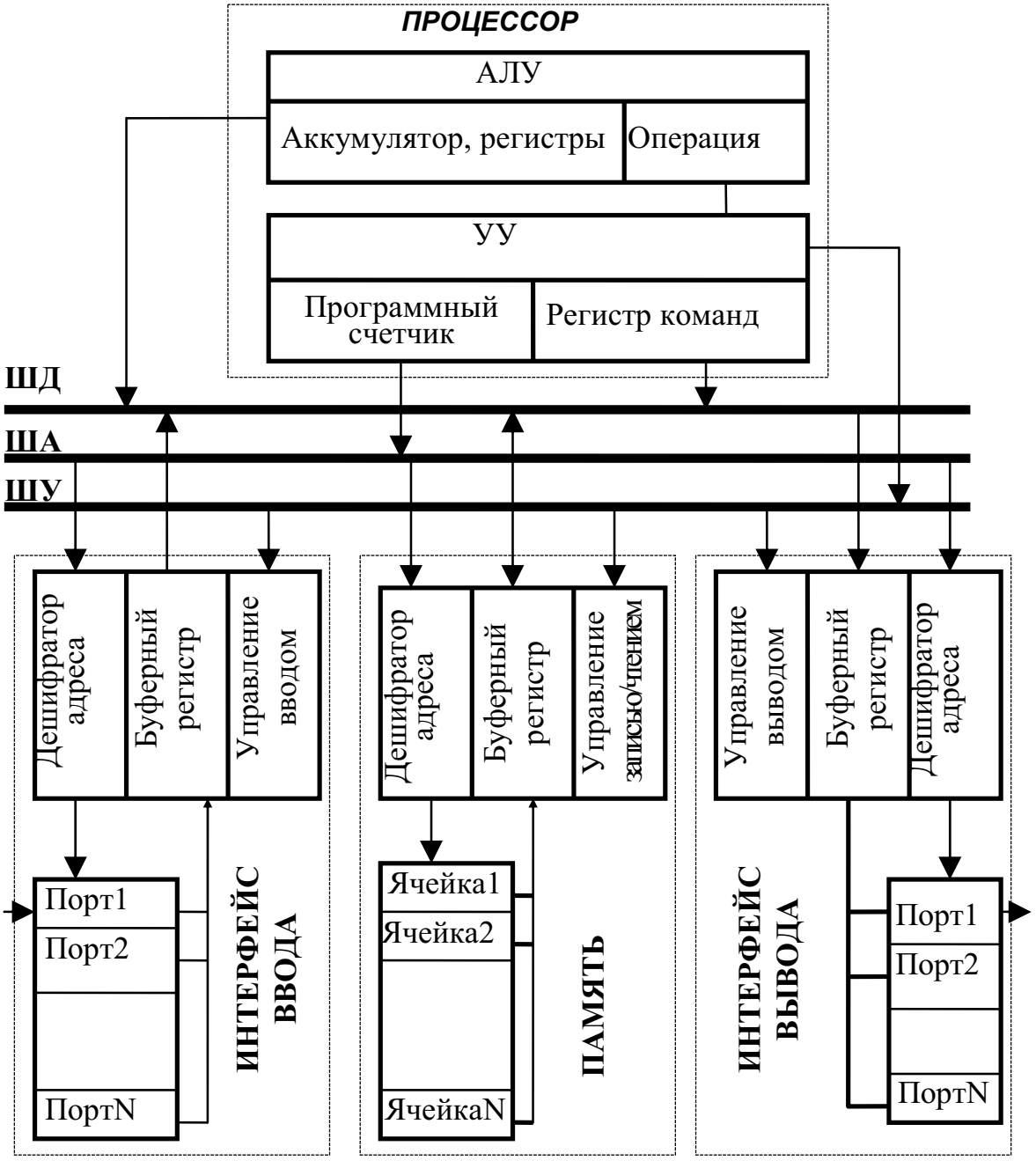


Рис. 2.2. Структурная схема вычислительной системы

Центральное устройство обработки, или процессор, является ядром всей системы. Процессор включает:

- управляющее устройство (УУ), которое координирует действия других элементов в компьютере;

- арифметико-логическое устройство (АЛУ), которое является цифровой логической схемой, выполняющей действия над данными, как задано управляющим устройством;
- комплект регистров, которые являются высокоскоростными ячейками хранения, используемыми для временного хранения данных, адресов и другой информации в пределах процессора.

АЛУ, регистры, память и устройства ввода/вывода создают информационный канал ВС.

Каждое АЛУ имеет определенный набор **типов данных**, которыми оно может манипулировать, и набор операций, которые оно может выполнять над этими данными. Большинство АЛУ поддерживают операции с двоичными целыми числами различных размеров. Некоторые также включают операции, чтобы манипулировать недвоичными числами и числами с плавающей точкой, а также различными нечисловыми данными. Типичные операции АЛУ включают:

- арифметические: сложение, вычитание, умножение, деление, сравнение;
- логические: И, ИЛИ, исключающее ИЛИ, инверсия, проверка битов;
- сдвиг и циклическое перемещение битов данных.

Управляющее устройство процессора выполняет выборку команд программы из памяти, декодирует команды и выполняет их, формируя управляющие сигналы. Управляющее устройство координирует все действия АЛУ, памяти и устройств ввода/вывода, непрерывно циклически выполняя ряд операций, которые заставляют команды выбираться из памяти и выполняться. Эта последовательность операций называется **циклом команды**.

Цикл команды включает пять основных шагов.

1. Команда выбирается из памяти в управляющее устройство процессора и помещается в **регистр команды**. Местоположение команды в памяти, ее **адрес**, определяется содержимым специального регистра, который называется **программным счетчиком**. Содержимое программного счетчика автоматически изменяется после выборки команды, указывая на адрес следующей команды.
2. Управляющее устройство декодирует команду, то есть определяет из кода команды, какие операции нужно выполнить.
3. Данные, участвующие в операции (они называются **операнды**), считываются из **портов ввода**, извлекаются из **памяти** или доступных **регистров** хранения, указанных в команде.
4. Выполняется операция над выбранными операндами.
5. Результат записывается в **регистр**, **ячейку памяти** или **порт вывода**.

Программные инструкции (команды) и данные хранятся и извлекаются из памяти ВС. Если одна и та же память используется для команд и данных, говорят, что ВС имеет **фоннеймановскую архитектуру**, в честь Джона фон Неймана, имя которого связывается с появлением первого компьютера, хранившего программу. Большинство универсальных ВС имеет такую архитектуру. ВС, которая использует память для команд и отдельную память для данных, имеет **гарвардскую архитектуру**. Многие **микроконтроллеры** входят в эту категорию. Кроме того, многие быстродействующие процессоры используют такую архитектуру, так как в этом случае команда и данные могут быть доступными параллельно. Структурные схемы систем с фоннеймановской и гарвардской архитектурой представлены на рис. 1.2.

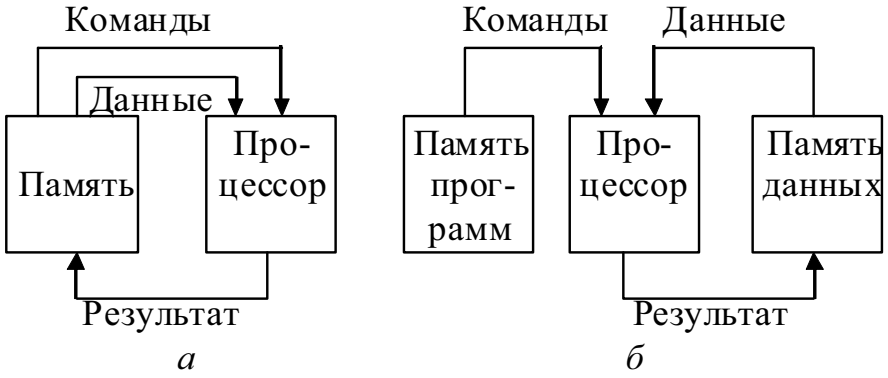


Рис. 2.3. Структура систем с фоннеймановской (а) и гарвардской архитектурой (б)

Дадим определение использованному выше термину «архитектура». Одно из определений слова «архитектура» (от греч. *architekton* – строитель) – это художественный характер, стиль постройки. С понятием архитектуры обычно отождествляется и понятие архитектоника, означающее выражение закономерности строения, органическое сочетание частей в одно стройное целое, их расположение, композицию. Говоря об архитектуре вычислительных систем, следует отметить, что, хотя это понятие и условно, оно прочно вошло в «вычислительный лексикон».

Под архитектурой микропроцессора понимают принцип его внутренней организации, общую структуру, конкретную логическую структуру отдельных устройств, совокупность команд и взаимодействие между аппаратной частью (устройствами, входящими в состав микропроцессора) и программами обработки информации системой, выполненной на основе микропроцессора. Иначе говоря, архитектуру микропроцессора определяют как совокупность его свойств и характеристик, рассматриваемую с позиции пользователя.

Рассмотрим кратко назначение и структуру других частей системы. Более подробно они будут рассмотрены в отдельных главах.

Основная память системы (внешняя по отношению к процессору) состоит из **постоянного запоминающего устройства (ПЗУ)** и **оперативного запоминающего устройства (ОЗУ)**.

Постоянное запоминающее устройство – это устройство, в котором хранится программа (и при необходимости совокупность констант), и доступ к нему возможен только для чтения данных. При выключении питания информация в ПЗУ сохраняется.

Содержимое ПЗУ не может быть стерто. Оно используется как память программы, составленной заранее изготовителем в соответствии с требованиями ее пользователей. В таких случаях говорят, что программа жестко «зашита» в запоминающем устройстве. Чтобы осуществить иную программу, необходимо применить другое ПЗУ или его часть. Из ПЗУ можно только выбирать хранимые там слова, но нельзя вносить новые, стирать и заменять записанные слова другими. Оно подобно напечатанной таблице: можно лишь считывать имеющиеся там числа, но заменять их или вносить новые невозможно. Помимо ПЗУ используются также ППЗУ и РППЗУ. Программируемое постоянное запоминающее устройство (ППЗУ) отличается от ПЗУ тем, что пользователь может самостоятельно запрограммировать ПЗУ (ввести в него программу) с помощью специального устройства – программатора, но только один раз (после введения программы содержимое памяти уже нельзя изменить).

Репрограммируемое постоянное запоминающее устройство (РППЗУ), называемое также стираемым ПЗУ, имеет особенность: хранимая информация может стираться несколько раз (при этом она разрушается). Иначе говоря, РППЗУ допускает перепрограммирование, осуществляемое с помощью программатора. Это облегчает исправление обнаруженных ошибок и позволяет изменять содержимое памяти.

Оперативное запоминающее устройство – это память, доступная как для чтения, так и для записи информации. При выключении электропитания такой памяти информация в ней разрушается.

Оперативное запоминающее устройство (ОЗУ), которое иначе называют запоминающим устройством с произвольной выборкой (ЗУПВ) или произвольным доступом (ЗУПД), служит памятью данных, подлежащих обработке, и результатов вычислений, а также программ, которые часто меняются.

Память (как ПЗУ, так и ОЗУ) построена как массивы однородных запоминающих ячеек. **Запоминающая ячейка** – минимальный доступный элемент памяти – представляет собой многоразрядный регистр (чаще всего 8-разрядный), в котором хранится число в двоичном коде.

Все ячейки памяти пронумерованы, чтобы можно было обратиться к каждой из них отдельно.

Уникальный (единственный) номер ячейки памяти называется ее адресом.

Емкостью адресуемой памяти называется наибольшее количество ячеек памяти, которое может быть обслужено данным процессором, и этот объем определяется **разрядностью адресной шины**. Характерное свойство памяти заключается в том, что время, требуемое для доступа к любой из ячеек памяти, не зависит от адреса этой ячейки.

ОЗУ допускает как запись, так и считывание слов. По отношению к этому запоминающему устройству приемлема аналогия с классной доской, на которой мелом записаны числа: их можно многократно считывать, не разрушая, а при необходимости – стереть число и записать на освободившемся месте новое. Следует иметь в виду, что информация, содержащаяся в ОЗУ, исчезает, стирается, если прерывается напряжение питания.

Информация передается между компьютером и внешним миром через различные устройства ввода/ вывода. Программы обычно передаются в память компьютера из таких внешних устройств, как магнитные или оптические устройства хранения. Данные, которые должны использоваться программой, могут передаваться в память с клавиатур, сканеров, магнитных дисков, аналого-цифровых преобразователей и других входных устройств. Программа может выводить данные в различные типы периферийных устройств. Электронно-лучевые трубки и жидкокристаллические панели часто используются, чтобы отобразить результаты вычислений, а различные типы принтеров используются чтобы распечатать результаты. Цифроаналоговые преобразователи, графопостроители, магнитные диски и другие устройства постоянного хранения информации также могут быть использованы в качестве выходных устройств.

Обмен данными между ВС и внешними устройствами происходит через интерфейс ввода/вывода. Интерфейсом мы назвали устройство сопряжения. Это упрощенное определение. В вычислительной технике этот термин имеет более широкое определение.

Под интерфейсом понимают совокупность электрических, механических и программных средств, позволяющих соединять модули системы между собой и с периферийными устройствами. Его составными частями служат аппаратные средства для обмена данными между узлами и программные средства – протокол, описывающий процедуру взаимодействия модулей при обмене данными.

В ВС применяют специальные интерфейсные схемы для сопряжения периферийных устройств с системой (они показаны в виде модулей

интерфейса ввода и интерфейса вывода). Основой этих схем являются порты ввода/вывода – схемы, спроектированные для обмена данными с конкретными периферийными устройствами. Порт – это адресуемый многорежимный буферный регистр ввода/вывода с выходными трехстабильными схемами, логикой управления и разъемом для подключения устройств ввода-вывода. Так же, как и ячейки памяти, порты пронумерованы, то есть имеют свой уникальный адрес.

Для многих ВС характерно несоответствие между относительно высокой скоростью обработки информации внутри микропроцессора и низкой скоростью обмена данными между модулями через интерфейс. Характеристики последнего в значительной степени определяют эффективность и производительность системы в целом.

Отдельные части ВС объединены между собой шинами.

Шиной *называют группу физических проводников (линий передачи), используемых для передачи информации в виде электрических сигналов, предназначенных для выполнения определенной функции (по одной линии на каждый передаваемый бит).*

Особенность структуры ВС заключается в магистральной организации связей между входящими в ее состав модулями. Связь осуществляется с помощью трех шин. По ним передается вся информация и сигналы, необходимые для работы системы. Эти шины соединяют процессор с памятью и интерфейсами ввода/вывода, в результате чего создается возможность обмена данными между процессором и другими модулями системы, а также передачи управляющих сигналов. Рассмотрим назначение и функции каждой из трех шин.

Шина данных (*ШД*) – это двунаправленная шина: по ней данные могут направляться либо в процессор, либо из него. При этом необходимо подчеркнуть, что невозможна одновременная передача данных в обоих направлениях. Эти процедуры разнесены во времени в результате применения временного мультиплексирования.

Шина адреса (*ША*) – это шина, по которой информация передается только в одном направлении – от микропроцессора к модулям памяти или ввода/вывода. По шине передаются многоразрядные двоичные числа. Каждое из них соответствует адресу определенной ячейки памяти или устройства ввода/вывода.

Шина управления (*ШУ*) служит для передачи сигналов, обусловливающих взаимодействие, синхронизацию работы всех модулей системы и внутренних узлов процессора. Одна часть линий шины управления служит для передачи сигналов, выходящих из процессора, а по другой части линий передаются сигналы к микропроцессору.

Достоинством шинной структуры является возможность подключения к ВС новых модулей, например нескольких блоков ОЗУ и ПЗУ для получения требуемой емкости памяти.

При обращении к конкретной ячейке памяти или к порту ввода/вывода с целью записи или считывания слова реализуется цикл шины. Выделяют **цикл чтения** шины (ввод данных в процессор) и **цикл записи** (вывода данных из процессора), а также несколько специальных циклов, которые определяются конкретной архитектурой процессора. В циклах чтения и записи процессор помещает на шину адреса код ячейки памяти или порта ввода/вывода, к которым происходит обращение. С адресной шины адрес ячейки/порта в форме двоичного кода подается на входы **дешифратора адреса**. При этом на выходе дешифратора появится сигнал, открывающий доступ к соответствующей ячейке или порту.

Цикл чтения заключается в выборке байта (или слова), хранимого в ячейке/порте указанного адреса (содержимое ячейки при этом остается неизменным). Для выполнения операции необходимо, чтобы в **схему управления чтением/записью** поступил с **шины управления** сигнал, устанавливающий режим ВЫВОД. Этот сигнал формируется устройством управления процессора. При этом **буферный регистр** пропускает байт из ячейки памяти, определенной адресом, на **шину данных**. При отсутствии управляющего сигнала выходы буферного регистра блокируются и связи ячейки памяти с шиной данных нет.

Цикл записи предполагает ввод нового байта (слова) в ячейку/порт. Если там имелся байт, то он стирается, и его место занимает новый байт. При выполнении цикла с шины управления поступает сигнал, устанавливающий режим ВВОД, буферный регистр открывается и становится возможным прием байта с шины данных в ячейку памяти по указанному адресу.

Вычислительные системы классифицируются согласно уровням интеграции компонентов, из которых они построены, и возможностям решения различных задач. Мини-ЭВМ является большой машиной, электрическая схема которой размещается на многих платах. Эти машины обладают высоким быстродействием и производительностью и используются в качестве главных компьютеров в больших системах управления или для решения крупных задач. МикроЭВМ представляет собой небольшую систему, используемую в системах управления на низком уровне или как персональный компьютер. Процессоры микроЭВМ строятся на основе больших интегральных схем называемых микропроцессорами.

Микропроцессор *– это одна или несколько цифровых интегральных схем, предназначенных для построения процессоров ВС.*

Термин «микропроцессор» подчеркивает размеры схем, но не их возможности. Современные микропроцессоры обладают параметрами, сравнимыми с параметрами больших компьютеров в недавнем прошлом.

Множество выпускаемых промышленностью универсальных микропроцессоров можно разделить по конструктивному признаку на две разновидности:

- однокристальные микропроцессоры с **фиксированной длиной (разрядностью)** слова и определенной **системой команд** (термины поясняются в следующем разделе);
- многокристальные (секционированные) микропроцессоры с наращиваемой разрядностью слова и микропрограммным управлением. Они составляются из двух и более БИС.

Внутренняя логическая организация однокристальных микропроцессоров в значительной степени подобна организации ЭВМ общего назначения. Это дает возможность при разработке микропроцессорной системы на основе однокристального микропроцессора опираться на методы проектирования и использования обычных ВС.

Структура многокристального микропроцессора, микропрограммное управление позволяют достичь гибкости в его применении, улучшить характеристики и сравнительно простыми средствами организовать распараллеливание отдельных машинных операций, что повышает производительность микроЭВМ, выполняемых на таких микропроцессорах.

Однако, хотя возможности многокристальных микропроцессоров выше, затраты на разработку аппаратных средств на их основе также существенно выше, чем на однокристальных. Поэтому при значительном прогрессе в разработке однокристальных микропроцессоров в микроЭВМ многокристальные микропроцессоры в настоящее время не используются.

Особую ветвь в микропроцессорной технике составляют в настоящее время однокристальные микроЭВМ (в зарубежной литературе они называются микроконтроллерами).

Микроконтроллер *– это законченная микроЭВМ, реализованная в виде единой микросхемы, включающей процессор, память, различные устройства ввода/вывода и интерфейсы внешних устройств.*

Так как реализовать составные части микроЭВМ в одной микросхеме в полном объеме затруднительно, большинство микроконтроллеров имеют меньшие возможности, чем микропроцессоры общего назначения. Микроконтроллеры первоначально использовались во встроенных управляющих системах, в которых компьютер встраивается непо-

средственно в такие продукты, как двигатели автомобиля, бытовая техника, оборудование связи, промышленные управляющие системы.

2.2. Архитектура однокристальных микропроцессоров

2.2.1. Параметры микропроцессоров и общая характеристика их архитектуры

Функциональные возможности микропроцессоров различны и определяются многими техническими характеристиками. К основным из них, которыми пользуются при сопоставлении и выборе микропроцессоров, относятся следующие характеристики:

1. Технология изготовления: p-канальная МОП (p-МОП), n-канальная МОП (n-МОП), комплементарная МОП (к-МОП), биполярная ТТЛ, ТТЛ с диодами Шоттки (ТТЛДШ), инжекционной интегральной логики ($И^2Л$), эмиттерно-связанной логики (ЭСЛ). Информация о технологии изготовления дает представление о потреблении энергии и среднем быстродействии микропроцессора.
2. Разрядность (4; 8; 16; 32) – длина информационного слова, которое может быть одновременно обработано микропроцессором. Она может быть фиксированной или наращиваемой (у многокристальных микропроцессоров).
3. Емкость адресуемой памяти. Характеризует возможности микропроцессора по взаимодействию с запоминающим устройством и определяется количеством ячеек памяти, которые может адресовать процессор. Емкость адресуемой памяти однозначно связана с разрядностью шины адреса микропроцессора.
4. Быстродействие. В справочниках наиболее часто его характеризуют продолжительностью выполнения одной операции (или числом операций «регистр–регистр» в секунду), а также тактовой частотой синхронизации микропроцессора и продолжительностью цикла простой команды.
5. Мощность потребления.
6. Питающие напряжения (число уровней, номиналы).
7. Конструктивные данные: габаритные размеры корпуса, число выводов.
8. Условия эксплуатации (интервал рабочих температур, относительная влажность воздуха, допускаемые вибрационные нагрузки и т. п.).
9. Надежность.
10. Стоимость.

Рассматривая характеристики, необходимо коснуться вопроса о поколениях микропроцессоров. В первые годы развития микропроцессоров в литературе различали их поколения. Однако в настоящее время

такая классификация практически уже не имеет смысла. Дело в том, что понятие «поколение» для микропроцессоров носит совсем иной характер, чем для больших машин. У ЭВМ каждое новое поколение имело более высокие основные технико-экономические характеристики по отношению к предыдущему поколению и поэтому вытесняло его. В микропроцессорной технике появление новой разработки не исключает применения ранее созданных микропроцессоров, а расширяет технические возможности применения микропроцессорных систем. Различные «поколения» микропроцессоров существуют совместно в течение продолжительного периода, часто взаимно дополняя (а не исключая) друг друга. Есть немало примеров того, что в одном устройстве работают два микропроцессора, моменты появления которых разделяет несколько лет (при условной классификации – два-три поколения).

Кроме технических характеристик, на возможность использования того или иного микропроцессора влияет его архитектура. Архитектура характеризует аппаратные средства данного микропроцессора и индивидуальна для каждого из них. Однако можно выделить общие моменты, присущие архитектурам различных микропроцессоров. Составляющие части архитектуры микропроцессора можно разделить на 2 группы. Первая группа определяет схемотехнику построения процессора и всей микропроцессорной системы в целом. В эту группу можно включить функциональную схему микропроцессора, набор управляющих сигналов, с которыми оперирует микропроцессор. Ко второй группе относятся аппаратные части архитектуры, определяющие программное обеспечение микропроцессорной системы. Это программная модель микропроцессора, его система команд, формат команды, способы адресации операндов, типы данных, с которыми может работать микропроцессор.

Определим подробнее составные части архитектуры микропроцессора. В качестве примера возьмем простой однокристальный микропроцессор *Intel* 8085 (1821ВМ85 по российским обозначениям).

В 1971 г. фирмой Intel были выпущены первые микропроцессоры (МП) – 4-разрядный *Intel* 4004 и 8-разрядный *Intel* 8008. А в 1974 г. – *Intel* 8080, который обрабатывает 8-разрядные слова и имеет 16-разрядные адресную шину и указатель стека. Его улучшенным вариантом является микропроцессор *Intel* 8085, построенный по КМОП-технологии. Он содержит генератор тактовых импульсов, систему управления и устройство определения приоритета прерываний, интеграция которых снижает число составляющих микропроцессорную систему интегральных схем. Микропроцессор *Intel* 8085 имеет единственный уровень питающего напряжения +5 В. Он использует те же команды, что и *Intel* 8080, что делает оба устройства совместимыми. Микропроцессор и системы на его основе

просты по организации и могут быть легко поняты, что делает его очень удобным для начального знакомства с микропроцессорами вообще.

2.2.2. *Функциональная схема и набор интерфейсных сигналов*

Функциональная схема определяет внутреннюю структуру микропроцессора и взаимодействие его частей. Функциональная схема микропроцессора 8085 приведена на рис. 2.4. В состав микропроцессора входят арифметико-логическое устройство, управляющее устройство и блок внутренних регистров. Кратко охарактеризуем эти узлы.

Арифметико-логическое устройство (АЛУ), служащее ядром микропроцессора, как правило, состоит из двоичного сумматора со схемами ускоренного переноса, сдвигающего регистра и регистров для временного хранения операндов. Обычно это устройство выполняет по командам несколько простейших операций: сложение, вычитание, сдвиг, пересылку, логическое сложение (ИЛИ), логическое умножение (И), сложение по модулю 2.

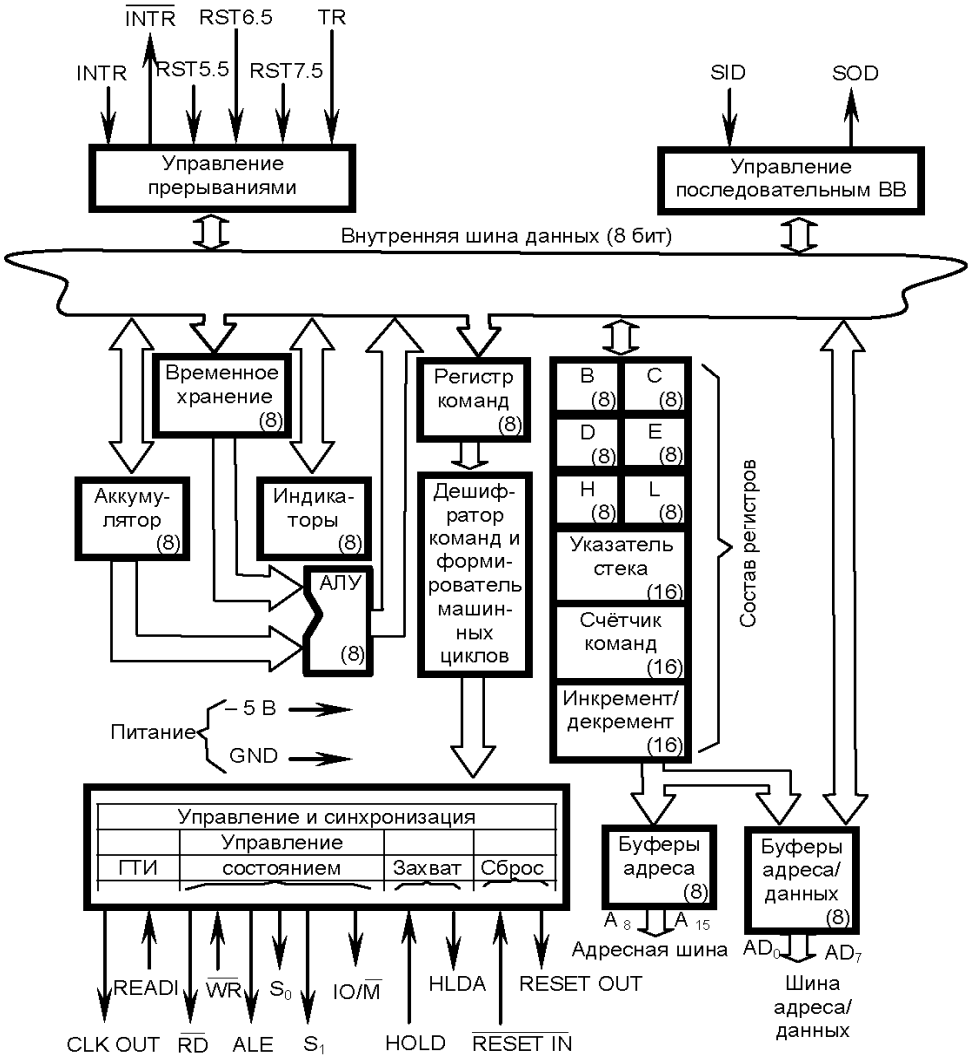


Рис. 2.4. Функциональная схема микропроцессора 8085

Регистром называется электронная схема для временного хранения двоичной информации (машинного слова). Ее строят на триггерах, общее число которых определяет разрядность регистра. Каждый триггер регистра используется для ввода, хранения и вывода одного разряда (1 или 0) двоичного числа. Разрядность регистра выбирают соответственно длине хранимого в нем слова. Регистры, которые служат только для ввода, хранения и вывода двоичной информации, называют регистрами хранения. От них отличаются сдвигающие регистры, которые помимо выполнения указанных функций позволяют осуществлять сдвиг двоичного числа вправо или влево (а иногда – в обоих направлениях). Если в регистр хранения вводят числа в параллельном коде, то есть одновременно во все триггеры, то ввод чисел в сдвигающий регистр часто производят в последовательном коде, подавая последовательно один разряд за другим, хотя возможен и ввод чисел в параллельном коде.

Операндом называют число или символ, участвующий в машинной операции. Типичным примером операнда, используемого при процедуре обработки данных микропроцессором, служит байт.

Устройство управления (УУ) «руководит» работой АЛУ и внутренних регистров в процессе выполнения команды. Согласно коду операции, содержащемуся в команде, оно формирует внутренние сигналы управления блоками микропроцессора. Адресная часть команды совместно с сигналами управления используется для считывания данных из определенной ячейки памяти (записи данных в ячейку). По сигналам УУ осуществляется выборка каждой новой, очередной команды.

Блок внутренних регистров, расширяющий возможности АЛУ, служит внутренней памятью микропроцессора: он используется для временного хранения данных и команд. Он также выполняет некоторые процедуры обработки информации. Обычно этот блок содержит регистры общего назначения и специальные регистры: регистр-аккумулятор, буферный регистр адреса, буферный регистр данных, счетчик команд, регистр команд, регистры стека, регистр признаков.

Регистры общего назначения (РОН), число которых может изменяться от 4 до 64, в значительной мере определяют вычислительные возможности микропроцессора. В микропроцессоре 8085 это регистры B, C, D, E, H, L. Их основная функция – хранение операндов, то есть данных, подлежащих обработке. Но они могут выполнять и роль специальных регистров. Все РОН доступны программисту, который их рассматривает как сверхоперативное запоминающее устройство. Иногда в технической документации к микропроцессору содержатся рекомендации по использованию РОН.

Регистр-аккумулятор (А), обычно называемый просто аккумулятором, предназначен для временного хранения операнда или промежуточного результата арифметических и логических операций, производимых АЛУ. При выполнении какой-либо операции с двумя операндами в этом регистре содержится один из используемых операндов, а после выполнения операции – ее результат. Разрядность аккумулятора равна разрядности информационного слова микропроцессора. Часто ввод и вывод всех данных в микропроцессоре производятся через аккумулятор. Встречаются микропроцессоры с двумя и более аккумуляторами, что позволяет повысить гибкость работы и эффективность решения задач.

Буферный регистр адреса – специальный регистр, служащий для приема и хранения адресной части исполняемой команды. Иначе говоря, в нем содержится до выдачи на адресную шину адрес ячейки внешней памяти или порта ввода/вывода. Возможное количество адресов (непосредственно адресуемых слов памяти) определяется разрядностью этого регистра. Так, в 16-разрядном регистре можно, изменяя нули и единицы отдельных разрядов двухбайтового слова, поместить $2^{16} = 65536$ адресов.

Буферный регистр данных служит для временного хранения выбранного из памяти слова перед выдачей его на внешнюю шину данных. Разрядность этого регистра определяется количеством байтов информационного слова микропроцессора (для хранения однобайтового слова необходим 8-разрядный регистр, двухбайтового слова – 16-разрядный).

Счетчик команд – счетчик, содержащий адрес ячейки памяти, в которой помещены байты выполняемой команды. Обычно команды определенной программы находятся в последовательно расположенных ячейках памяти: для однобайтовой команды число, указывающее адрес каждой последующей ячейки, на единицу больше числа, отмечающего адрес данной ячейки. Поэтому переход к следующей команде достигается увеличением числа, содержащегося в счетчике команд, на единицу (для возврата к предыдущей команде содержимое счетчика должно быть уменьшено на единицу). В ходе выполнения текущей команды, при передаче команды из памяти в микропроцессор, содержимое счетчика команд увеличивается на единицу и образуется адрес очередной команды. Вот почему говорят, что счетчик команд предназначен для хранения адреса команды, следующей в программе по порядку за выполняемой командой. Возможна ситуация, когда требуется после данной команды использовать команду, хранимую не в соседней, а в другой, например удаленной, ячейке памяти. Тогда по сигналу УУ в счетчик команд заносится адрес удаленной ячейки.

Регистр команд принимает и хранит код очередной команды, адрес которой находится в счетчике команд. По сигналу УУ в него передается из регистра хранимая там информация.

В состав практически любого микропроцессора входят средства работы со стеком. Название «стек» происходит от английского слова *stack*, что в дословном переводе означает «штабель» (дров), кипа (бумаг). В микропроцессорах стек представляет собой набор регистров, хранящих данные (адреса возврата, используемые при обращении к подпрограммам; состояния внутренних регистров). Этот набор организован таким образом, что слово данных выбирается по принципу «вошедший последним – выходит первым» – подобно тому, как из штабеля дров первым берут полено, положенное последним (в английском языке этот принцип определяется выражением *Last-In-First Out* и аббревиатура *LIFO* иногда встречается как название стековой памяти). При записи в стек очередного слова все находящиеся в нем слова смещаются на один регистр вниз (процесс такой записи называют «проталкиванием»). После выборки слова из стека оставшиеся слова сдвигаются на один регистр вверх (процесс считывания называют «выталкиванием» вверх). Стек может быть выполнен не только на внутренних регистрах микропроцессора, составляя его часть, но и может находиться во внешнем оперативном запоминающем устройстве, занимая там выделенную для него зону. В последнем случае стек получается более емким, однако для обращения к нему необходим специальный регистр – указатель стека. Указатель стека – регистр, служащий для хранения адреса последней занятой ячейки стека, которую называют вершиной. Содержащееся в регистре число указывает, где находится вершина стека. Когда в стек записывается очередное слово, то число в указателе стека соответственно увеличивается. Извлечение слова из стека сопровождается, наоборот, уменьшением числа, находящегося в указателе стека. Кроме такой процедуры предусматривается и возможность считывания без разрушения содержимого любой ячейки стека при неизменном числе, хранимом в указателе стека.

Регистр признаков (индикаторы), или F-регистр, представляет собой набор триггеров, называемых флажками. В зависимости от результатов операций, выполняемых АЛУ, каждый триггер устанавливается в состояние 0 или 1. Флажковые биты, определяющие содержимое регистра, индицируют условные признаки: **нулевого результата**, **знака результата**, **переполнения** и т. п. Эта информация, характеризующая состояние процессора, важна для выбора дальнейшего пути вычислений.

Для структуры микропроцессора характерно наличие внутренней шины данных, соединяющей между собой его основные части. Разрядность внутренней шины данных, то есть количество передаваемых по ней одновременно (параллельно) битов числа, соответствует разрядности слов, которыми оперирует микропроцессор. Следует иметь в виду, что по шине данных передаются не только обрабатываемые АЛУ слова, но и

командная информация. Следовательно, недостаточно высокая разрядность шины данных может ограничить состав (сложность) команд и их число. Поэтому разрядность шины данных относят к важным характеристикам микропроцессора – она в большой мере определяет его структуру. Шина данных работает в режиме двунаправленной передачи. Это означает, что по ней можно передавать слова в обоих направлениях, но, разумеется, не одновременно: требуется применение специальных буферных схем и мультиплексного режима обмена данными между микропроцессором и внешней памятью. Мультиплексный режим (от англ. multiple – многократный, множественный) – режим одновременного использования канала передачи большим числом абонентов с разделением во времени средств управления обменом. В современных микропроцессорах иногда в мультиплексном режиме используются также шины адреса и данных.

Интерфейсные сигналы – это сигналы, которые вырабатывает микропроцессор на своих внешних выводах и которые используются составными частями микропроцессорной системы. Эти сигналы формируют внешние шины микропроцессора: адреса, данных и управления. Шина управления обычно содержит сигналы синхронизации записи и чтения информации по шине данных, причем либо отдельно для памяти и для портов ввода/вывода, либо включается дополнительный сигнал, показывающий, откуда будет произведена передача. Если используется мультиплексирование шин, вводятся специальные сигналы, облегчающие демультиплексирование и формирование отдельных внешних шин. В число интерфейсных сигналов включают также сигналы, обеспечивающие поддержку специальных режимов работы процессора: режим прерываний и режим прямого доступа к памяти.

Режим прерываний – это специальный режим выполнения подпрограмм, когда переход к подпрограмме инициируется не самой программой, а специальным входным сигналом микропроцессора, возбуждаемым внешним устройством.

В режиме прямого доступа к памяти передача информации между внешним устройством и памятью происходит без участия процессора, что приводит к значительному повышению скорости передачи.

Подробнее эти режимы освещены в главе, где обсуждаются вопросы ввода/вывода информации.

2.2.3. Программная модель и система команд

Программная модель показывает программно доступные части микропроцессора и микропроцессорной системы. Программная модель позволяет программисту проще ориентироваться в структуре микропроцессорной системы, не отвлекая его внимания на ненужные при программи-

ровании детали. На рис. 2.5 приведена программная модель микропроцессорной системы на основе 8-разрядного микропроцессора 8085. На схеме показан микропроцессор и его программно-доступные части: АЛУ, аккумулятор (A), регистр флагов (F), регистры общего назначения (B,C,D,E,H,L), программный счетчик (PC), регистр указатель стека (SP). Подробно описаны биты регистра флагов: S – бит знака (повторяет старший бит аккумулятора), Z – флаг нулевого результата (устанавливается, если в аккумуляторе 0), AC – флаг вспомогательного переноса (из младшей тетрады в старшую), P – флаг четности, CY – флаг переноса (из старшего значащего бита). На схеме показаны также память(M) и порты ввода/вывода(I/O).

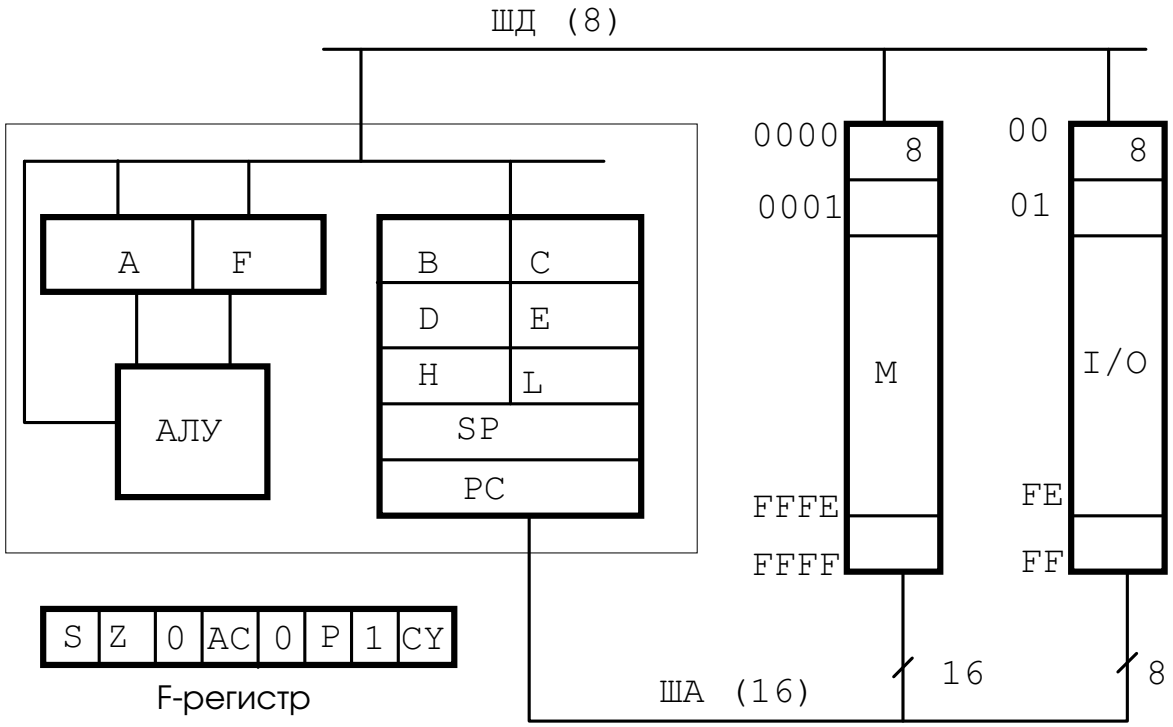


Рис. 2.5. Программная модель микропроцессорной системы на основе Intel 8085

Микропроцессор – это программно-управляемое устройство, все действия в котором выполняются с помощью команд. Системой команд называется совокупность всех элементарных команд, выполняемых микропроцессором.

Команды из системы команд микропроцессора принято делить на группы. Для большинства микропроцессоров можно выделить шесть основных групп команд: передача данных, арифметические операции, логические операции, операции сдвига, операции передачи управления и управление микропроцессором.

Группа команд передачи данных загружает регистры микропроцессора данными из ячеек памяти, сохраняет содержимое регистров в па-

мяти, переносит данные из одного регистра в другой или из одной ячейки памяти в другую. К этой же группе обычно относятся команды загрузки данных в стек и извлечение данных из стека.

Арифметические команды, такие как команды сложения, вычитания, умножения и деления, выполняют двоичные арифметические операции над целыми двоичными операндами. Не все микропроцессоры поддерживают все четыре функции. Например, многие микроконтроллеры обеспечивают только команды сложения и вычитания и вынуждают программистов писать короткие программы, чтобы выполнить умножение или деление. Многие микропроцессоры включают команды инкремента и декремента двоичных чисел.

Инкрементом называется увеличение содержимого регистра или ячейки памяти на единицу. Соответственно, декремент – это уменьшение содержимого регистра или ячейки памяти на единицу.

Микропроцессоры, которые поддерживают форматы десятичных чисел, имеют специальные инструкции для выполнения арифметических операций с двоично-десятичными числами. Некоторые микропроцесссоры имеют непосредственные команды сложения и вычитания двоично-десятичных чисел, однако большинство используют для сложения обычные команды двоичного сложения с последующей операцией коррекции результата. Очень часто необходимо выполнять арифметические операции над многоразрядными двоичными числами, то есть числами, разрядность которых больше разрядности микропроцессора. В этом случае используются операции сложения и вычитания, использующие флаг переноса.

Группа логических команд выполняет булевы операции «И», «ИЛИ», «Исключающее ИЛИ» над битами операндов. Эти операции дают микропроцессору возможность устанавливать, сбрасывать, инвертировать или проверять отдельные биты или группы битов в ячейках памяти, регистрах или портах ввода/вывода. Команда «И» позволяет сбрасывать выбранные биты слова в нуль. Второй операнд тогда должен содержать набор битов, называемых маской, имеющей нуль в каждой позиции, которая должна быть сброшена в нуль, и единицу в каждой позиции, которая должна остаться неизменной. Подобные маски могут быть созданы для команды «ИЛИ» (установка выбранных битов в единицу) и для команды «Исключающее ИЛИ» (инвертирования выбранных битов). Многие устройства ввода/вывода имеют специальный регистр состояния, в битах которого отражается способность устройства выполнять операции. Команда «И» может выделить отдельные биты в регистре состояния, чтобы определить, находятся они в состоянии нуля или единицы.

Команды сдвига и вращения сдвигают биты операнда вправо или влево. Это может быть использовано для выделения битов или совмещения нескольких битов в операнде, для перевода данных из параллельной в последовательную форму, для выполнения операций умножения и деления на числа, равные степеням двойки. Операция логического сдвига сдвигает биты вправо или влево на один бит, свободный бит замещается нулем. Для чисел без знака это эквивалентно делению или умножению на два. Команда арифметического сдвига вправо выполняет деление на два над двоичными числами в дополнительном коде путем сохранения старшего знакового бита, в то время как операнд сдвигается на один бит вправо. Некоторые микропроцессоры позволяют выполнять операции многократных сдвигов за одну команду. Команды циклических сдвигов (вращений) выполняют операцию сдвига с заполнением освобождающегося бита битом, выдвигаемым из начала или конца операнда. Используются также операции циклических сдвигов через бит переноса. В этих операциях бит переноса выступает как дополнительный бит операнда.

Нормальный порядок выполнения команд в программе – это последовательное их выполнение в направлении возрастания адресов. Чтобы изменить этот порядок, используются специальные команды ветвлений (передачи управления). Эти команды прерывают обычный порядок выполнения команд и позволяют перейти в произвольное место программы, обеспечивая тем самым программирование разветвляющихся и циклических алгоритмов. В систему команд микропроцессора обычно включают команды безусловных и условных переходов. Команды безусловных переходов могут передавать управление в любое место памяти путем загрузки программного счетчика новым адресом команды. Команды условных переходов перед выполнением перезагрузки командного счетчика проверяют состояние флагового регистра микропроцессора. Если проверяемое условие истинно, то осуществляется переход, в противном случае программа продолжает обычное выполнение. В современных микропроцессорах, кроме того, есть специальные команды организации циклов. В этих командах один из регистров микропроцессора используется в качестве счетчика и при каждом прохождении цикла декрементируется, а при достижении нулевого состояния цикл прекращается. Для поддержки модульного программирования в систему команд вводятся команды работы с подпрограммами. Команды перехода к подпрограммам обычно записывают текущее состояние программного счетчика в стек, чтобы сохранить адрес команды, следующей за командой вызова. Команда возврата из подпрограммы извлекает из

стека сохраненный адрес и загружает его в программный счетчик. Тем самым осуществляется возврат к прерванной программе.

Команды управления микропроцессором позволяют изменять режимы работы микропроцессора. Набор команд микропроцессора 8085 приведен в приложении.

2.2.4. Формат команды, способы адресации операндов

Команда должна сообщить процессору, какие действия он должен выполнить, где взять данные для выполнения операций и куда поместить результат. Так как микропроцессор – это цифровая логическая схема, любая команда должна быть представлена в виде двоичного числа, закодированного в определенном формате. В этом числе закодированы ответы на все вышеприведенные вопросы. В соответствии с этим принято делить код команды на поля. Эти поля включают: поле кода операции, которое определяет, что должна сделать команда, и поле операндов, которые показывают, где берутся данные для выполнения операции. В некоторых случаях конкретные значения данных могут быть составной частью кода команды.

Команда может состоять из одного, двух или большего количества байтов, последовательно расположенных в памяти. Первый байт команды содержит код операции. Считанный в начале интервала выполнения команды, называемого циклом команды, ее первый байт поступает из памяти по внутренней шине данных в регистр команд, где хранится в течение всего цикла. Дешифратор кода операции дешифрирует содержимое регистра команд – определяет характер операции и адреса операндов. Эта информация передается в УУ, которое вырабатывает управляющие сигналы, направляемые в блоки микропроцессора, которые участвуют в выполнении данной команды. Возможен случай, когда код операции несет всю необходимую информацию для выполнения команды. Тогда выполнение начинается сразу после считывания первого байта команды. Если же в команде содержится более одного байта, то остальные байты, несущие информацию об адресе ячейки памяти, где хранятся данные, передаются либо в буферный регистр адреса, либо в один из РОН. Только после завершения всей процедуры считывания команды или, иначе говоря, после получения полной информации о местонахождении операндов и о том, какая операция должна выполняться над ними, начинается обработка.

Чтобы выбрать операнд из ячейки памяти или записать результат в память, адрес этой ячейки должен быть определен в команде.

Механизм указания адреса операнда в команде и способ получения (вычисления) исполнительного адреса данного, используемого в команде, называется способом (методом) адресации операндов.

В современной микропроцессорной технике разработано и используется большое количество различных способов адресации операндов, обеспечивающих гибкое использование данных программистом.

В команде, выполняемой микропроцессором, данные могут указываться либо непосредственно значениями, либо адресами этих значений. В первом случае **непосредственно в команде** уже указано то значение данного, которое нужно использовать в операции. Такой метод указания операнда носит название **непосредственного**.

В большинстве микропроцессоров адрес памяти может быть указан **прямо** или **косвенно**. При **прямой адресации** адрес ячейки находится **непосредственно в команде**. Прямая адресация операндов очень удобна, однако у нее есть ряд недостатков, существенно ограничивающих применимость этого способа. Во-первых, длина команды с таким способом адресации велика, особенно для современных микропроцессоров с большой адресуемой памятью. Во-вторых, этот способ делает программу неперемещаемой в памяти, то есть перенос программы в другую область памяти (с другими адресами) делает ее неработоспособной.

Однако часто программисту не бывает известно, по какому конкретному адресу будет расположено данное. В этом случае используется **косвенная адресация**. При **косвенной адресации** адрес ячейки памяти находится в **регистре** микропроцессора или в какой-либо другой **ячейке памяти**. При косвенной адресации операнд, специфицированный в команде, содержит адрес того регистра или ячейки памяти, где находится нужное данное. Такой способ адресации делает программу перемещаемой в памяти, команда с таким способом адресации короткая, но время ее выполнения увеличивается, так как необходимо выполнять промежуточные операции по выбору данных. Многие процессоры поддерживают варианты косвенного метода адресации – **автоинкрементный** и **автодекрементный**. В этих методах данные выбираются так же, как и в косвенном методе, но **регистр**, указанный в команде и используемый в качестве указателя адреса данного, после выполнения команды **автоматически увеличивается** или **уменьшается** на **1**.

Широко распространены в микропроцессорной технике **базовый** и **индексный** методы адресации (в том числе может быть их комбинация **базово-индексный** метод). Данные очень часто сохраняются в виде таблиц, списков или аналогичных структур. Вышеназванные способы по-

зволяют микропроцессору эффективно работать с такими данными. Адрес данного в таких структурах можно разбить на две части: базу (начальный адрес всего массива) и смещение или индекс (адрес относительно базы). В **базовом** методе **адрес базы** хранится **в регистре** в неизменном виде, а смещение изменяется. В команде приводятся указание на регистр, в котором хранится адрес базы, и смещение в виде непосредственного операнда. В **индексном** методе, наоборот, **индекс хранится в регистре**, а базовый адрес указывается непосредственно в команде.

В командах передачи управления широко применяется **относительный** метод адресации. В этом методе исполнительный адрес получается как **сумма** содержимого **программного счетчика** и **смещения**, непосредственно приводимого в команде. В команде **смещение** обычно указывается в виде числа **со знаком** в **дополнительном коде**.

МП *Intel* 8085 использует пять способов адресации:

- неявная (код операции сам по себе показывает, какие операнды должны быть использованы);
- регистровая;
- непосредственная;
- прямая;
- косвенная регистровая.

Дополнительно имеется комбинированный способ, который использует сочетания различных способов адресации.

Примеры решения задач

1. Нарисуйте схему цифровой системы, имеющей 4 входных и 1 выходной сигнал. Выходной сигнал должен принимать значение 1, если по крайней мере 3 входных сигнала принимают значение 1.

Решение

Строим таблицу истинности для функции выходного сигнала (табл. 2.2).

По теореме Шеннона, любая функция N логических переменных может быть записана как логическая сумма логических произведений каждой комбинации входных переменных и соответствующего им значения выходной переменной. Применяя эту теорему, получим логическую функцию:

$$Y = X_1 \cdot X_2 \cdot X_3 \cdot \overline{X}_4 + X_1 \cdot X_2 \cdot \overline{X}_3 \cdot X_4 + X_1 \cdot \overline{X}_2 \cdot X_3 \cdot X_4 + X_1 \cdot X_2 \cdot X_3 \cdot X_4 + \overline{X}_1 \cdot X_2 \cdot X_3 \cdot X_4 .$$

Таблица 2.2

Таблица истинности (пример 1)

Входы				Выход
X_4	X_3	X_2	X_1	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Далее, используя карту Карно, минимизируем эту функцию (рис. 2.6).

x_3x_4 \ x_1x_2	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	0	1	1	1
10	0	0	1	0

Рис. 2.6. Карта Карно

Получаем следующее выражение:

$$Y = X_1 \cdot X_2 \cdot X_4 + X_1 \cdot X_2 \cdot X_3 + X_2 \cdot X_3 \cdot X_4 + X_1 \cdot X_3 \cdot X_4 .$$

Приведенное выражение можно реализовать, используя 4-входовый логический элемент OR и 4 трехвходовых элемента AND (рис. 2.7).

2. Повторите предыдущий пример, если все входные сигналы одновременно не могут принимать 1 значение.

Решение

В этом случае комбинация входных сигналов $X_1X_2X_3X_4$ не определена. Для неопределенных или невозможных комбинаций мы можем использовать любое наиболее подходящее значение. Если мы выберем это значение равным 1, то мы можем использовать предыдущее решение.

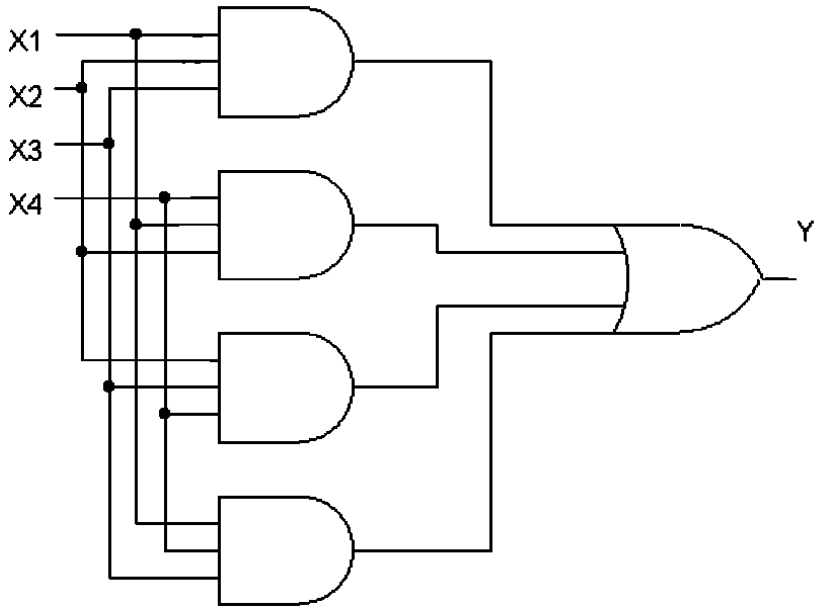


Рис. 2.7. Схема для примера 1

3. Разработайте схему контроля паритета. Нарисуйте блок-схему, в которой используется контроль паритета при передаче информации по линии связи.

Решение

В цифровых системах часто выполняется операция определения, является ли сумма бит в байте нечетной (это называется нечетный паритет) или четной (четный паритет). Для построения схемы, выполняющей такую операцию, можно использовать элементы «Исключающее ИЛИ». Выходной сигнал для схемы «Исключающее ИЛИ» равен 1, только в том случае, когда входные сигналы имеют противоположное значение. Другими словами выходной сигнал равен 1, если сумма входных бит равна 1. На рис. 2.8 приведена схема 4-разрядного контроллера паритета. Из рисунка можно заключить, что Z = 1 (или Y = 0), если сумма входных битов X_1, X_2, X_3, и X_4 нечетна. Теперь, если вход P' заземлен (P' = 0), то P = 0 – для нечетного паритета и P = 1 – для четного паритета.

Представленная на рис. 2.8 система может быть использована не только для контроля паритета, но и для генерации бита паритета P. Независимо от паритета 4-битового входного слова, паритет 5-битового кода X_1, X_2, X_3, X_4 и P нечетный. Это утверждение следует из того факта, что

если сумма битов X_1, X_2, X_3, и X_4 нечетна (четна), то P равно 0 (1), и, соответственно, сумма X_1, X_2, X_3, X_4 и P всегда нечетна. Использование дополнительного бита паритета – эффективный путь повышения достоверности передаваемой информации. На рис. 2.9 бит паритета P_1 генерируется и передается вместе с 4-разрядным словом информации. В приемнике определяется паритет полученного 5-разрядного слова. Если выход P_2 равен 0, принимается, что при передаче данных не произошло ошибок. Если P_2 = 1, то это показывает, что передача произошла с ошибками. Нужно заметить, что одиночным контролем паритета можно определить ошибки, вызванные повреждением нечетного количества бит.

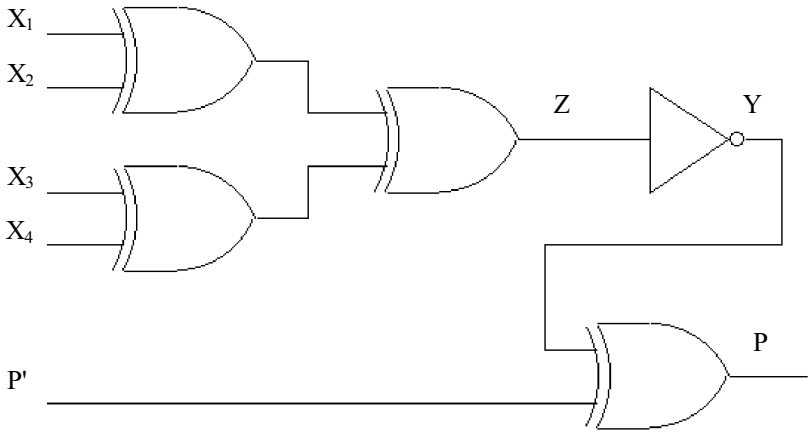


Рис. 2.8. Контроллер паритета

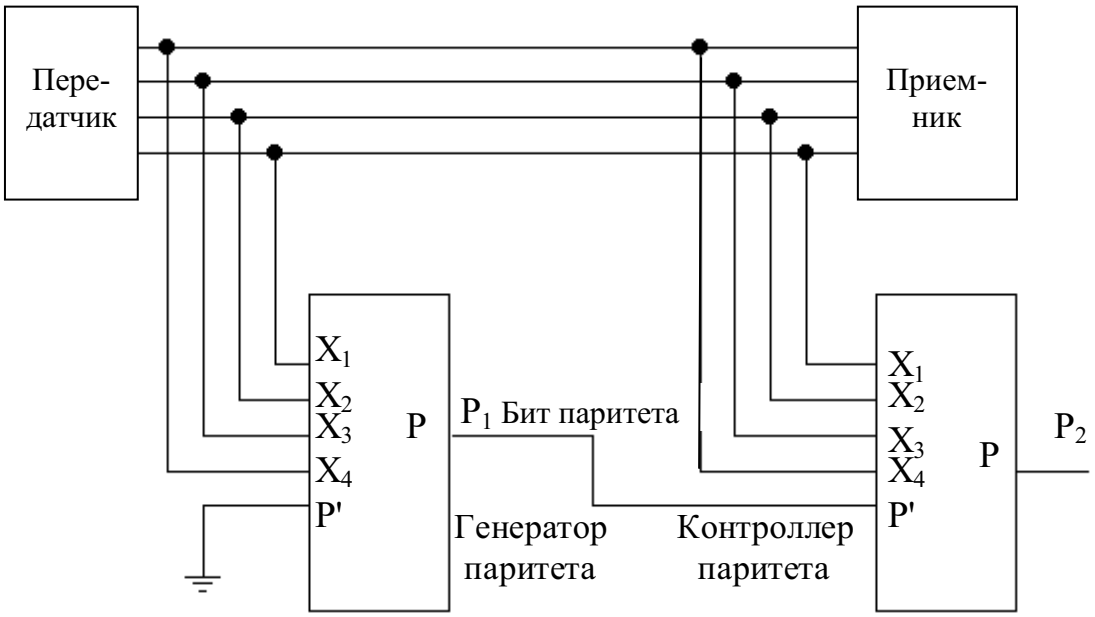


Рис. 2.9. Передача данных с контролем паритета

4. Используя мультиплексор, реализуйте логическую функцию:

$$Y = \overline{A}\overline{B}\overline{C}\overline{D} + A\overline{B}\overline{C}D + \overline{A}B\overline{C}D + \overline{A}B\overline{C}\overline{D} + \overline{A}\overline{B}CD + A\overline{B}C\overline{D} + \overline{A}BCD + \overline{A}BC\overline{D}\,.$$

Решение

Мультиплексор выполняет функцию выбора одного входа из N входов данных и передачу с выбранного входа на один выход. Таким образом, мультиплексор имеет N входов данных, $\log_2 N$ управляющих входов (определяют выбираемый вход) и выход данных. Мультиплексор можно использовать для реализации логических функций. Покажем это для нашей задачи. Составим таблицу истинности для задачи (табл. 2.3). Объединим в таблице соседние строки попарно. Каждая пара строк будет ассоциироваться с одним входом мультиплексора.

Таблица 2.3

Таблица истинности (пример 4)

A	B	C	D	Y	Входы данных
0 0	0 0	0 0	0 1	1 0	$\overline{D}$ (D_0)
0 0	0 0	1 1	0 1	0 1	D (D_1)
0 0	1 1	0 0	0 1	1 1	1 (D_2)
0 0	1 1	1 1	0 1	1 1	1 (D_3)
1 1	0 0	0 0	0 1	0 1	D (D_4)
1 1	0 0	1 1	0 1	1 0	$\overline{D}$ (D_5)
1 1	1 1	0 0	0 1	0 0	0 (D_6)
1 1	1 1	1 1	0 1	0 0	0 (D_7)

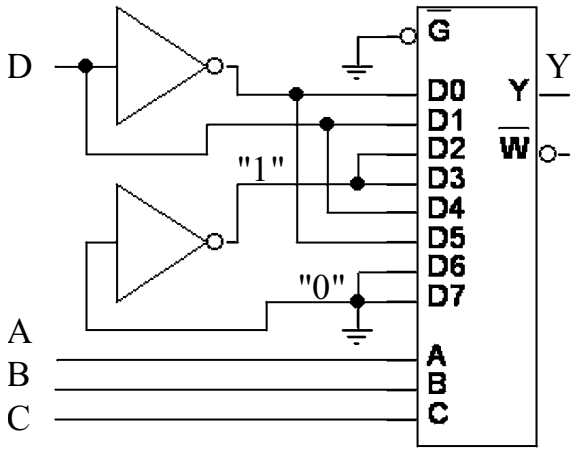


Рис. 2.10. Реализация логической функции (пример 4)

Для реализации будем использовать мультиплексор «8 входов на 1 выход» (например, микросхему 74AS151) (рис. 2.10). Входы A, B, C – управляющие, D0…D7 – входы данных мультиплексора.

Вопросы и задания для повторения

Структура компьютерной системы и назначение основных частей

2.1. Дайте определение для элемента «И-НЕ» и приведите таблицу истинности для него.

2.2. Дайте определение для элемента «ИЛИ-НЕ» и приведите таблицу истинности для него.

2.3. Покажите, что функция «И» может быть выполнена с использованием операций «ИЛИ» и «НЕ», а операция «ИЛИ» с использованием операций «И» и «НЕ». Запишите логические выражения для этих функций.

2.4. Реализуйте функции «И», «ИЛИ» и «НЕ», используя только элементы «И-НЕ», а затем только «ИЛИ-НЕ». Напишите соответствующие логические выражения.

2.5. Дайте определение для логического элемента «исключающее ИЛИ». Проверьте, что следующие логические выражения описывают функцию «исключающее ИЛИ»:

a) $X_1 \cdot \overline{X}_2 + \overline{X}_1 \cdot X_2$;

b) $(X_1 + X_2) \cdot (\overline{X}_1 + \overline{X}_2)$;

c) $(X_1 + X_2) \cdot \overline{(X_1 \cdot X_2)}$.

Нарисуйте логические схемы, реализующие данные выражения.

2.6. Реализуйте логическую функцию, используя элементы «И», «ИЛИ» и «НЕ».

$$Y = \overline{X}_1 \cdot \overline{X}_2 \cdot \overline{X}_3 \cdot \overline{X}_4 + X_1 \cdot \overline{X}_2 \cdot \overline{X}_3 \cdot \overline{X}_4 + X_1 \cdot X_2 \cdot \overline{X}_3 \cdot \overline{X}_4 + \overline{X}_1 \cdot X_2 \cdot \overline{X}_3 \cdot \overline{X}_4 +$$
$$X_1 \cdot \overline{X}_2 \cdot X_3 \cdot \overline{X}_4 + X_1 \cdot X_2 \cdot X_3 \cdot \overline{X}_4 + X_1 \cdot X_2 \cdot X_3 \cdot X_4 + \overline{X}_1 \cdot \overline{X}_2 \cdot \overline{X}_3 \cdot X_4 +$$
$$X_1 \cdot \overline{X}_2 \cdot \overline{X}_3 \cdot X_4 + X_1 \cdot X_2 \cdot \overline{X}_3 \cdot X_4 + \overline{X}_1 \cdot X_2 \cdot \overline{X}_3 \cdot X_4.$$

2.7. Повторите решение предыдущей задачи, используя элементы: a) «И-НЕ»; b) «ИЛИ-НЕ».

2.8. Постройте таблицу истинности для двоичного полусумматора. Полусумматором называется логическая схема для арифметического суммирования бит. Он имеет 2 входа (2 1-битовых числа) и 2 выхода (D – результат сложения и C – перенос). Напишите логические выражения для D и C.

2.9. Постройте таблицу истинности для полного сумматора (два входных бита и вход переноса). Запишите логические выражения для выходов (сумма и перенос). Реализуйте схему полного сумматора.

2.10. Нарисуйте схему для сложения 2-, 4-битовых двоичных чисел, используя одноразрядные полные двоичные сумматоры.

2.11. Что такое дешифратор? Приведите таблицу истинности, логические выражения для выходных сигналов и логическую схему, реализующую 3-входовый двоичный дешифратор.

2.12. Что такое мультиплексор? Реализуйте логическую схему мультиплексора, имеющего 4 входа и 1 выход.

2.13. Что такое приоритетный шифратор? Приведите таблицу истинности для приоритетного шифратора «4 на 2».

2.14. Что такое триггер-защелка? Как его можно построить с использованием логических элементов. Покажите, что полученная схема имеет 2 устойчивых состояния.

2.15. Приведите таблицы истинности для триггеров:
a) R–S; b) J–K; c) D; d) T.
Нарисуйте схемные обозначения для каждого типа триггеров.

2.16. Покажите, как можно использовать J–K триггер в качестве D-триггера и T-триггера. Приведите соответствующие таблицы истинности.

2.17. Что такое регистр? Как он работает?

2.18. Что такое «состязания»? Какие меры принимаются в цифровых схемах для исключения этого явления?

2.19. Объясните, как сдвиговый регистр может использоваться для преобразования параллельных данных при передаче в последовательные и наоборот. Нарисуйте временные диаграммы процессов.

2.20. Нарисуйте схему 4-разрядного двоичного счетчика-делителя на основе: a) J–K; b) D триггеров. Нарисуйте временные диаграммы сигналов.

2.21. Объясните, как можно построить счетчик с произвольным коэффициентом пересчета.

2.22. Что такое постоянное запоминающее устройство (ПЗУ)? Что хранят в ПЗУ? Как еще можно использовать ПЗУ?

2.23. Что такое масочное ПЗУ, однократно программируемое ПЗУ, перепрограммируемое ПЗУ?

Архитектура микропроцессора

2.24. Дайте определения терминам: микропроцессор, микропроцессорная система, микропроцессорный комплект, микрокомпьютер, микроконтроллер.

2.25. Чем микропроцессор отличается от микроконтроллера? Какие функции должны быть реализованы в микропроцессоре и в микроконтроллере?

2.26. Какие характеристики микропроцессора в основном определяют эффективность вычислительной или управляющей системы на основе микропроцессора?

2.27. Дайте определения терминам: аппаратные средства, программное обеспечение, архитектура, процессор (CPU), порт, интерфейс.

2.28. Что такое шина? Какие типы шин используются в микропроцессорной системе?

2.29. Что такое команда, система команд, операнд, код операции, цикл выполнения команды?

2.30. Является ли утверждение, что в микропроцессоре должно быть реализовано по возможности большее количество команд, правильным?

2.31. Нарисуйте функциональную схему микропроцессора. Перечислите его основные части.

2.32. Перечислите внутренние регистры микропроцессора. Что хранится в каждом из этих регистров?

2.33. Что такое стек? Какие аппаратные средства нужны для организации в памяти стека?

2.34. Что такое программная модель микропроцессора? Перечислите основные программно-доступные средства микропроцессора.

2.35. Перечислите основные группы команд микропроцессора.

2.36. Что такое формат команды? Объясните, как микропроцессор получает необходимую информацию из команды.

2.37. Что такое способ адресации? Какие основные способы адресации используются в микропроцессорах?

Глава 3
АРХИТЕКТУРА 16-РАЗРЯДНОГО УНИВЕРСАЛЬНОГО МИКРОПРОЦЕССОРА

В главе рассматривается архитектура достаточно развитого 16-разрядного микропроцессора. Во второй главе уже рассмотрены общие подходы к анализу микропроцессоров и дан пример простого микропроцессора. Поэтому мы будем изучать микропроцессор по предложенному ранее плану.

Для рассмотрения мы выберем микропроцессор семейства 80X86 фирмы *Intel*. Фактически, этот микропроцессор на многие годы стал эталоном архитектуры современных микропроцессоров, на нем построено множество ВС. Этот микропроцессор в модифицированном виде применяется и в настоящее время, поэтому кроме теоретического материал имеет и практическое значение.

Микропроцессор 8086 был представлен в 1978 г. и нашел применение как основа построения процессора микроЭВМ. Сегодня в мире имеются буквально миллионы систем, основанных на этом микропроцессоре. С количеством программного обеспечения, написанного для 8086, не конкурирует никакая другая архитектура.

Однако в начале 80-х годов стало ясно, что замена 8086 стала необходима. Система на основе микропроцессора 8086 требовала множество дополнительных схем, чтобы выполнить даже несложную разработку. Фирма *Intel* почувствовала потребность интегрировать обычно используемые внешние устройства системы на тот же самый кремниевый кристалл, что и центральный процессор. В 1982 г. *Intel* удовлетворила эту потребность, представляя семейство микропроцессоров 80186. Первоначальный 80186 объединил расширенный центральный процессор 8086 с шестью обычно используемыми внешними устройствами системы. Параллельно *Intel* прикладывала усилия по разработке микропроцессора 80286. Этот микропроцессор начал путь к самой высокоэффективной архитектуре *Intel*, которая сегодня включает микропроцессоры *Intel* 386, *Intel* 486 и *Pentium*.

В 1987 г. *Intel* объявила второе поколение семейства микропроцессоров 80186: 80C186/C188. Семейство 80C186 совместимо с семейством 80186 и имеет расширенный набор элементов. Высокоэффективный

процесс производства позволил вдвое повысить тактовую частоту и вчетверо снизить потребляемую мощность. Следующий важный шаг произошел в 1990 г. с введением в семейство 80C186EB. Центральный процессор 8086 был повторно разработан как статический модуль, внешние устройства семейства были так же повторно разработаны, как статические модули со стандартными сопряжениями.

В 1991 г. семейство 80C186 было снова расширено введением трех новых изделий: 80C186XL, 80C186EA и 80C186EC. Они отличаются сниженным энергопотреблением и повышенным быстродействием.

3.1. Функциональная схема микропроцессора 8086

Микросхема 8086 представляет собой однокристальный высокопроизводительный 16-разрядный микропроцессор с фиксированной системой команд. Микропроцессор предназначен для использования в качестве центрального процессорного устройства при построении средств вычислительной техники – от простейших одноплатных микроЭВМ до высокопроизводительных мультипроцессорных систем.

Микропроцессор обладает высоким быстродействием, обеспечивает возможность **прямой адресации памяти** объемом **до 1 Мб**, **65536 устройств ввода** и **65536 устройств вывода**. Для вычисления адресов операндов, размещенных в памяти, используется **24 режима адресации**. Микропроцессор имеет **векторную структуру прерываний** и обеспечивает обработку до **256 запросов прерываний** трех типов: внешних, внутренних и программных.

Архитектурной особенностью микропроцессора 8086 является наличие аппаратно-программных средств, позволяющих упростить построение **мультипроцессорных систем** на его основе. Эти средства обеспечивают синхронизацию работы нескольких независимых (выполняющих собственные потоки команд) процессоров, имеющих общие ресурсы, а также синхронизацию параллельной работы микропроцессора и сопроцессоров (специализированных процессоров, аппаратно реализующих команды сложных процедур). Микропроцессор 8086 характеризуется двумя режимами работы – **минимальным** и **максимальным**, которые различаются способом формирования сигналов обмена и, соответственно, возможностями реализуемых систем.

Функциональная схема микропроцессора приведена на рис. 3.1. Структура микропроцессора 8086 ориентирована на параллельное выполнение функций выборки и выполнения команд и состоит из **устройства сопряжения канала** (**УСК**), **устройства обработки** (**УО**) и устройства управления и синхронизации.

Устройство сопряжения канала предназначено для формирования физического адреса памяти, выборки команд из памяти и записи их в очередь команд, чтения операндов команд из памяти или регистров ввода/вывода, записи результатов выполнения команд в память или регистры ввода/вывода.

В УСК входят: шесть 8-разрядных **регистров очереди команд**; четыре 16-разрядных **сегментных регистра**; 16-разрядный **регистр адреса (указателя) команды**; 16-разрядный **регистр обмена**; 16-разрядный **сумматор адреса**.

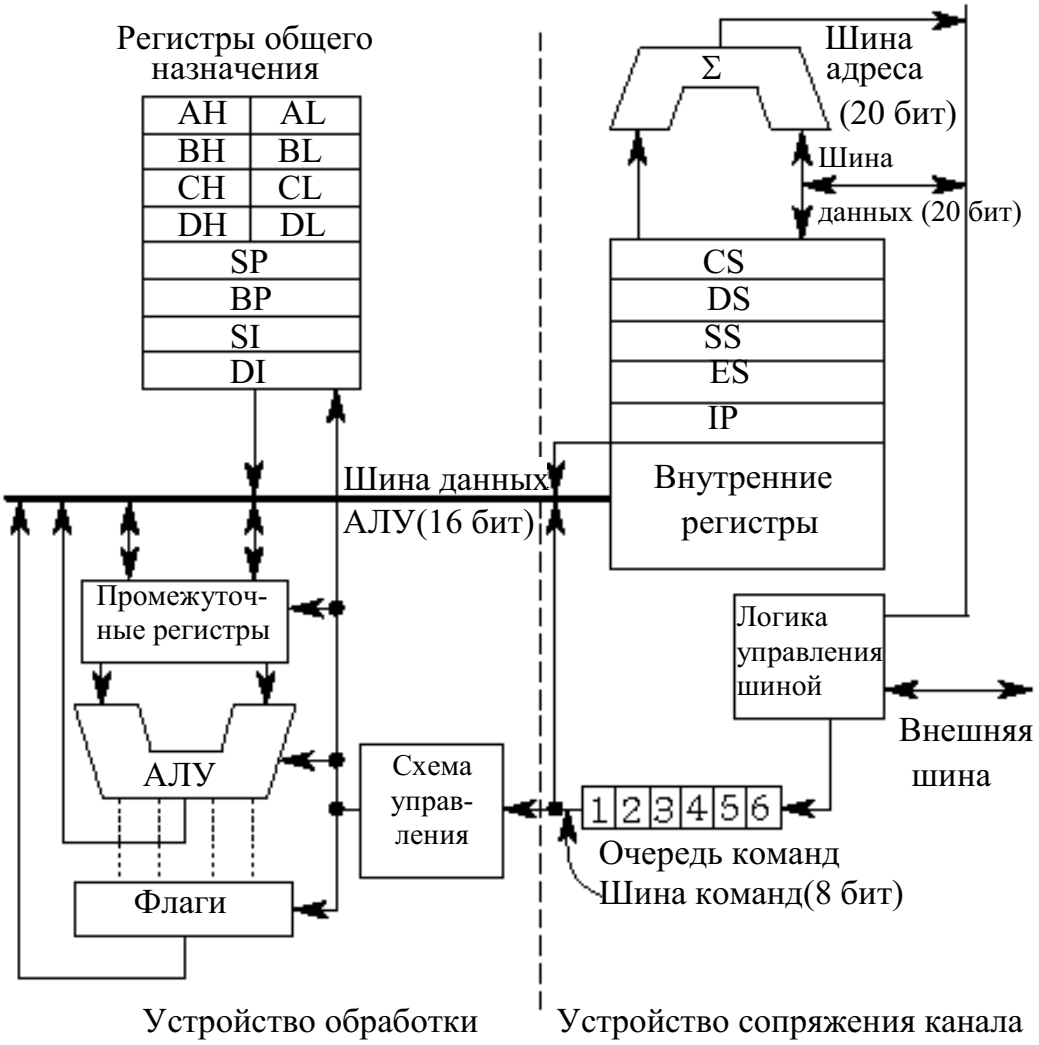


Рис. 3.1. Функциональная схема микропроцессора

Устройство обработки предназначено для выполнения операций по обработке данных. Команды, выбранные из памяти и записанные в регистры очереди команд УСК, по запросам от УО поступают через 8-разрядную магистраль команд на микропрограммное устройство управления, которое декодирует команды и вырабатывает соответствующую последо-

вательность микрокоманд, управляющую процессом выполнения текущей операции. УО не имеет непосредственной связи с внешней магистралью системы и обменивается данными через регистр обмена с УСК.

В устройство обработки входят: 16-разрядное **арифметико-логическое устройство**, восемь 16-разрядных **регистров общего назначения**, 16-разрядный **регистр признаков** состояния микропроцессора.

Команды всегда выбираются из памяти как слова, независимо от четности или нечетности адреса, по которому производится чтение команды.

Отличительной особенностью 8086 является возможность **аппаратной перестройки внутренней структуры** схемы управления и синхронизации. Выбор режима функционирования этой схемы предоставляет разработчику системы возможность выбора подмножества выходных управляющих сигналов в соответствии со степенью сложности проектируемой микропроцессорной системы. Системная настройка обеспечивается специальным выводом выбора режима **MN/MX**.

Микропроцессор позволяет обрабатывать 256 типов прерываний с номерами от 0 до 255, которые делятся на внешние аппаратные, внутренние аппаратные и программные. Запросы на внешние прерывания формируются внешними по отношению к микропроцессору устройствами. Запросы на внутренние прерывания формируются при выполнении определенных команд или по некоторым условиям при выполнении команд. По любому прерыванию управление передается программе (процедуре) обслуживания прерывания посредством вектора прерывания, выбираемого из таблицы векторов прерывания, располагаемой в памяти.

Запросы на внешние прерывания воспринимаются и обрабатываются после выполнения текущей команды. **Внешние прерывания** поступают на микропроцессор по двум внешним выводам (INT и NMI) и делятся на **маскируемые** и **немаскируемые**.

Вопросы, связанные с использованием режима прерываний, обсуждаются более подробно далее в шестой главе, которая посвящена принципам организации обмена данными между микропроцессором и внешними устройствами.

3.2. Интерфейсные сигналы микропроцессора, циклы обмена с шиной

Микропроцессор взаимодействует с системой посредством внешних интерфейсных сигналов. Эти сигналы могут быть разделены на три группы: **адрес/данные**, **управление** и **служебные**. Назначение некоторых выводов микропроцессора зависит от его режима работы. Режим работы определяется специальным сигналом **MN/MX**. Минимальный

режим включается при подаче на вход MN/MX логической 1, максимальный – при подаче на вход MN/MX логического 0. Графическое представление микропроцессора показано на рис. 3.2. На этом рисунке отрицание названия сигнала подразумевает, что активный уровень сигнала низкий. Обозначения выводов микропроцессора в минимальном режиме, если функции выводов в минимальном режиме отличаются от их функций в максимальном, даются в круглых скобках.

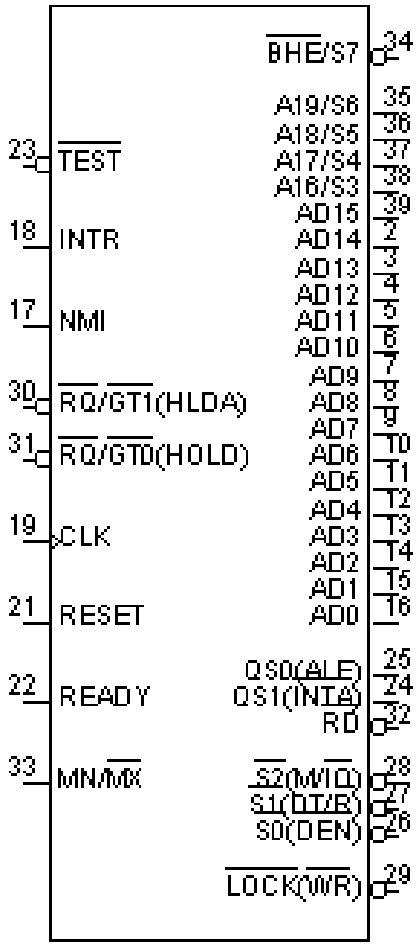


Рис. 3.2. Графическое представление микропроцессора 8086

Теперь опишем основные сигналы микропроцессора.

Сначала мы рассмотрим сигналы адреса/данных. На рис. 3.2 эти сигналы имеют мнемонику **AD** или **A**. Микропроцессор 8086 имеет совмещенную шину адреса и данных, обычно называемую мультиплексированной шиной адреса/данных. Временное мультиплексирование данных и адресов позволяет сократить требуемое количество выводов корпуса микропроцессора. Шина может быть разделена с помощью внешних регистров-защелок на отдельные шины адреса и данных. Выводы шины адреса (**A19 … A16**) указывают четыре старших адресных бита. Активный уровень этих сигналов – высокий. Сигналы шины адреса/данных (**AD0…AD15**) образуют мультиплексированную шину адреса/данных микропроцессора.

Сигнал **BHE** (разрешение старшего байта шины) используется, чтобы разрешить передачу данных по старшей половине шины данных (**D8…D15**). Сигнал **BHE** должен иметь низкий уровень для разрешения передачи старшего байта данных.

Сигнал **BHE** не нуждается в промежуточном запоминании в регистре-защелке. Передача слова или байта по шине данных определяется значениями сигналов **A0** и **BHE**.

Сигналы управления различаются в минимальном и максимальном режимах. В минимальном режиме микропроцессор вырабатывает все необходимые системе сигналы управления. В максимальном режиме необходим внешний системный контроллер. Микропроцессор выраба-

тывает сигналы состояния, которые поступают на этот контроллер, а он вырабатывает все необходимые управляющие сигналы.

Далее описаны сигналы для микропроцессора в минимальном режиме.

Сигнал **ALE** (разрешение фиксации адреса) обеспечивается процессором, чтобы зафиксировать адрес на мультиплексированной шине адреса/данных. Активный уровень сигнала **ALE** – высокий, правильное значение адреса гарантируется по срезу данного сигнала.

Строб записи (**WR**) указывает, что данные с шины данных должны быть написаны в память или устройство ввода/вывода. Этот сигнал имеет активный низкий уровень.

Строб чтения (**RD**) – сигнал с активным низким уровнем, который указывает, что процессор выполняет цикл чтения памяти или устройства ввода/вывода.

Передача/прием данных (**DT/R**) управляет направлением потока данных через внешний приемопередатчик шины данных. Когда сигнал имеет низкий уровень, данные передаются к процессору. При высоком уровне процессор передает данные на шину данных.

Разрешение данных (**DEN**) активизирует приемопередатчики шины данных. **DEN** активен всегда, когда происходит передача данных. Активный уровень – низкий. **DEN** не активен всякий раз, когда **DT/R** изменяет состояние.

Память/УВВ (**M/IO**) – определяет, куда предаются данные. Когда уровень сигнала низкий, данные передаются к устройству ввода/вывода. Когда высокий – данные передаются в память.

Запрос на прерывание (**INTR**) – сигнал требования прерывания внешним устройством. Это входной сигнал, его активный уровень – высокий. В ответ микропроцессор обеспечивает сигнал подтверждения прерывания (**INTA**) с активным низким уровнем. Это сигнал маскируемого прерывания.

Немаскируемое прерывание (**NMI**) вызывает прерывание с вектором 2. Немаскируемое прерывание нельзя запретить программно.

HOLD (активен при высоком уровне сигнала) указывает, что другое устройство управления передачей данных по шине требует внешнюю шину. Процессор генерирует сигнал **HLDA** в ответ на запрос. Одновременно с формированием сигнала **HLDA**, процессор переводит шины в высокоимпедансное состояние. После снятия сигнала **HOLD** процессор переводит сигнал **HLDA** в неактивное состояние. Когда процессор должен выполнять цикл шины, он будет снова управлять локальной шиной и шиной управления.

В **максимальном режиме** сигнал **LOCK** указывает, что другие устройства управления передачей данных по шине системы не могут получить

прав управления. Сигнал **LOCK** имеет активный низкий уровень. Сигнал **LOCK** формируется по специальной команде префикса блокировки (**LOCK**) и активизируется в начале первого цикла передачи данных, связанного с командой, следующей непосредственно после префикса блокировки, и остается активным до завершения этой команды. Если сигнал **LOCK** активен, выборки команд из памяти в очередь команд не происходит.

Выводы **QS0** и **QS1** несут информацию о состоянии очереди команд процессора. Табл. 3.1 показывает возможные состояния очереди команд.

Сигналы состояния цикла шины (**S0**…**S2**) кодируют тип машинного цикла, выполняемого микропроцессором, как показано в табл. 3.2.

В максимальном режиме **HOLD-HLDA** протокол трансформируется в протокол управления доступом к шине Запрос/Разрешение (**RQ/GT**). Это позволяет другим сопроцессорам включаться в общую систему с микропроцессором 8086.

Таблица 3.1

Операции с очередью команд

QS1	QS0	Операции с очередью команд
0	0	Нет операций
0	1	Первый байт кода операции выбирается из очереди
1	1	Выборка следующего байта из очереди
1	0	Очередь команд пуста

Таблица 3.2

Типы машинных циклов

S2	S1	S0	Тип цикла
0	0	0	Подтверждение прерывания
0	0	1	Чтение УВВ
0	1	0	Запись УВВ
0	1	1	Останов
1	0	0	Выборка команды
1	0	1	Чтение данных из памяти
1	1	0	Запись данных в память
1	1	1	Пассивный (нет операций)

CLK, **RESET**, **READY** – сервисные сигналы. Активный уровень сигнала **RESET** заставляет процессор немедленно закончить текущее действие, очистить внутреннюю логику и перейти в неактивное состояние. Процессор начинает выбирать команды через несколько циклов тактовых сигналов, после того как **RESET** возвращается в неактивное

состояние. Входной сигнал **CLK** должен быть подключен к генератору синхронизации. Сигнал **READY** сообщает процессору, что память или устройство ввода/вывода закончило передачу или прием данных. Соединение **READY** с уровнем логической 1 будет всегда устанавливать состояние готовности для процессора.

Для пользователя действия, выполняемые микропроцессором, представляют собой последовательность циклов канала по обмену информацией с памятью или периферийными устройствами. Каждый цикл канала микропроцессора состоит, как минимум, из четырех машинных тактов T1–T4. Машинный такт начинается по спаду импульса синхронизации **CLK** и продолжается один период синхронизации.

Любой цикл шины можно условно разделить на две фазы:

1) **фаза передачи адреса/статуса;**
2) **фаза передачи данных.**

Фаза передачи адреса начинается перед началом такта T1 и продолжается в течение этого такта. Фаза передачи данных начинается в такте T2 и заканчивается в такте T4. В такте T1 на канал адреса/данных всегда выдается адресная информация. В этом же такте вырабатывается сигнал **ALE**, который позволяет идентифицировать начало цикла канала и используется как стробирующий импульс для занесения адресной информации во внешний регистр адреса.

В такте T2 производится переключение направления работы канала адреса/данных. Передача данных по каналу происходит в тактах T3 и T4. Длительность цикла канала может быть удлинена использованием управляющего сигнала **READY**. Этот сигнал позволяет разработчику синхронизировать скорость работы внешней памяти со скоростью работы микропроцессора путем введения дополнительных тактов ожидания между тактами T3 и T4. В течение тактов ожидания данные на канале остаются неизменными. Между тактом T4 текущего цикла и тактом T1 следующего цикла канала процессор может вводить дополнительные (холостые) такты, предназначенные для выполнения внутренних действий. Моменты введения этих тактов и их число зависят от состояния очереди команд и выполняемой команды в УО.

На рис. 3.3 представлена типовая временная диаграмма выполнения циклов чтения и записи.

Цикл чтения начинается с выработки сигнала **ALE**. Этот сигнал используется для занесения адресной информации во внешний регистр адреса. В такте **T2** канал **A/D** переключается в высокоомное состояние, вырабатывается сигнал **RD**, который используется для чтения адресуемого устройства. Для управления шинными формирователями, обеспе-

чивающими развязку канала адреса/данных микропроцессора от системного канала данных, используются сигналы **DT/R** и **DEN**.

Цикл записи (как и цикл чтения) начинается с выдачи сигнала **ALE** и адреса на шину адреса/данных. В такте Т2 непосредственно за выдачей адреса на шину **A/D** выдаются данные для записи в адресуемое устройство. Эта информация остается истинной на канале данных до окончания такта Т4. Сигнал **WR** вырабатывается в начале такта Т2 и остается в этом состоянии до начала такта Т4.

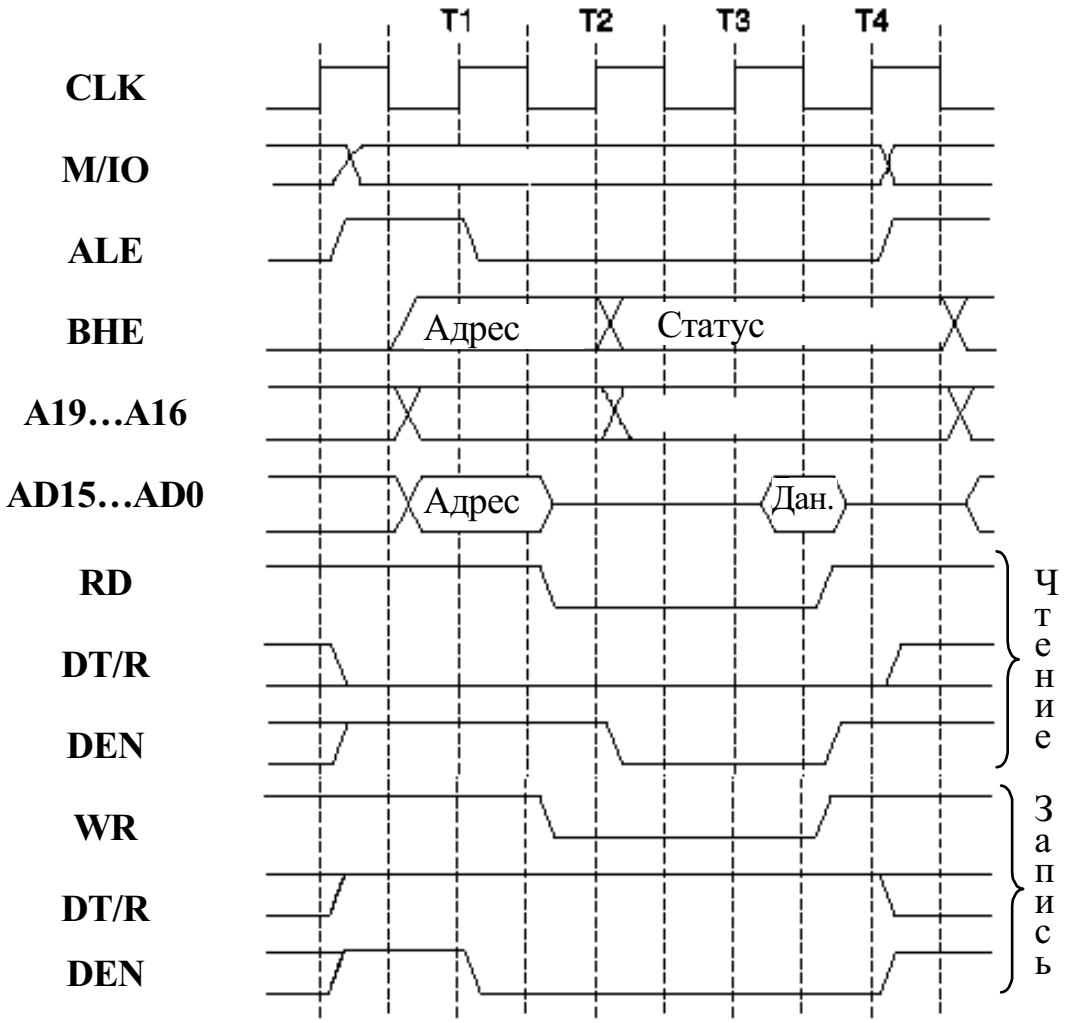


Рис. 3.3. Временная диаграмма циклов чтения и записи

3.3. Программная модель микропроцессора

Программно-доступными функциональными частями микропроцессора являются **регистры общего назначения**, **сегментные регистры**, регистры-**указатели** адреса (**индексные регистры**) и **регистр признаков**. Они показаны на рис. 3.4.

Общие регистры используются для хранения операндов и результатов выполнения команд и делятся на две группы: регистры данных (регистры общего назначения), индексные регистры и указатели.

В группу регистров данных входят: регистр аккумулятора AX; регистр указателя базы данных BX, регистр счетчика циклов CX, регистр данных DX.

В группу индексных регистров и регистров-указателей входят регистр указателя стека SP, регистр указателя базы стека BP, регистр индекса источника SI, регистр индекса приемника DI.

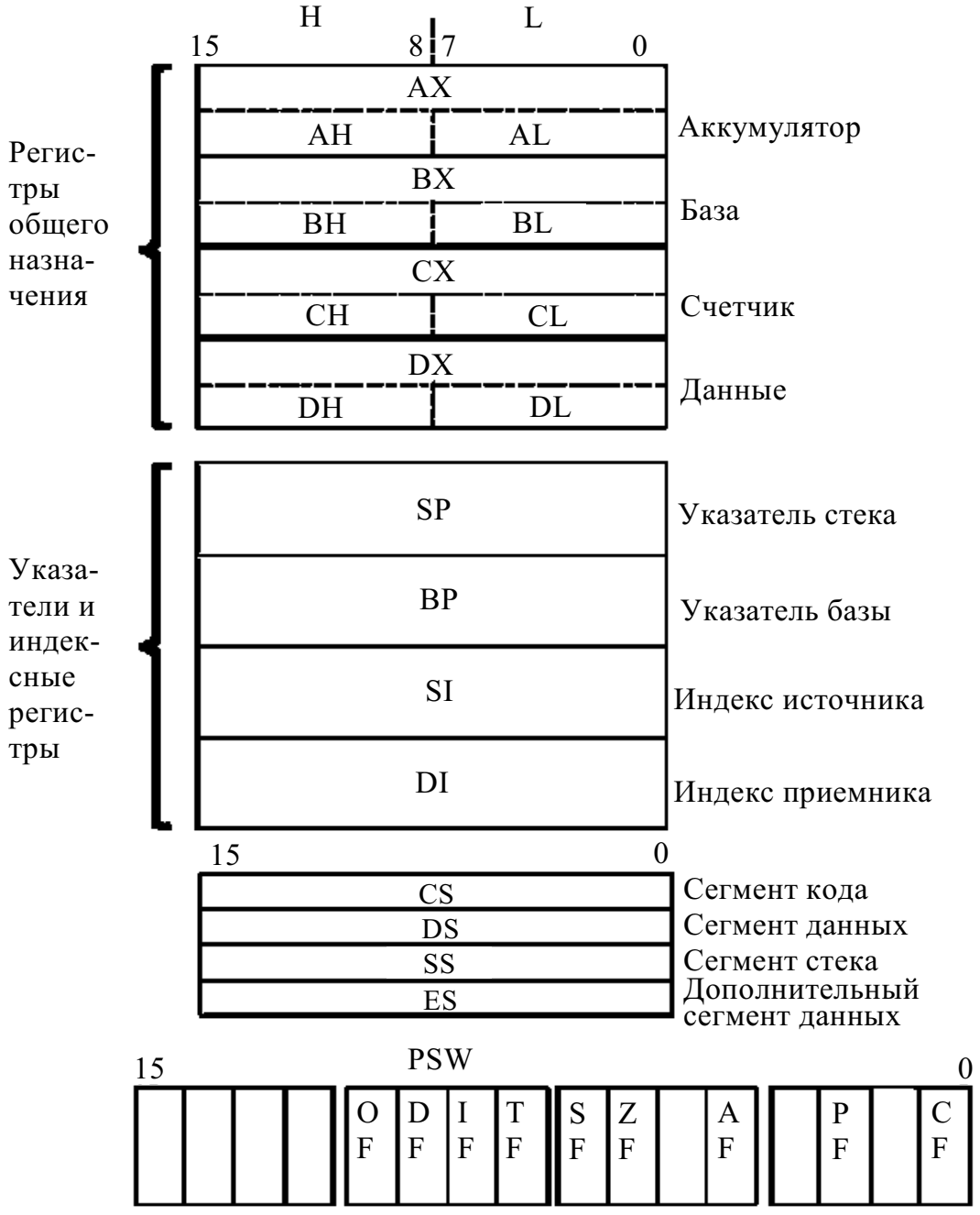


Рис. 3.4. Программно-доступные регистры микропроцессора

Старшие и младшие восемь разрядов группы регистров общего назначения могут быть адресованы раздельно. Они образуют набор 8-разрядных регистров общего назначения (AH, AL, BH, BL, CH, CL, DH, DL), причем регистрам AH, BH, CH, DH соответствуют старшие восемь разрядов, а регистрам AL, BL, CL, DL – младшие восемь разрядов группы регистров. Некоторые команды по умолчанию используют регистры для выполнения определенных функций. Они перечислены в табл. 3.3.

Таблица 3.3

Использование общих регистров

Регистр	Операция
AX	Умножение и деление слов, ввод/вывод слов
AL	Умножение и деление байтов, ввод/вывод байтов, десятичная арифметика, перекодирование
AH	Умножение и деление байтов
BX	Перекодирование
CX	Строковые операции, циклы
CL	Сдвиги переменных
DX	Умножение и деление слов, косвенная адресация портов ввода/вывода
SP	Стековые операции
SI	Операции со строками
DI	Операции со строками

Сегментные регистры используются для организации сегментной адресации памяти и предназначены для хранения базовых адресов текущих сегментов памяти. В 8086 имеется четыре 16-разрядных сегментных регистра: кода **CS**, данных **DS**, стека **SS**, дополнительного сегмента данных **ES**.

Разряды регистра признаков содержат признаки состояния микропроцессора, которые разделены на две группы: признаки результата и признаки управления. В группу признаков результата входят:

- признак переполнения **OF**, указывающий на переполнение в случае выполнения операций над целыми числами;
- признак знака **SF**, указывающий на знак результата;
- признак нуля **ZF**, указывающий на равенство нулю результата;
- признак вспомогательного переноса **AF**, указывающий на перенос из третьего разряда или на заем в третий разряд результата при выполнении арифметических операций;

- признак четности **PF**, указывающий на четное число единиц в младшем байте результата;
- признак переноса **CF**, указывающий на перенос из старшего разряда или на заем в старший разряд результата.

В группу признаков управления входят:

- признак направления **DF**, указывающий направление обработки строк данных;
- признак разрешения прерывания **IF**, разрешающий или запрещающий маскируемые прерывания;
- признак пошагового режима **TF**, управляющий пошаговыми прерываниями.

Микропроцессор обеспечивает формирование 20-разрядного адреса для адресации ячейки внешней памяти. Память организована как линейная последовательность ячеек памяти объемом в 1 Мб с адресами от 00000H до FFFFFH. Структурными единицами памяти являются: ячейка, слово, двойное слово и сегмент.

Ячейка памяти – минимальная адресуемая единица памяти, используемая для запоминания 8-разрядных данных (байта данных).

Слово памяти – две последовательные ячейки памяти, которые используются для запоминания 16-разрядных данных (слова данных), причем младшие восемь разрядов всегда хранятся в ячейке памяти с меньшим адресом, а старшие – с большим. При адресации 16-разрядных данных указывается адрес первой ячейки слова памяти. Слово памяти может располагаться в памяти как по четному, так и по нечетному адресу. Чтение (запись) данных из слова памяти по четному адресу осуществляется за одно обращение к памяти, а по нечетному – за два обращения.

Двойное слово памяти – четыре последовательные ячейки памяти или два последовательных слова памяти, которые используются для запоминания 32-разрядных данных. При адресации 32-разрядных данных указывается адрес первой ячейки двойного слова памяти. Двойное слово памяти также может иметь четный или нечетный адрес.

Для достижения максимальной производительности слова и двойные слова данных должны размещаться в памяти по четным адресам.

Программы, написанные для микропроцессора 8086, рассматривают 1 Мб памяти как группу сегментов, определяемых конкретным применением.

Сегмент памяти участок памяти, емкость которого может изменяться от 16 до 65536 байт, начинается с адреса, кратного 10 Н. Каждому сегменту соответствует непрерывная и отдельно адресуемая область памяти. Примеры сегментации показаны на рис. 3.5.

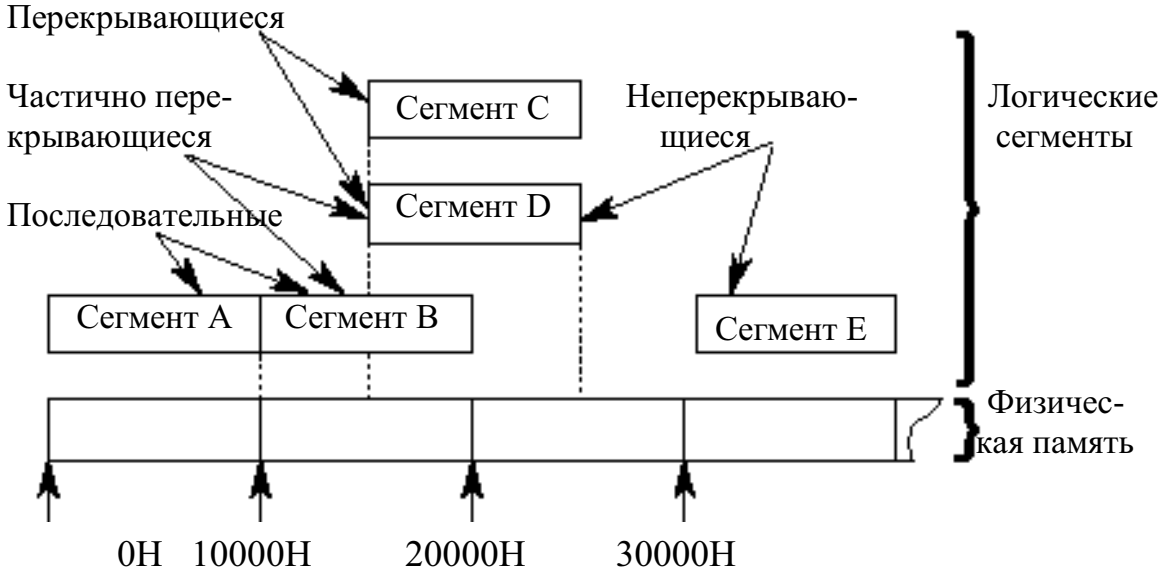


Рис. 3.5. Сегментация памяти микропроцессора

Сегменты могут следовать друг за другом непрерывно, с интервалом, или могут перекрываться. Максимальное количество следующих непрерывно друг за другом сегментов емкостью 65536 байт равно 16.

Микропроцессор позволяет независимо адресовать четыре программных сегмента в памяти, называемых текущими сегментами команд, данных, стека и текущим дополнительным сегментом.

Сегментирование памяти совместно с позиционно-независимыми командами передачи управления позволяет создавать динамически перемещаемые программные модули.

Физически область памяти для 8086 организуется как два банка памяти по 512 кб (рис. 3.6): старший банк (**D15–D8**) и младший банк (**D7–D0**). Для адресации ячеек памяти в каждом банке используются разряды шины адреса микропроцессора. Байт данных с четным адресом пересылается по линиям **D7–DO** канала данных, а байт данных с нечетным адресом – по линиям **D15–D8** канала данных. Микропроцессор вырабатывает сигналы **BHE** и **A0**, используемые для выбора соответствующего банка памяти.

Младший банк, содержащий четно адресуемые байты, выбирается при **A0** = 0. Старший банк, содержащий нечетно адресуемые байты, выбирается при **BHE** = 0 (табл. 3.4).

Программы имеют дело не с физическими, а с логическими адресами. Логический адрес состоит из начального адреса сегмента и смещения.

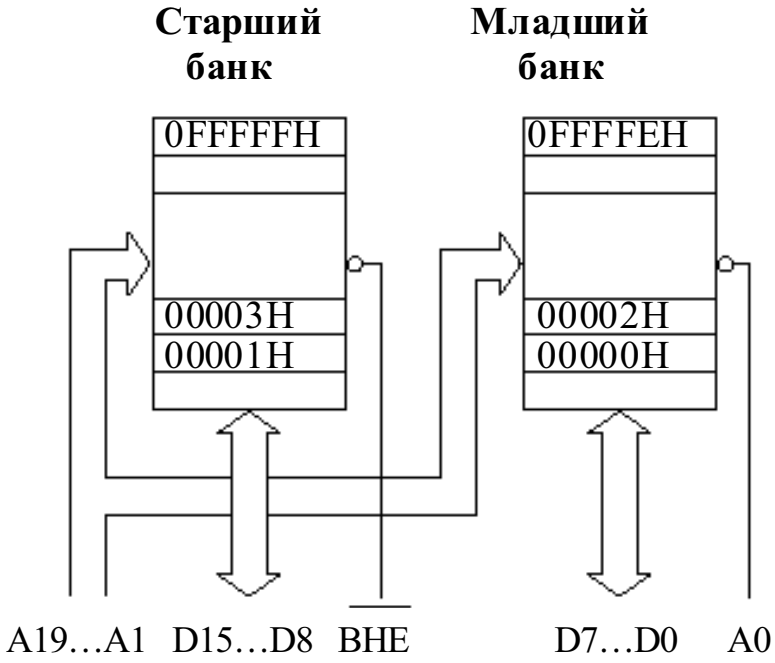


Рис. 3.6. Физическая модель памяти

Таблица 3.4

Кодирование сигналов BHE и A0

BHE	**A0**	Операция
0	0	Передача слова
0	1	Передача байта по старшей половине шины данных (**D15…D8**)
1	0	Передача байта по младшей половине шины данных (**D7…D0**)
1	1	Запрещено

Сегмент – это логическая единица памяти объемом до 64 кб. Каждый сегмент состоит из последовательных ячеек памяти. Сегменты независимы и адресуются отдельно.

Программа назначает сегменту базовый адрес в памяти, с которого сегмент начинается. Чтобы адреса сегментов можно было хранить в регистрах микропроцессора они должны быть 16-разрядными. Однако базовый адрес сегмента в общем случае 20-разрядный. Поэтому младшие 4 двоичных разряда базового адреса сегмента берутся нулевыми и в написании опускаются. Таким образом, сегмент может начинаться не с любого физического адреса, а только с адреса, у которого младшие

4 разряда равны 0. Кроме базового адреса при логической адресации указывается смещение.

Смещение – это число, определяющее в байтах расстояние от базового адреса начала сегмента до конкретной ячейки памяти.

Базовый адрес начала сегмента хранится в одном из сегментных регистров микропроцессора, а смещение определяется из команды или находится в общем регистре.

Способ адресации определяет, как в команде вычисляется смещение. Результат этого вычисления называется ***эффективным адресом (EA)****.*

В табл. 3.5 показано, как по умолчанию определяются логические адреса ячеек памяти при выполнении микропроцессором различных действий.

Таблица 3.5

Источники логического адреса

Тип обращения к памяти	Сегмент по умолчанию	Альтернативный сегмент	Смещение
Выборка команд	CS	нет	IP
Операции со стеком	SS	нет	SP
Переменные (исключая перечисленные ниже)	DS	CS, ES, SS	EA
Цепочка-источник	DS	CS, ES, SS	SI
Цепочка-приемник	ES	нет	DI
BP как регистр базы	SS	CS, DS, ES	EA

В общем случае микропроцессор может выбирать ячейки памяти из сегментов, которые указаны в таблице как альтернативные источники адреса сегмента. Для этого используется специальная команда – **префикс замены сегмента**.

3.4. Способы адресации операндов

При выполнении любой программы процессор обращается к памяти, в которой хранятся команды и данные. В командах преобразований данных определяются адреса, которые указывают местоположение необходимых данных, а в командах передачи управления определяются адреса команд, которым передается управление, т. е. адреса переходов. Способ или метод определения в команде адреса операнда (адреса перехода), называется режимом адресации или просто адресацией. В наиболее простом режиме адресации, называемом **прямой адресацией**, адрес находится в самой команде. Однако использование этого режима, хотя и предусмотрено в большинстве современных процессоров, приводит к чрезмерной длине команд, особенно в условиях постоянно увеличивающейся емкости памяти. Поэтому в настоящее время в процессорах

применяется много других режимов адресации, направленных на достижение следующих целей:

- определение адреса памяти наименьшим числом бит, что сокращает длину команд;
- вычисление адреса относительно текущей команды (так называемая относительная адресация), обеспечивающее загрузку программ без модификации в любую область памяти;
- осуществление доступа к ячейкам памяти, адреса которых вычисляются при выполнении программы, что упрощает доступ к регулярным структурам данных;
- оперирование адресами в форме, которая наиболее удобна для таких структур данных, как массивы и стеки.

Для современных процессоров разработано более двух десятков режимов адресации, которые в той или иной степени удовлетворяют приведенным выше требованиям. Режимы адресации значительно расширяют гибкость и удобство пользования системой команд. Назначением режима адресации является указание способа формирования эффективного (или исполнительного) адреса EA. Этот адрес является либо адресом операнда (в командах, оперирующих данными), либо адресом перехода (в командах передачи управления). В МП 8086 **эффективный адрес** памяти представляет собой 16-битное беззнаковое целое, являющееся смещением относительно базы некоторого сегмента. Полный (физический) адрес памяти формируется с привлечением одного из сегментных регистров. Режимы адресации подразделяются на **прямые** и **косвенные**. При **прямой адресации** эффективный **адрес** либо содержится **в команде**, либо вычисляется с использованием значения, находящегося в команде, и содержимого, указанного в команде регистра (или двух регистров). При **косвенной адресации** эффективный адрес в команде определяет регистр или ячейку памяти, содержащую **окончательный эффективный адрес**. Большинство современных процессоров допускает только один уровень косвенности, хотя принципиально этот уровень может быть любым. МП 8086 имеет организацию типа «регистр – память». С точки зрения адресации это означает, что его команды адресуют максимум **два операнда** и что не допускается одновременная адресация двух ячеек памяти. Первым операндом в двухоперандной (или двухадресной) команде обычно является содержимое регистра или ячейки памяти, а вторым – содержимое регистра или непосредственный операнд в команде. Ниже будет показано, что приведенная нумерация («первый» и «второй» операнды) в МП 8086 является довольно условной и при желании может быть изменена. Наиболее общий формат двухоперандной команды приведен на рис. 3.7 (штриховыми линиями обозначены необязательные байты команды).

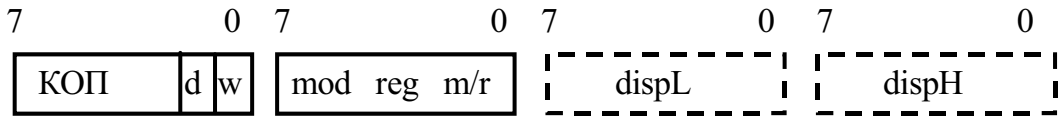


Рис. 3.7. Формат двухоперандной команды

Первый байт команды содержит **код операции** (КОП) и два однобитных поля – направления **d** и слова **w**. Поле d определяет направление передачи: если d = l, то направление в МП, а если d = 0, то направление из МП. Само направление относится ко второму операнду – регистру, определяемому полем **reg** второго байта команды. Этот байт называется **постбайтом** (или просто байтом) **режима адресации**. Поле w идентифицирует тип (размер) операнда: если w = l, то команда оперирует словом, а если w = 0 – байтом. Таким образом, в зависимости от значений полей d и w имеются четыре возможности (табл. 3.6).

Таблица 3.6

Кодирование полей d и w

d	w	Передача или операция
0	0	Байт из регистра в память или регистр
0	1	Слово из регистра в память или регистр
1	0	Байт в регистр из памяти или регистра
1	1	Слово в регистр из памяти или регистра

Участвующие в операции регистры или регистр и ячейку памяти идентифицирует **постбайт**, состоящий из трех **полей**: **mod** – режим, **reg** – регистр и **r/m** – регистр/память. Поле reg определяет второй операнд, обязательно находящийся в регистре. Способ кодирования внутренних регистров МП в поле **reg** представлен в табл. 3.7. Из таблицы видно, что регистры данных **AX**–**DX**, а также указательные и индексные регистры **SP**, **BP**, **SI**, **DI** адресуются одинаковым образом. Данное обстоятельство подчеркивает правомерность объединения всех этих регистров в группу общих регистров.

Таблица 3.7

Адресация внутренних регистров МП

Поле reg (и r/m)	8-битные регистры	16-битные регистры
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Поле **mod** определяет используемый режим адресации. В частности, оно показывает, как интерпретируется поле r/m для нахождения первого операнда. Если mod = 11, операнд содержится в регистре, а в остальных случаях первый операнд находится в памяти. Когда поле mod содержит 11 (регистровая адресация), поле r/m определяет 8- или 16-битный регистр в соответствии с таблицей. В остальных случаях адресуется память, и поле mod определяет, как используется (необязательное) смещение **disp**, находящееся во втором и третьем байтах команды:

- 00, disp = 0 – смещение отсутствует;
- 01, disp = dispL – команда содержит 8-битное смещение, которое расширяется со знаком до 16 бит;
- 10, disp = dispH, dispL – команда содержит два байта смещения.

В случае косвенной адресации поле r/m определяет, каким образом формируется эффективный адрес операнда.

Приведенные правила имеют только одно исключение: если mod = 00 и r/m = 110, то EA = dispH, dispL. Здесь в команде содержится абсолютный адрес памяти. Таким образом, операнд в памяти можно адресовать **прямо** (16-битное смещение) или **косвенно** (с 8- или 16-битным смещением). Во втором случае память можно адресовать через базовый регистр (BP или BX), через индексный регистр (SI или DI), а также через комбинацию базового и индексного регистров. Всего получается 24 режима адресации.

Рассмотрим стандартные режимы адресации микропроцессора 8086 с учетом приведенных выше особенностей формирования эффективного адреса.

Регистровая адресация. При использовании регистровой адресации операнд находится в одном из общих регистров МП, а в некоторых командах – в одном из сегментных регистров. Регистр может быть определен в байте кода операции или в постбайте режима адресации. В двухоперандных командах определяются два регистра. Регистры идентифицируются 3-битными полями reg и r/m (когда mod = 11). В зависимости от значения бита w в операции участвует 8- или 16-битный регистр (или регистры). Команды, оперирующие содержимым регистров, оказываются **наиболее короткими** и выполняются за **минимальное время**, так как не требуют цикла шины для обращения к памяти. В ассемблерных программах регистры обозначаются зарезервированными именами: **AL**, **AH**, **AX**, **BL**, **BH**, **BX** и т. д.

Непосредственная адресация. Непосредственные операнды представляют собой константы длиной 8 или 16 бит, содержащиеся в командах. **Доступ** к таким операндам осуществляется очень **быстро**, так как

они находятся во внутренней очереди команд и циклов шины для считывания операндов из памяти не требуется. Непосредственные операнды **изменить** в ходе выполнения команды **невозможно**. Данный режим предусмотрен для большинства двухоперандных команд. Наличие в командах постбайта режима адресации позволяет манипулировать непосредственными операндами и содержимым регистров или ячеек памяти. Однако в МП нет команд непосредственной загрузки сегментных регистров и включения константы в стек, что вызывает некоторые неудобства при программировании. Стандартный непосредственный операнд имеет длину 16 бит, а короткий – 8 бит (при необходимости он расширяется со знаком до 16 бит). Константы в командах применяются для нескольких целей, например для инициализации регистров и переменных в памяти, в качестве масок для манипуляций отдельными битами, для сравнения переменных с граничными значениями и т. д. Непосредственные операнды находятся в конце формата команд после смещения, если оно имеется, причем первым следует младший байт константы.

Прямая адресация. Прямая адресация представляет собой простейший режим адресации: эффективный адрес EA берется прямо из поля смещения команды, и никакие регистры для его вычисления не привлекаются. Этот режим применяется для обращения к простым переменным, которые называются также скалярами. Разновидностью данного режима является длинная прямая адресация, в которой команда содержит базовый адрес сегмента и смещение. В этом случае обеспечивается прямой доступ к операнду с любым логическим (и физическим) адресом. Однако длинная прямая адресация допускается только в командах межсегментных переходов и вызовов и не может применяться в командах, оперирующих данными.

Косвенная регистровая адресация. В этом режиме эффективный адрес EA операнда находится в одном из базовых или индексных регистров (рис. 3.8). Из четырех адресных регистров BP, BX, SI и DI в косвенной адресации могут использоваться только регистры BX, SI и DI. Косвенная адресация через указатель базы BP моделируется при помощи базовой адресации с нулевым смещением. Данный режим с некоторыми вариантами применяется во всех современных процессорах. Он позволяет вычислять адреса во время выполнения программы, что часто требуется, например, для обращения к различным элементам регулярных структур данных. При модификации содержимого регистра одна и та же команда оперирует различными ячейками памяти. Для изменения содержимого регистра применяется команда загрузки эффективного ад-

реса LEA, а также все арифметические и логические команды. Отметим, что в команде безусловного перехода и команде вызова подпрограммы CALL с регистровой косвенной адресацией допускается указывать любой 16-битный общий регистр.

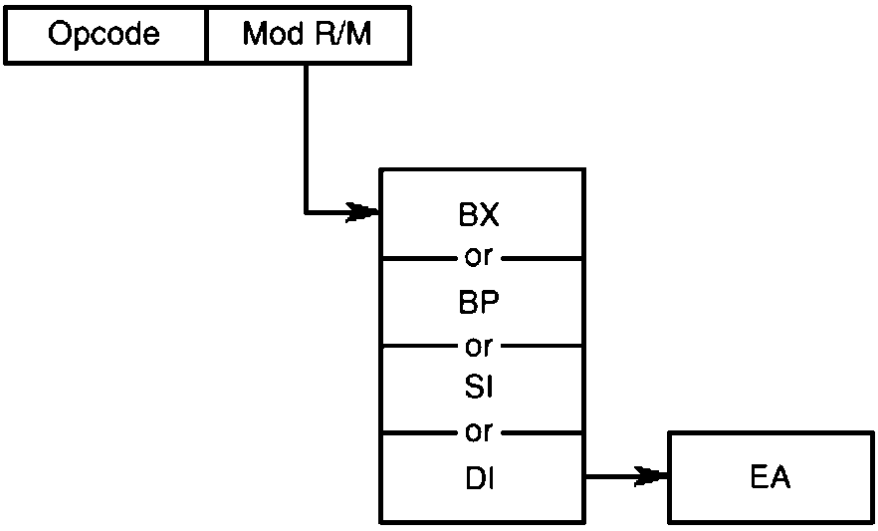


Рис. 3.8. Косвенная регистровая адресация

Базовая адресация. При базовой адресации (называемой также адресацией по базе или адресацией типа **база плюс смещение**) эффективный адрес операнда является **суммой** значения **смещения** и содержимого регистров **BX** или **BP**. Напомним, что при определении BP в качестве базового адреса шинный интерфейс обращается к операнду в текущем сегменте стека (если нет префикса замены сегмента). Это делает базовую адресацию с регистром BP очень удобным средством обращения к данным, находящимся в стеке, что требуется, например, при передаче подпрограммам параметров через стек. Смещения, содержащиеся в команде, могут иметь длину 8 или 16 бит и интерпретируются как знаковые целые. Размер смещения программа-ассемблер выбирает автоматически с учетом атрибутов операндов. В языке ассемблера используются два обозначения базовой адресации. Первое обозначение имеет вид [BREG] DISP и соответствует адресации структуры данных типа «запись» в языке ПАСКАЛЬ. Второе обозначение имеет вид [BREG + DISP] и явно указывает на необходимость суммирования содержимого регистра и смещения при вычислении эффективного адреса. В обоих случаях обозначение BREG подразумевает один из базовых регистров.

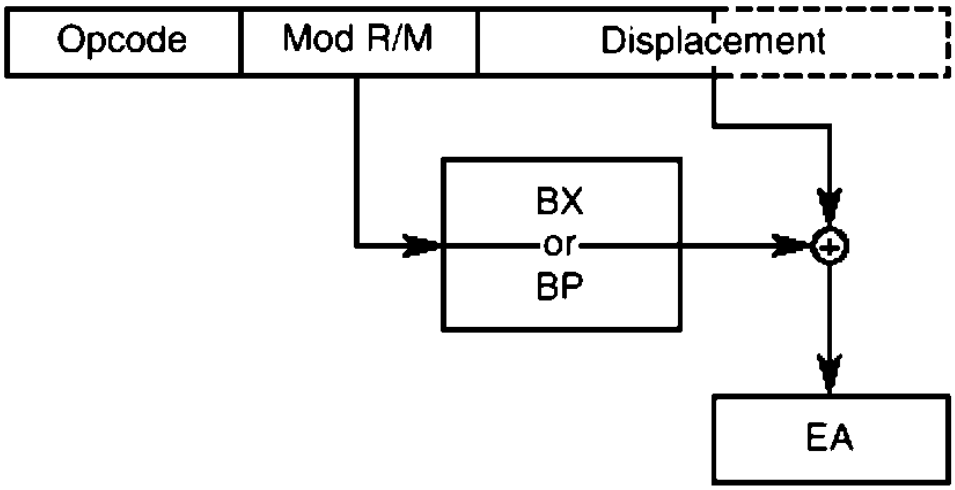


Рис. 3.9. Базовая адресация

Индексная адресация. В режиме индексной адресации (называемой также адресацией с индексированием, адресацией типа «база плюс индекс») эффективный адрес вычисляется как сумма **смещения**, находящегося в команде, и содержимого регистров **SI** или **DI**. Данный режим обычно применяется для обращения к элементам одномерных массивов. Смещение определяет фиксированный начальный адрес массива, а значение в индексном регистре определяет нужный элемент. Так как все элементы массива имеют одинаковый размер, простые манипуляции над содержимым индексного регистра позволяют обращаться к любому элементу массива. По существу, режимы базовой и индексной адресации в МП 8086 аналогичны. Это объясняется тем, что базовые адреса и индексные значения имеют одинаковую длину 16 бит. В языке ассемблера индексная адресация обозначается в виде ADDR16 [IREG], как это принято в языках высокого уровня. Здесь ADDR16 – 16-битное смещение, а IREG обозначает один из индексных регистров SI или DI. Таким образом, структура данных может размещаться в различных областях памяти, а модификация содержимого базового регистра обеспечивает доступ к этим областям. Базовый регистр указывает на начало структуры данных, а требуемый элемент адресуется с помощью смещения (расстояния) от базы.

Базово-индексная адресация. Данный режим называется также адресацией типа «**база плюс индекс плюс смещение**». Здесь эффективный адрес равен сумме содержимого базового регистра, индексного регистра и смещения, находящегося в команде (рис. 3.10). Базовая индексная адресация является наиболее гибким режимом, так как два компонента адреса можно определять и варьировать при выполнении программы. Регистры BP и BX используются как базовые, а регистры SI и

DI – как индексные, т. е. всего получается четыре комбинации регистров. В МП 8086 данный режим расширен и допускает в командах 8- или 16-битные смещения, которые также суммируются при вычислении эффективного адреса. Смещение считается знаковым целым, представленным в дополнительном коде. При просмотре исходной программы ассемблер по выражению, находящемуся в поле операнда, определяет необходимость задания смещения и его размер. Данный режим обеспечивает подпрограммам удобный способ адресации массива, находящегося в стеке. Регистр BP обычно адресует некоторую отсчетную точку в стеке после того, как подпрограмма включила в стек содержимое внутренних регистров МП и выделила область стека для локальных переменных. В этом случае регистр BP выполняет функцию указателя стекового кадра. Расстояние начала массива от отсчетной точки представляется значением смещения, а индексный регистр предназначен для адресации отдельных элементов массива. С помощью базовой индексной адресации возможно также обращение к элементам массива, содержащегося в матрице, т. е. двумерном массиве. В ассемблерных программах базово-индексная адресация обозначается в виде [BREG] ADDR16 [IREG], т. е. как комбинация базовой и индексной адресаций.

Относительная адресация. В режиме относительной адресации эффективный адрес вычисляется как сумма фиксированного смещения, находящегося в команде, и текущего значения программного счетчика **PC**. При этом значение PC равно адресу байта, следующего за текущей командой (предполагается, что команда с относительной адресацией считана из памяти и PC адресует следующую по порядку команду). В МП 8086 относительная адресация применяется только в командах условных и безусловных переходов, вызовов программ и управления итерациями (или циклами). Команда содержит короткое 8-битное смещение, которое расширяется со знаком до 16 бит, а затем прибавляется к содержимому PC. Следовательно, этот режим обеспечивает передачу управления в диапазоне +128…–127 байт от текущей команды. Короткие (SHORT) переходы часто требуются в циклических программах, и на них приходится до 80 % команд передачи управления. Следует отметить, что программист в ассемблерных программах указывает не значение смещения, а абсолютный адрес перехода, т. е. метку команды, которой необходимо передать управление. Значение смещения автоматически вычисляет программа-ассемблер. Значение смещения не зависит от размещения программы в адресном пространстве памяти, поэтому относительная адресация обеспечивает позиционную независимость программ.

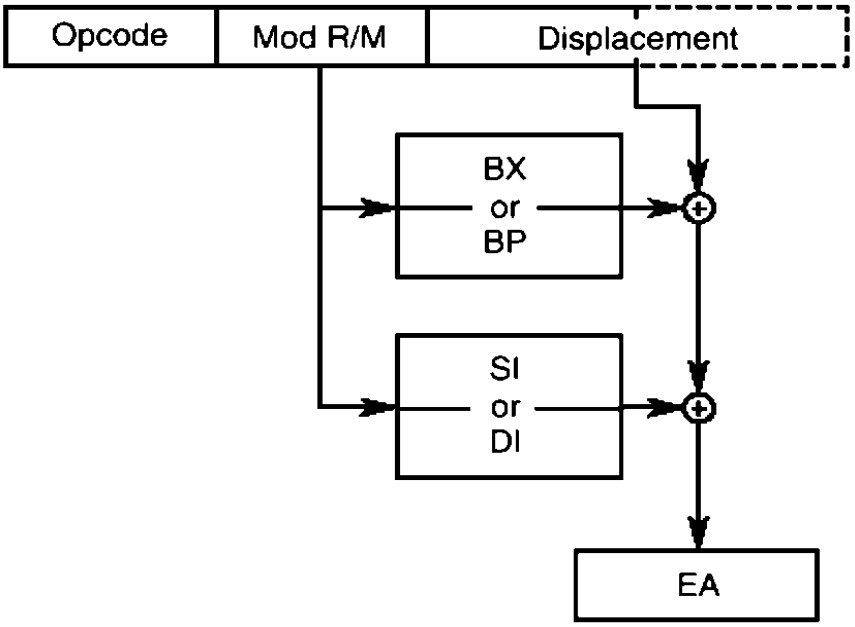


Рис. 3.10. Базово-индексная адресация

Адресация цепочек. Для обращения к операндам цепочечных команд обычные режимы адресации памяти не применяются. Вместо этого неявно используются индексные регистры. При выполнении цепочечной команды предполагается, что регистр SI адресует первый байт (слово) цепочки-источника, а регистр DI – первый байт (слово) цепочки-получателя. В повторяющихся цепочечных операциях МП автоматически корректирует содержимое регистров SI и DI по мере перехода к следующим элементам цепочек.

Адресация портов ввода-вывода. Если порты ввода-вывода отображены на память, то для обращения к ним применяются любые режимы адресации операндов, находящихся в памяти. В этой же ситуации возможно использование цепочечных команд при наличии соответствующего аппаратного интерфейса. Для доступа к портам, находящимся в адресном пространстве ввода-вывода, используются два режима. В прямой адресации номер порта представляет собой 8-битный непосредственный операнд, находящийся во втором байте команды, что обеспечивает обращение к фиксированным портам 0–255. Косвенная адресация портов ввода-вывода аналогична регистровой косвенной адресации операндов в памяти: номер порта находится в регистре DX и имеет диапазон 0–65535. С помощью предварительной инициализации регистра DX одна и та же команда может обращаться к любому порту в адресном пространстве ввода-вывода. К группе соседних портов возможно обращаться с помощью простого программного цикла, который модифицирует содержимое регистра DX.

Режим адресации и время выполнения команды. Как следует из предыдущего материала, эффективный адрес памяти получается в результате некоторых вычислений. Время их выполнения, а следовательно, и время выполнения команды будет варьироваться в зависимости от сложности вычислений. Время вычисления эффективного адреса зависит от режима адресации. Оно равно числу тактов синхронизации, необходимому для определения эффективного адреса при условии, что команда находится во внутренней очереди команд. Наличие перед командой префикса замены сегмента увеличивает величины времени выполнения на два такта синхронизации.

Примеры решения задач

1. Раскодируйте следующую команду, представленную в виде шестнадцатеричного числа и запишите ее в ассемблерном виде: 23C7H.

Решение

Первый байт этого числа является кодом операции. Его формат

0 0 1 0 0 0 1 1

opcode d w

Этот код операции соответствует операции «И» (AND); w = 1 означает, что команда обрабатывает 2-байтовое слово.

Второй байт числа – постбайт режима адресации. Его формат

1 1 0 0 0 1 1 1

mod reg r/m

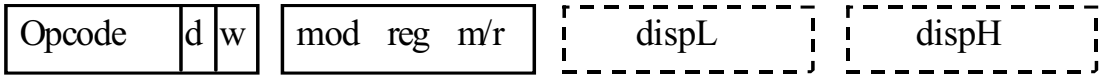
Поле mod = 11 свидетельствует о том, что в команде использован регистровый метод адресации операндов, и показывает, что поле **r/m** обрабатывается так же, как и поле **reg**. Поле reg = 000 кодирует регистр AX. Поле m/r = 111 кодирует регистр DI. Поле d = 1 показывает, что регистр DI является источником данных, а регистр AX – приемником. Ассемблерная запись команды:

AND AX,DI.

2. Закодируйте команду MOV 01AF[DI], DX.

Решение

Определим сначала действие, выполняемое командой, и способ адресации операндов. Команда копирует слово данных из регистра (DX) в ячейку памяти. Ячейка памяти адресуется с использованием индексного способа адресации с использованием смещения. Общий формат команды следующий:



Opcode определяется из таблицы кодировки, например, взятой из фирменного руководства программиста. Это следующий код – 100010dw. Поле w равно 1, потому что команда пересылает слово информации. Поле d равно 0, потому что поле reg является источником информации. Из таблиц (смотри, например, в учебнике) определяем:

r/m = 101 ((DI)+ disp.);

reg = 010 (DX регистр);

mod = 10 (disp = dispL,dispH).

Как результат получаем двоичный код команды 10001001 10010101 10101111 00000001 или 8995AF01H в шестнадцатеричном коде.

Вопросы и задания для повторения

Основные характеристики и функциональная схема микропроцессора

3.1. Какие составные части вычислительной системы включены в состав микросхемы микропроцессора 8086?

3.2. Перечислите основные характеристики микропроцессора.

3.3. Что такое минимальный и максимальный режимы работы микропроцессора? Как выбирается соответствующий режим?

3.4. Каков объем адресуемой памяти микропроцессора? Сколько портов можно адресовать?

3.5. Может ли микропроцессор одновременно передавать данные и адресную информацию? Почему?

3.6. Каково назначение регистра очереди команд?

3.7. Объясните, как формируется физический адрес ячейки памяти микропроцессором 8086. Каковы преимущества и недостатки сегментации памяти?

Интерфейсные сигналы и циклы шины

3.8. На какие группы делятся интерфейсные сигналы микропроцессора?

3.9. Как изменяется назначение сигналов микропроцессора в зависимости от режима работы?

3.10. Перечислите адресные сигналы микропроцессора. Поясните, как сигналы A0 и $\overline{\text{BHE}}$ используются для определения разрядности передаваемых данных.

3.11. Какие выводы микропроцессора используются для передачи данных?

3.12. Перечислите управляющие сигналы микропроцессора в минимальном и максимальном режиме. Покажите, как получить управляющие сигналы $\overline{\text{WR}}$, $\text{DT}/\overline{\text{R}}$, $\text{M}/\overline{\text{IO}}$ из сигналов состояния $S_0 \ldots S_2$ в максимальном режиме.

3.13. Объясните, почему для работы микропроцессора необходимы сигналы синхронизации.

3.14. Что вызывает сигнал RESET?

3.15. Объясните назначение сигнала READY.

3.16. Перечислите циклы шины микропроцессора 8086. Нарисуйте временные диаграммы для основных циклов.

3.17. Нарисуйте временные диаграммы цикла чтения памяти с двумя тактами ожидания.

Программная модель микропроцессора

3.18. Перечислите основные группы программно-доступных регистров микропроцессора. Определите назначения этих регистров.
Почему регистры общего назначения могут использовать либо как 16-разрядные регистры, либо как два 8-разрядных регистра? Почему регистры-указатели всегда используются только как 16-разрядные?

3.19. Что такое сегментный регистр? Перечислите сегментные регистры и укажите их назначение.

3.20. Что такое флаговый регистр (F-регистр)? Объясните назначение битов этого регистра.

3.21. Что подразумевают под физическим и логическим адресом? Что такое эффективный адрес, и как он вычисляется?

3.22. Какие средства используются в микропроцессоре для работы со стеком?

Форматы команд, способы адресации операндов

3.23. Перечислите основные режимы адресации микропроцессора. Нарисуйте диаграммы вычисления эффективного адреса для каждого режима.

3.24. Определите эффективный адрес операнда для следующих режимов адресации, если (BX) = 1234H, (DS) = 3100H, disp H, L = 1A33H: a) регистровая адресация; b) косвенная регистровая адресация; c) базовая адресация; d) прямая адресация. (Считать, что команда, если необходимо, использует регистр BX.)

3.25. Покажите структуру команды микропроцессора. Объясните назначение каждого поля.

3.26. Декодируйте следующие машинные команды микропроцессора и напишите их ассемблерные формы: a) 10001001 11000001; b) 10001000 00000000; c) 10001011 10011100 10000000 11010001.

3.27. Декодируйте следующие машинные команды микропроцессора, заданные как шестнадцатеричные числа, и напишите их ассемблерные формы: a) 81C70F30; b) 01C1; c) 02DD.

3.28. Вычислите адреса переходов в командах передачи управления для следующих режимов адресации, если (IP) = 1000H; disp = 0F2H; (BX) = 0DE01H: a) относительная адресация; b) регистровая адресация (используя BX).

Глава 4
СИСТЕМА КОМАНД 16-РАЗРЯДНОГО УНИВЕРСАЛЬНОГО МИКРОПРОЦЕССОРА

Система команд микропроцессора содержит 135 машинных команд, которые могут быть разделены на шесть категорий: команды пересылки данных, арифметические команды, команды поразрядной обработки данных, строковые команды, команды передачи управления, команды управления микропроцессором.

Команды пересылки данных предназначены для пересылки содержимого операнда-источника на место операнда-приемника. Существует четыре группы команд пересылки данных: общего назначения, ввода/вывода, логического адреса, признаков.

Арифметические команды предназначены для выполнения основных арифметических операций (сложение, вычитание, умножение и деление) над порядковыми и целыми двоичными числами, над упакованными и распакованными двоично-десятичными числами, а также для преобразования форматов данных.

Команды поразрядной обработки данных предназначены для выполнения логических операций и операций линейного и циклического сдвигов (арифметических и логических) на один или n разрядов.

Команды обработки элементов строк данных предназначены для пересылки, сравнения, записи в память, загрузки в аккумулятор элементов строк данных. Команды обработки строк совместно с префиксом повторения позволяют организовать аппаратные циклы для обработки элементов строк длиной до 64 кб.

Команды передачи управления предназначены для организации перехода в программе. Существует четыре класса таких команд: безусловная передача управления, условная передача управления, управление циклами, команды прерываний.

Команды управления микропроцессором позволяют программно управлять его различными функциями и делятся на две группы: команды управления состоянием признаков, команды синхронизации работы микропроцессора с внешними событиями.

Команды микропроцессора обеспечивают выполнение операций над одним или двумя операндами, и результат операции может записываться по адресу любого из операндов. В зависимости от типа команды операнды могут быть расположены в программно-доступных регистрах, непосредственно в коде команды, в памяти и регистрах ввода/вывода. Непосредственные данные могут быть типа байта или слова. Операнды в программно-доступных регистрах могут быть типа байта или слова, а для команд умножения и деления – типа двойного слова.

4.1. Команды передачи данных

Как любой современный процессор, МП 8086 имеет обширный набор команд, предназначенных для пересылок данных между регистрами, а также между регистрами и памятью. Команды передач данных удобно разделить на четыре подгруппы: **общие** команды **передачи данных**; команды **ввода-вывода**; команды **загрузки адресов**; команды **передачи данных из флагового регистра**. Отличительной особенностью команд передач данных является то, что они **не модифицируют состояния флажков**. Исключение составляют только команды POPF и SAHF, которые прямо воздействуют на регистр флагов.

Подгруппа общих команд передачи данных включает в себя команды, осуществляющие передачи: *регистр–регистр, регистр–память* и *память–регистр*. Наиболее мощной среди них является команда MOV (передать, переслать).

Эта команда имеет следующее обобщенное представление:

MOV dst,src; dst: = (src) .

Команда MOV осуществляет передачу содержимого источника src в получатель dst и имеет несколько форматов. Самой гибкой и универсальной является команда **MOV mem/reg,mem/reg**. Она содержит постбайт режима адресации.

Команды **MOV ac, mem** и **MOV mem, ac** предназначены для загрузки и запоминания содержимого аккумуляторов. Например, команда **MOV AL,BETA** передает в аккумулятор AL из памяти байт с адресом BETA, а команда **MOV ALPHA,AX** запоминает содержимое аккумулятора AX в слове памяти с адресом ALPHA. При выполнении последней команды содержимое регистра AL оказывается в байте с адресом ALPHA, а содержимое регистра AH – в байте с адресом ALPHA+1. Напомним, что оба байта находятся в текущем сегменте данных (если специально не определен другой сегмент) и указанные адреса представляют собой смещения в этом сегменте. Обе команды имеют длину 3 байта и выполняются за десять тактов синхронизации.

Команды **MOV sreg,mem/reg** и **MOV mem/reg,sreg** осуществляют передачи между сегментными регистрами и регистрами/памятью. В них передаются только слова, а ячейка памяти может быть определена с помощью любого допустимого режима адресации. Отметим, что в команде MOV sreg, mem/reg нельзя указывать сегментный регистр кода CS, так как при этом результат операции не определен. Обе команды имеют длину 2–4 байта в зависимости от режима адресации. Время выполнения в формате *регистр–регистр* составляет два такта синхронизации; при задании памяти первая команда выполняется за (8+EA), а вторая – за (9+EA) тактов синхронизации. Команда MOV sreg, mem/reg применяется для инициализации сегментных регистров, т. е. для определения сегментов. Если, например, в сегментный регистр DS необходимо загрузить 0F000, то потребуются команды:

```
MOV   AX,0F000H ; Инициализация
MOV   DS, AX       ; регистра DS на 0F000 .
```

Команда обмена **XCHG** имеет два формата. Первый формат позволяет обменять содержимое любого общего регистра и ячейки памяти, а также любой пары общих регистров. Ячейка памяти адресуется с использованием любого допустимого режима адресации. Команда имеет длину 2–4 байта и выполняется за четыре (обмен между регистрами) или 17+EA тактов синхронизации (обмен между регистром и ячейкой памяти). Второй формат команды XCHG предназначен для обмена содержимого любого общего регистра и аккумулятора AX. Отметим, что в команде XCHG нельзя указывать сегментные регистры. Команда XCHG AX,AX используется как команда пустой операции NOP.

Однобайтная команда преобразования **XLAT** заменяет содержимое аккумулятора AL на байт из 256-байтной таблицы, начальный адрес которой находится в регистре BX. Другими словами, содержимое AL используется как индекс таблицы, адресуемой регистром BX. Алгоритм выполнения команды XLAT состоит из двух шагов:

1) прибавить содержимое регистра AL к содержимому регистра BX;
2) использовать результат как смещение в сегменте данных (относительно DS) и поместить адресуемый байт из памяти в регистр AL.

Команда XLAT обычно применяется для быстрого преобразования символов из одного символьного кода в другой. Время ее выполнения составляет 11 тактов синхронизации.

Команды **LEA**,**LDS**,**LES** отличаются от других команд передачи тем, что при их выполнении в адресуемый регистр (регистры) передаются не собственно данные, а адреса. Основное применение команд LEA, LDS и LES – это инициализация регистров перед выполнением цепочечных команд. При выполнении команды загрузки эффективного адреса **LEA reg,mem** вычисляется эффективный адрес EA памяти (в со-

ответствии с указанным режимом адресации) и его значение (а не адресуемое им слово в памяти!) загружается в указанный общий регистр. Такая операция может потребоваться, например, для загрузки в регистр BX начального адреса таблицы, которая необходима для выполнения команды XLAT. Команды LDS и LES выполняют почти одни и те же действия: вычисляется эффективный адрес EA памяти, который суммируется с содержимым регистра DS; затем слово из памяти по полученному адресу загружается в адресуемый командой общий регистр, а следующее слово из памяти загружается в регистр DS (команда LDS) или ES (команда LES). Длина всех рассматриваемых команд может быть 2–4 байта. Отметим, что обычно в команде LDS указывается регистр SI, а в команде LES – регистр DI, так как в цепочечных операциях регистр SI ассоциируется с регистром DS, а регистр DI – с регистром ES.

Однобайтные команды **LAHF** и **SAHF** введены для упрощения программной совместимости микропроцессоров 8086 и 8080. Команда LAHF осуществляет передачу младшего байта регистра флажков в регистр AH (состояния самих флажков при этом не изменяются), а команда SAHF реализует обратную передачу – содержимое регистра AH передается в младший байт регистра флажков (старший байт регистра флажков не изменяется). Обе команды выполняются за четыре такта синхронизации. В микропроцессоре 8080 двухбайтное слово состояния процессора PSW образовано содержимым регистра флажков и аккумулятора. Оно фигурирует в двух командах: команда PUSH PSW включает слово состояния в стек, а команда POP PSW извлекает PSW из стека. Наличие в МП 8086 команд LAHF и SAHF позволяет легко эмулировать эти действия программно.

Каждая команда включения в стек **PUSH** имеет соответствующую ей команду извлечения из стека **POP**. Для адресации вершины стека, находящейся в текущем сегменте стека и содержащей последний включенный в стек элемент данных, предназначен указатель стека **SP**. Все стековые команды манипулируют только словами и сопровождаются автоматической модификацией указателя стека: при включении в стек производится декремент, а при извлечении из стека – инкремент SP. До выполнения стековых команд необходимо инициализировать регистры **SP** и **SS**. Кроме того, каждой команде **POP** должна предшествовать команда **PUSH**. Отметим, что команды вызова CALL и возврата RET используют стек автоматически. Команда PUSH часто используется для передачи через стек параметров подпрограммам. Команда **PUSH mem/reg** включает в стек содержимое адресуемого регистра или ячейки памяти, а команда **POP mem/reg** извлекает содержимое вершины стека и передает его в общий регистр или ячейку памяти. Однобайтные команды **PUSH reg** и **POP reg** производят соответствующие стековые операции с общими регистрами.

Команды **PUSH sreg** и **POP sreg** выполняют стековые операции с сегментными регистрами. Отметим, что указание в них регистра CS не допускается.

Наконец, однобайтные команды **PUSHF** и **POPF** предназначены для временного запоминания в стеке и последующего восстановления из стека содержимого 16-битного регистра флажков. С их помощью можно изменять состояние флажка TF, так как команд прямого воздействия на этот флажок нет. Изменение осуществляется с помощью включения в стек регистра флажков, необходимого изменения восьмого бита копии содержимого регистра флажков в памяти и последующего ее извлечения из стека. Последние четыре команды выполняются за то же время, что и команды PUSH reg и POP reg.

Таблица 4.1

Команды передачи данных

Мнемоника	Действие	Описание
Общего назначения		
MOV dest,src	(dest)←(src),	Передача байта или слова
PUSH src	(SP) ← (SP) – 2 ((SP) + 1:(SP))←(src)	Загрузка слова в стек
POP dest	(dest)← ((SP)+1:(SP)) (SP) ← (SP) + 2	Извлечение слова из стека
XCHG dest,src	(temp) ← (dest) (dest) ← (src) (src) ← (temp)	Обмен байтов или слов
XLAT	AL ← ((BX) + (AL))	Преобразование байта
Ввод/вывод		
IN acc, port	(AL) ← (port) or (AX) ← (port)	Ввод байта или слова
OUT port, acc	(dest) ← (AL) or (dest) ← (AX)	Вывод байта или слова
Загрузка адресов		
LEA dest, src	(dest) ← EA	Загрузка эффективного адреса
LDS dest, src	(dest) ← (EA) (DS) ← (EA + 2)	Загрузка указателя с использованием DS
LES dest, src	(dest) ← (EA) (ES) ← (EA + 2)	Загрузка указателя с использованием ES
Передача флагов		
LAHF	(AH)←(SF,ZF,AF,PF,CF)	Загрузка AH флагами
SAHF	(SF,ZF,AF,PF,CF)←(AH)	Запоминание AH во флагах
PUSHF	(SP) ← (SP) – 2 ((SP) + 1:(SP)) ← Flags	Загрузка флагов в стек
POPF	Flags ← ((SP) + 1:(SP)) (SP) ← (SP) + 2	Извлечение флагов из стека

В байте кода операции всех команд ввода-вывода находится бит w, следовательно каждая команда может передавать байты и слова. Ввод и вывод осуществляются через аккумуляторы **AL** (байты) и **AX** (слова). Двухбайтные команды **IN ac, port** и **OUT port, ac** содержат во втором байте прямой адрес порта. Команда IN загружает данные из адресуемого порта в аккумулятор, а команда OUT выполняет передачу данных в противоположном направлении. Эти команды выполняются за десять тактов синхронизации и адресуют порты в диапазоне 00–FF. Однобайтные команды **IN ac, DX** и **OUT DX, ac** также допускают передачи байт и слов, но теперь 16-битный адрес порта находится в регистре DX и максимальный адрес порта равен FFFF. Косвенная адресация через регистр DX удобна в тех случаях, когда адрес нужного порта вычисляется по ходу выполнения программы. Обе команды выполняются за восемь тактов синхронизации. В технической литературе ввод-вывод с прямыми адресами портов иногда называют статическим, а с переменными адресами – динамическим.

Команды передачи данных сведены в табл. 4.1.

4.2. Команды арифметических операций

Микропроцессор 8086 имеет широкий набор команд, реализующих арифметические операции, что позволяет применять его в сложных системах обработки данных. Арифметические операции выполняются над целыми числами четырех типов: беззнаковыми двоичными, знаковыми двоичными, упакованными десятичными и неупакованными десятичными. Сначала рассмотрим операции над двоичными числами, а десятичную арифметику рассмотрим отдельно. Длина беззнаковых двоичных чисел составляет 8 или 16 бит, и все они считаются значащими, т. е. учитываются при определении значения числа. Диапазон значений 8-битных чисел составляет 0–255, а 16-битных – 0–65535. Имеются команды сложения, вычитания, умножения и деления чисел, представленных в таких форматах. Знаковые двоичные числа также могут быть 8- и 16-битными. Старший (левый) бит определяет знак числа: 0 – число положительное, 1 – число отрицательное. Числа представляются в стандартном дополнительном коде. Диапазон значений составляет –128…+127 для 8-битных чисел и –32768…+32767 – для 16-битных чисел. Число нуль содержит во всех разрядах нули и считается положительным. Для знаковых двоичных чисел имеются специальные команды умножения и деления, а сложение и вычитание, благодаря применению дополнительного кода, реализуются теми же командами, что и для беззнаковых чисел. При выполнении арифметических операций особенности (или признаки) полученного результата фиксируются в 6 битах ре-

гистра флажков. Состояния большинства флажков проверяются командами условных переходов; имеется также специальная команда INTO прерывания при переполнении. Команды арифметических операций влияют на разные флажки, но в соответствии с общими правилами.

1. Флажок **CF** фиксирует значение переноса (при сложении) и заема (при вычитании) из старшего бита; его можно использовать для обнаружения переполнения при сложении беззнаковых двоичных чисел.
2. Флажок AF фиксирует значение переноса (при сложении) и заема (при вычитании) из младшей тетрады; этот флажок введен только для команд десятичной коррекции и для других целей не используется.
3. Флажок SF соответствует значению старшего бита (бит 7 или бит 15) результата операции; для чисел со знаком отражает знак результата, для беззнаковых – интерпретируется как цифра, а не как знак.
4. Флажок ZF фиксирует получение нулевого результата.
5. Флажок PF устанавливается в 1, если младшие 8 бит результата операции содержат четное число единиц; в противном случае он сбрасывается в 0 (наличие флажка PF упрощает реализацию функций контроля символьных данных).
6. Флажок OF в операциях со знаковыми числами фиксирует арифметическое переполнение, т. е. выход результата за диапазон представимых значений (при использовании дополнительного кода переполнение определяется сложением по модулю 2 значений переносов из знакового бита и старшего значащего бита).

Арифметические команды приведены в табл. 4.2. Все команды, кроме команды INC reg, имеют в коде операции бит w, следовательно могут оперировать байтами и словами.

Общее представление команды сложения имеет вид

ADD dst, src; dst: = (dst) + (src).

Команда производит сложение операндов dst и src и помещает сумму на место dst.

Команды сложения с переносом имеют обобщенное представление в виде

ADC dst,src; dst: = (dsf) + (src) + (CF).

Они имеют такие же форматы, как и команды ADD, и выполняются за такое же время. Единственное отличие их заключается в том, что в сложении наряду с операндами участвует флажок CF, значение которого прибавляется к младшему биту результата сложения операндов.

Команда инкремента имеет обобщенное представление:

INC dst; dst: = (dst)+1.

Она позволяет увеличить на единицу содержимое любого общего регистра или ячейки памяти. Команда INC воздействует на все арифметические флажки, за исключением флажка CF – его состояние не изменяется. Операнд dst считается беззнаковым двоичным числом.

Команды вычитания отличаются от соответствующих команд сложения только выполняемой операцией.

Команда декремента предназначена для уменьшения на единицу содержимого регистра или ячейки памяти. Она не влияет на состояние флажка переноса CF.

Команда **NEG** изменения знака (или образования дополнительного кода) имеет следующее общее представление: **NEG dst;** dst = 0 – (dst). Если операнд равен нулю, его значение не изменяется. Попытка изменить знак байта 80 или слова 8000 не модифицирует операнд, но устанавливает флажок переполнения OF. Команда влияет на все флажки, причем флажок CF всегда устанавливается в единицу, кроме случая, когда операнд равен нулю, тогда флажок CF = 0.

Команды сравнения очень похожи на соответствующие команды вычитания. Единственное их отличие заключается в том, что результат вычитания операндов нигде не запоминается. Следовательно, данные команды производят так называемое неразрушающее сравнение операндов. Состояния всех флажков определяются получающимся результатом и могут быть (кроме флажка AF) проверены командами условного перехода.

Таблица 4.2

Арифметические команды

Сложение		
ADD dest, src	(dest) ← (dest) + (src)	Сложение байтов или слов
ADC dest, src	(dest) ← (dest) + (src) + (CF)	Сложение с переносом
INC dest	(dest) ← (dest) + 1	Инкремент байта или слова
AAA	If ((AL) and 0FH) > 9 or (AF) = 1 then (AL) ← (AL) + 6 (AH) ← (AH) + 1 (AF) ← 1 (CF) ← (AF) (AL) ← (AL) and 0FH	ASCII-коррекция при сложении
DAA	**If ((AL) and 0FH) > 9 or (AF) = 1 then (AL) ← (AL)+6; (AF)←1** if (AL) > 9FH or (CF) = 1 then (AL) ← (AL) + 60H, (CF)←1	Десятичная коррекция при сложении

Продолжение табл. 4.2

Вычитание		
SUB dest, src	(dest) ← (dest) – (src)	Вычитание байта или слова
SBB dest, src	(dest) ← (dest) – (src) – (CF)	Вычитание с заемом
DEC dest	(dest) ← (dest) – 1	Декремент байта или слова
NEG dest	Если операнд – байт, то (dest) ← 0FFH – (dest) (dest) ← (dest) + 1, если операнд – слово: (dest) ← 0FFFFH – (dest) (dest) ← (dest) + 1	
CMP dest, src	(dest) – (src)	Сравнение байтов или слов
AAS	If ((AL) and 0FH) > 9 or (AF) = 1 then (AL) ← (AL) – 6 (AH) ← (AH) – 1 (AF) ← 1 (CF) ← (AF) (AL) ← (AL) and 0FH	ASCII-коррекция при вычитании
DAS	If ((AL) and 0FH) > 9 or (AF) = 1 then (AL) ← (AL) – 6; (AF)←1 if (AL) > 9FH or (CF) = 1 then (AL) ← (AL) – 60H, (CF)←1	Десятичная коррекция для вычитания
Умножение		

MUL src	Если src – байт:
	(AX) ← (AL) × (src)
	If (AH) = 0
	then (CF) ← 0
	else (CF) ← 1 (OF) ← (CF)
	Если src – слово:
	(DX:AX) ← (AX) × (src)
	If (DX) = 0
	then (CF) ← 0
	else (CF) ← 1 (OF) ← (CF)
		Умножение байта или слова без знака

IMUL src	Если src – байт: (AX) ← (src) × (AL) Если src – слово: (DX:AX) ← (src) × (AX)	Знаковое умножение байта или слова
AAM	(AH) ← (AL) / 0AH (AL) ← (AL) *mod* 0AH	ASCII-коррекция для умножения

Окончание табл. 4.2

Деление		
DIV src	Если src – байт: (AL) ← (src) / (AX) (AH) ← (src) *mod* (AX) Если src – слово: (AX) ← (src) / (DX:AX) (DX) ← (src) *mod* (DX:AX)	Деление байта или слова без знака
IDIV src	Если src – байт: (AL) ← (src) / (AX) (AH) ← (src) *mod* (AX) Если src – слово: (AX) ← (src) / (DX:AX) (DX) ← (src) *mod* (DX:AX)	Деление байта или слова со знаком
AAD	(AL) ← (AH) × 0AH + (AL) (AH) ← 0	ASCII-коррекция при делении

В МП 8086 имеются две команды умножения: для беззнаковых и знаковых двоичных чисел. Умножение десятичных чисел требует наличия специальных команд коррекции. Команда умножения беззнаковых целых выполняет умножение адресуемого операнда и содержимого аккумулятора. В операции над байтами функции аккумулятора выполняет регистр AL, а 16-битное произведение образуется в регистрах AH–AL. Регистр AH называется расширением (ext) аккумулятора AL. Если src идентифицирует слово, оно умножается на содержимое аккумулятора AX, а произведение длиной 32 бита формируется в регистрах DX–AX. В этой операции расширением аккумулятора AX является регистр DX. Когда старшая половина произведения отличается от нулевой, флажки OF и CF устанавливаются в единицу, показывая наличие значащих цифр произведения в регистрах AH и DX. В противном случае флажки OF и CF принимают нулевые значения. Состояния остальных флажков после выполнения команды MUL не определены. Команда **IMUL src;** осуществляет практически такие же действия, что и команда MUL src; но сомножители и произведение интерпретируются как знаковые двоичные числа в дополнительном коде. Если старшая половина произведения, находящаяся в регистрах AH или DX, не является расширением знака младшей половины произведения (не содержит 00 (0000) или FF (FFFF) при умножении байт (слов)), флажки OF и CF устанавливаются в единицу. Это означает, что в старшей половине находятся значащие цифры произведения. В противном случае OF, CF = 0. Состояния остальных флажков после выполнения команды IMUL не определены.

В МП 8086 имеются две команды деления, операндами которых являются беззнаковые и знаковые двоичные числа. Команда деления беззнаковых чисел производит деление аккумулятора и его расширения (AH–AL, DX–AX для 8- и 16-битного делителя, соответственно) на содержимое src. Частное формируется в регистре AL (или AX), а остаток – в регистре AH (или DX). Дробное частное округляется до целого путем отбрасывания дробной части результата. Состояния всех флажков не определены. Если частное превышает разрядность аккумулятора (больше FF или FFFF) или делитель является нулем, генерируется прерывание типа 0, а частное и остаток не определены. При возникновении прерывания выполняются следующие действия: содержимое регистра флажков включается в стек; флажки IF и TF сбрасываются в 0; содержимое сегментного регистра CS включается в стек; в регистр CS загружается слово из памяти по адресу 00002, содержимое PC включается в стек; в PC загружается слово из памяти по адресу 00000. В результате этих действий МП переходит к подпрограмме обработки прерывания типа 0, полный адрес которой сегмент:смещение берется из ячеек 00000 и 00002. Если генерирование прерывания типа 0 нежелательно (например, при отсутствии соответствующей подпрограммы обработки), необходимо до выполнения операции деления проверить возможность возникновения прерывания.

Команда IDIV осуществляет почти такие же действия, как и команда DIV, но делимое и делитель считаются знаковыми числами. Диапазон представления частного составляет –127…+127, когда делителем является байт, и –32767…+32 767, когда делитель – слово. При выходе частного за эти диапазоны и при попытке деления на нуль генерируется прерывание типа 0. Состояния флажков после выполнения команды деления IDIV неизвестны.

Микропроцессор 8086 допускает два представления десятичных чисел: упакованный формат (BCD-формат) и неупакованный (ASCII-формат). В BCD-формате байт содержит две десятичные цифры (по одной в каждой тетраде), представленные в коде 8421; в ASCII-формате байт содержит одну десятичную цифру (в младшей тетраде), а старшая тетрада содержит либо 0011 (символьный код ASCII), либо нули.

Сложение BCD-чисел выполняется в два этапа: сначала байты операндов суммируются, как обычные двоичные числа, по правилам двоичной арифметики, а затем осуществляется коррекция результата. Анализ двоичного сложения BCD-чисел показывает, что неправильный BCD-результат появляется в двух ситуациях: получена недопустимая

тетрада, т. е. тетрада, двоичный эквивалент которой больше 9; получена допустимая тетрада, но при сложении из нее возник двоичный перенос с весом 16, в то время как правильный вес единицы переноса должен быть равен 10. Отметим, что перенос из младшей тетрады фиксируется флажком AF, а из старшей – флажком CF. Коррекция двоичной суммы BCD-чисел, полученной с помощью команд ADD или ADC и находящейся в регистре AL, выполняется однобайтной командой десятичной коррекции DAA. Алгоритм коррекции состоит из двух шагов:

1) если AF = 1 или младшая тетрада регистра AL содержит запрещенную комбинацию (в диапазоне чисел 10…15), то к содержимому AL прибавляется 06H и флажок AF устанавливается в единицу;
2) если CF = 1 или старшая тетрада регистра AL содержит запрещенную комбинацию, то к содержимому AL прибавляется 60H и флажок CF устанавливается в единицу.

Команда DAA, в соответствии с полученным в регистре AL результатом, воздействует на все флажки, за исключением флажка переполнения OF, состояние которого после выполнения команды DAA не определено.

Вычитание BCD-чисел, как и сложение, выполняется в два этапа: сначала операнды вычитаются как двоичные числа с помощью команды SUB или SBB, а затем результат, находящийся в регистре AL, корректируется командой DAS десятичной коррекции для вычитания. Так как в МП 8086 операция вычитания выполняется путем сложения уменьшаемого и дополнительного кода вычитаемого, действия команды DAS описываются следующим образом:

1) если AF = 1 или младшая тетрада регистра AL содержит запрещенную тетраду, то из содержимого регистра AL вычитается 06 и флажок AF устанавливается в единицу;
2) если CF = 1 или старшая тетрада регистра AL содержит запрещенную тетраду, то из содержимого регистра AL вычитается 60 и флажок CF устанавливается в единицу.

Пример. Вычислить произведение N-разрядного неупакованного двоично-десятичного числа, расположенного в памяти (адрес первой цифры в регистре DI), и одноразрядного двоичного числа, расположенного в регистре DX. Результат поместить в память (адрес первого байта в регистре SI).

```
       MOV CX,N        ; В CX занести количество цифр
       MOV [DI],0      ; Сбросить первый байт произведения
       AND DL,OFH      ; Скорректировать множитель
MULT:  MOV AL,[SI]     ; Очередная цифра множимого
       INC SI          ; Продвинуть указатель
       AND AL,OFH      ; Сбросить старшую тетраду
       MUL DL          ; Умножить
       AAM             ; Скорректировать произведение
       ADD AL,[DI]     ; Прибавить в произведение
       AAA             ; Скорректировать сумму
       MOV [DI],AL     ; Запомнить результат
       INC DI          ; Продвинуть указатель
       MOV [DI],AH     ; Старшая цифра произведения
       DEC CX          ; Проверить окончание умножения
       JNZ MULT        ;
```

4.3. Команды логических операций и команды сдвигов

Логические операции, реализуемые в МП 8086, представлены булевыми операторами **NOT** (инверсия), **AND** (конъюнкция), **OR** (дизъюнкция), **XOR** («Исключающее ИЛИ», т. е. сложение по модулю 2) и командой **TEST**, которая выполняет конъюнкцию операндов, но не изменяет их значений (неразрушающая проверка). Все логические операции являются поразрядными, т. е. выполняются независимо для всех бит операндов. Команды данной группы описаны в табл. 4.3. Обобщенное представление команд логических операций имеет следующий вид:

AND dst, src; dst: = (dst)^ (src)
OR dst, src; dst: == (dst)\/ (src)
XOR dst, src; dst: = (dst)@ (src)
TEST dst, src; $(\underline{\text{dst}}) \wedge (\text{src})$
NOT src; src: = $(\overline{\text{src}})$.

Унарная команда инверсии NOT не влияет на состояние флажков. Бинарные команды AND, OR, XOR и TEST воздействуют на арифметические флажки следующим образом:

- флажки OF и CF всегда переводятся в нулевое состояние, так как межразрядные связи при выполнении операций отсутствуют;
- состояния флажков SF, ZF и PF зависят от полученного результата и определяются по тем же правилам, что и в командах арифметических операций; состояние флажка AF не определено.

Команда поразрядной конъюнкции AND в основном применяется для перевода в нулевое состояние тех бит операнда, которые определяются другим операндом – маской. Маска должна содержать нули в сбрасываемых битах и единицы в остальных.

Команда поразрядной дизъюнкции OR применяется для установки в 1 определенных бит операнда с помощью маски, а также упаковки байт или слов из полей других элементов данных.

С помощью команды «Исключающего ИЛИ» XOR можно инвертировать определенные биты операнда, сравнивать операнды на абсолютное равенство и переводить регистр в нулевое состояние.

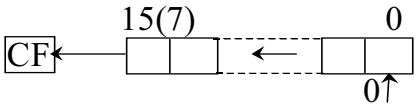
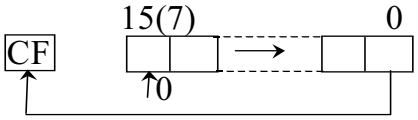
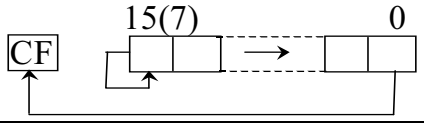
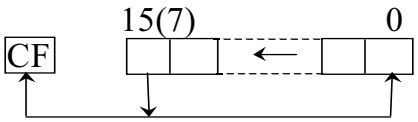
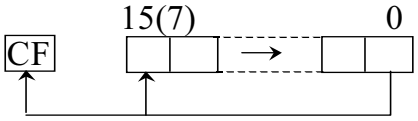
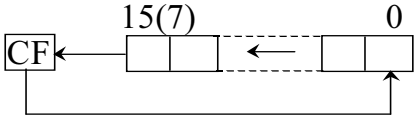
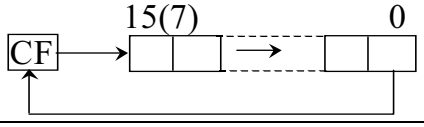
Коды операций всех команд сдвигов содержат бит w, и, следовательно, имеется возможность сдвигать байты и слова. Поле операнда имеет вид mem/reg, count. Здесь mem/reg стандартным образом адресует общий регистр или ячейку памяти, а count (счет или счетчик) определяет число сдвигов. Этот операнд может быть указан как константа 1 (статический сдвиг) или как регистр CL. В первом случае осуществляется сдвиг на 1 бит, а во втором – число сдвигов определяется содержимым регистра CL, которое должно являться беззнаковым целым двоичным числом. Таким образом, число сдвигов можно задать переменной, вычисляемой во время выполнения программы (так называемый динамический сдвиг). Выбор константы 1 или регистра CL идентифицирует бит в коде операции: если v = 0, то count = 1, а если v = l, то count = (CL).

Команды сдвигов подразделяются на команды циклических сдвигов (ротаций) и обычных. В циклических сдвигах выдвигаемый бит помещается на место освобождающегося бита.

При выполнении команд сдвигов флажки модифицируются следующим образом: состояние флажка AF всегда не определено; флажок CF всегда содержит значение последнего выдвинутого бита в однобитных сдвигах флажок OF = 1, если операция изменила значение старшего (знакового) бита операнда; при сдвиге на несколько бит состояние флажка OF не определено; циклические сдвиги влияют только на флажки OF и CF; в обычных сдвигах флажки SF, ZF и PF модифицируются в соответствии с полученным результатом. Команды RCL и RCR называются командами циклического сдвига влево и вправо через перенос, так как флажок CF расширяет сдвигаемый операнд на 1 бит. В МП 8086 можно выполнить сдвиги содержимого любого общего регистра и байта или слова в памяти на любое число бит, вплоть до максимума, равного 255.

Таблица 4.3

Команды логических операций и сдвигов

Мнемоника	Действие	Описание
Логические команды		
NOT dest	Если операнд – байт: (dest) ← FFH – (dest) Если операнд – слово: (dest) ← FFFFH – (dest)	Инверсия байта или слова
AND dest, src	(dest) ← (dest) and (src) (CF) ← 0; (OF) ← 0	Операция «И» над байтом или словом
OR dest, src	(dest) ← (dest) or (src) (CF) ← 0; (OF) ← 0	Операция «ИЛИ» над байтом или словом
XOR dest, src	(dest) ← (dest) xor (src) (CF) ← 0; (OF) ← 0	Операция «Исключающее ИЛИ» над байтом или словом
TEST dest, src	(dest) and (src) (CF) ← 0; (OF) ← 0	Проверка байта или слова
Сдвиги		
SHL dest, count SAL dest, count		Логический/арифметический сдвиг влево
SHR dest, count		Логический сдвиг вправо
SAR dest, count		Арифметический сдвиг вправо
Вращения		
ROL dest, count		Циклический сдвиг влево
ROR dest, count		
RCL dest, count		Циклический сдвиг влево через перенос
RCR dest, count		Циклический сдвиг вправо через перенос

Команды SHL и SHR реализуют логический сдвиг влево и вправо, соответственно. Для логического сдвига характерно, что в освобождающийся бит загружается нуль, а выдвигаемый бит теряется. Команды SAL и SAR предназначены для арифметического сдвига влево и вправо. Арифметический сдвиг вправо отличается от логического сдвига тем, что знаковый бит не сдвигается, а тиражируется в соседнем правом бите, сохраняя знак числа. Арифметический сдвиг влево в дополнительном коде не отличается от логического сдвига. Поэтому мнемоники SAL и SHL обозначают одну и ту же машинную команду и введены для удобства программирования.

Команды арифметического сдвига по существу реализуют умножение и деление чисел на два в некоторой целой степени. Однако для отрицательных чисел команда SAR не дает такого же результата, как деление командой IDIV.

4.4. Команды передачи управления

При выполнении линейных программных фрагментов операционное устройство выбирает байты из очереди команд и интерпретирует их в соответствии с семантикой команд. Как только в очереди появляются два свободных байта, устройство шинного интерфейса инициирует цикл шины, выбирает из программной памяти очередное слово и помещает его в очередь команд. При этом соответственно корректируется программный счетчик PC. Такие действия соответствуют естественному порядку выполнения команд. Физический адрес программной памяти определяется содержимым сегментного регистра кода CS и программного счетчика PC.

На практике обойтись только линейными программами невозможно. В разветвляющихся и циклических программах, а также при организации подпрограмм необходимо выполнять не следующую по порядку команду, а команду, находящуюся в другой ячейке программной памяти и определяемую адресом перехода. Специальные команды, которые модифицируют указатели программной памяти (регистры PC и CS), называются командами передачи управления. В системе команд МП 8086 имеется обширный набор таких команд. Данные команды не изменяют состояний флажков, за исключением команды возврата из прерывания IRET.

Сегментная организация программной памяти определяет две основные разновидности команд передачи управления. Передача управления в пределах текущего сегмента кода (программы) называется внутрисегментной – при этом модифицируется только PC и адрес перехода представляется одним словом. Такая передача управления называется еще близкой (тип NEAR). Передача управления за пределы текущего сегмента кода называется межсегментной – при этом необходимо мо-

дифицировать содержимое регистров PC и CS и адрес перехода представляется двумя словами (сегмент: смещение). Данная передача управления называется еще далекой (тип FAR), так как она позволяет перейти к любой ячейке адресного пространства памяти.

Команды передачи управления 8086 подразделяются на команды безусловных переходов, условных переходов, вызовов подпрограмм, возвратов, управления циклами и команды прерываний (табл. 4.4).

Таблица 4.4

Команды передачи управления

Условные переходы		
Jcond disp8	If cond = 1 then (IP)←(IP) + disp8	Переход, если проверяемое условие истинно
Безусловные переходы		
CALL dest dest может быть регистром или указателем на ячейку памяти или непосредственным смещением	If inter-segment then (SP) ← (SP) – 2 ((SP)+1:(SP))←(CS) (CS) ← SEG (SP) ← (SP) – 2 ((SP) +1:(SP)) ← (IP) (IP) ← dest	Вызов процедуры
RET		Возврат из процедуры
JMP dest	If inter-segment then (CS) ← SEG (IP) ← dest	Переход
Управление циклами		
LOOP disp8	(CX) ← (CX) – 1 if (CX) ≠ 0 then (IP) ← (IP) + disp8	Цикл
LOOPE/LOOPZ disp8	(CX) ← (CX) – 1 if (ZF) = 1 and (CX) ≠ 0 then (IP) ← (IP) + disp8	Цикл, если равно/нуль
LOOPNE/LOOPNZ disp8	(CX) ← (CX) – 1 if (ZF) = 0 and (CX) ≠ 0 then (IP) ← (IP) + disp8	Цикл, если не равно/не нуль
JCXZ disp8	If (CX) = 0 then (IP) ← (IP) + disp8	Переход, если регистр CX = 0
Прерывания		
INT num		Прерывание с заданным номером
INTO		Прерывание, если переполнение
IRET		Возврат из прерывания

При выполнении команд **безусловных переходов** происходит модификация PC или PC и CS, а их прежнее содержимое теряется.

Двухбайтная команда JMP dispL содержит во втором байте смещение, которое интерпретируется как знаковое целое. При выполнении команды значение смещения прибавляется (с расширением знака до 16 бит) к содержимому PC, которое соответствует адресу команды, находящейся после команды JMP. Диапазон значений байта смещения составляет –128…+127. Если смещение положительное, осуществляется переход вперед, а если отрицательное – переход назад. Данная команда удобна для организации коротких программных циклов. Относительная адресация обеспечивает позиционную независимость, так как значение смещения не зависит от положения программы в памяти.

Трехбайтная команда JMP disp производит такое же действие, как предыдущая команда, но содержит 16-битное смещение. Оно попрежнему интерпретируется как знаковое целое, поэтому область перехода увеличивается до –32768…+32767 байт относительно адреса команды, находящейся после команды JMP disp.

Команда JMP mem/reg реализует косвенный безусловный переход в программе. Здесь адресом перехода, загружаемым в PC, служит содержимое 16-битного общего регистра или слова памяти, определяемое постбайтом режима адресации. Отметим, что необязательные байты dispL, dispH в команде относятся к режиму адресации памяти.

В системе команд 8086 есть 19 двухбайтных команд **условных переходов**, называемых также разветвлениями. Все они имеют единый формат. При выполнении этих команд анализируется некоторое условие, закодированное текущими состояниями флажков (а в команде JCXZ – содержимым регистра CX), и в зависимости от удовлетворения условия переход осуществляется или нет. Данные команды позволяют проверить оба состояния всех флажков арифметических операций (кроме флажка AF), а также ряд комбинаций состояний нескольких флажков. Если условие истинно, управление передается по адресу перехода путем прибавления к содержимому PC однобайтного знакового смещения (с расширением знака до 16 бит), а если условие ложно, выполняется следующая по порядку команда. Таким образом, все условные переходы являются короткими. Двухбайтные команды условных переходов обеспечивают диапазон переходов –128…+127 байт. Обычно этого диапазона достаточно, но иногда требуется условно перейти к метке, находящейся вне указанного диапазона. Эту проблему можно решить двумя способами. В обоих

способах используется расширенный диапазон переходов команды JMP, равный –32768…+32 767 байт.

В первом способе, называемом векторным переходом, условный переход осуществляется к метке промежуточной команды JMP, которая и передает управление в нужную точку. Естественно, при этом способе предполагается наличие места для команды JMP в диапазоне условного перехода. Удобно размещать эту промежуточную команду сразу после другой команды JMP или после команды RET.

Команду JCXZ удобно помещать в начале цикла, особенно в том случае, если возможна ситуация, при которой цикл (со счетчиком CX) не выполняется ни разу. При программировании может возникнуть необходимость передачи управления за пределы действия команд условного перехода. Пусть, например, при ZF = 1 необходимо передать управление команде с меткой MORE, которая находится через 0400 байт от данной точки в программе. В этом случае приходится использовать две команды:

```
JNZ     NEXT     ; Флажок ZF = 0
JMP     MORE     ; Флажок ZF = 1
NEXT:
```

Ассемблер сформирует команду JMP с двухбайтным смещением.

Таблица 4.5

Интерпретация условных переходов

Мнемоника	Проверяемое условие	“Переход, если…”
JA/JNBE	(CF or ZF) = 0	выше/не ниже и не равно
JAE/JNB	CF = 0	выше или равно/не ниже
JB/JNAE	CF = 1	ниже/не выше и не равно
JBE/JNA	(CF or ZF) = 1	ниже или равно/не выше
JC	CF = 1	перенос
JE/JZ	ZF = 1	равно/нуль
JG/JNLE	((SF xor OF) or ZF) = 0	больше/не меньше и не равно
JGE/JNL	(SF xor OF) = 0	больше или равно/не меньше
JL/JNGE	(SF xor OF) = 1	меньше/не больше и не равно
JLE/JNG	((SF xor OF) or ZF) = 1	меньше или равно/не больше
JNC	CF = 0	нет переноса
JNE/JNZ	ZF = 0	не равно/не нуль
JNO	OF = 0	нет переполнения
JNP/JPO	PF = 0	нет паритета/паритет нечетный
JNS	SF = 0	знак положительный
JO	OF = 1	переполнение
JP/JPE	PF = 1	паритет/паритет четный
JS	SF = 1	знак отрицательный

Следует отметить, что большинство команд условных переходов имеет две (и даже три) мнемоники, подчеркивающие содержательный смысл проверяемого условия и введенные для удобства программирования.

Команды позволяют проверить все отношения между знаковыми и беззнаковыми числами. Фигурирующие в определении команд термины «больше» и «меньше» относятся к знаковым числам, представленным в дополнительном коде, а «выше» и «ниже» – к беззнаковым.

Команда **вызова подпрограммы CALL** передает управление с автоматическим сохранением адреса возврата. В поле операнда этой команды находится метка первой команды вызываемой подпрограммы. При переходе к подпрограмме необходимо временно запомнить адрес команды, находящейся после команды CALL. Этот адрес называется адресом возврата. После того как подпрограмма закончит свои действия, завершающая ее команда возврата RET передает управление по запомненному адресу возврата. Удобным местом хранения адресов возвратов, который обеспечивает очень простую организацию вложенных подпрограмм, является стек.

Сегментная организация памяти МП 8086 влияет на реализацию команд вызовов. Как и команды безусловного перехода, вызовы могут быть внутрисегментными и межсегментными. В первом случае вызываемая подпрограмма находится в текущем сегменте кода (тип NEAR), а во втором – в произвольном (тип FAR). В соответствии с этим, в стеке приходится запоминать содержимое либо только PC, либо PC и CS.

Как видно, команды CALL имеют такие же форматы, как и команды безусловных переходов; отсутствует только формат CALL dispL. По воздействию на регистры PC и CS команды CALL также соответствуют командам JMP, но дополнительно они запоминают в текущем сегменте стека (адресуемом регистром SS) адрес возврата с соответствующей модификацией указателя стека SP. Напомним, что включение в стек сопровождается декрементом указателя стека и что SP адресует последние включенные в стек данные.

В МП 8086 нет команд условных вызовов подпрограмм, и поэтому при необходимости механизм условных вызовов реализуется двумя командами.

Команда прямого межсегментного вызова **CALL addr** имеет длину 5 байт и позволяет вызвать подпрограмму, находящуюся в любой области адресного пространства памяти. Этой командой выполняются следующие действия: содержимое SP уменьшается на 2; в адресуемую регистрами SP и SS ячейку памяти пересылается содержимое CS; содержимое SP уменьшается на 2; в адресуемую регистрами SP и SS ячейку

памяти записывается содержимое PC; в PC загружается смещение, в CS загружается сегментный адрес seg.

Команда **CALL mem** осуществляет косвенный межсегментный вызов подпрограмм через память. Текущее содержимое регистров PC и CS запоминается в стеке, после чего слово из адресуемой ячейки памяти загружается в PC, а следующее слово – в регистр CS.

Однобайтная команда RET реализует **внутрисегментный возврат**. Ее действия заключаются в том, что верхнее слово стека передается в PC, а содержимое указателя стека SP увеличивается на 2.

Три команды **управления циклами** (или итерациями) применяются для организации программных циклов. В них предусматривается использование регистра CX в качестве счетчика цикла. Второй байт интерпретируется как знаковое целое и при необходимости передачи управления в начале цикла прибавляется к содержимому PC. Следовательно, диапазон переходов этих команд составляет –128… +127 байт от следующей команды. В ассемблерных программах поле операнда команд управления циклами содержит метку первой команды цикла. Метка должна находиться в диапазоне переходов, как и во всех командах условной передачи управления.

Когда выполняется команда LOOP (повторить цикл), производится декремент регистра CX и, если (CX) ≠ 0, смещение прибавляется к PC – происходит переход к началу цикла; в противном случае ((CX) = 0 и цикл окончен) выполняется следующая по порядку команда. Другими словами, команда LOOP MORE эквивалентна двум командам:

```
DEC     CX
JNZ     MORE.
```

Следовательно, данную пару команд можно заменить одной командой LOOP. Каждая такая замена экономит байт объектного кода и один такт, поскольку время выполнения команды LOOP составляет 5 (переход не происходит) или 17 (переход происходит) тактов синхронизации. Несущественное на первый взгляд ускорение выполнения операции проверки окончания цикла может оказаться заметным при большом числе повторений цикла.

Мнемоники LOOPE и LOOPZ определяют одну и ту же машинную команду, которая производит декремент регистра CX, а затем передает управление в начало цикла (по-прежнему путем прибавления смещения к PC), если (CX) ≠ 0 и ZF = 1. В противном случае будет выполняться следующая по порядку команда. Следовательно, по сравнению с командой LOOP данная команда вводит дополнительное условие повторения цикла – единичное состояние флажка ZF. Время выполнения составляет 6 или 18 тактов синхронизации.

Мнемоники LOOPNE и LOOPNZ также определяют одну и ту же машинную команду. Действия и время ее выполнения аналогичны команде LOOP, но дополнительным условием перехода к началу цикла является нулевое состояние флажка ZF.

В микропроцессоре 8086 имеются три команды, относящиеся к прерываниям. Первые две команды позволяют вызвать подпрограмму обработки прерывания примерно так же, как это делают аппаратные (внешние) запросы прерываний по входу INT, когда они разрешены, или запросы немаскируемых прерываний по входу NMI. Реагируя на программное прерывание, микропроцессор не выполняет цикла подтверждения прерывания.

Команда **INT type** вызывает подпрограмму обработки, определяемую типом прерывания. Тип прерывания зависит от значения бита v в коде операции, который, в свою очередь, зависит от того, имеется или отсутствует операнд в ассемблерной записи команды INT. Если v = 0, второй байт команды отсутствует и тип прерывания принимается равным трем, – это прерывание контрольной точки, или контрольного останова. Если v = l, тип прерывания задается вторым байтом команды и может принимать значение от 0 до 255.

Выполнение команды **INT** инициирует следующую последовательность действий: декремент указателя стека на 2; включение в стек содержимого регистра флажков (как в команде PUSHF); сброс флажков IF и TF (запрещение восприятия прерываний и покомандной работы); декремент указателя стека на 2; включение в стек содержимого регистра CS; определение значения ADDRESS путем умножения кода типа прерывания на 4; загрузка в регистр CS слова памяти по адресу ADDRESS+2; декремент указателя стека на 2; включение в стек содержимого PC; загрузка в PC слова памяти по адресу ADDRESS. В результате этих действий осуществляется межсегментный косвенный вызов подпрограммы обработки прерывания через память, причем адрес памяти однозначно определяется типом прерывания.

Команду прерывания можно использовать для вызова супервизора, т. е. как запрос на обслуживание операционной системой. Для каждого вида обслуживания, которое требуется от операционной системы, в прикладной программе определяется свой собственный тип прерывания. Кроме того, программное прерывание применяется для отладки подпрограмм обслуживания прерываний от периферийных устройств.

Однобайтная команда INT (тип прерывания равен 3) используется в процессе отладки прикладных программ. Вызываемая ею подпрограмма по адресу 000CH обычно является частью пакета отладочных программ.

Команда программного прерывания короче команды CALL межсегментного вызова, и, кроме того, она запоминает в стеке содержимое регистра флажков, что часто требуется для подпрограмм обслуживания прерывания. При этом вызванная подпрограмма обязательно должна заканчиваться командой возврата из прерывания IRET.

Однобайтная команда возврата из прерывания IRET предназначена для выхода из подпрограмм обработки прерываний, инициированных аппаратно или программно. По существу действия команды IRET противоположны действиям команды INT: слово из вершины стека передается в PC; производится инкремент SP на 2; слово из вершины стека извлекается в CS; производится инкремент SP на 2; слово из вершины стека передается в регистр флажков; производится инкремент SP на 2.

4.5. Цепочечные команды

Под цепочкой понимается последовательность любых контекстно связанных байт или слов, находящихся в смежных ячейках памяти. В системе команд МП 8086 имеется пять однобайтных команд, предназначенных для обработки одного элемента цепочек. Цепочечной команде может предшествовать специальный однобайтный префикс повторения REP, который вызывает повторение действия команды над следующим элементом. Благодаря такому префиксу повторения цепочки данных обрабатываются значительно быстрее, чем при организации программного цикла. Повторение рассчитано на максимальную длину цепочки 64 кб и может заканчиваться по нескольким условиям. Кроме того, повторяющуюся операцию можно прерывать и возобновлять. Команды могут иметь операнд-источник, операнд-получатель или то и другое одновременно. Подразумевается, что цепочка-источник по умолчанию находится в текущем сегменте данных, но допускается префикс замены сегмента. Цепочка-получатель должна находиться только в текущем дополнительном сегменте. Ассемблер не использует операнды команд для адресации цепочек. Вместо этого содержимое регистра SI всегда считается смещением текущего элемента цепочки-источника, а содержимое регистра DI – смещением текущего элемента цепочки-получателя. Эти регистры необходимо соответственно инициализировать до выполнения цепочечной команды с помощью команд загрузки адреса LEA, LDS и LES.

При выполнении цепочечной команды содержимое регистров SI и DI автоматически модифицируется, чтобы адресовать следующие элементы цепочек. Флажок направления DF определяет автоинкремент (DF = 0) или автодекремент (DF = 1) индексных регистров.

Если команде предшествует префикс повторения, то после каждого ее выполнения производится декремент регистра-счетчика CX, поэтому

его необходимо предварительно инициализировать на требуемое число повторений. Когда содержимое CX достигает нуля, управление передается следующей команде.

Собственно цепочечная команда оперирует одним элементом цепочки. Повторяющееся (циклическое) выполнение команды обеспечивает однобайтный префикс повторения REP. Наличие этого префикса позволяет, например, передать одной командой цепочку произвольной длины (не превышающей максимальной) из одной области памяти в другую при соответствующей инициализации регистров DS:SI, ES:DI и CX.

Префикс повторения имеет пять мнемокодов: REP, REPE, REPZ, REPNE и REPNZ. Они определяют только два объектных кода префикса и введены для лучшей передачи содержательного смысла команды. Префикс повторения не влияет на состояние флажков. Префикс REP используется с командами MOVS и STOS и инициирует действие "повторять, пока не достигнут конец цепочки", т. е. до тех пор, пока содержимое регистра CX не достигнет нуля. Префиксы REPE и REPZ действуют аналогично и представляют собой такой же байт, как и префикс REP. Они используются с командами CMPS и SCAS и оперируют с флажком ZF = 1, состояние которого определяется результатом исполнения этих команд. Префиксы REPNE и REPNZ действуют аналогично предыдущим префиксам, но флажок ZF должен быть равен нулю. В противном случае повторение заканчивается.

При выполнении цепочечных операций МП реагирует на прерывание до обработки следующего элемента цепочки. После возврата из прерывания операция возобновляется с точки прерывания. Однако возобновление операции будет неправильным, если, кроме любого префикса повторения, определены еще один или два префикса (например, замены сегмента или блокировки). Во время обработки прерывания МП помнит действие только одного непосредственно предшествующего команде префикса. Когда осуществлен возврат из прерывания, любые дополнительные префиксы не действуют. Поэтому, если с цепочечной командой должны использоваться несколько префиксов, необходимо запрещать прерывания на время ее выполнения (напомним, что немаскируемые прерывания запретить нельзя). При этом следует учитывать, что временной интервал, в течение которого прерывания запрещены, при обработке длинных цепочек может оказаться неприемлемым.

Команда MOVS. Команда передачи цепочки имеет следующее обобщенное представление: **MOVS dst.src;** dsi: = (src).

Данная команда передает байт или слово из цепочки src, адресуемой регистром SI, в цепочку dst, адресуемую регистром DI, и, соответственно, модифицирует указатели SI и DI для адресации следующих

элементов цепочек. Состояния флажков не модифицируются. При использовании с префиксом построения REP команда MOVS осуществляет блоковую передачу память–память. Тип цепочки ассемблер определяет по атрибутам операндов. Обычно ассемблер (в зависимости от версии) допускает использование для передачи цепочки мнемокоды MOVSB и MOVSW, которые определяют тип элементов цепочки (B-байт и W-слово). При этом операнды в команде могут отсутствовать.

Команда CMPS. Команда сравнения цепочек имеет следующее обобщенное представление: **CMPS dst,src;** (src) – (dst).

Эта команда производит вычитание байта или слова цепочки dst, адресуемой регистром DI, из байта или слова цепочки src, адресуемой регистром SI. В зависимости от результата вычитания устанавливаются флажки, но сами операнды не изменяются. Регистры-указатели продвигаются на следующие элементы цепочек. Если, например, после команды CMPS находится команда JL (перейти, если меньше), то переход осуществляется в том случае, если элемент src меньше элемента dst.

Когда перед командой CMPS находится префикс REPE (или REPZ), операция интерпретируется как «сравнивать, пока не достигнут конец цепочек или пока элементы цепочек будут не равны». При наличии префикса REPNE (или REPNZ) операция приобретает смысл «сравнивать, пока не достигнут конец цепочек (или пока элементы цепочек будут равны)». Таким образом, команду CMPS удобно применять для нахождения одинаковых или различающихся элементов цепочек.

Команда SCAS. Команда сканирования (или просмотра) цепочки имеет обобщенное представление: **SCAS dst;** (ac) – (dst).

Эта команда вычитает элементы цепочки dst (байт или слово), адресуемое регистром DI, из содержимого аккумулятора AL (байт) или AX (слово). В соответствии с полученной разностью устанавливаются флажки, но значения операндов не изменяются. С префиксом REPE (или REPZ) данную команду можно использовать для поиска элемента цепочки со значением, отличающимся от заданного значения. Если перед командой SCAS находится префикс REPNE (или REPNZ), то операция интерпретируется как "просматривать до тех пор, пока не будет достигнут конец цепочки или значение элемента цепочки не будет равно отыскиваемому значению".

Команда LODS. Команда загрузки цепочки в аккумулятор имеет обобщенное представление: **LODS src;** ac: = (src).

Когда выполняется эта команда, элемент цепочки (байт или слово), адресуемый регистром SI, загружается в аккумулятор AL или AX, а ука-

затель SI продвигается на следующий элемент цепочки. Состояния флажков не изменяются. Обычно эта команда с префиксом повторения не применяется, но ее удобно использовать в программных циклах вместо команд MOV ac, src и INC SI (или DEC SI) – в зависимости от направления продвижения по цепочке. Допускается использование мнемокодов LODSB и LODSW, указывающих тип элемента цепочки.

Команда STOS. Команда запоминания содержимого аккумулятора в цепочке с обобщенным представлением: **STOS dst;** dst: = (ac) – передает байт (слово) из аккумулятора AL (AX) в элемент цепочки, адресуемый регистром DI, и продвигает DI на следующий элемент. Сегментный адрес для этой команды всегда находится в регистре ES, и префикс замены сегмента не используется, а при его наличии – игнорируется. Состояния флажков не изменяются. С префиксом повторения эта команда представляет собой удобное средство инициализации цепочки на фиксированное значение, например нуль или пробел.

4.6. Команды управления микропроцессором

Команды данной группы обеспечивают программное управление различными функциями МП.

Команды первой подгруппы предназначены для управления состояниями отдельных флажков, а второй – для синхронизации МП с внешними событиями. Последние не влияют на состояния флажков.

Команды CLC, CMC и STC выполняют, соответственно, сброс, инвертирование и установку в 1 флажка CF. Их удобно применять с командами сдвига через перенос RCR и RCL. Команды CLD и STD осуществляют сброс и установку флажка направления DF. Состояние этого флажка определяет автодекремент или автоинкремент индексных регистров SI и DI в цепочечных командах. Команды CLI и STI управляют состоянием флажка прерываний IF. После выполнения команды CLI флажок IF сброшен и МП не распознает аппаратные прерывания на входе INT (маскируемые прерывания запрещены). Однако немаскируемые прерывания на входе NMI и программные прерывания МП распознает и соответственно реагирует на них. При подтверждении прерывания флажок IF автоматически сбрасывается. Команда STI переводит флажок IF в состояние 1, разрешая восприятие прерываний на входе INT. Ожидающее прерывание не распознается до завершения команды, находящейся после команды STI.

Таблица 4.6

Команды управления процессором

Операции с флагами	
STC	Установить флаг переноса
CLC	Сбросить флаг переноса
CMC	Инвертировать флаг переноса
STD	Установить флаг направления
CLD	Сбросить флаг направления
STI	Установить флаг разрешения прерываний
CLI	Сбросить флаг разрешения прерываний
Внешняя синхронизация	
HLT	Останов
WAIT	Ожидание
ESC	Обращение к сопроцессору
LOCK	Блокировка шины
Нет операций	
NOP	Пустая операция

Команда HLT (останов) заставляет МП перейти в состояние останова. Из этого состояния МП может быть выведен или сигналом сброса CLR, или аппаратным прерыванием на входе NMI, или запросом прерывания на входе INT, если эти прерывания разрешены. Данную команду можно использовать вместо бесконечного программного цикла, когда программа должна ожидать внешнего прерывания.

Команда WAIT (ожидание) переводит МП в состояние ожидания, в котором он периодически через пять тактов синхронизации проверяет сигнал на входной линии TEST. При появлении на ней активного уровня МП переходит к выполнению следующей за WAIT команды. Команда LOCK, называемая также префиксом блокировки, заставляет МП (работающий в максимальном режиме) выдать сигнал LOCK на время выполнения следующей за префиксом команды. Префикс LOCK может находиться перед любой командой.

Команда ESC представляет собой средство, с помощью которого внешний процессор (сопроцессор) может получать предназначенные для него команды и операнды в процессе работы МП. Сам МП по этой команде не делает ничего, кроме обращения к памяти за операндом и выдачи его на шину.

Команда NOP (нет операции) не производит никаких видимых действий. Она может применяться для замены в программе ненужных байт без изменения общей длины программы, а также в программных циклах задержки.

Примеры решения задач

1. Первоначально все флаги микропроцессора сброшены, DX содержит число 5A0AH и CL содержит 02H. Определите содержимое соответствующих регистров и флагов после выполнения команды RCL DX,CL.

Решение

Приведенная команда сдвигает содержимое операнда-приемника (DX) влево на количество бит, определяемое содержимым регистра CL. Флаг переноса (CF) считается при этом «частью» операнда-приемника, так что старший бит операнда сдвигается во флаг переноса, а бит из флага сдвигается в младший бит операнда. В нашем случае происходит два сдвига. Определим содержимое DX.

DX = 5A0AH = 01011010 00001010B, CF = 0.

После первого сдвига DX = 10110100 00010100, CF = 0.

После второго сдвига DX = 01101000 00101000, CF = 1.

Команда не влияет на флаги, кроме CF и OF. Но состояние флага OF в нашем случае неопределенное, так как количество сдвигов больше 1. Таким образом, после выполнения команды содержимое регистров и флагов следующее: DX = 6828H, CF = 1, OF не определено, CL и другие флаги не изменяются.

2. Напишите программу вычисления булевой функции: $Y = X_0\overline{X}_1X_2\overline{X}_3\overline{X}_4X_5X_6X_7 + X_0\overline{X}_1\overline{X}_2X_3\overline{X}_4X_5\overline{X}_6X_7$. Начальные данные вводятся через порт 0AH как байт данных. Биты этого байта соответствуют логическим переменным x. Результат нужно записать в порт 10H как младший бит.

Решение

Функция Y равна 1 («истина») в следующих случаях:

$X_0\overline{X}_1X_2\overline{X}_3\overline{X}_4X_5X_6X_7 = 1$ и $X_0\overline{X}_1\overline{X}_2X_3\overline{X}_4X_5\overline{X}_6X_7 = 1$.

В первом случае входной байт должен быть равным 11100101B = 0E5H. Во втором случае равным 10101001B = 0A9H. Программа может быть следующей:

```
        MOV   BL,0E5H   ; загрузка в BL 0E5H
        MOV   CL,0A9    ; загрузка в CL 0A9H
INP:    N   AL,0AH      ; ввод данных из порта
        XOR   BL,AL     ; сравнение с первым числом
        JZ   OUT1       ; к метке OUT1, если равно
        JMP   NEXT      ; к следующей проверке
OUT1:   MOV   AL,01H    ; в AL 1
```

```
                OUT   10H,AL      ; вывод в порт 10H
                JMP   DONE
NEXT:           XOR   CL,AL       ; сравнение с вторым числом
                JZ    OUT1        ; к метке OUT1, если равно
OUT0:           MOV   AL,0        ; вывод числа 0
                OUT   10H,AL      ; в порт 10H
DONE:
```

Вопросы и задания для повторения

4.1. Сформулируйте основные правила для написания программ на языке ассемблера микропроцессора 8086.

4.2. Напишите машинные коды и ассемблерные нотации команд, выполняющих следующие действия:
a) (AX)←(BX); b) (BX)←(AX); c) (CL)←((SI)+0A50H);
d) (AX) ←00F1H; e) (AL)←((BP)+(DI)).

4.3. Напишите ассемблерные нотации команд, вычислите значения соответствующих регистров после выполнения команды, эффективные адреса операндов и физические адреса ячеек памяти:
a) (AX)←(AX)+(BX); b) (CX)←(CX)+2468H;
c) (AL)←(AL)+((BX)+(SI)); d) ((BX))←0000;
e) (BX)←(BX)+((SI)+1EH).
Исходные значения регистров (AX) = 01A8H, (BX) = 20ECH, (CX) = 80A0H, (SI) = 0100H, (DS) = 0A00H, (SS) = 1000H. Начальное содержимое ячеек памяти можно взять произвольно.

4.4. Объясните, какие действия проводит каждая команда. Определите содержимое операндов и флагов после выполнения команды, вычислите эффективные адреса операндов и физические адреса ячеек памяти.
a) XCHG BL,10[BP]
DS = 00F0H BX = 0A1FEH
SS = 0F00H BP = 1000H
(адресуемая ячейка) = 0D1H;
b) OR DX,[BP],[DI]
DS = 0100H DX = 9559H
SS = 0200H BP = 0A00H
DI = 0200H CF = 0,ZF,PF,SF = 1
(адресуемая ячейка) = 3F04H;
c) MOV ES:100[SI],AL
DS = 0F10H AX = 3AB9H
ES = 10F0H SI = 2100H
ZF,CF = 0, PF,SF = 1.

4.5. Напишите программу обмена содержимого двух ячеек памяти M1 и M2, находящихся в одном сегменте памяти. Подсчитайте время выполнения этой программы, если частота синхронизации микропроцессора f = 5 MHz.

4.6. Напишите программу сложения N 16-разрядных слов, последовательно расположенных в памяти. Результат тоже должен находиться в памяти. Подсчитайте время выполнения этой программы, если частота синхронизации микропроцессора f = 5 MHz и N = 100.

4.7. Напишите программу вычисления выражения $Z = 2\cdot(X\cdot Y+T)$, где X, Y, T – 8-разрядные числа без знака, расположенные в памяти. Адреса ячеек памяти находятся в регистрах BX, CX и DX, соответственно. Если $Z > 100$, результат нужно записать в порт 1FH, в противном случае – в регистр AX.

Глава 5
ПРОЕКТИРОВАНИЕ СИСТЕМ НА БАЗЕ МИКРОПРОЦЕССОРОВ INTEL

Микропроцессоры 8085 и 8086 могут быть использованы как **ядро процессора** в вычислительной системе. Перед началом проектирования процессора на основе микропроцессоров мы должны принять во внимание множество их функциональных особенностей. Нужно отметить следующие основные **особенности**:

1. Необходимость **синхронизации** (**внутренней** для 8085 и **внешней** для 8086).
2. **Мультиплексная** шина **адреса/данных** микропроцессоров.
3. Шины микропроцессоров имеют довольно **низкую нагрузочную способность**.
4. Возможность работы в **минимальном и максимальном режиме** (для 8086).

Чтобы решить проблемы проектирования микропроцессорных систем с минимальными затратами, выпускаются **специальные наборы микросхем** (микропроцессорные комплекты). Обычно каждый микропроцессор имеет собственный **комплект БИС** (больших интегральных схем), который применяется вместе с микропроцессором.

5.1. Микропроцессорная система на основе микропроцессора 8085

Микропроцессор 8085 сконструирован таким образом, что на его основе можно построить простую вычислительную систему с минимальным набором микросхем. Например, представленная на рис. 5.1 система содержит их только три. В схеме использован МП *Intel* 8085, который управляет шиной системы и двумя другими специальными ИС интерфейса с периферией. Составляющие интерфейса системы, представленной на рисунке, являются ИС *Intel* 8155 и 8355.

Микросхема 8155 содержит 2048 бит памяти статического ОЗУ, организованного в память 256x8 бит; она содержит также три порта ввода/вывода и таймер. Два порта ввода/вывода являются универсальными, по 8 бит каждый. Третий (6 бит) может быть использован как порт вво-

да, вывода или выводы порта могут быть использованы в качестве системы сигналов управления для двух других 8-разрядных портов. Схема 8185 программируема и содержит регистр состояния и 14-разрядный счётчик-таймер.

Микросхема 8355 является интерфейсом периферии, содержит ПЗУ ёмкостью 16 384 бит, организованное в блок памяти 2048x8 бит, и два универсальных порта ввода/вывода по 8 бит каждый.

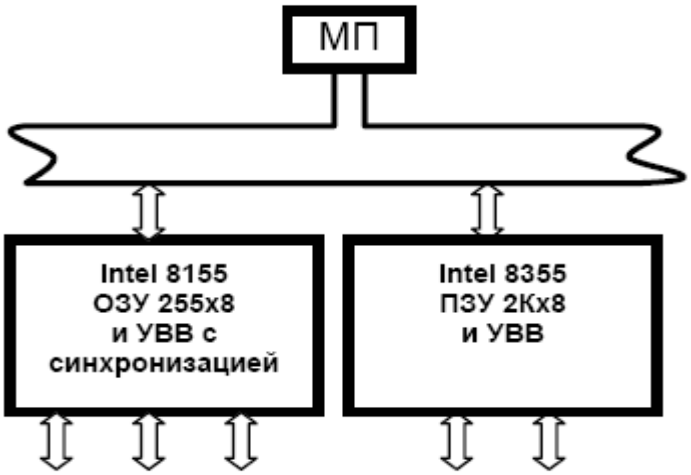


Рис. 5.1. Блок-схема простейшей системы на основе МП 8085

Восьмиразрядный МП *Intel* 8085 выпускается в корпусе DIP (с двусторонней упаковкой выводов) с 40 выводами, расположение которых приведено на рис. 5.2. В табл. 5.1 приведено название выводов и их назначение.

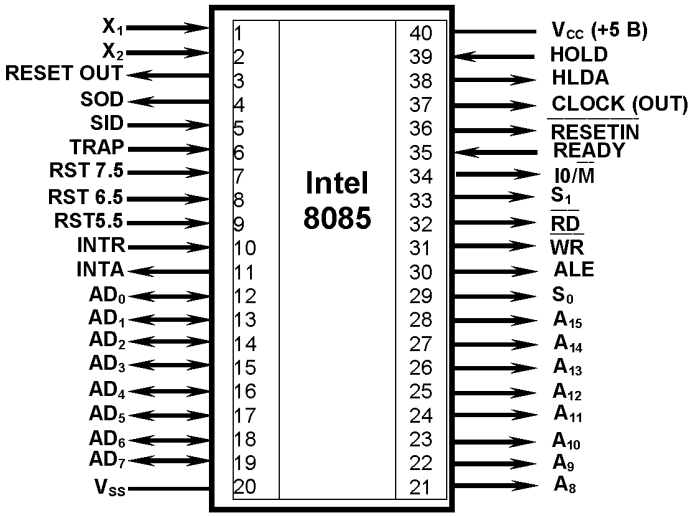


Рис. 5.2. Расположение выводов МП 8085

МП *Intel* 8085 имеет 16 адресных линий и 8 линий шины данных. Восемь старших разрядов шины адреса выведены на выводы A_8–A_{15}. Эти выводы являются выходами или могут быть в состоянии высокого со-

противления (в третьем состоянии). Корпус только с 40 выводами ограничивает количество сигналов, с помощью которых МП может общаться с системой. По этой причине выводы с 12 по 19 использованы как равноценные линии шины адреса/данных (AD_0–AD_7). Поэтому этот микропроцессор называется устройством с мультиплексированной шиной данных/адреса. Адресные шины восьми младших разрядов разделяют выводы с линиями шины данных. Мультиплексировать – значит выбирать линии поочерёдно. То есть мультиплексировать шину адреса/данных означает использовать сначала шину для передачи адреса, затем использовать её же для выдачи или получения данных. Микропроцессор 8085 снабжён специальным сигналом для того, чтобы информировать периферийные устройства, производит ли мультиплексированная шина операции на адресной шине или на шине данных. Это специальный сигнал, называемый сигналом разрешения адреса (ALE).

Таблица 5.1

Назначение выводов МП 8085

Выводы	**Назначение**	**Описание**
$AD_0 – AD_7$	Двунаправленная, три состояния	Шина адреса/данных
$AD_8 – AD_{15}$	Выход, три состояния	Шина адреса
ALE	Выход*	Разрешение захвата адреса
RD	Выход, три состояния	Управление считыванием
WR	Выход, три состояния	Управление записью
IO/M	Выход, три состояния	Указатель устройства для передачи данных (УВВ или память)
S_0 , S_1	Выход	Указатель состояния шины
READY	Вход	Вызов состояния ожидания
SID	Вход	Ввод последовательных данных
SOD	Выход	Вывод последовательных данных
HOLD	Вход	Требование захвата
HLDA	Выход	Подтверждение состояния захвата
INTR	Вход	Запрос прерывания
TRAP	Вход	Запрос немаскированного прерывания
RST 5.5 RST 6.5 RST 7.5	Вход	Запрос аппаратного векторного прерывания
INTA	Выход	Подтверждение запроса на прерывание
RESET IN	Вход	Сброс системы
RESET OUT	Выход	Сброс периферии
X_1 , X_2	Вход	Соединение кристалла или внешнего ГТИ
CLK	Выход	Сигнал внутреннего ГТИ
V_{CC} , V_{SS}	–	Питание, земля

Необходимо отметить, что выводы мультиплексированной шины двунаправлены или могут быть в положении трёх состояний. Вывод управления ALE является выходным.

Выводы V_{CC} и V_{SS} являются выводами питания, подсоединёнными к источнику +5 В.

МП 8085 снабжён внутренним генератором тактовых импульсов, входы которого X_1 и X_2 обычно соединены с частотозадающей цепью (чаще всего с кварцевым резонатором). Внутренняя частота МП является половиной частоты кристалла.

Многие выводы МП 8085, показанные на рис. 5.2, выполняют функции управления:

$\overline{\text{RD}}$ и $\overline{\text{WR}}$ – используются для информации устройства памяти или УВВ, т. е. определяют, наступило ли время послать или принять данные по шине данных (в нашем случае по мультиплексированной шине).

$\overline{\text{RESET IN}}$ – вход сброса. Шины адреса/данных и линии управления находятся в состоянии высокого сопротивления в ходе сброса. Когда МП сбрасывается, вывод RESET OUT (относится к операции сброса) выдаёт сигнал в периферийные устройства, информируя их, что операция сброса закончена.

CLK – выход генератора тактовых импульсов.

INTR – вход запроса прерывания. Прерывание INTR в 8085 может быть разрешено или запрещено командами программы (маскировано). Кроме входа маскированного запроса на прерывание (INTR), МП снабжён четырьмя другими прерываниями: **TRAP, RST7.5, RST6.5, RST5.5** (от прерывания наивысшего приоритета до самого низшего, соответственно). Сигнал TRAP или один из трёх сигналов RST влекут за собой ветвление МП по вызываемому специальному адресу. Команды рестартов RST могут быть разрешены или запрещены программно, но прерывания по входу TRAP таким образом запрещены быть не могут. Запрос на прерывание INTR вызывает переход к новому адресу, указанному специальной командой, выданной периферией, когда активизируется выход, подтверждающий получение запроса на прерывание (INTR).

SID и SOD – это слаборазвитые ввод и вывод последовательных данных, соответственно. Отдельный бит данных на выводах SID загружается в наиболее значимый разряд (бит 7) аккумулятора командой RIM. Вывод выхода SOD активизируется или сбрасывается командой SIM в МП.

READY – это вход, который информирует МП, что периферия готова выдать или принять данные. Если READY имеет низкий уровень в цикле считывания или записи, МП его интерпретирует как требование перейти в состояние ожидания. В этих условиях МП будет ждать до тех

пор, пока периферия не просигнализирует, что она готова передать или получить данные. Затем будем продолжать выполнение цикла записи или считывания. Вход READY удобен при использовании очень медленных, по сравнению со скоростью обработки данных в МП, устройств памяти или периферии.

HOLD – входной сигнал требования захвата – оповещает МП, что другое устройство хочет использовать шины адреса и данных (это может производиться в ходе ПДП). По получении сигнала HOLD МП завершает текущую операцию, затем выводы данных и адреса RD, WR и IO/M переводятся в третье состояние, т. е. исключается взаимодействие с передачами данных на шинах.

HLDA – выход подтверждения состояния захвата указывает периферии, что запрос HOLD был получен и микропроцессор не будет управлять шинами в следующем цикле тактовых импульсов.

IO/$\overline{W}$, S_0 и S_1 – выходы, являющиеся сигналами управления, которые информируют периферию о типе машинного цикла, выполняемого МП (табл. 5.2).

Таблица 5.2

Машинные циклы МП 8085

Сигналы управления МП 8085			Состояние машинного цикла
IO/M	S_1	S_0	
0	0	1	Запись в память
0	1	0	Считывание из памяти
1	0	1	Запись в УВВ
1	1	0	Извлечение из УВВ
0	1	1	Извлечение КОП
1	1	1	Подтверждение запроса на прерывание
*	0	0	Останов
*	**	**	Ожидание
*	**	**	Сброс

Примечания: * *– третье состояние (высокое сопротивление);*
** *– не оговаривается.*

Вариант микропроцессорной системы, построенной на основе однокристального микропроцессора 8085, содержащего постоянное запоминающее устройство на микросхеме 573РФ2, дешифратор адреса 555ИД7, регистр хранения 555ИР22 и многоцелевую микросхему 8155 приведена на рис. 5.3.

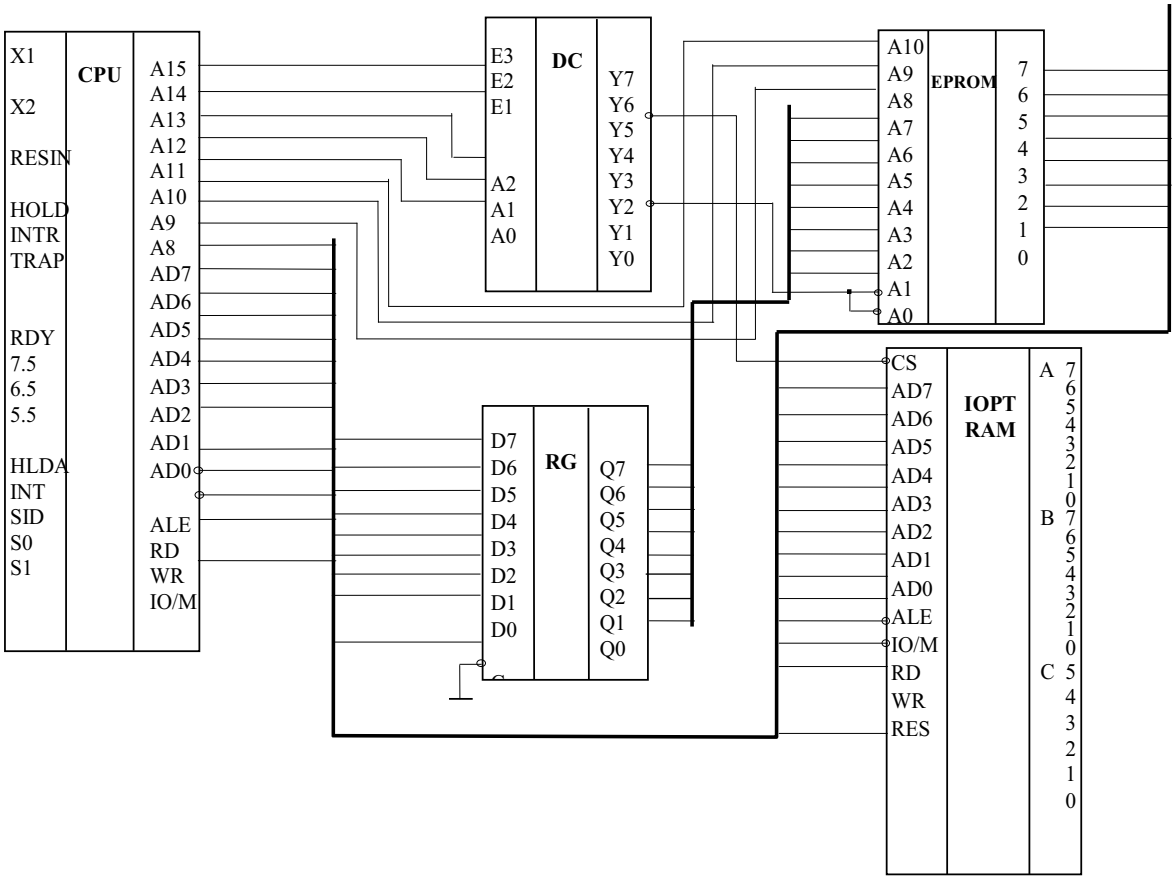


Рис. 5.3. Микропроцессорная система на основе МП 8085

Микропроцессор 8085 является модернизированным вариантом микропроцессора 8080. По сравнению с прототипом 8085 обладает повышенным быстродействием и имеет встроенный тактовый генератор, а также несколько особенностей работы. Во-первых, это мультиплексированная шина адреса/данных. При передаче данных из микропроцессора сначала передается адрес, причем младший байт адреса должен быть зафиксирован в дополнительном регистре (в нашей системе это 555ИР22), а затем данные по линиям AD0 – AD7 (линии A8 – A15 содержат при этом старший байт адреса). Во-вторых, при обращении к внешним устройствам адрес внешнего устройства, который является однобайтовым, передается одновременно и по старшему и по младшему байту шины адреса. Это учтено в нашей системе тем, что адресный дешифратор подключен к старшему байту шины адреса и используется одновременно и для выбора банков памяти (573РФ2), и для выбора внешних устройств (8155). Сделать то же самое для микропроцессора 8080 нельзя.

Специализированная БИС 8155 – микросхема, которая содержит на одном кристалле ОЗУ объемом 256 байт, параллельный интерфейс ввода/вывода

(2 порта по 8 линий и 1 порт по 6 линий) и 14-разрядный программируемый таймер. Адресное пространство микросхемы показано на рис. 5.4. Микросхема может выполнять роль статической памяти, если на вход CS подан сигнал нулевого уровня и сигнал IO/M = 0. Сигналы RD и WR являются сигналами управления чтения/записи в ОЗУ. При активном (равном нулю) сигнале CS и IO/M = 1 происходит обращение к портам ввода/вывода микросхемы. В микросхеме их имеется 6 (назначение и адреса приведены на рис. 5.4).

Регистр управления содержит управляющее слово для программирования режима работы и направления передачи информации через порты A,B и C и управления таймером. Формат регистра управления показан на рис. 5.5.

Работа таймера программируется двумя регистрами, формат которых показан на рис. 5.6.

Слово состояния, формат которого приведен на рис. 5.7, определяет состояние таймера и портов параллельного обмена.

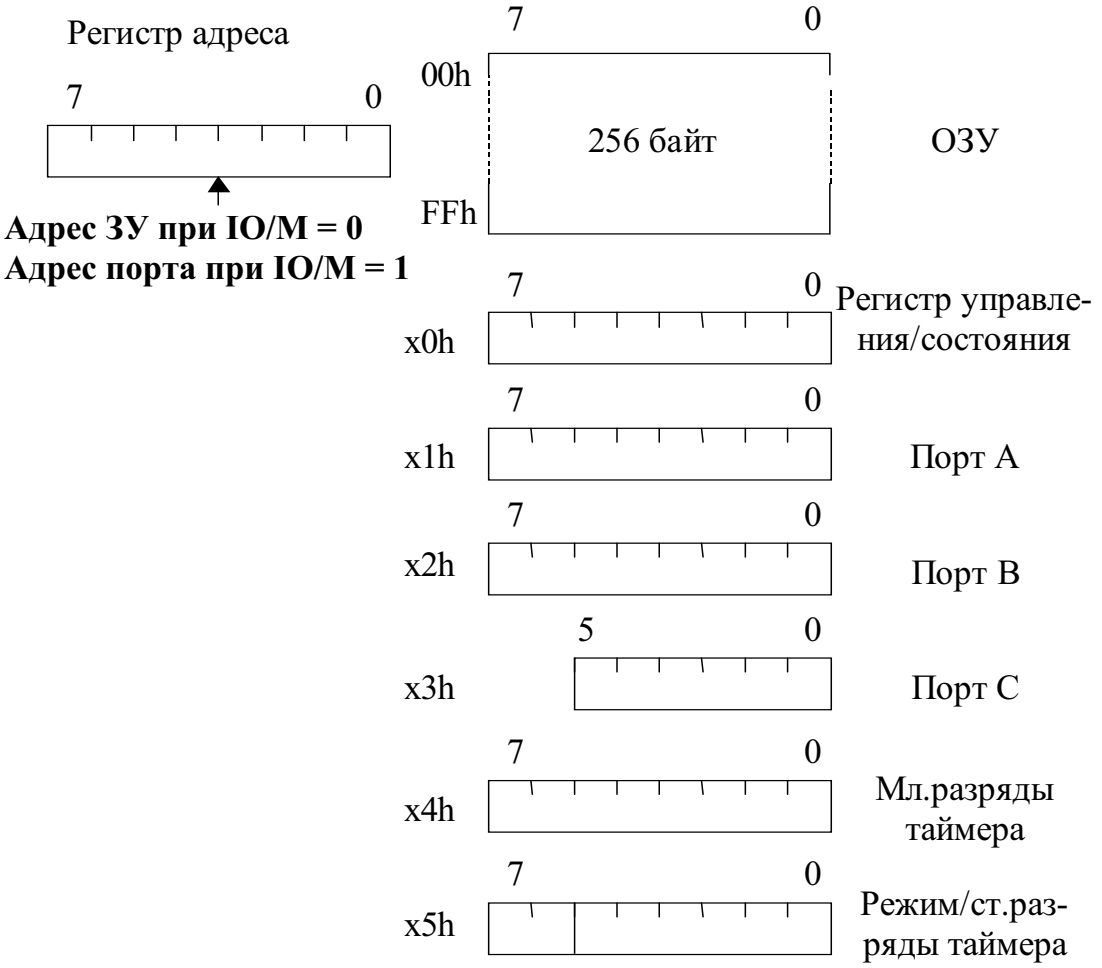


Рис. 5.4. Адресное пространство микросхемы 8155

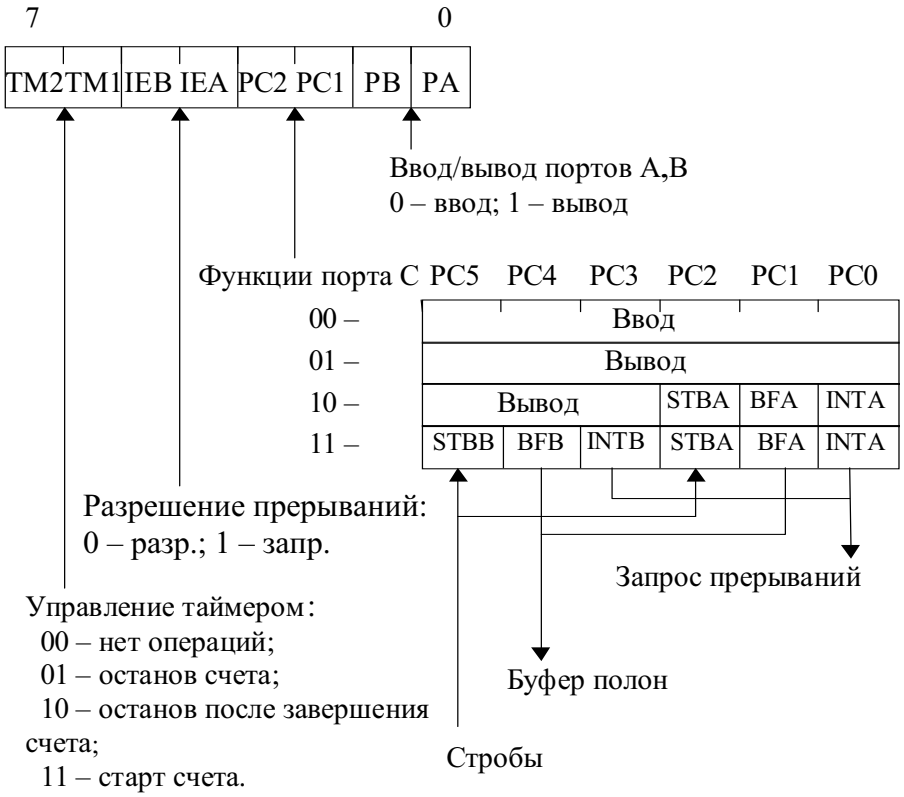
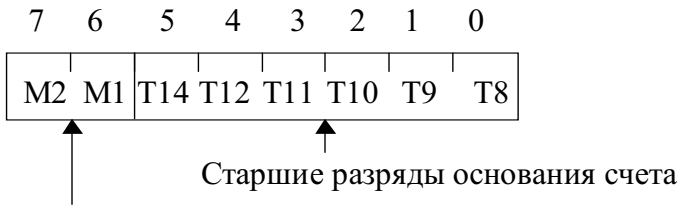
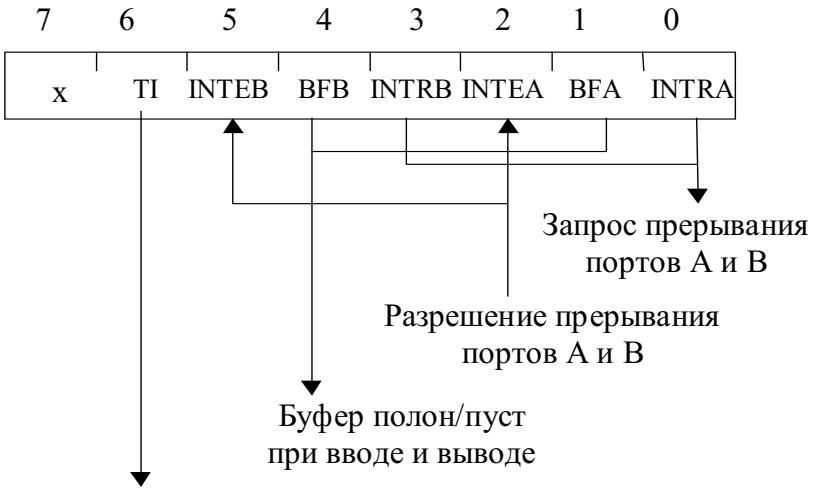


Рис. 5.5. Формат регистра управления



Режим таймера:
- 00 – формирование импульса длительностью первой половины периода счета;
- 01 – формирование меандра с периодом основания счета, автоматическая загрузка;
- 10 – формирование одиночного импульса по окончании счета;
- 11 – формирование одиночного импульса по окончании счета, автоматическая загрузка.

Рис. 5.6. Формат регистров управления таймером



Запрос прерывания от таймера (устанавливается после окончания счета, сбрасывается после чтения или запуска счета)

Рис. 5.7. Формат слова состояния

5.2. Микропроцессорная система на основе микропроцессора 8086 в минимальном режиме

Рассмотрим, как упомянутые выше особенности микропроцессора 8086 учитываются при разработке микропроцессорных систем, в которых микропроцессор работает в минимальном режиме.

Специализированный кристалл – генератор синхронизации 8284 – разработан для синхронизации работы микропроцессора. Графическое представление этого кристалла приведено на рис. 5.8.

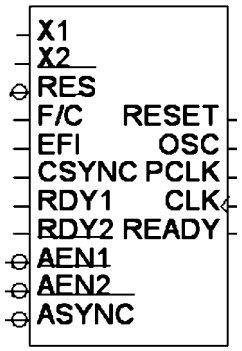


Рис. 5.8. Внешний генератор 8284 для систем на основе микропроцессора 8086

Выводы **X1** и **X2** обеспечивают подключение параллельного **резонансного контура** для внутреннего генератора (**F/C** имеет низкий уровень). **EFI** может соединяться с внешним генератором (**F/C** имеет высокий уровень). Частота генератора внутренне делится на три, чтобы получить **синхросигнал (CLK)**. Вывод **RESET** указывает, что центральный процессор устанавливается в исходное состояние и может использоваться как сигнал сброса системы. Его активный уровень – высокий, сигнал синхронизован с синхросигналом процессора **CLK** и продолжается целое число тактов, соответствующих

длине сигнала **RES**. Этот сигнал может быть асинхронным. **RES** внутренне синхронизирован.

Асинхронные входные сигналы готовности (**RDY1**, **RDY2**) сообщают процессору, что адресуемая память или устройство ввода/вывода закончили передачу или прием данных. Вход **READY** изменяет свое состояние синхронно с **CLK** и принимает активное высокое значение в соответствии с сигналом **RDY2** или **RDY1**. Сигналы **RDY1** и **RDY2** используются, если сигналы **AEN1** и **AEN2,** соответственно, имеют низкий уровень. Соединение **RDY** с уровнем логической 1 (**AEN** должен иметь низкий уровень) всегда утверждает состояние готовности для центрального процессора. Если одна из линий **RDY** не использована, она должна быть подключена к низкому логическому уровню, чтобы управление осуществлялось через другой вход **RDY**.

Специальные внешние регистры-защелки должны использоваться для разделения информации на шине адреса/данных. Для этих целей выпускаются специальные восьмибитовые регистры, например 82С82. Микропроцессорный сигнал **ALE** обеспечивает строб для фиксации физической адресной информации. Адрес представлен на мультиплексной шине адреса/данных в течение состояния Т1. Срез импульса **ALE** происходит в середине Т1 и обеспечивает правильную фиксацию адреса.

В большинстве случаев требуется буферизации шины данных системы. Следующие случаи требуют дополнительных внешних приемопередатчиков на шине данных:

- емкостная нагрузка на шине адреса/данных становится слишком большой;
- текущая нагрузка на шине адреса/данных превышает предельные технические параметры устройства;
- память или устройства ввода/вывода не могут обеспечить перевод своих выводов в высокоомное состояние, чтобы предотвратить конфликтную ситуацию при обращении к шине.

Микропроцессор генерирует два сигнала управления **DEN** и **DT/R** для управления двунаправленными буферами или приемопередатчиками. Линии связи между процессором и приемопередатчиками носят название **локальная шина**. Линии связи между приемопередатчиками и памятью или устройствами ввода/вывода называются **буферизированной шиной**. Полностью буферизированная система не имеет никаких устройств, подключенных к локальной шине. Частично буферизированная система имеет устройства как на локальной, так и на буферизированной шине. В полностью буферизированной системе **DEN** непосредственно управляет выводом приемопередатчика. Частично буферизированная система требует, чтобы **DEN** был использован совместно с дру-

гим сигналом для предотвращения включения приемопередатчика при доступе к локальной шине. **DT/R** всегда соединяется непосредственно с приемопередатчиком. Однако может потребоваться инвертор, если полярность **DT/R** не соответствует полярности, требуемой приемопередатчику. **DT/R** переходит в состояние низкого уровня (0) только для чтения памяти и устройств ввода/вывода, выбора команды и циклов подтверждения прерывания.

Сначала мы рассмотрим систему, в которой микропроцессор работает в минимальном режиме. В этом режиме он генерирует все необходимые сигналы управления. Блок-схема системы на основе микропроцессора 8086 показана на рис. 5.9. Система состоит из микропроцессора, генератора синхронизации, 3-х регистров-защелок, 2-х приемопередатчиков, устройства памяти и устройств ввода/вывода.

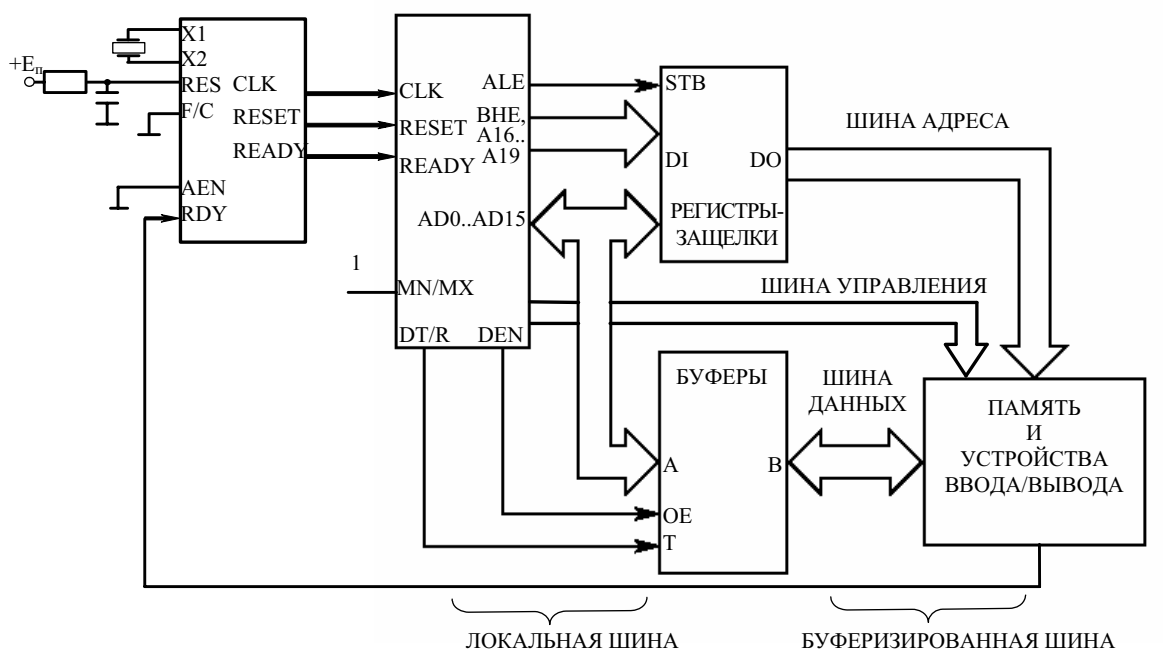


Рис. 5.9. Блок-схема системы с микропроцессором в минимальном режиме

В системе, представленной на блок-схеме, микропроцессор выполняет все функции по обработке информации и формированию всех необходимых сигналов управления. Внешний генератор обеспечивает формирование синхронизирующего сигнала **CLK**, сигналов начального сброса **RESET** и готовности **READY**. При включении питания системы конденсатор, подключенный к входу **RES**, некоторое время заряжается через резистор. В это время входной сигнал **RES** имеет низкое значение и **RESET** активен. По сигналу **RESET** все составляющие системы устанавливаются в начальное состояние. Адресные выводы микропроцессо-

ра связаны с входами регистров-фиксаторов. Поскольку число адресных сигналов 21 (**AD0** … **AD15**, **AD16** … **AD19**, **BHE**), требуется 3 регистра. Адрес сохраняется в регистре по сигналу **ALE** от микропроцессора. В таком случае в течение всего цикла шины адрес находится в регистрах и доступен для составляющих системы. Выводы регистров-фиксаторов формируют буферизированную адресную шину.

Кроме того, линии шины адреса/данных (**AD0**…**AD15**) микропроцессора связаны со входами буферных приемопередатчиков с высокой нагрузочной способностью. Приемопередатчики 8-разрядные, поэтому требуется 2 кристалла. Приемопередатчики управляются сигналами микропроцессора **DEN** и **DT/R**. В течение цикла чтения шины сигнал **DT/R** имеет низкий уровень и данные передаются от выводов **B** к выводам **A**. Сигнал **DEN** разрешает работу приемопередатчиков. В течение цикла записи сигнал **DT/R** имеет высокий уровень и данные передаются от выводов **A** к выводам **B.**

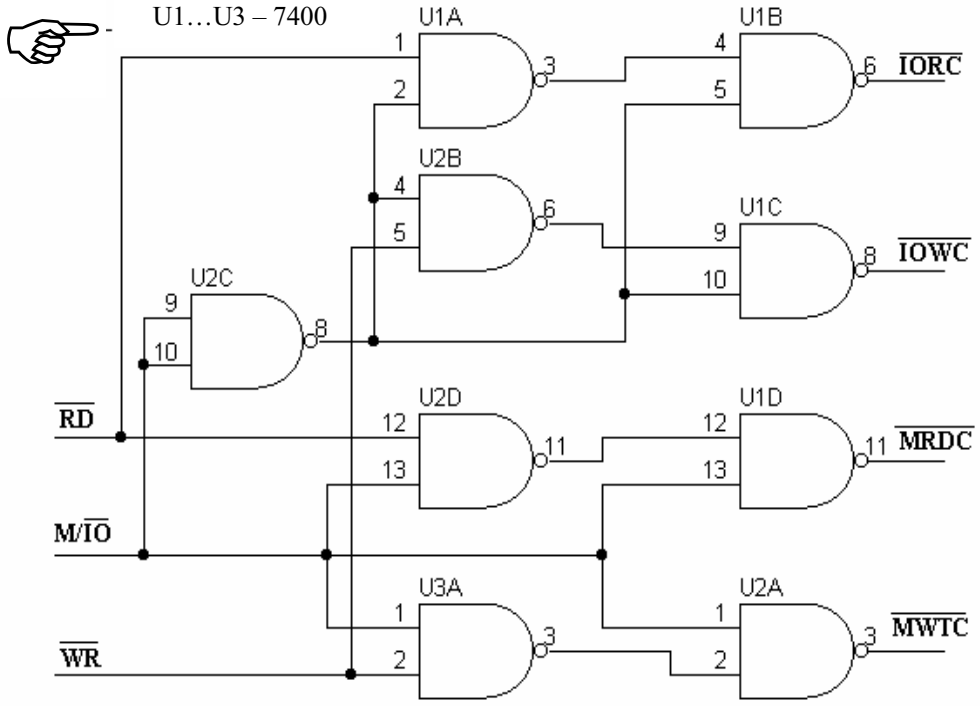


Рис. 5.10. Схема формирования управляющих сигналов

RD, **WR**, **M/IO** – сигналы шины управления микропроцессорной системы. Однако память и устройства ввода/вывода в большинстве случаев требуют других сигналов. Это сигналы **MWTC**, **MRDC**, **IOWC** и **IORC**. **MWTC** и **MRDC** – это сигналы, разрешающие запись в память и чтение из памяти, соответственно. **IOWC** и **IORC** разрешают запись и чтение данных из портов ввода/вывода. Сигналы могут быть сформиро-

ваны из сигналов микропроцессора при помощи дополнительных логических схем. Пример такой схемы показывается на рис. 5.10.

5.3. Многопроцессорные и многопользовательские системы

В настоящее время все большее распространение находят мультипроцессорные вычислительные системы. Разделение проблем между несколькими одновременно работающими процессорами по существу увеличивает производительность системы. Однако в таких системах имеются специфические проблемы, связанные с совместным использованием общих ресурсов системы. Общедоступным ресурсом становятся также шины системы. Некоторые устройства могут передать данные по шине и управлять ею.

Микропроцессор 8086 допускает разработку на его основе мультипроцессорных систем, так как в нем заложена возможность синхронизации работы нескольких процессоров. В мультипроцессорных системах, выполненных на основе 8086, возможно использование процессоров двух типов: независимых и подчиненных (сопроцессоров). Независимый процессор выполняет свой собственный поток команд. Сопроцессор отличается от независимого тем, что следит за выполнением потока команд центральным процессором, идентифицирует в этом потоке свои команды, выполняет их и таким образом расширяет набор команд центрального процессора.

Для обеспечения возможности разделения системного канала при работе нескольких процессоров МП 8086 по команде **LOCK** вырабатывает сигнал блокировки канала **LOCK**, который запрещает другим процессорам пользоваться системным каналом на время выполнения команды, следующей за командой **LOCK**. Данная команда совместно с командой **XCHG** может быть использована для координации доступа к совместно используемым ресурсам через «семафор» – программно-управляемый признак в памяти.

Микропроцессор 8086 может быть синхронизирован по отношению к внешним событиям с помощью команды **WAIT** и входного сигнала **TEST**.

В систему команд 8086 входит команда **ESC** (расширение), которая предоставляет другому процессору (сопроцессору) возможность получения команд и данных из программы, выполняемой микропроцессором. Команда **ESC** совместно с командой **WAIT** используется для организации параллельных процессов (программ) в мультипроцессорной системе. При этом сопроцессор следит за состоянием очереди команд микропроцессора через сигналы **QS0** и **QS1** (табл. 2.1).

Максимальный режим (вывод MN/MX подключен к шине «Общий») ориентирован на применение микропроцессора в сложных одно- и многопроцессорных системах. В системах максимальной конфигурации функции управления каналом берет на себя системный контроллер, который декодирует три сигнала состояния микропроцессора **SA0-SA2**.

В максимальном режиме сигналы **QS0** и **QS1** являются альтернативными сигналам **ALE** и **WR** (минимальный режим), соответственно. В случае максимального режима сигналы **WR** и **ALE** больше не доступны и должны быть сгенерированы внешними аппаратными средствами.

В максимальном режиме сигналы **HOLD** и **HLDA** меняются на сигналы **RQ/GT0** и **RQ/GT1**. Эти двунаправленные сигналы могут использоваться для разделения локальной шины между процессорами при помощи механизма подтверждения, который представляет из себя трехфазный цикл: запрос, предоставление и очистка. В первом процессоре, требующем доступа к шине, генерируется импульс на линии **RQ/GT**. Микропроцессор отвечает импульсом на той же самой линии, сообщая о переходе в состояние «Подтверждение запроса», и освобождает шину. В течение последующего цикла интерфейс шины микропроцессора отключен от общей шины. Однако микропроцессор продолжает выполнять команды, пока следующая команда не будет требовать доступа к шине или очередь команд не очистится. Когда другой процессор закончит операции на шине, он генерирует сигнал на линии **RQ/GT**, и микропроцессор возобновляет управление шиной. Обмен сигналами происходит синхронно, и оба взаимодействующих процессора должны быть синхронизированы от одного генератора.

Линия **RQ/GT0** имеет больший приоритет, чем линия **RQ/GT1**. Если запросы шины происходят одновременно на обеих линиях, управления шиной получает процессор, производящий запрос по линии **RQ/GT0**.

Чтобы облегчить проектирование мультипроцессорных систем в дополнение к регистрам-защелкам адреса и буферным приемопередатчикам, используются микросхемы **контроллера шины** и **арбитра шины**. **Контроллер шины,** используя сигналы состояния от микропроцессора, формирует необходимые управляющие сигналы для системы. **Арбитр шины** необходим в мультипроцессорных системах и облегчает совместное использование шины системы несколькими процессорами.

Графическое представление контроллера шины 8288 показано на рис. 5.11.

Контроллер шины 8288 обеспечивает формирование сигналов управления шиной и необходимую логику, чтобы связать микропроцессор 8086 с шиной системы.

Входы **S0**…**S2** контроллера шины должны быть связаны с выходами состояния S0…S2 микропроцессора. Эти сигналы могут также использовать другие процессоры, соединяющиеся с локальной шиной. Контроллер шины опрашивает S0...S2 сигналы в начале каждой фазы синхронизации. Если происходит изменение состояния сигнала от пассивного до активного (табл. 2.2), контроллер начинает формировать цикл шины. Цикл начинается с формирования сигнала **ALE** и сигнала направления передачи **DT/R**. Затем, в зависимости от требуемого цикла, формируются другие управляющие сигналы. Ввод разрешения доступа **(AEN)** присутствует, чтобы отключать буферные схемы от шины, когда другие устройства управления передачей данных по шине осуществляют контроль шины. Контрольные сигналы **ALE**, **INTA**, **MRDC**, **MWTC**, **IOWC** и **IORC** контроллера 8288 идентичны сигналам, формируемым микропроцессором в минимальном режиме.

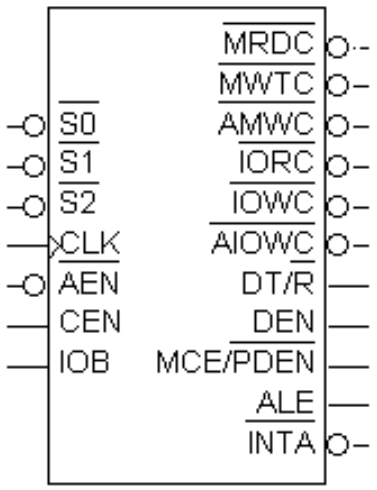


Рис. 5.11. Контроллер шины 8288

Удлиненные сигналы записи (**AIOWC** и **AMWC**) формируются на один такт синхронизации раньше обычных сигналов записи, чтобы получить более длинный импульс записи. Такой импульс часто требуется для периферийных ИС и статической оперативной памяти. Обычные сигналы гарантируют наличие данных до сигнала записи, что является необходимым для динамических кристаллов памяти и периферийных устройств, которые стробируют данные в начале импульса записи. Удлиненные сигналы не гарантируют наличия данных в начале записи импульса.

Необходимо отметить, что сигнал **DEN** в максимальном режиме инвертирован по сравнению с сигналом в минимальном режиме, что в некоторых случаях является более удобным.

Набор интерфейсных кристаллов микропроцессора 8086 включает микросхему арбитра шины 8289. Это интегральная схема, предназначенная для использования как устройство, реализующее синхронизацию доступа нескольких устройств для управления передачей данных по шине. Графическое представление микросхемы показано на рис. 5.12. Арбитр шины также использует состояние цикла шины **S0**…**S2** микропроцессора. Сигнал **AEN** позволяет доступ к шине системы буферных приемопередатчиков процессора. Поскольку шина системы может запрашивать любые устройства управления передачей данных по шине, необходимо принять во внимание их приоритет. Чтобы понять методы приоритетного арбитража, достаточно рассмотреть только несколько основных сигналов арбитра шины.

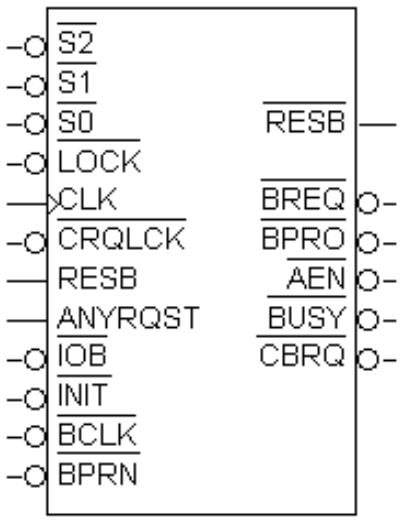


Рис. 5.12. Арбитр шины

BREQ (запрос шины) – выходной сигнал, который арбитр шины генерирует для запроса цикла шины.

BPRN – входной сигнал приоритета шины. Сигнал сообщает арбитру шины, что он может использовать шину системы по следующему срезу импульса синхронизации шины **BCLC**. Если **BPRN** активен, данный арбитр шины имеет самый высокий приоритет. Удаление сигнала сообщает контроллеру, что он потерял приоритет.

BPRO – выход приоритета шины. Данный сигнал может быть установлен арбитром, который имеет приоритет доступа к шине и не использует ее, позволяя использование шины арбитрами с более низким приоритетом. Данный сигнал используется при последовательном методе установки приоритета, когда вывод BPRO одного арбитра соединен с входом BPRN следующего арбитра с меньшим приоритетом.

BUSY является двунаправленной линией шины системы, с которой связаны соответствующие вводы/выводы (типа открытый коллектор) всех арбитров. Этот сигнал сообщает арбитру о готовности шины (если **BUSY** = 1, шина доступна). Когда арбитр использовал шину, он удаляет сигнал **BUSY**, выключая транзистор выходного каскада. Это позволяет другим арбитрам осуществить доступ к шине системы.

В мультипроцессорных системах арбитр устраняет конфликты из-за доступа к шине и согласует асинхронно работающие процессоры. Когда имеются одновременные запросы шины, арбитр разрешает конфликт и возвращает шину процессору с самым большим приоритетом. Обычно арбитр может быть установлен на один из нескольких методов арбитража приоритетов.

Схема параллельного управления доступом к шине показана на рис. 5.13. Сигналы **BREQ** от арбитров поступают в приоритетный кодер, который формирует бинарный номер активной линии BREQ с самым большим приоритетом. Полученный номер поступает на декодере для выбора соответствующей строки **BPRN**, на которой формируется активный сигнал, возвращаемый в арбитр шины с самым большим приоритетом. Этот арбитр открывает доступ к шине системы, как только это будет возможно **(BUSY** = 1**)**. Открывая доступ к шине, выбранный арбитр формирует сигнал **BUSY =** 0, отключая все оставшиеся арбитры от шины.

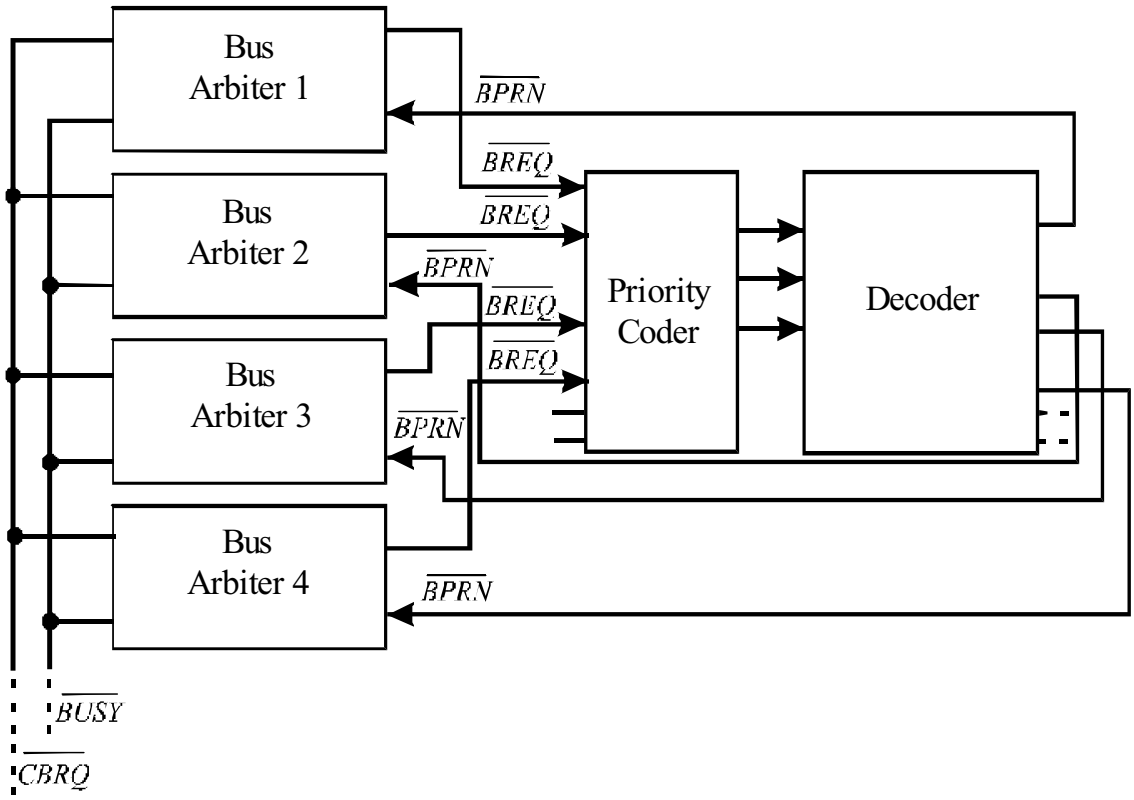


Рис. 5.13. Блок-схема параллельного арбитража приоритетов

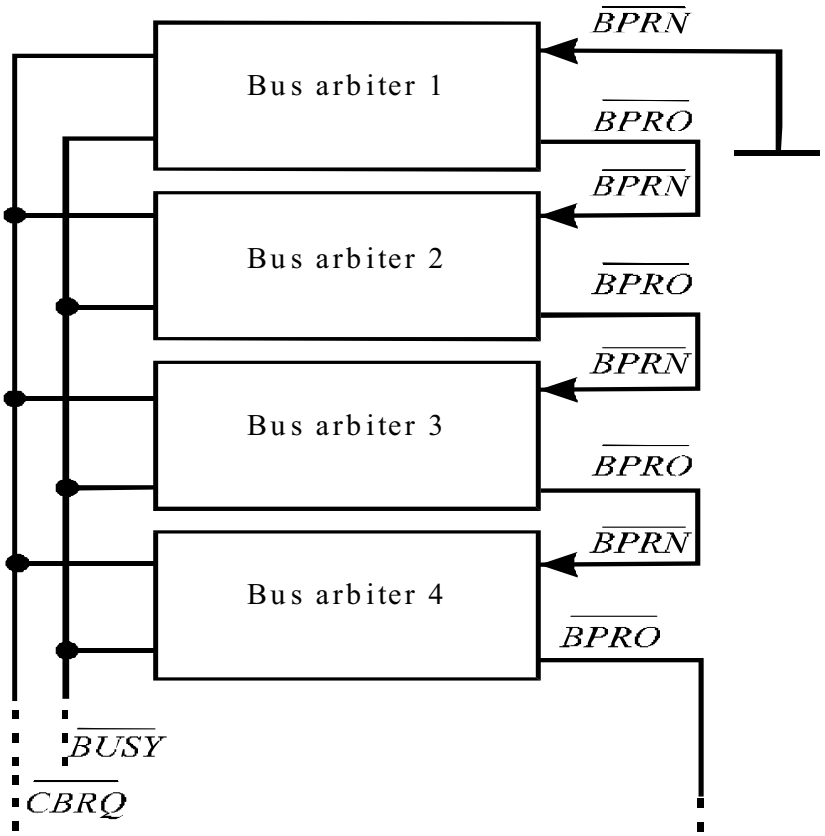


Рис. 5.14. Схема последовательного арбитража приоритетов

Последовательный приоритетный метод арбитража реализуется организацией цепочки арбитров шин. Для этих целей вывод **BPRO** арбитра с большим приоритетом связан со входом **BPRN** смежного арбитра с меньшим приоритетом. На рис. 5.14 приоритеты арбитров уменьшаются сверху вниз. Если арбитр шины не запрашивает шину, он передает сигнал **BPRN** к выводу **BPRO**. Когда арбитр запрашивает шину, доступ обеспечивается, только если сигнал **BPRN** = 0 (и, конечно, если **BUSY** = 1).

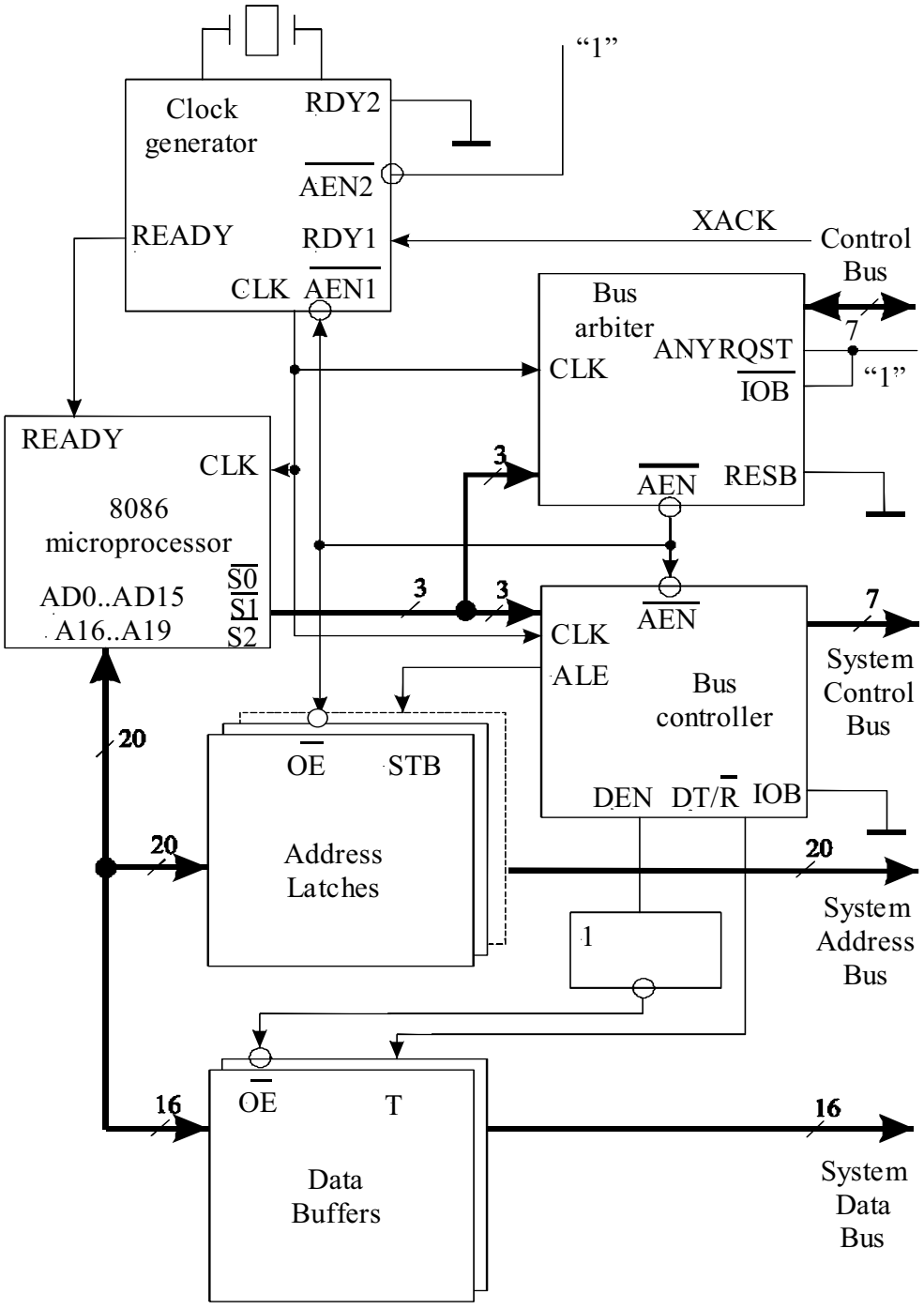


Рис. 5.15. Микропроцессорная система с многопользовательской шиной

Циклический метод арбитража подобен параллельному методу, но приоритеты арбитра переназначаются динамически. Приоритеты устанавливаются более сложной схемой, которая назначает только что обслуженному арбитру самый низкий приоритет и, соответственно, переназначает приоритеты оставшихся арбитров. При применении этого метода все арбитры имеют приблизительно равные возможности использования шины системы.

Из рассмотренных методов предпочтение обычно отдается последовательному, поскольку он предполагает использование существенного числа арбитров шины и не требует усложнения схемы.

Пример микропроцессорной системы показан на рис. 5.15. В ней используются арбитр шины и контроллер шины для связи с многопользовательской шиной системы. Шина системы состоит из адресной шины, шины передачи данных и шины управления. Некоторые устройства системы могут быть подключены к ней. Каждое устройство реализует доступ к шине через промежуточные регистры адреса и буферные приемопередатчики и имеет собственную шину контроллера и арбитра шины. Арбитры шины связаны между собой отдельной шиной управления для разрешения конфликтов и распределения приоритетов каждому арбитру.

Вопросы и задания для повторения

Примеры решения задач

1. Выполните преобразование сигналов микропроцессора $\overline{RD}, \overline{WR}, M/\overline{IO}$ в сигналы $\overline{IORC}$, $\overline{IOWC}$, $\overline{MRDC}$, $\overline{MWTC}$, используя логические элементы «И-НЕ».

Решение

Построим таблицу истинности для задачи (табл. 5.3) и найдем логические выражения, описывающие выходные сигналы. Заметим, что логические функции определены не для каждой комбинации входных сигналов. Знак "×" показывает, что данная комбинация сигналов невозможна. Упростим логические функции, используя карты Карно. Пример карты для сигнала IORC показан на рис. 5.16. В карте Карно "×" может рассматривать или как 1 или как 0 в зависимости от удобства разработчика. Группируя элементы карты, как показано сплошными линиями, получим следующее выражение для IORC:

$$IORC = RD + M/IO.$$

Применяя законы Де Моргана, получим $IORC = \overline{\overline{RD} \cdot \overline{M/IO}}$.

Таблица 5.3

Таблица истинности для задачи 1

Входы			Выходы				Циклы
RD	WR	M/IO	MRDC	MWTC	IORC	IOWC	
0	0	0	×	×	×	×	
0	0	1	×	×	×	×	
0	1	0	1	1	0	1	IO-чтение
0	1	1	0	1	1	1	Чтение памяти
1	0	0	1	1	1	0	IO-запись
1	0	1	1	0	1	1	Запись памяти
1	1	0	1	1	1	1	Нет операции
1	1	1	1	1	1	1	Нет операции

RD·WR / M/IO	00	01	11	10
00	x	0	1	1
01	x	1	1	1

Рис. 5.16. Карта Карно для сигнала IORC

По аналогии получим выражения:

$$\overline{IOWC} = \overline{\overline{WR} \cdot \overline{M/IO}},$$

$$\overline{MRDC} = \overline{\overline{RD} \cdot M/IO},$$

$$\overline{MWTC} = \overline{\overline{WR} \cdot M/IO}.$$

Реализация полученных выражений приведена на рис. 5.17.

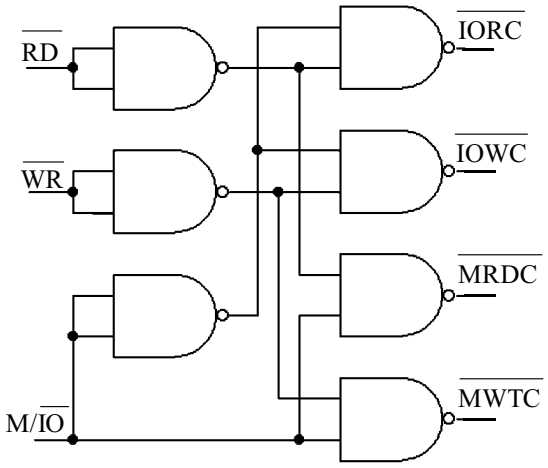


Рис. 5.17. Схема для задачи 1

Замечание. Приведенное решение не единственное. К примеру, если в карте Карно группировать ячейки, как показано штриховыми линиями, логические выражения и схема будут другими.

2. Нарисуйте блок-схему вычислительной системы на основе микропроцессора 8086 и сопроцессора 8087.

Решение

Микропроцессор 8086 является универсальным микропроцессором. Однако на практике часто необходимо проводить вычисления с числами в плавающем формате или вычислять сложные математические функции. Такие задачи могут быть решены микропроцессором, но затраченное время будет слишком велико. Для таких задач разработан специализированный процессор, который называют математическим сопроцессором. Он работает в паре с основным процессором, помогая ему более эффективно проводить вычисления.

Микросхема 8087 – математический сопроцессор, предназначенный для работы совместно с микропроцессором 8086. Сопроцессор поддерживает все необходимые типы данных и операций, программно и аппаратно совместим с микропроцессором. Все команды и типы данных используются программистом таким же образом, что и в основном процессоре.

Сопроцессор 8087 следит за локальной шиной основного процессора, ожидая появления специальной команды основного процессора ESC. Эта команда обеспечивает механизм получения команды из потока команд основного процессора и позволяет использовать режимы адресации основного процессора. Микропроцессор 8086 выполняет команду NOP для команды ESC, но при этом осуществляет вычисления адреса операнда в памяти и помещает его на шину системы. Сопроцессор 8087 может сохранить этот адрес или, если необходимо, провести чтение данных. Чтобы прочитать или записать несколько последовательных слов данных, 8087 должен взять управление шиной системы на себя. Слежение за выборкой команд осуществляется через шину данных и биты цикла S2…S0, а выполнение команд отслеживается через сигналы состояния очереди команд QS0 и QS1. Для выполнения всех названных операций микропроцессор 8086 должен работать в максимальном режиме. Если 8087 собирается захватить шину системы, он должен использовать сигналы RQ/GT.

Так как команды выбираются из памяти только основным процессором, за командой ESC должна следовать команда WAIT для предотвращения выполнения команды, когда предыдущая еще не закончена сопроцессором. Команда WAIT проверяет вход микропроцессора. Если

вход TEST имеет низкий логический уровень, выполнение потока команд продолжается, иначе микропроцессор переходит в режим ожидания. Вход TEST управляется выходом BUSY сопроцессора 8087.

Блок-схема системы показана на рис. 5.18.

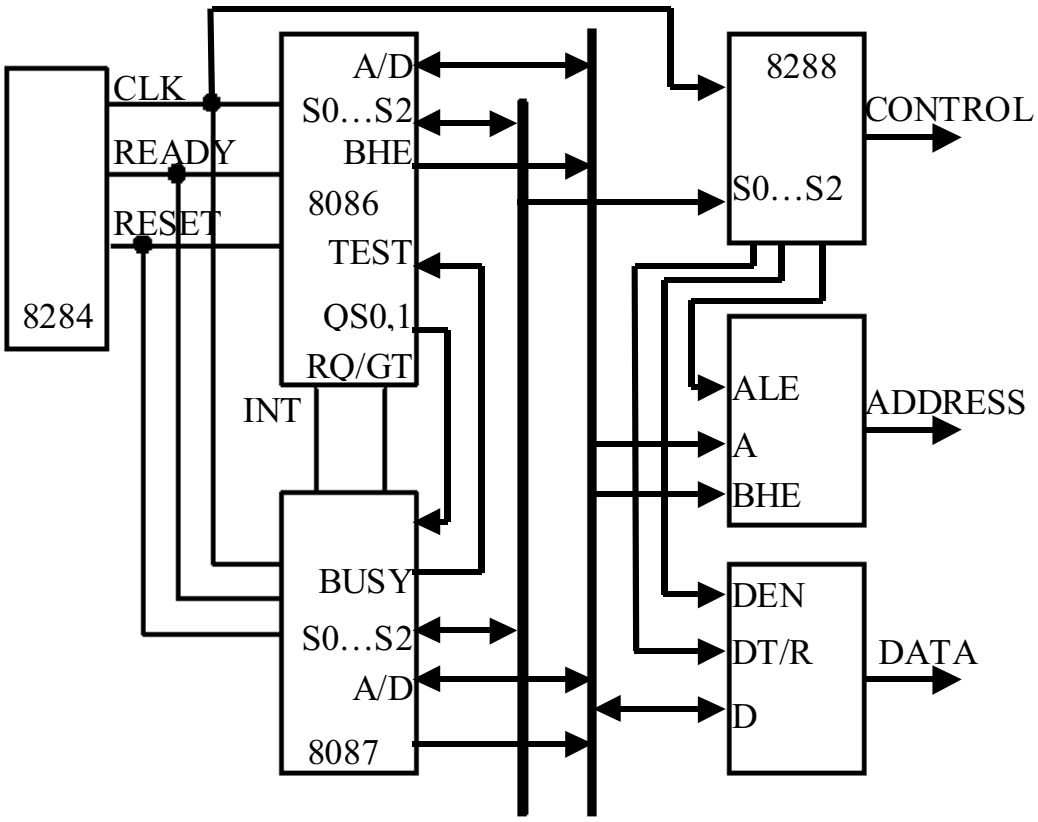


Рис. 5.18. Блок-схема микропроцессорной системы (пример 2)

Задачи и упражнения

Микропроцессорные системы на основе микропроцессора 8086 в минимальном режиме

5.1. Нарисуйте схему системного синхрогенератора на микросхеме 8284: a) с кварцевым резонатором; b) LC-контуром; c) внешним генератором.

5.2. Нарисуйте цепи для сигнала RESET микросхемы 8284. Вычислите требуемые номиналы R и C, если длительность сигнала RESET должна составлять 1 миллисекунду.

5.3. Нарисуйте цепи сигнала READY микросхемы 8284. Объясните назначение этого сигнала.

5.4. Почему микропроцессор использует мультиплексированную шину адреса/данных? Покажите, как можно разделить эти шины в системе.

5.5. Нарисуйте принципиальную схему разделения шин адреса и данных. Постарайтесь минимизировать количество использованных микросхем.
5.6. Объясните, что необходимо для увеличения нагрузочной способности шины данных. Нарисуйте принципиальную схему.
5.7. Выполните преобразование сигналов $\overline{RD}, \overline{WR}, M/\overline{IO}$ микропроцессора в сигналы $\overline{IORC}$, $\overline{IOWC}$, $\overline{MRDC}$, $\overline{MWTC}$, используя мультиплексор.

Многопроцессорные и многошинные системы

5.8. Перечислите основные характеристики микропроцессора 8086 в максимальном режиме.
5.9. Что такое контроллер шины? Почему мы должны его использовать в максимальном режиме? Нарисуйте условное обозначение контроллера шины 8288. Объясните назначение его выходов.
5.10. Что такое арбитр шины? Почему он нужен в многопроцессорной системе? Нарисуйте условное обозначение арбитра шины 8289. Объясните назначение выводов.
5.11. Нарисуйте блок-схему приоритетного арбитража шины. Объясните ее работу.
5.12. Что такое системная шина, локальная шина, шина ввода/вывода в многошинной системе?
5.13. Нарисуйте блок-схему системы на основе микропроцессора 8086 в максимальном режиме: a) с локальной и системной шинами; b) с системной шиной и шиной ввода/вывода.

Глава 6
ОРГАНИЗАЦИЯ ПАМЯТИ МИКРОПРОЦЕССОРНЫХ СИСТЕМ

Микропроцессорные системы используют память для хранения команд, данных и другой информации. Системы памяти отличаются друг от друга по **способам доступа** к ним, по **объему памяти**, **энергонезависимости**, **стоимости хранения** в расчете на бит информации, **времени доступа**.

Вычислительные системы используют обычно целую иерархическую структуру систем памяти, как это показано на рис. 6.1. Память вычислительной системы можно разделить на **внутреннюю** память, если любой ее элемент доступен процессору непосредственно, и **внешнюю**, если это не так. Прямо или произвольно адресуемая память представляет собой последовательность нумерованных ячеек, доступ к которым осуществляется с помощью адресных сигналов, определяющих номер ячейки, и специальных стробирующих сигналов, определяющих момент чтения или записи.

Системы внешней памяти используются для хранения больших объемов информации. К ним относятся накопители на магнитных дисках, накопители на магнитных лентах, оптические системы, такие как CDROM, и другие приборы. При взаимодействии с системами внешней памяти вычислительная система переносит блоки информации из нее во внутреннюю память и выбирает данные уже из нее.

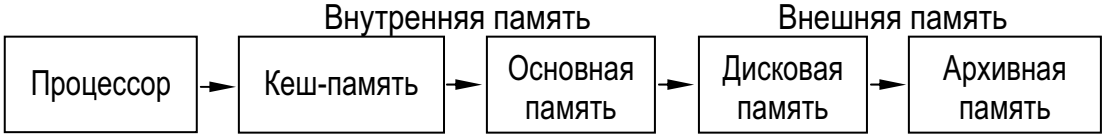


Рис. 6.1. Иерархическая структура памяти

В этой главе рассматриваются принципы построения и функционирования **внутренней памяти**. Системы вторичной памяти доступны для процессора как периферийное оборудование через порты ввода/вывода.

6.1. Основные принципы организации памяти

С точки зрения системы команд память – это набор **слов,** каждое из которых имеет уникальный **адрес,** показывающий расположение слова в памяти. Концепция адресов памяти эквивалента концепции телефонных номеров. Каждый телефон имеет свой собственный номер в некотором поле возможных номеров. Подобно этому, каждая ячейка памяти имеет адрес, который определяет модуль памяти и расположение ячейки в этом модуле.

Каждое слово памяти содержит один или более адресуемых байтов. Количество адресуемых байтов определяется разрядностью микропроцессора. Например, восьмибитные микропроцессоры имеют байтовую организацию памяти. За одно обращение микропроцессор может обработать только один байт информации. Шестнадцатиразрядные микропроцессоры могут обращаться к одному или двум байтам одновременно. Современные 32-разрядные микропроцессоры могут работать с 32-разрядными словами, 16-разрядными словами и 8-разрядными байтами. Поэтому память для этих микропроцессоров организована таким образом, чтобы допускать обращение к одному, двум или четырем байтам одновременно. Количество адресуемых ячеек памяти зависит от количества бит шины адреса микропроцессора. 8-разрядные микропроцессоры и микроконтроллеры имеет 16-битную шину адреса, позволяя адресовать 64 кб памяти. 16-разрядные микропроцессоры позволяют адресовать несколько мегабайтов памяти. Наконец, современные микропроцессоры используют 64-разрядную шину адреса, что позволяет адресовать фактически бесконечный объем памяти.

Под памятью цифровых вычислительных систем понимают совокупность технических средств, предназначенных для приема (записи), хранения и выдачи (считывания) информации, представленной двоичным кодом.

Основными характеристиками запоминающих устройств (ЗУ) являются:

- **информационная емкость**, определяемая максимальным объемом хранимой информации в битах или байтах;
- **быстродействие**, характеризуемое **временем выборки информации** из ЗУ и временем цикла обращения к ЗУ с произвольным доступом или временем поиска и количеством переданной в единицу времени информации в ЗУ (или из ЗУ) с последовательным доступом;
- **энергопотребление**, определяемое электрической мощностью, потребляемой ЗУ от источников питания в каждом из режимов работы;
- **стоимость хранения информации** в расчете на один бит;
- **энергонезависимость**, то есть способность сохранять информацию в ЗУ после выключения электропитания;
- надежность, масса, габаритные размеры и др.

Развитие средств вычислительной техники связано с устойчивой тенденцией к увеличению информационной емкости и быстродействия вычислительных систем. Память современных универсальных компьютеров должна иметь информационную емкость $10^7 \ldots 10^{12}$ байт при времени выборки 2…20 нс. Эти параметры должны сочетаться с высокой надежностью, низкой стоимостью, малым энергопотреблением и приемлемыми массой и габаритными размерами. На современном уровне развития техники невозможно реализовать память с требуемыми параметрами в виде единого устройства, поэтому ЗУ современных вычислительных систем имеют многоуровневую иерархическую структуру, позволяющую в определенной мере удовлетворить всем предъявляемым требованиям. Непосредственно связанные с процессором модули памяти составляют верхние уровни иерархии. Они имеют максимальное быстродействие, но относительно малую информационную емкость. На остальных уровнях иерархии модули памяти располагаются по мере увеличения информационной емкости и связанного с этим уменьшения быстродействия. Запоминающие устройства верхних уровней иерархической памяти образуют внутреннюю память, а ЗУ нижних уровней – внешнюю память компьютеров.

Система памяти, состоящая из различных по техническим характеристикам модулей ЗУ, с точки зрения пользователя должна функционировать как единый блок памяти, обладающий быстродействием, близким к быстродействию верхнего уровня, и емкостью нижних уровней. Практически даже при самой эффективной организации обмена информацией между уровнями невозможно избежать потерь времени при обращении к данным, размещенным на нижних уровнях, что непосредственно сказывается на производительности вычислительной системы. Тем не менее, в силу технических ограничений, связанных, в основном, с возможностями элементной базы, в настоящее время не существует альтернативных решений, позволяющих строить одноуровневые системы памяти.

Для построения ЗУ современных ЭВМ используется обширный арсенал технических средств.

По виду носителя информации ЗУ могут быть: ферромагнитные, электромагнитные, сегнетоэлектрические, оптические, ультразвуковые, на основе сверхпроводимости и электронные. Среди последних значительное место занимают полупроводниковые ЗУ, выполненные в виде микросхем основной элементной базы внутренних ЗУ современных вычислительных систем.

Скорости работы процессоров значительно увеличиваются из года в год. В результате проектирование запоминающих систем, которые могут передавать информацию без задержек процессора, становится более трудным. Кристаллы полупроводниковой памяти с короткими временами доступа очень дороги. Альтернативой построения памяти большой

емкости из дорогих высокоскоростных кристаллов является использование маленькой высокоскоростной памяти, чтобы сохранять информацию, наиболее вероятно используемую процессором, в то время как остальная информация хранится в основной памяти. Основная память может быть построена на менее быстродействующих и поэтому менее дорогих кристаллах памяти. Быстродействующая память небольшого объема называется **кеш**ем или **кеш-памятью**.

Обращения к памяти часто демонстрируют свойство, называемое локализацией. Понятие «**пространственная локализация**» обращения означает, что если произошло обращение к ячейке ЗУ, то имеется высокая вероятность, что следующее обращение будет к следующей ячейке памяти. Программы имеют тенденцию показывать высокую степень пространственной локализации, потому что команды выбираются от последовательных ячеек ЗУ, пока не происходит ветвления. Понятие «**временная локализация**» означает, что если произошло обращение к ячейке памяти по некоторому адресу, то имеется высокая вероятность, что по этому адресу будут обращаться снова в ближайшем будущем. Данные часто показывают хорошую временную локализацию так же, как и команды, которые содержатся в циклах.

Кеш-память использует локализацию обращений к памяти, помещая в высокоскоростной памяти копию области оперативной памяти, которая наиболее вероятно будет затребована процессором. Каждое обращение к памяти выполняется сначала как обращение к кешу. Если требуемая информация там обнаружена, то считается, что происходит удачное обращение к кешу, и информация быстро передается в процессор. Если происходит неудачное обращение к кешу, то есть если информация в кеше не найдена, то нужно обратиться к более медленной оперативной памяти.

Среднее время обращения к памяти, при наличии кеша, определяется формулой:

$$T_{access} = H\,T_{cache} + (1 - H)\,(T_{cache} + T_{main}),$$

где H – коэффициент совпадения кеша, то есть процент удачных обращений к кешу; T_{cache} и T_{main} – времена доступа к кешу и основной памяти, соответственно.

Например, запоминающая система с коэффициентом совпадения 90 %, T_{cache} = 20 нс и T_{main} = 100 нс имела бы среднее время обращения T_{access} = 30 нс, которое является намного ближе к таковому для кеш-памяти, чем для более медленной основной памяти.

Записи в память в системах с кеш-памятью могут быть обработаны двумя различными способами. При использовании способа **запись через кеш** (write through) каждый элемент записывается непосредственно в основную память в каждом цикле записи, кеш модифицируется одно-

временно. При использовании способа **обратной записи** (write back) все записи делаются только в кеш, оставляя кеш и содержание оперативной памяти временно неодинаковыми. Позже, когда информация в кеше должна быть заменена, все измененные элементы кеша копируются назад, в оперативную память. Это уменьшает общее количество обращений к основной памяти.

6.2. Элементная база запоминающих устройств

Большинство вычислительных систем до начала 70-х годов использовали память на основе ферритовых колец. Некоторые специальные вычислительные системы, особенно применяемые в космосе, где необходима радиационная стойкость, используют такую память до сих пор. Существенный недостаток такой памяти – очень большие габариты. Однако достоинством является энергонезависимость. В других вычислительных системах в основном применяется полупроводниковая память.

Элементной базой полупроводниковых ЗУ являются большие интегральные микросхемы (БИС) памяти. В настоящее время основной объем промышленного выпуска микросхем памяти составляют микросхемы оперативных ЗУ (ОЗУ) и постоянных ЗУ (ПЗУ). Отличие ОЗУ от ПЗУ состоит в том, что в ОЗУ можно записывать и считывать информацию, а в ПЗУ – только считывать.

Микросхемы оперативных ЗУ (БИС ОЗУ) отличаются наиболее широкой номенклатурой среди микросхем памяти. Они выпускаются во всевозможных конструктивно-технологических вариантах емкостью от 16 бит до 16 Мбит.

Интегральные ПЗУ выпускаются трех видов:

- **масочные ПЗУ** (**ROM**), программируемые в процессе изготовления с помощью индивидуальных фотошаблонов (масок) по заказам потребителя;
- **однократно программируемые ПЗУ** (**ППЗУ, PROM**), в которых запись информации производится потребителем путем избирательного нарушения однородности исходной матрицы ЭП импульсами электрического тока;
- **репрограммируемые ПЗУ** (**РПЗУ, EPROM, EEPROM**), в которых запись информации может изменяться неоднократно электрическим способом.

Свойства постоянных запоминающих устройств отражены в табл. 6.1.

Таблица 6.1

Полупроводниковые системы памяти

Память	Энергонезависимость	Высокая плотность элементов	Низкое энергопотребление	Однотранзисторная запоминающая ячейка	Возможность перезаписи в системе	Хранение данных и программ
ROM/PROM	√	√	√	√		
EPROM	√	√	√	√		
EEPROM	√		√		√	√
Flash	√	√	√	√	√	√
SRAM					√	√
DRAM		√			√	√

Основной составной частью микросхемы **ОЗУ** является массив элементов памяти, объединенных в матрицу накопителя. Элемент памяти (ЭП) может хранить один бит (0 или 1) информации. Каждый ЭП имеет свой адрес. Для обращения к ЭП необходимо его «выбрать» с помощью кода адреса, сигналы которого подводят к соответствующим выводам микросхемы.

Для ввода и вывода информации служит вход и выход микросхемы. Для управления режимом микросхемы памяти необходим сигнал «Запись-считывание», значение 1 которого определяет режим записи бита информации в ЭП, а 0 – режим считывания бита информации из ЭП. Такую организацию матрицы накопителя, при которой одновременно можно записывать или считывать один бит, называют одноразрядной. Некоторые микросхемы ОЗУ имеют одноразрядную организацию. Но большая часть из них имеют многоразрядную организацию, иначе называемую «словарной». У таких микросхем несколько информационных входов и столько же выходов, и поэтому они допускают одновременную запись (считывание) многоразрядного кода, который принято называть «словом».

Микросхемы ОЗУ по типу ЭП разделяют на **статические (SRAM)** и **динамические (DRAM)**. В микросхемах статических ОЗУ в качестве ЭП применены статические триггеры на биполярных или МДП-транзисторах. Как известно, статический триггер способен при наличии напряжения питания сохранять свое состояние неограниченное время. Число состояний, в которых может находиться триггер, равно двум, что и позволяет использовать его для хранения двоичной единицы информации.

В микросхемах динамических ОЗУ элементы памяти выполнены на основе электрических конденсаторов, сформированных внутри полупроводникового кристалла. Такие ЭП не могут долгое время сохранять свое состояние, определяемое наличием или отсутствием электрического заряда, и поэтому нуждаются в периодическом восстановлении (регенерации). Микросхемы динамических ОЗУ отличаются от микросхем статических ОЗУ большей информационной емкостью, что обусловлено меньшим числом компонентов в одном ЭП и, следовательно, более плотным их размещением в полупроводниковом кристалле. Однако динамические ОЗУ сложнее в применении, поскольку нуждаются в организации принудительной регенерации и в усложнении устройства управления.

Микросхемы ПЗУ построены также по принципу матричной структуры накопителя. Функции ЭП в микросхемах ПЗУ выполняют перемычки в виде проводников, диодов или транзисторов между шинами строк и столбцов в накопителе. В такой матрице наличие перемычки соответствует, например, 1, а ее отсутствие – 0. Микросхемы ПЗУ имеют словарную организацию, и поэтому информация считывается в форме многоразрядного кода, т. е. словом. Совокупность ЭП в матрице накопителя, в которой размещается слово, называют ячейкой памяти (ЯП). Число ЭП в ЯП определяет ее разрядность n. Каждая ЯП имеет свой адрес, и для обращения к определенной ЯП нужно подвести сигналы кода, соответствующего данной ячейке адреса. Число ячеек памяти равно 2^m, где m – количество двоичных разрядов адреса, а информационная емкость микросхемы – $2^m \times n$ бит.

Занесение информации в микросхемы ПЗУ, т. е. их программирование, осуществляют, в основном, двумя способами. Один способ заключается в формировании в накопителе перемычек в местах пересечения строк и столбцов матрицы через маску на заключительной технологической стадии изготовления микросхемы ПЗУ. Такие микросхемы ПЗУ называют масочными. Другой способ программирования микросхемы ПЗУ основан на пережигании легкоплавких перемычек в тех пересечениях шин строк и столбцов, куда должен быть записан 0 или 1 в зависимости от принятого кодирования. В исходном состоянии такая микросхема имеет в матрице перемычки во всех пересечениях строк и столбцов. Пользователь осуществляет программирование ПЗУ с помощью специального устройства для программирования, называемого программатором.

Микросхемы ПЗУ, масочные (ПЗУМ) и программируемые пользователем (ППЗУ) допускают однократное программирование, поскольку оно осуществляется формированием или разрушением соединений в матрице. Один из вариантов реализации ПЗУ ориентирован на программирование заданных логических функций. Такие ПЗУ называют программируемыми логическими матрицами (ПЛМ).

Существует разновидность микросхем ПЗУ, допускающая неоднократное перепрограммирование (репрограммирование). Элементом памяти в микросхемах репрограммируемых ПЗУ (РПЗУ) является МДП-транзистор, обладающий свойством переходить в состояние проводимости под воздействием импульса программирующего напряжения и сохранять это состояние длительное время (тысячи часов). Данный эффект обусловлен накоплением электрического заряда в подзатворном диэлектрике. Если на транзистор не воздействовать импульсом программирующего напряжения, то он сохранит закрытое для электрического тока состояние. Для стирания информации перед новым циклом программирования необходимо вытеснить накопленный под затвором заряд. В зависимости от способа выполнения этой операции микросхемы РПЗУ разделяют на два вида: со стиранием электрическим сигналом (РПЗУ-ЭС) и ультрафиолетовым светом (РПЗУ-УФ), которым полупроводниковый кристалл облучают через специальное окно в крышке корпуса. Микросхемы РПЗУ сохраняют информацию длительное время без питания, т. е. являются энергонезависимыми.

Обратимся к рис. 6.2, на котором представлено условное изображение микросхемы статического ОЗУ К561РУ2. Число адресных входов $A_0 - A_7$ (A_0 – младший разряд) позволяет определить информационную емкость микросхемы: $2^8 = 256$ бит. Наличие одного информационного входа DI и одного выхода DO (прямого и инверсного) указывает на одноразрядную организацию микросхемы памяти: 256x1 бит.

Для управления режимом работы предусмотрены два сигнала: CS (ВМ – выбор микросхемы) и WR (запись-считывание). Управляющий вход CS является инверсным. Сигнал CS разрешает или запрещает обращение к микросхеме по информационным входу и выходу. Наличие на входе CS-сигнала с уровнем логической 1 однозначно определяет режим хранения. При этом выход принимает высокоомное состояние, при котором он электрически отключен от приемника информации.

Для обращения к микросхеме для записи или считывания одного бита информации необходимо подать разрешающий сигнал CS с нулевым уровнем и сигнал WR с соответствующим режиму уровнем: при записи – 1, при считывании – 0. В любом режиме вход и выход развязаны, т. е. не могут влиять на состояние друг друга. Таким свойством обладают микросхемы с выходами на три состояния.

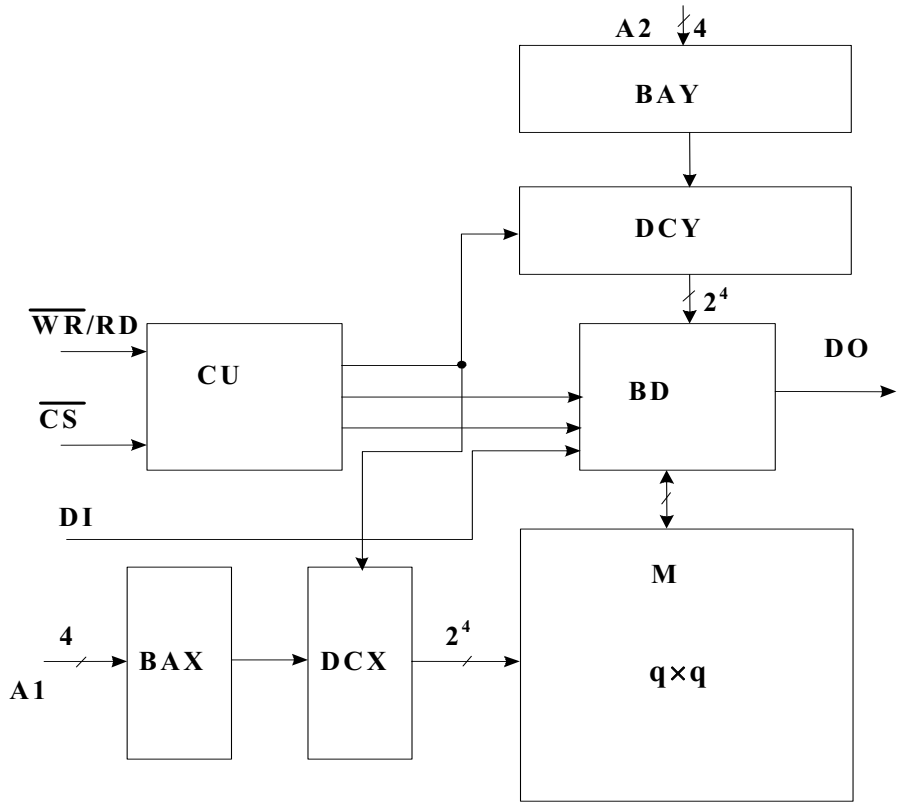


Рис. 6.2. Структура микросхемы ЗУ с одноразрядной организацией

Учитывая отмеченную особенность, можно объединить вход и выход микросхемы и подключить их к общей информационной шине, по которой информация подается к микросхеме и выводится из нее.

Для построения ОЗУ на микросхемах с одноразрядной организацией необходимо объединить микросхемы с тем, чтобы обеспечить возможность записи информации в ОЗУ, ее хранение и считывание в форме многоразрядного цифрового кода, т. е. слова.

Очевидно, что решение этой задачи существенно упрощается при использовании микросхем со словарной организацией. В обширной номенклатуре микросхем статических ОЗУ микросхемы со словарной организацией представлены большим числом типов. Один из них – микросхема КР537РУ8 – приведен на рис. 6.3. Особенность микросхемы состоит в том, что она имеет организацию 2048х8 бит и, следовательно, допускает запись или считывание информации 8-разрядными словами (байтами). Информационные входы и выходы микросхемы совмещены и обладают свойствами двунаправленной проводимости. Другая особенность приведенной микросхемы заключается в наличии сигнала ОЕ.

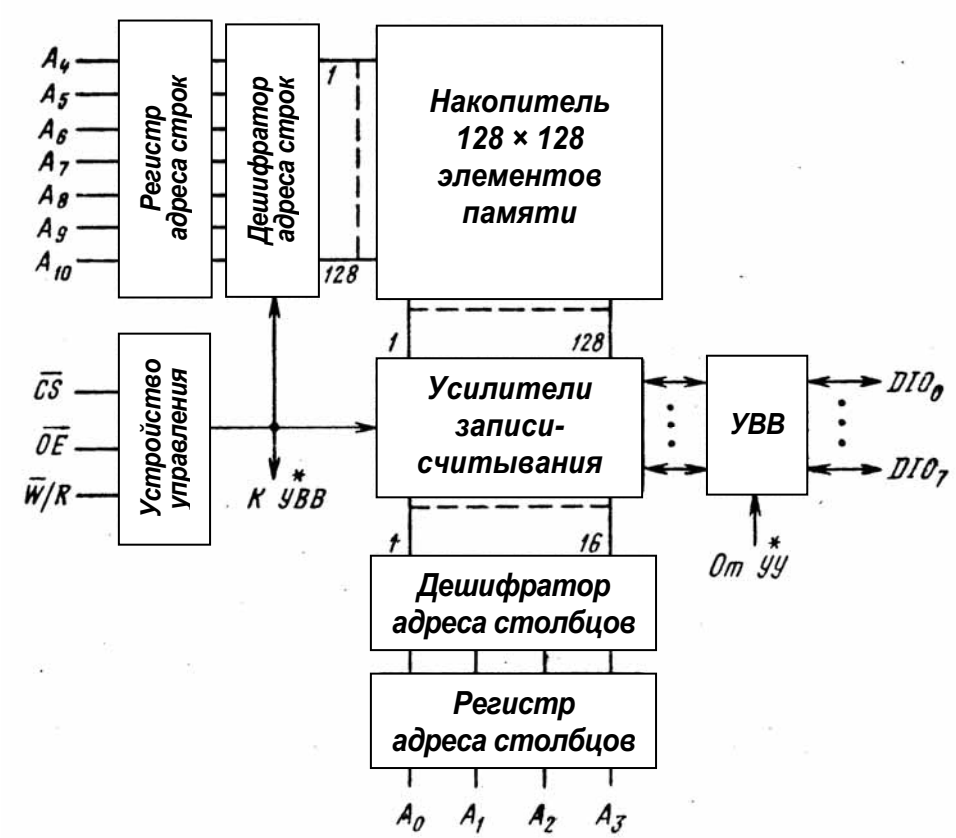


Рис. 6.3. Структура микросхемы ЗУ со словарной организацией

Микросхемы динамических ОЗУ имеют более сложное управление, чем микросхемы статических ОЗУ. Это обусловлено необходимостью организации принудительной регенерации хранимой микросхемой информации, осуществляемой с помощью специальных внешних устройств с интервалом, определяемым периодом регенерации. Для микросхем серии К565 этот период равен 2 мс. Микросхемы динамических ОЗУ в своем большинстве построены с мультиплексированием кода адреса: вначале в микросхему вводят код адреса строки, фиксируя его во входном регистре стробирующим сигналом RAS, затем вводят код адреса столбца, фиксируя его сигналом CAS. Число адресных выводов таким образом уменьшается вдвое: у микросхемы с информационной емкостью 16 кбит их всего восемь. В микросхеме функция сигнала выбора кристалла выполняет сигнал RAS.

В режиме регенерации микросхема работает по циклу *считывание–модификация–запись*, находясь при этом в состоянии изоляции от информационных входа и выхода, так как сигнал CAS не активен. Следовательно, адресованы оказываются только строки. Это говорит о том,

что регенерация информации происходит во всех элементах памяти строки. Перебирая адреса строк, устройство регенерации обеспечивает восстановление информации во всей матрице накопителя. Время, необходимое для регенерации информации в микросхеме, определяют произведением числа строк на время одного цикла регенерации.

Микросхемы статических ОЗУ подразделяют по виду управляющих сигналов на асинхронные и тактируемые. Для тактируемых ОЗУ установлено требование подавать сигнал CS импульсом. Важным моментом в этом требовании является то, что переход микросхемы в активное состояние происходит в момент поступления сигнала CS. Асинхронные микросхемы допускают подачу управляющих сигналов уровнями или импульсами.

6.3. Принципы построения ЗУ на микросхемах памяти

Организация памяти МП системы определяется, прежде всего, разрядностью шин адреса и данных процессора. Кристалл памяти организован как $2^k \times n$, что означает, что имеются k линий адреса и n линий ввода/вывода данных, и, таким образом, мы имеем 2^k n-битных адресуемых ячеек. Дешифратор адреса в пределах кристалла выбирает одну ячейку, соответствующую каждому k-битному адресу, и записывает в нее или считывает из нее информацию.

Использование ЗУ, организованных как $2^k \times n$ в системе, которая имеет I линий адреса, где $I > k$, означает, что система может адресовать 2^{I-k} таких устройств. Эти ЗУ обычно организовываются иерархически в банки, как показано на рис. 6.4. Адрес памяти разделяется на поля: код банка, номер кристалла и адрес «на кристалле». Каждая часть адреса декодируется на различном уровне иерархии. На высшем уровне выбирается один из банков, на следующем выбирается кристалл в пределах банка, «на кристалле» выбирается одна ячейка в пределах выбранной микросхемы.

Если разрядность шины данных системы m больше, чем число входов данных n на кристалле памяти, то нужно использовать m/n кристаллов, чтобы создать один адресуемый блок памяти.

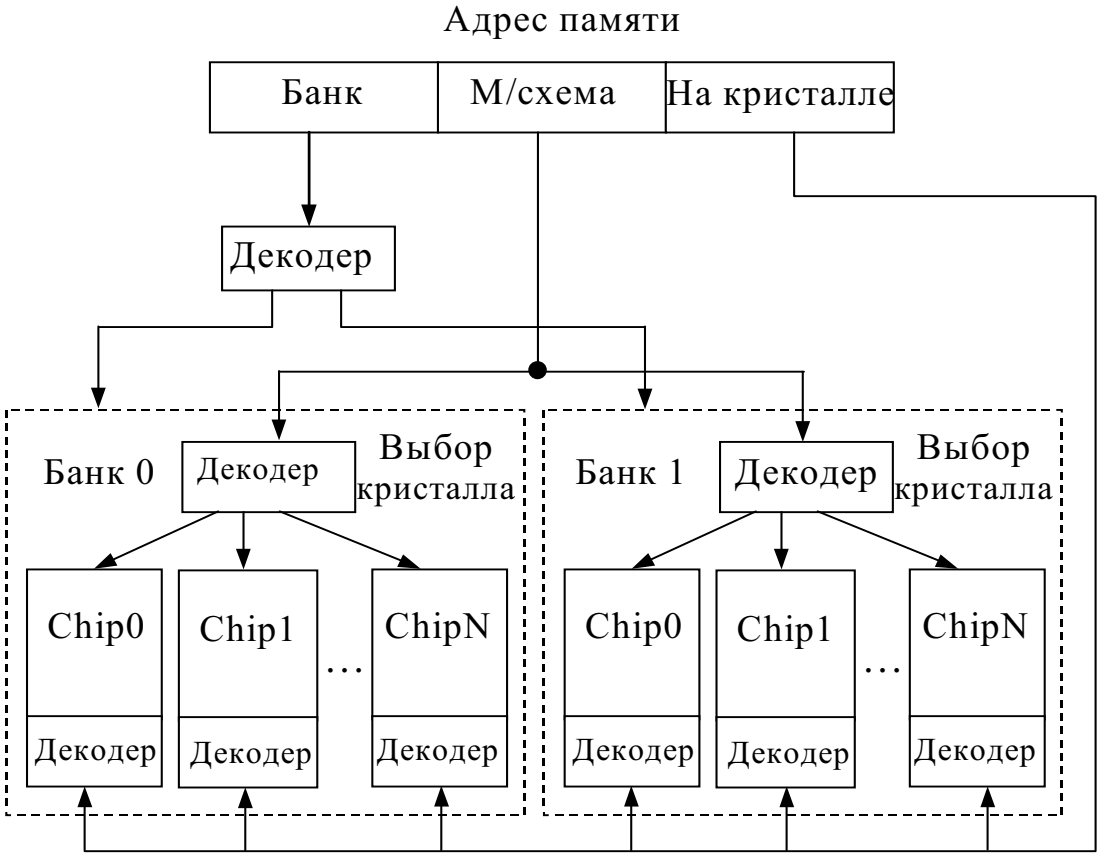


Рис. 6.4. Иерархическое декодирование памяти

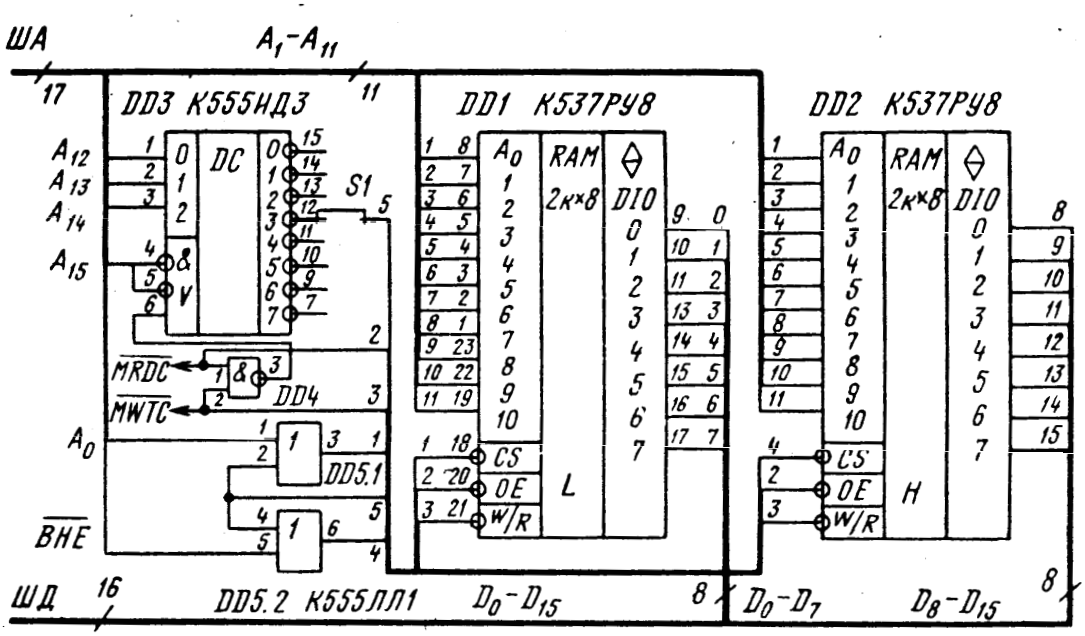


Рис. 6.5. Модуль ОЗУ на микросхемах с байтовой организацией

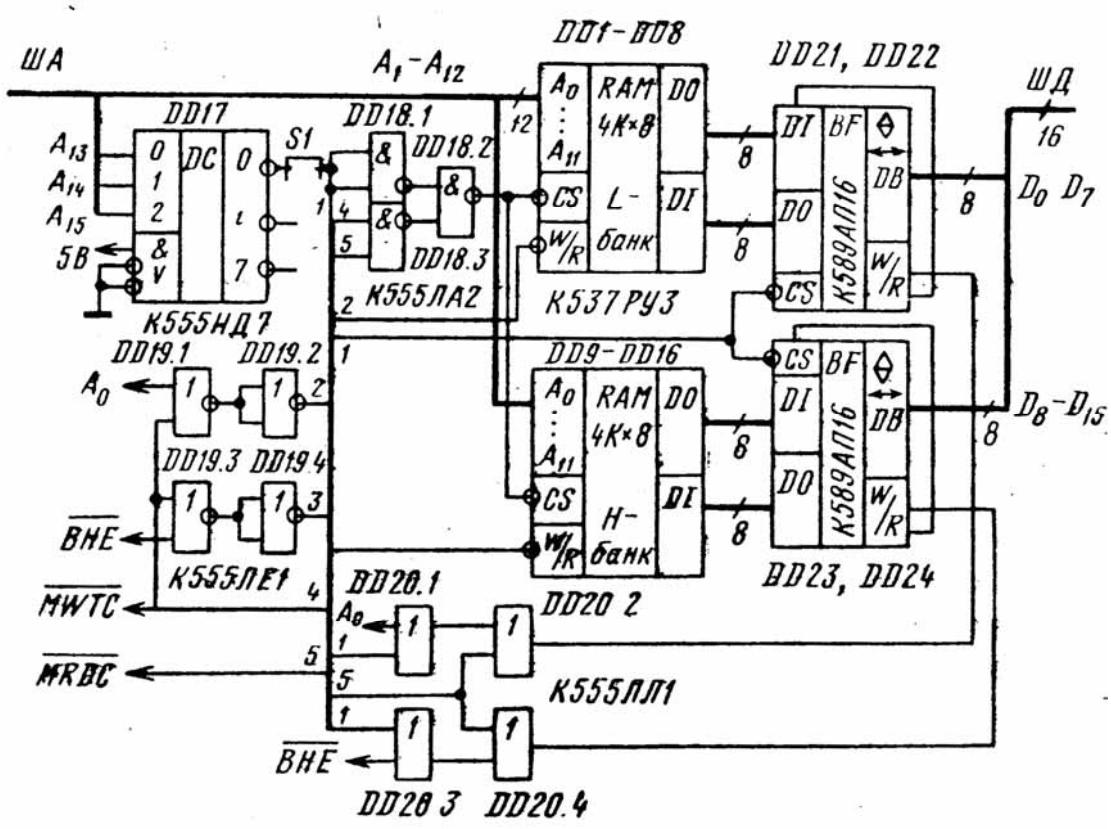


Рис. 6.6. Модуль статического ОЗУ на микросхемах с битовой организацией

Пример построения блока ОЗУ, ориентированный на 16-разрядную шину данных с использованием микросхем с байтовой организацией, приведен на рис. 6.5. Модуль ОЗУ состоит из двух банков емкостью по 2 кб каждый. Банк L хранит младшие байты, банк H – старшие. Информационные выводы младшего банка подключены к младшим линиям шины данных, а выводы старшего банка – к старшим разрядам шины данных. Управление доступом к банкам и режимом их работы осуществляют сигналами MRDC, MWTC, BHE, A_0. Блок ОЗУ выполняет операции записи и считывания как 16-разрядного слова, так и любого из двух байтов. Сигналы выбора банков формирует логическая схема на элементах **ИЛИ** в зависимости от значения сигналов A_0, **BHE** и при наличии сигнала разрешения, снимаемого с выхода дешифратора. Этот сигнал формируется, если на дешифратор со старших разрядов шины адреса поступает нужный адрес.

При использовании микросхем памяти, не имеющих специального входа для сигнала разрешения считывания, усложняется схема устройства управления. Вариант построения блока ОЗУ на таких микросхемах приведен на рис. 6.6. Модуль состоит из двух банков, информационные входы и выходы которых соединены с 16-разрядной шиной данных через специ-

альные двунаправленные буферы. Селектор адреса и логическая схема при обращении к блоку формируют сигнал выбора для микросхем памяти и буфера шины данных. Сигналы управления режимом записи и считывания для микросхем формируется специальной логической схемой.

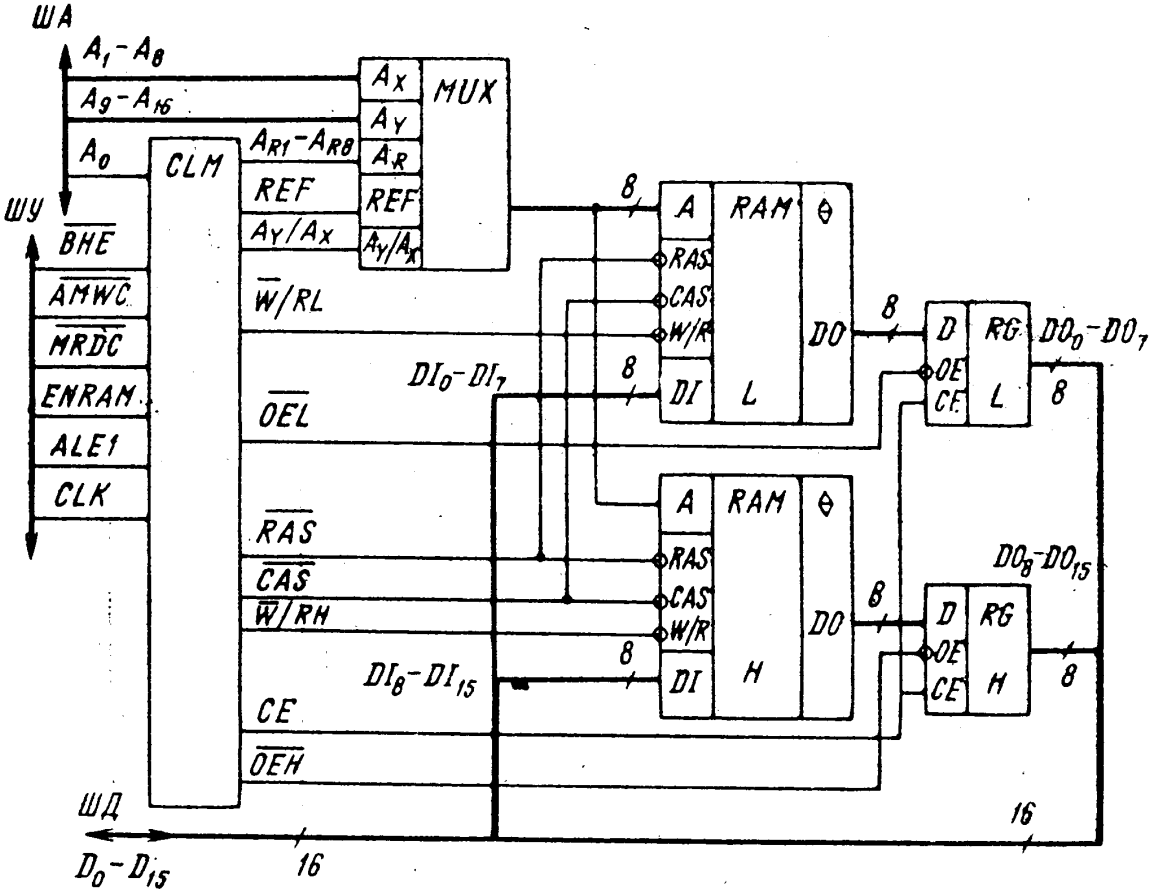


Рис. 6.7. Функциональная схема динамического ОЗУ

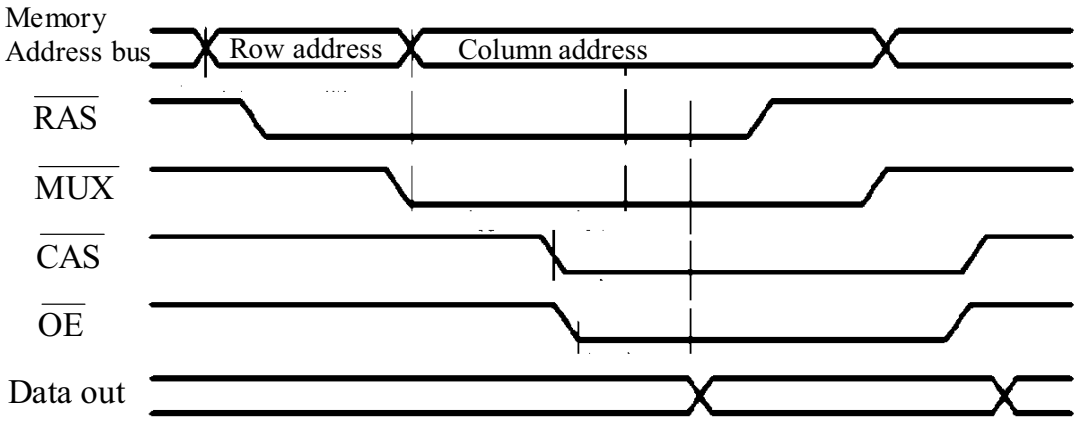


Рис. 6.8. Временная диаграмма цикла чтения динамической памяти

Структурная схема блока динамического ОЗУ приведена на рис. 6.7. Структура ОЗУ также ориентирована на 16-разрядную шину

данных и включает 2 байтовых банка данных, имеющих раздельные входы управления режимом записи/считывания, два буферных регистра с раздельным управлением разрешения выхода, мультиплексор и контроллер. Мультиплексор обеспечивает последовательный во времени ввод адресного кода строк и столбцов в модуль ОЗУ. В режиме регенерации мультиплексор коммутирует на входы адреса адрес регенерации.

Основная идея состоит в том, что при обращении к памяти адрес разбивается на две части: адрес строки и адрес столбца, передаваемые друг за другом. В функции контроллера динамического ОЗУ входит формирование сигналов управления **RAS**; **CAS**; **MUX**; **WE**; **OE** в соответствии с временной диаграммой (рис. 6.8) и обеспечение регенерации микросхем динамической памяти.

Вопросы и задания для повторения

Примеры решения задач

1. Сколько транзисторов экономится при переходе от 4-транзисторной схемы динамического элемента памяти к 1-транзисторной схеме элемента динамической памяти, если общий объем адресуемой памяти запоминающего устройства составляет 256 кб?

Решение

Четырехтранзисторная ячейка динамической памяти представлена на рис. 6.9. Информация в данной ячейке хранится на паразитных емкостях C1 и C2 между затвором и истоком транзисторов Q1 и Q2,соответственно. Транзисторы Q и Q’ образуют цепь регенерации, которая нужна одна на всю линию запоминающих ячеек.

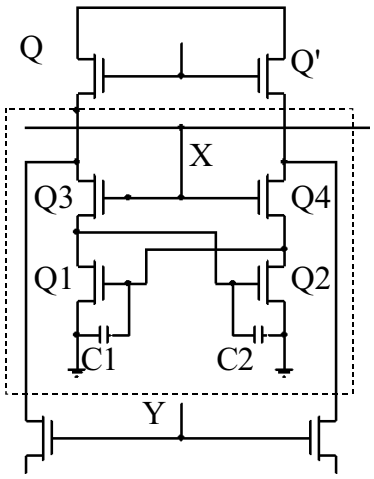


Рис. 6.9. 4-транзисторная ячейка

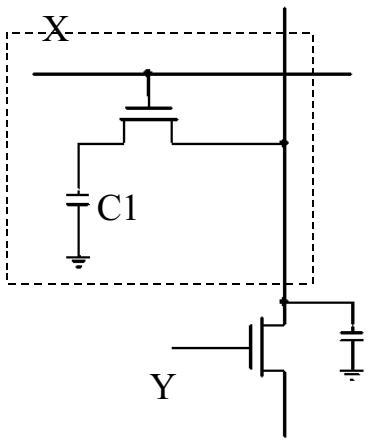


Рис. 6.10. 1-транзисторная ячейка

Возможно построение схемы запоминающей ячейки с использованием одного транзистора (рис. 6. 10). Такая ячейка используется для хранения данных в запоминающих устройствах большого объема. По сравнению с предыдущей схемой в каждой запоминающей ячейке экономится 3 транзистора. Кроме того, мы экономим на транзисторах, используемых в схеме регенерации. Таким образом, для памяти объемом в 256 кбит общее количество сэкономленных транзисторов равно $2^{18}\times3+2\times512 = 787456$.

2. Нарисуйте принципиальную схему модуля памяти, состоящего из 128 кб ОЗУ и 64 кб ПЗУ.

Решение

Сначала выберем микросхемы, которые будем использовать для решения задачи. Выберем микросхему перепрограммируемого ПЗУ 27128 с организацией 16Kx8 бит и микросхему HM62256 с организацией 32Kx8 для ОЗУ. В нашем случае необходимо использовать четыре микросхемы первого типа и четыре микросхемы второго типа. Схема модуля памяти представлена на рис. 6.11. В данной системе использована 20-разрядная шина адреса и 16-разрядная шина данных. Одноименные входы адреса микросхем соединены с соответствующими линиями шины адреса. CE разрешает работы микросхемы и WE разрешает запись в память. Сигналы CE всех микросхем ОЗУ объединены между собой так же, как и сигналы WE. Входы/выходы данных соединены с линиями шины данных. Дешифраторы U7, U13 определяют диапазон адресов, отводимых под ОЗУ и ПЗУ. В нашем случае под ОЗУ отведены адреса в 0000H…FFFFH, под ПЗУ – адреса в диапазоне 10000H…7FFFFH.

Задачи

Основные принципы организации памяти

6.1. Объясните, почему компьютерная память организована и адресуется побайтно. Почему микропроцессор 8086 может также адресовать и двухбайтовые слова?

6.2. Перечислите основные параметры памяти. Как эти параметры влияют на стоимость и производительность вычислительной системы?

6.3. Микропроцессор имеет 16-разрядную шину данных и 32-разрядную шину адреса. Каков может быть адресуемый объем памяти для этого микропроцессора? Как должна быть организована эта память?

6.4. Что такое кеш-память? Почему ее необходимо использовать в высокопроизводительных вычислительных системах? В чем преимущества применения кеш-памяти?

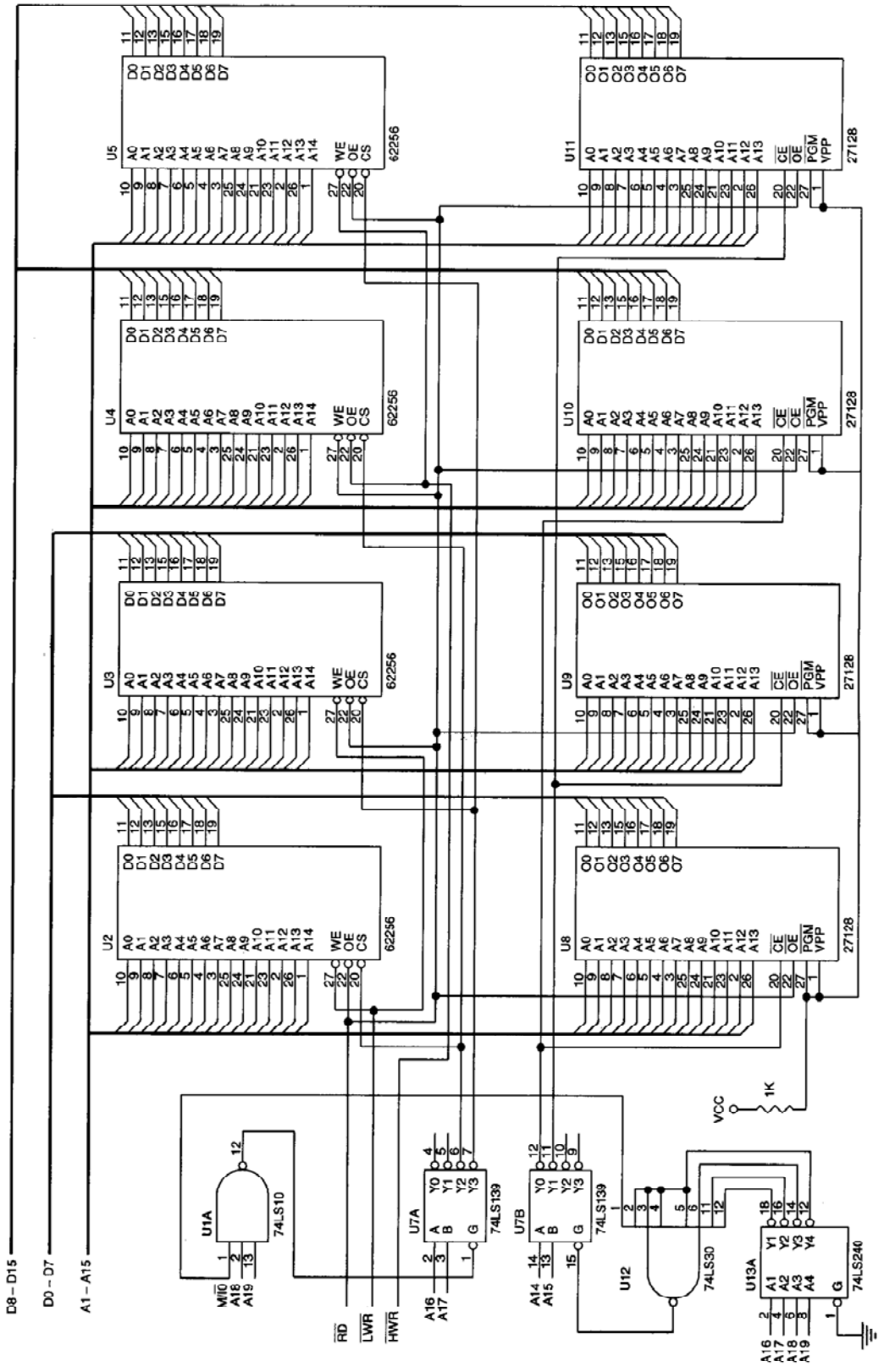


Рис. 6.11. Модуль памяти

6.5. Что такое пространственная и временная локализация? Какие стратегии записи в кеш-память существуют?
6.6. Что такое виртуальная адресация? Почему она используется в вычислительных системах с большим адресуемым объемом памяти? Какие конкретно способы применяются на практике?

Полупроводниковые микросхемы памяти

6.7. Перечислите основные классы полупроводниковых микросхем памяти и их основные параметры.
6.8. Что такое ПЗУ? Нарисуйте блок-схему модуля ПЗУ. Какие аппаратные средства необходимы для построения ПЗУ?
6.9. Что такое масочные ПЗУ, однократно программируемые ПЗУ? Как осуществляется их программирование?
6.10. Перечислите несколько примеров применения ПЗУ.
6.11. Что означают акронимы (сокращения) EPROM и EEPROM?
6.12. Перечислите преимущества полупроводниковых элементов памяти по сравнению с элементами на основе ферритовых колец. Какие преимущества, в свою очередь, память на ферритах имеет по сравнению с полупроводниковыми ОЗУ?
6.13. Нарисуйте схему построения 6-транзисторной статической ячейки памяти. Объясните, как она работает.
6.14. Сколько транзисторов экономится при переходе от 6-транзисторной статической ячейки памяти к 4-транзисторной динамической ячейке в памяти объемом 64 кб?
6.15. Нарисуйте схему 4-транзисторной динамической ячейки памяти. Почему появляется необходимость в дополнительной схеме регенерации информации в ячейке?
6.16. Нарисуйте схему 1-транзисторной ячейки динамической памяти. Поясните, как она работает.

Организация систем памяти

6.17. Почему память микропроцессорных систем имеет иерархическую организацию? Что такое банк памяти?
6.18. Объясните, как построить на основе микросхем статической памяти с организацией 1024x1 бит (например, AM93L415) память объемом 1024x8 бит.
6.19. Объясните, как построить на основе микросхем ППЗУ с организацией 8Kx8 бит (например, 27C64) память объемом 64Kx8 бит.
6.20. Объясните, как построить на основе микросхем статической памяти с организацией 4096x1 бит (например, AM9044) память объемом 16Kx8 бит.

6.21. Нарисуйте принципиальную схему модуля памяти объемом 16 кб на основе микросхем перепрограммируемого ПЗУ с ультрафиолетовым стиранием.
6.22. Нарисуйте принципиальную схему модуля статического ОЗУ объемом 8 кб, используя: a) микросхемы с организацией 4Kx1 бит, b) микросхемы с организацией 2Kx8 бит.
6.23. Спроектируйте систему памяти для микропроцессора 8086 32Kx16 бит ОЗУ и 16Kx16 бит ПЗУ.
6.24. Нарисуйте блок-схему памяти для микропроцессора 8086 на основе банков 64 кб каждый.
6.25. Нарисуйте принципиальную схему модуля динамического ОЗУ объемом 64 кб, используя: a) микросхемы с организацией 64Kx1 бит, b) микросхемы с организацией 64Kx4 бит.

Глава 7
ОРГАНИЗАЦИЯ ВВОДА/ВЫВОДА В МИКРОПРОЦЕССОРНОЙ СИСТЕМЕ

Каждое **внешнее устройство** должно быть связано с помощью интерфейса с шиной данных микропроцессорной системы так, чтобы данные могли быть переданы командами программы между этим устройством и микропроцессором. Каждый интерфейс устройства ввода/вывода должен быть **адресуемым** и отвечать на сигналы шины управления микропроцессора. Наиболее общий подход состоит в том, чтобы использовать один или большее количество **регистров** в интерфейсе устройства как **буферы** между устройством и процессором. Мы знаем эти буферные регистры как **порты**. Данные, посланные внешнему устройству, записываются сначала в регистр в интерфейсе устройства, откуда они уже передаются внешнему устройству. Аналогично процессор обращается к данным, пересылаемым от внешнего устройства, читая регистр в интерфейсе устройства.

Мы будем делить интерфейсы на **пользовательские** и **стандартные**. Интерфейсы делятся также по способу передачи данных на **параллельные** и **последовательные**.

Чтобы облегчить и ускорить процесс обмена данными между микропроцессорной системой и внешними устройствами, часто используются специальные режимы обмена данными. Это режимы **прерывания** и **прямого доступа к памяти**.

7.1. Схемотехника и программирование пользовательских интерфейсов

Устройство, которое дополнительно подключено к микропроцессорной системе для поддержки ввода/вывода данных, а также временного и архивного хранения данных, называется **периферийным** (или **внешним**) **устройством**. Микропроцессорные системы без внешних устройств (клавиатура, дисплей, накопители информации на магнитных носителях, принтер) оказываются бесполезными. Поэтому разработчики вычислительных систем уделяют большое внимание организации взаи-

модействия системы с внешними устройствами. Основное требование при этом – надежная, без потерь передача информации.

Аппаратные средства и программное обеспечение, необходимые для связи периферийных устройств с микропроцессорной системой, называются ***интерфейсом****. Аппаратные средства интерфейса включают сигналы обмена данными, а также оборудование, обеспечивающее этот обмен. Порядок следования сигналов во времени называется* ***протоколом обмена****.*

Интерфейсы принято делить на **магистральные** и **радиальные**. Магистральным называется такой интерфейс, к которому подключаются одновременно несколько устройств. Такой интерфейс обычно используется для связи блоков самой вычислительной системы. Для связи микропроцессорной системы и внешнего устройства используется обычно радиальный интерфейс (для каждого внешнего устройства используется свой канал связи). По протоколу обмена интерфейсы делят на **параллельные** и **последовательные**. Стандартные интерфейсы общего применения обычно ориентированы на байтовый (восьмибитовый) обмен информацией. В **параллельном** интерфейсе байт данных передается по восьми линиям **одновременно**, а в **последовательном –** по одной линии бит за битом **последовательно**. Естественно, под линией здесь понимаются два провода (информационный и общий).

Интерфейс ввода/вывода данных должен иметь специальные регистры для промежуточного хранения данных. Данные сначала записываются процессором в этот регистр, а затем периферийное устройство получает их. Микропроцессорная система может иметь много таких регистров, и процессор должен иметь возможность их **адресовать**. Имеются два способа отображения этих регистров в адресном пространстве микропроцессорной системы. Во-первых, **регистры** могут составлять **часть пространства адресов памяти** системы. В этом случае регистры доступны как обычные ячейки оперативного запоминающего устройства. Все микропроцессорные команды, которые работают с памятью, могут работать и с регистрами ввода/вывода. Чтение и запись в регистры стробируются сигналами управления микропроцессора, предназначенными для управления памятью. Недостаток этого пути – уменьшение объема памяти, доступного для хранения данных. Второй способ адресации регистров состоит в использовании **отдельного адресного пространства** для отображения регистров ввода/вывода. В этом случае регистры называются **портами** ввода/вывода. Для записи и чтения информации из портов ввода/вывода используются **специальные команды** микропроцессора. Для микропроцессора 8086, например, имеются команды **IN** и

OUT. Кроме того, микропроцессор использует для управления портами ввода/вывода иные сигналы, чем для управления ячейками памяти.

Пример схемы пользовательского интерфейса показан на рис. 7.1. На рисунке показано подключение светодиода (VD1) к шинам микропроцессорной системы. Светодиод в данной схеме рассматривается как периферийное устройство. В качестве порта вывода используется D-триггер. Логическая схема определяет адрес этого порта. Информация, которая записывается в порт, стробируется сигналом IOWC. В данном примере использован второй способ адресации портов ввода/вывода.

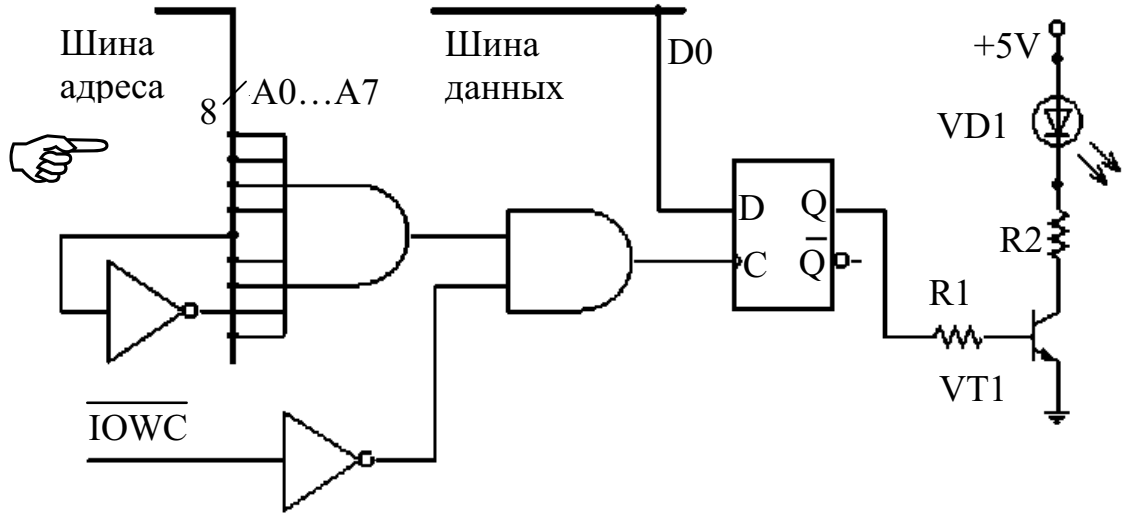


Рис. 7.1. Пример схемы пользовательского интерфейса

Для внешнего устройства данные часто необходимы в аналоговом виде, в то время как микропроцессорная система работает с двоичными числами. В этом случае для преобразования сигналов необходимы **цифроаналоговые преобразователи** (ЦАП). Данные от различных датчиков (температуры, давления и др.) часто передаются как непрерывные аналоговые напряжения или токи. Чтобы микропроцессорная система могла их обработать, аналоговые сигналы должны быть преобразованы в цифровые двоичные сигналы, которые могут читаться микропроцессором. Подобные функции выполняют **аналого-цифровые преобразователи** (АЦП). Интерфейсные схемы при использовании АЦП и ЦАП строятся так же, как и в обычном случае. В составе интерфейса должен быть регистр, куда помещаются данные. Этот регистр может быть частью ЦАП или АЦП, но чаще это дополнительный регистр. Кроме него, в интерфейсе должен быть предусмотрен еще один регистр, который используется для управления АЦП или ЦАП. Процессор просто передает параллельные цифровые данные в регистры в интерфейсе устройства. Преобразование и передача данных между этими регистрами и внеш-

ними устройствами осуществляются автоматически электронными схемами интерфейса.

Интерфейсы дисководов, графических дисплеев и других сложных устройств ввода/вывода работают подобным же образом. Несмотря на существенные различия между характеристиками различных типов периферийных устройств, большинство из них рассматривается микропроцессором как **набор адресуемых регистров**. Эти интерфейсы включают в себя специальные регистры **управления** и **состояния**, в которые микропроцессор передает команды интерфейсу или считывает с них текущее состояние внешнего устройства.

Ниже приведен фрагмент программы передачи одного байта данных на принтер. Интерфейс принтера представлен для микропроцессора двумя портами: регистром данных (Dat Reg) и регистром состояния (Stat Reg). Программа читает и проверяет младший бит регистра состояния устройства. Если этот бит – 1, устройство не готово принять данные от процессора, и программа ожидает в цикле, когда бит поменяет значение. Если бит – 0, устройство готово принимать данные, и микропроцессор посылает информацию в регистр данных интерфейса устройства.

```
Check:  IN    AL,Stat Reg ; чтение регистра состояния
AND     AL, 00000001B     ; проверка младшего бита
JNE     Check             ; регистра состояния
MOV     AL,Data           ; пересылка в регистр данных
OUT     Dat reg,AL        ; интерфейса.
```

Для того, чтобы облегчить создание электронных схем интерфейсов, разработано большое число различных специализированных больших интегральных схем. Одна из них – БИС 8255 (российский аналог 580ВВ55) – программируемое периферийное параллельное устройство ввода/вывода, предназначенное для использования в микропроцессорных системах. Микросхема используется как универсальный компонент параллельных интерфейсов. Конфигурация и режим работы БИС 8255 могут быть запрограммированы разработчиком интерфейса таким образом, что для связи периферийного устройства и микропроцессора обычно не требуется никаких дополнительных логических схем. Блок-схема программируемого периферийного адаптера параллельного интерфейса показана на рис. 7.2.

Буфер шины данных – двунаправленный 8-разрядный буфер с трехстабильными выходами, который используется для связи адаптера 8255 с шиной данных микропроцессорной системы. Данные передаются или принимаются БИС с шины системы при выполнении команд от микропроцессора. Команды и информация о состоянии адаптера также передаются через буфер шины данных.

В функции **устройства управления** входит управление всеми внутренними и внешними передачами данных, а также слов управления и состояния. Оно принимает информацию с шин данных и адреса микропроцессорной системы и, в свою очередь, вырабатывает управляющие сигналы для портов.

Каждый порт может быть программно настроен на выполнение определенных функций, и микропроцессор должен передать в адаптер специальные управляющие слова, которые содержат информацию о режимах, в которых должны работать порты. Для этого в БИС 8255 есть специальный регистр управляющих слов. Этот регистр может быть записан микропроцессором.

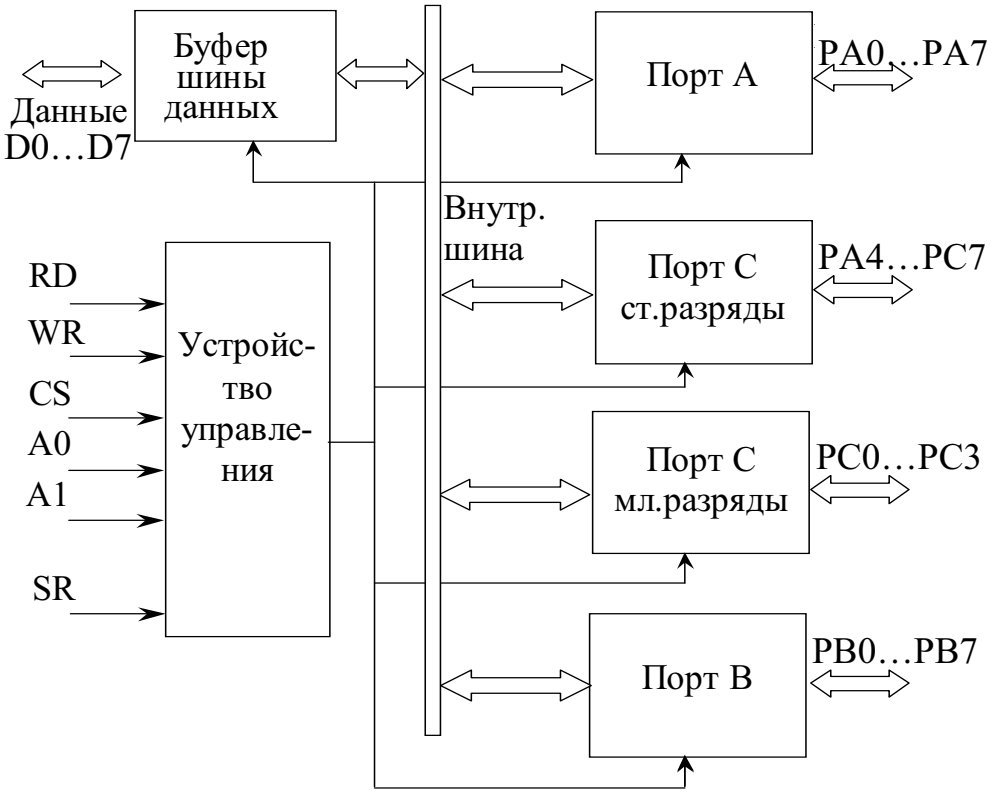


Рис. 7.2. Блок-схема БИС 8255

Микросхема 8255 содержит 3 восьмиразрядных порта (A, B и C). Все они могут быть сконфигурированы программно, и каждый имеет свои особенности, что расширяет функциональные возможности микросхемы.

Порт С может быть разделен на 2 четырехразрядных полупорта. Каждый полупорт можно использовать самостоятельно или совместно с портами A и B для передачи служебной информации.

Адаптер параллельного интерфейса может работать в трех режимах, которые устанавливаются программно:

1) режим 0 – простой ввод/вывод;
2) режим 1 – стробируемый ввод/вывод;
3) режим 2 – двунаправленный ввод/вывод.

Когда на микросхему подается сигнал сброса, все порты устанавливаются в режим ввода данных и все 24 выхода портов переходят в состояние 1. После сигнала сброса все порты остаются в режиме простого ввода. Если микросхема должна использоваться именно в этих режимах, то дополнительного программирования не требуется. Если требуются другие режимы, то необходимо записать в регистр управляющих слов соответствующее слово инициализации. Перепрограммирование режимов работы микросхемы может быть сделано в любой момент работы программы.

Режимы для **порта А** и **порта В** могут быть определены отдельно, в то время как для **порта С** каждая из двух частей может быть запрограммирована отдельно для режима 0 или использоваться как вспомогательная часть портов А и В для режимов 1 и 2. Все выходные регистры, включая триггеры состояния, будут сброшены всякий раз, когда режим изменяется. Режимы могут быть скомбинированы так, что адаптер может быть приспособлен почти к любой структуре интерфейса ввода/вывода.

Интерфейсы последовательных каналов ввода/вывода в микропроцессорных системах реализуются с использованием БИС 8251 (российский аналог – 580ВВ51). Микросхема представляет собой универсальный синхронно-асинхронный приемопередатчик последовательной связи, выполняющий функции приема и преобразования параллельных форматов слов в последовательные для их передачи по каналам связи и последовательных форматов, принимаемых из каналов связи в параллельный формат для ввода в процессор. Микросхема может быть запрограммирована для работы в пяти режимах: асинхронная передача, асинхронный прием, синхронная передача, синхронный прием с внутренней синхронизацией, синхронный прием с внешней синхронизацией.

На рис. 7.3 приведена структурная схема БИС 8251. **Передатчик** принимает данные с шины данных, преобразует их в последовательный код, добавляет служебные разряды и выдает их на выход передатчика TxD под управлением сигналов синхронизации со входа TxC. **Приемник** принимает данные со входа RxD, преобразует их в параллельный код, исключает служебные символы и посылает на шину данных. Прием синхронизируется сигналами на входе RxC. В режиме асинхронной передачи/приема скорость передачи или приема кратна частоте сигналов на входе TxC/RxC. Коэффициент кратности устанавливается программно и равен: 1; 16 или 64.

Сигналы TxRDY и RxRDY используются для связи с процессором. Сигнал TxRDY указывает, что передатчик готов принять новое слово данных от процессора. В единичное состояние сигнал устанавливается после программного запуска передачи и после завершения передачи очередного слова данных, а сбрасывается в нулевое состояние после записи байта данных в регистр данных передатчика. Сигнал RxRDY показывает, что данные в приемнике готовы для ввода в процессор. Он устанавливается в единичное состояние после приема слова данных и сбрасывается после считывания данных процессором. Оба сигнала могут быть использованы как сигналы требования прерываний, в случае программной организации ввода/вывода сигналы не используются.

Схема управления содержит регистры управляющих слов, регистр состояния, схему управления модемом. Синхронизируется БИС сигналами, подаваемыми на вход CLK (обычно используют вторую фазу сигналов синхронизации микропроцессора). Сигнал SR, длительностью не менее 6 периодов синхронизации, используется для установки БИС в исходное состояние. Выход сигнала запроса приемнику терминала RTS программно устанавливается в 0 и используется как требование передачи данных от внешнего устройства. Сигнал на входе готовности приемника терминала CTS указывает (при CTS = 0), что передача данных внешним устройством разрешена. Выход сигнала запроса готовности передатчику терминала DTR можно использовать для синхронизации работы передатчика и управления скоростью выборки. Он устанавливается в 0 программно. Вход сигнала готовности передатчика терминала DSR указывает на готовность внешнего устройства к передаче, фиксируется в **слове состояния** и может быть проанализирован программой.

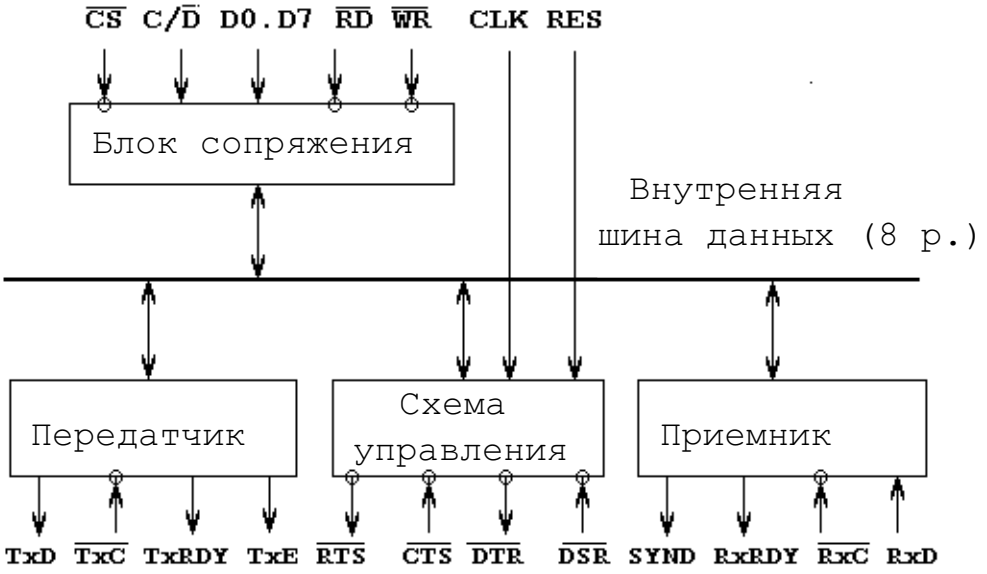


Рис. 7.3. Структурная схема контроллера

С процессором БИС сопрягается посредством шины данных D0...D7 и управляющих сигналов CS, C/D, RD и WR. На вход выбора кристалла CS подается сигнал логического 0 с селектора адреса, определяющего адрес, по которому обращаются к БИС при программировании. Вход C/D (управление/данные) обычно управляется разрядом A0 шины адреса. Входы, управляющие чтением и записью информации, RD и WR соединяются с линиями IORC и IOWR процессора (линии, управляющие чтением/записью во внешние устройства).

Микросхема содержит 7 программно-доступных 8-разрядных регистров: данных (РД), состояния (РС), режима (РР), команд (РК), первого синхросимвола (РСС1), второго синхросимвола (РСС2) и регистр передатчика. В асинхронном режиме работы регистры РСС1 и РСС2 не используются.

Перед использованием контроллер должен быть запрограммирован. Программирование осуществляется записью нужной информации в регистры БИС. Порядок записи управляющих слов и данных следующий: после сигнала сброса по линии SR производится запись управляющего слова режима в РР; идущие далее управляющие слова в зависимости от содержимого РР интерпретируется либо как первый синхросимвол и записывается в РСС1 (для синхронного режима), либо как команда (в асинхронном режиме): в синхронном режиме третье управляющее слово воспринимается как второй синхросимвол и записывается в РСС2, а четвертое слово – как команда.

В дальнейшем на микросхему могут поступать данные и команды в произвольном порядке (команды от данных микросхема отличает по сопровождающему их сигналу C/D, логическая 1 – команды, логический 0 – данные). Если необходимо сменить режим, нужно подать команду программного сброса или сигнал SR, после чего повторить процедуру начального программирования. Программный контроль за состоянием приемопередатчика возможен посредством **слова состояния**.

7.2. Специальные режимы ввода/вывода

7.2.1. Прерывания

Все микропроцессорные системы должны связываться с внешним миром. Типичная вычислительная система обычно имеет клавиатуру, дисковод и коммуникационный порт. Все они требуют внимания процессора в разное время. Имеются два различных способа обработки запросов ввода/вывода от периферийных устройств: **опрос** и **прерывания**.

Опрос требует, чтобы процессор проверял каждое периферийное устройство в системе периодически, чтобы определять, требует ли оно

обслуживания. Чаще всего процессору нужно многократно обращаться к внешнему устройству, прежде чем оно потребует какого-либо внимания. В большинстве случаев использование опроса снижает производительность системы. Время, использованное на проверку состояния внешнего устройства, – это время, потраченное без пользы для решаемой задачи.

Прерывания устраняют потребность в опросе путем формирования специального сигнала центральному процессору в тот момент, когда периферийное устройство требует обслуживания. Микропроцессор тогда прекращает выполнять основную задачу, сохраняет ее состояние и передает управление **программе обработки прерывания**. После окончания программы обработки прерывания восстанавливается первоначальное состояние процессора, и выполнение основной задачи продолжается с точки, в которой возникло прерывание.

В вычислительной системе часто происходят события, которые требуют прерывания нормального хода выполнения программы и выполнения некоторых специальных действий. Такие исключительные ситуации, или просто **исключения**, могут быть инициированы сигналом от внешнего устройства, или условиями, обнаруженными в процессоре.

Например, персональные компьютеры часто используют таймер, чтобы прервать процессор раз в секунду, чтобы заставить его модифицировать изображение часов, отображаемых на экране. Компьютеры, используемые в управлении производственным процессом, обычно прерываются датчиками, которые обнаруживают различные состояния оборудования, требующие немедленного внимания. Пример внутреннего состояния, требующего прерывания, попытка деления на 0, которая не может дать никакого результата. Этот тип исключительного состояния должен приостановить работу программы, чтобы прервать операцию и послать предупреждающее сообщение пользователю.

Основное **преимущество внешнего прерывания** состоит в том, что микропроцессор может работать **параллельно** с несколькими внешними процессами, каждый из которых обслуживается только тогда, когда процесс требует внимания. Когда обнаружена исключительная ситуация, процессор обычно отвечает следующей последовательностью операций.

1. Заканчивает текущую команду программы, чтобы достигнуть удобной точки остановки.
2. Сохраняет текущее состояние счетчика команд в стеке системы или в определенном регистре, сохраняя указатель на следующую команду, которая была бы выполнена, если бы программа не была прервана.

3. Определяет устройство, которое вызвало прерывание. Многие микропроцессоры выполняют специальную операцию подтверждения прерывания, чтобы позволить внешнему устройству, вызвавшему прерывание, идентифицировать себя с помощью уникального числа, называемого **вектором прерывания**.
4. Загружает счетчик команд начальным адресом программы, которая выполняет необходимые действия по обслуживанию внешнего устройства. Эта программа называется **программой обработки прерывания**. Если используется вектор прерывания, то он указывает на ячейки памяти, содержащие начальный адрес программы обработки прерывания.
5. Выбирает и выполняет команды программы обработки прерывания.
6. После завершения программы обработки прерывания выполняет **команду возврата из прерывания**, чтобы восстановить содержимое счетчика команд из стека, позволяя микропроцессору возвратиться к первоначальной программе, во время выполнения которой возникло прерывание.

Так как прерывания могут происходить в любое время, программа обработки прерывания должна вначале сохранить значения всех регистров, которые будут использоваться в пределах программы обработки прерывания, и затем восстановить их перед возвращением к основной программе. Это позволяет основной программе быть продолженной, как будто прерывания не происходило.

Таким образом, прерыванием называется переход к подпрограмме обработки какой-либо исключительной ситуации, вызываемый некоторым внешним по отношению к микропроцессору сигналом.

Микропроцессор 8086 имеет два источника внешних прерываний; вход **немаскируемого прерывания (NMI)** и вход **маскируемого прерывания (INTR)**. Для большинства микропроцессорных систем единственный вход маскируемого прерывания недостаточен. Большинство процессоров использует **специальный контроллер прерываний**, чтобы увеличить число доступных маскируемых прерываний.

Микропроцессор 8086 использует специальную микросхему контроллера прерываний 8259 (580ВН59). Блок-схема микросхемы контроллера приведена на рис. 7.4. Этот контроллер является стандартным для многих микропроцессорных систем и персональных компьютеров. Контроллер 8259 имеет восемь входов прерываний. Используя дополнительные контроллеры, число входов прерывания возможно довести до 64. Дополнительные контроллеры называются вспомогательными; первый контроллер является основным. Основной контроллер располагает по

приоритетам запросы на прерывание от вспомогательных контроллеров и своих входов IRn и передает запросы по одному на вход маскируемого прерывания процессора.

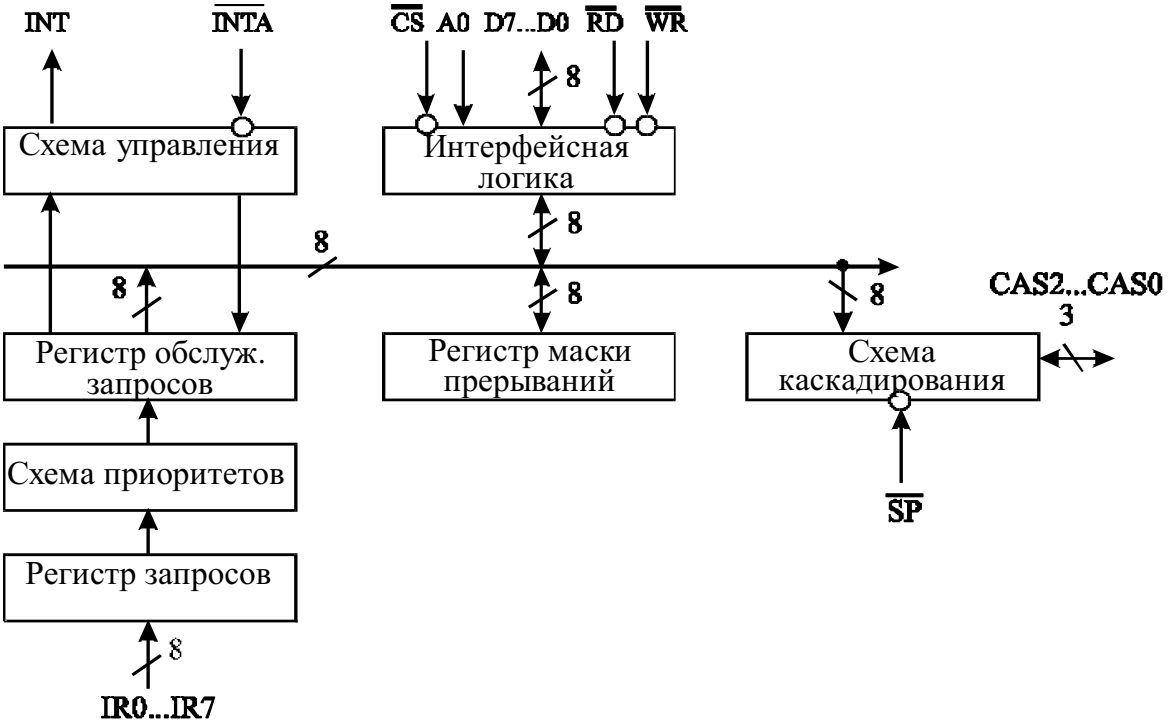


Рис. 7.4. Блок-схема контроллера прерываний

Микропроцессор 8086 может использовать до 256 различных прерываний. Каждое прерывание определяется его **номером** в пределах от 0 до 255. Каждому прерыванию соответствует **вектор прерывания**, который является номером прерывания, умноженным на 4. Вектор прерывания – указатель, который указывает на связанную с прерыванием **подпрограмму обработки прерывания**. Таблица векторов прерываний расположена в начальной области памяти микропроцессора. Таблица векторов прерываний имеет размер 1 кб (4 байта, умноженные на 256).

Прежде чем начать работать, каждый контроллер прерываний должен быть инициализирован последовательностью из управляющих слов инициализации. После того, как произошел сброс системы, состояния всех регистров контроллера не определены. Слова инициализации используются, чтобы установить необходимые режимы работы контроллера.

7.2.2. Прямой доступ к памяти

Во многих случаях большие блоки данных должны быть переданы между устройством ввода/вывода и памятью. Дисковод, например, обычно читает и записывает данные блоками, которые могут быть объемом в тысячи байт. Если использовать для управления передачей дан-

ных процессор, то значительная часть времени его работы будет потрачена на эту операцию. Даже если при передаче данных используется режим прерываний, все равно производительность системы в целом будет сильно падать. Это происходит потому, что микропроцессор не имеет команд прямой передачи данных от внешнего устройства в память и наоборот, и ему приходится использовать регистры общего назначения для временного хранения данных. В результате передача осуществляется в два этапа: сначала из внешнего устройства – в процессор, а затем из процессора – в память. Выгоднее было бы использование передачи напрямую, минуя процессор.

Прямой доступ к памяти (**ПДП**) позволяет передавать данные между памятью и периферийными устройствами **без вмешательства процессора.** Системы, которые используют прямой доступ к памяти, имеют специальное устройство, называемое **контроллер прямого доступа к памяти**. Контроллер берет в этом режиме под свой контроль шины системы и выполняет управление передачей данных между памятью и периферийным устройством. Когда контроллер ПДП принимает запрос от периферийного устройства, он посылает в процессор сигнал требования прямого доступа. В этом случае процессор вырабатывает сигнал подтверждения ПДП и переходит в пассивный режим, переводя внешние шины адреса и данных в высокоомное состояние и не вырабатывая сигналы управления. В случаях, если процессору не требуется обращение к внешним шинам системы, он может продолжать выполнение команд из очереди команд. Контроллер ПДП выполняет передачу данных самостоятельно. Если передачи данных в режиме прямого доступа происходят нечасто, то это не приводит к снижению производительности системы в целом, так как передача данных при прямом доступе прозрачна для процессора.

Передача данных в режиме ПДП начинается с запроса. Устройство, требующее прямой доступ, может иметь данные для передачи или может требовать данные от другого устройства. Кроме того, передачи данных в режиме ПДП могут быть инициализированы системным программным обеспечением без внешнего запроса.

Когда режим ПДП разрешается процессором, контроллер прямого доступа к памяти обеспечивает все необходимые сигналы шины для передачи данных. Источник и приемник данных для передачи программируемы и могут быть или в пространстве адресов ввода/вывода, или в пространстве адресов памяти.

Микропроцессорные системы на основе микропроцессора 8086 используют в качестве контроллера прямого доступа к памяти микросхему 8257. Интегральная схема 8257 – программируемый 4-канальный кон-

троллер ПДП. Каждый канал контроллера оборудован 16-разрядным регистром адреса ПДП и 16-разрядным регистром-счетчиком числа переданных байтов. Младшие 14 битов регистра-счетчика определяют количество циклов ПДП, таким образом, их максимальное количество равно 16384. Старшие 2 бита регистра-счетчика определяют тип операции ПДП для данного канала.

Логика арбитража в контроллере решает, какой канал имеет приоритет, если два канала одновременно запрашивают передачу данных. Каждый канал может получить или низкий, или высокий приоритет.

Перед использованием контроллер должен быть запрограммирован. При программировании контроллера вначале необходимо установить параметры для каждого канала. Программируются следующие параметры: указатель адреса ПДП, число передаваемых байт данных, тип операции ПДП.

7.3. Стандартные интерфейсы

Во многих случаях необходимо подключить к микропроцессорной системе какое-либо стандартное периферийное устройство. Такие устройства обычно имеют встроенные средства для подключения к вычислительной системе. Эти средства принято называть интерфейсом, но по принятой нами классификации их точнее будет называть стандартными интерфейсами. Стандартные интерфейсы **RS232C** и **Centronics** – одни из самых распространенных. RS232C – это последовательный, а Centronics – параллельный интерфейс.

Интерфейс RS232C определен стандартом Ассоциации электронной промышленности и подразумевает наличие оборудования двух типов: терминального – DTE (оконечное оборудование данных – ООД в русском варианте) и связного – DCE (аппаратура передачи данных – АПД). Терминальная аппаратура, например компьютер, может посылать и (или) принимать данные по последовательному интерфейсу. Оно оканчивает последовательную линию. Связная аппаратура – это устройства, которые могут упростить последовательную передачу совместно с терминальной аппаратурой. Наглядным примером связного оборудования является модем – устройство которое кодирует информацию при передаче по телефонной линии и декодирует при приеме. Схема обмена информации между двумя ЭВМ по телефонной линии приведена на рис. 7.5. В качестве ООД могут выступать также дисплей, принтер и т. д.

В большинстве систем, содержащих интерфейс RS232C, данные передаются асинхронно, то есть в виде последовательности пакетов данных. Каждый пакет содержит 1 байт данных, причем информация в па-

кете достаточна для его декодирования без отдельного сигнала синхронизации.

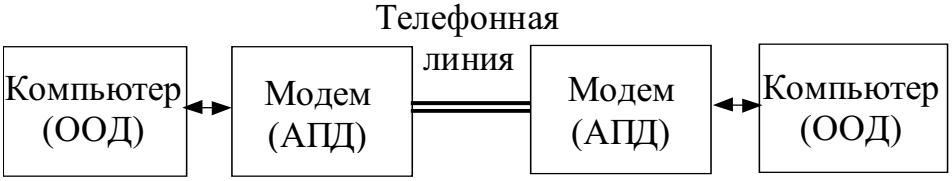


Рис. 7.5. Схема обмена по последовательному каналу

Чтобы передать байт данных по интерфейсу RS232C, необходимо ввести дополнительные биты, обозначающие начало и конец пакета. Кроме того, желательно добавить лишний бит для простого контроля ошибок по паритету (четности). Наиболее широко распространен формат, включающий в себя один стартовый бит, один бит паритета и два стоповых бита. Начало пакета данных всегда отмечает низкий уровень стартового бита. После него следует восемь бит данных. Бит паритета содержит 1 или 0 так, чтобы общее число единиц в восьмибитной группе было нечетным (нечетный паритет) или четным (четный паритет). Последними передаются два стоповых бита, представленных высоким уровнем напряжения. Таким образом, полное асинхронно передаваемое слово данных состоит из 12 бит (фактические данные содержат только 8 бит).

К сожалению, используемые в интерфейсе RS232C уровни сигналов отличаются от уровней сигналов, действующих в компьютере. Логический 0 (SPACE) представляется положительным напряжением в диапазоне от +3 до +25 В, а логическая 1 (MARK) – отрицательным напряжением в диапазоне от –3 до –25 В. На рис. 7.6 показан сигнал пакета данных для кода буквы А в том виде, в котором он существует на линиях интерфейса RS232C.

Сдвиг уровня, то есть преобразование ТТЛ-уровней в уровни интерфейса RS232C и наоборот, производится специальными схемами.

Сигналы интерфейса RS232C подразделяются на 3 класса.

Последовательные данные (например, TXD, RXD). Интерфейс RS232C обеспечивает два независимых последовательных канала данных: первичный (главный) и вторичный (вспомогательный). Оба канала могут работать в дуплексном режиме, т. е. одновременно осуществлять передачу и прием информации.

Управляющие сигналы квитирования (например, RTS, CTS). Сигналы квитирования – это средство, с помощью которого обмен сигналами позволяет АПД начать диалог с ООД до фактических передачи или приема данных по последовательной линии связи.

Сигналы синхронизации (например, TC, RC). В синхронном режиме (в отличие от более распространенного асинхронного) между уст-

ройствами необходимо передавать сигналы синхронизации, которые упрощают синхронизм принимаемого сигнала в цепях его декодирования.

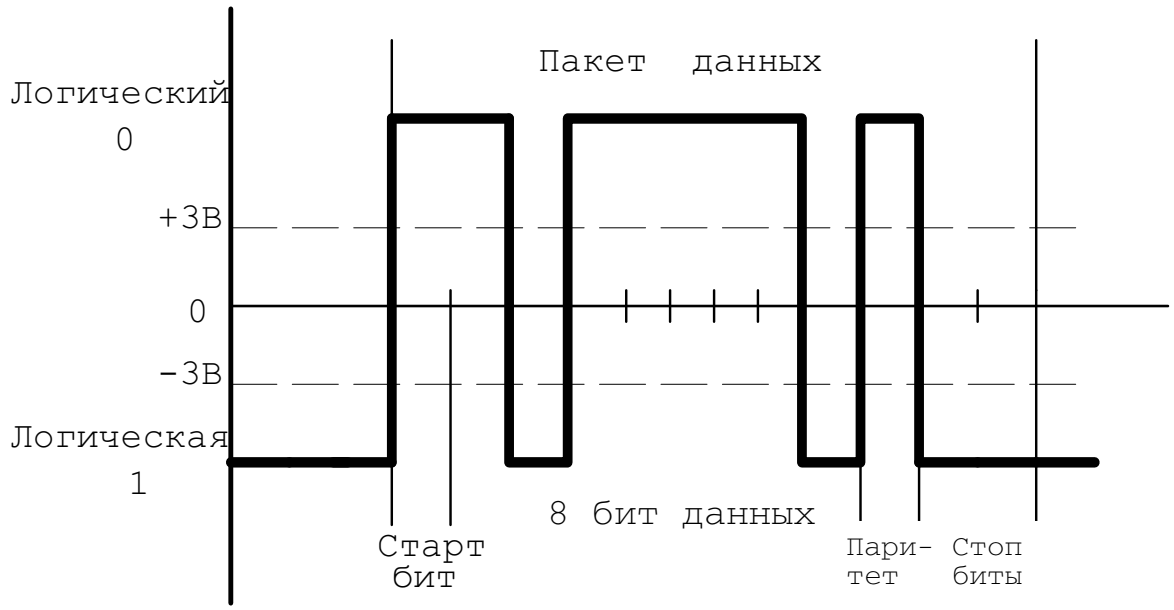


Рис. 7.6. Пакет посылки

Интерфейс **Centronics** обеспечивает радиальное подключение устройств с **параллельной** передачей информации к компьютеру. Передача данных осуществляется между одним источником (И) и одним приемником (П). Основным назначением интерфейса Centronics является подключение к компьютеру принтеров различных типов. Поэтому распределение контактов разъема, назначение сигналов, программные средства управления интерфейсом ориентированы именно на это использование. В то же время с помощью данного интерфейса можно подключать к компьютеру и другие внешние устройства.

Сигналы Centronics имеют следующее назначение (тип выходных каскадов для всех сигналов – ТТЛ):

DO...D7 – 8-разрядная шина данных для передачи из компьютера в принтер, логика сигналов положительная;

STROBE – сигнал стробирования данных; данные действительны как по переднему, так и по заднему фронту этого сигнала, сигнал говорит приемнику, что можно принимать данные;

ACK – сигнал подтверждения принятия данных и готовности приемника принять следующие данные;

BUSY – сигнал занятости приемника обработкой полученных данных и неготовности принять следующие данные, сигнал активен также при ошибке.

Остальные сигналы не являются обязательными.

Временная диаграмма цикла передачи данных представлена на рис. 7.7.

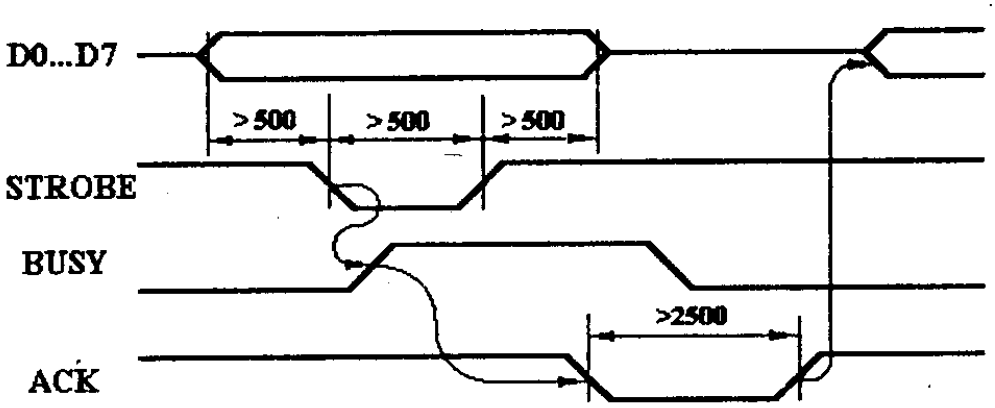


Рис. 7.7. Временные диаграммы цикла передачи данных в Centronics (все временные интервалы в наносекундах)

Перед началом цикла передачи данных источник должен убедиться, что сняты сигналы BUSY и ACK. После этого выставляются данные, формируется и снимается строб, снимаются данные. При получении строба приемник формирует сигнал BUSY, а после окончания обработки данных выставляет сигнал ACK, снимает BUSY и снимает ACK. Затем может начинаться новый цикл.

Все сигналы интерфейса Centronics передаются в уровнях ТТЛ и рассчитаны на подключение одного стандартного входа ТТЛ. Максимальная длина соединительного кабеля по стандарту – 1,8 м.

Вопросы и задания для повторения

Примеры

1. Постройте параллельный интерфейс между двумя микропроцессорными системами, используя два параллельных периферийных адаптера 8255 в режиме 1 или 2.

Решение

Два адаптера 8255 в режиме 1 или 2 не могут быть соединены между собой непосредственно. Необходима дополнительная логическая схема.

На первый взгляд кажется, что можно решить задачу, используя один дополнительный инвертор (для режима однонаправленного обмена) или два дополнительных инвертора (для режима двунаправленного обмена), как показано на рис. 7.8.

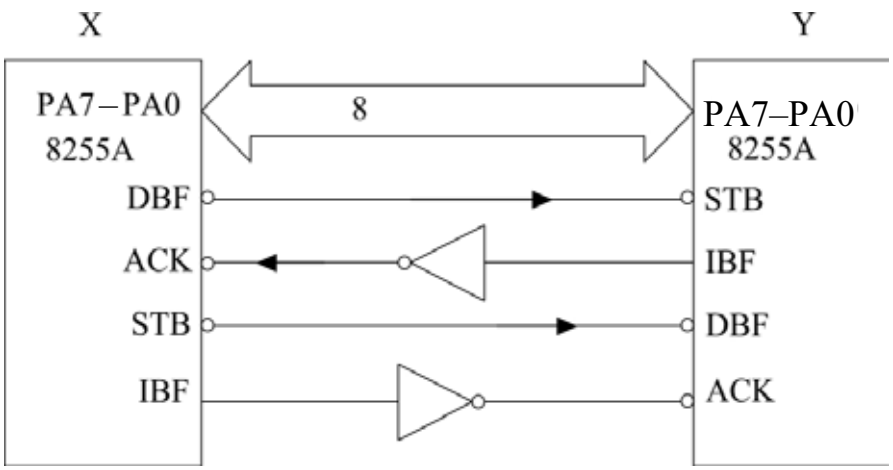


Рис. 7.8. Некорректное соединение адаптеров

Временная спецификация для микросхемы 8255 показывает, что данная схема не будет работать правильно. Дело в том, что сигналы «квитирования», вырабатываемые одной микросхемой, не дадут импульса строба данных STB нужной длительности для стробирования записи данных в другую микросхему.

Временная диаграмма на рис. 7.9 показывает процессы, протекающие в схеме на рис. 7.8. Здесь левая микросхема, обозначенная X, передает данные правой микросхеме (Y). Последовательность переходов управляющих сигналов, которые сопровождают передачу, показана на рис. 7.9 стрелками, начинающимися на спаде сигнала DBF_X и оканчивающимися на фронте сигнала STB_Y. На временной диаграмме t_{DELAY1} представляет задержку от фронта DBF_X до фронта STB_Y; t_{DELAY2} – задержка от фронта IBF_Y до среза ACK_X. Задержки t_{DELAY1} и t_{DELAY2} обусловлены инверторами и внутренними задержками самих адаптеров. Передача данных становится неуверенной, если длительность импульса STB_Y меньше минимально допустимой величины (500 нс). Длительность импульса можно определить из рис. 6.2, следуя по стрелкам от среза к фронту STB_Y; это t_{ST}. Получим

$$t_{ST} = t_{SIB} + t_{DELAY2} + t_{AOB} + t_{DELAY1} >= 500 \text{ нс}.$$

Для t_{SIB} и t_{AOB} минимальные значения в технической документации не определены (то есть, в принципе, они могут быть очень малыми), поэтому ими можно пренебречь, тогда

$$t_{DELAY1} + t_{DELAY2} >= 500 \text{ ns}.$$

Из наших рассуждений следует вывод, что последовательно с сигналами управления должна быть внесена дополнительная задержка величиной, как минимум, 500 нс. Такая задержка не может быть создана просто инверторами. Рис. 7.10 аналогичен рис. 7.8, но в него добавлены

элементы задержки Delay1 и Delay2. Требуемая задержка может быть создана или Delay1 или Delay2 или обеими вместе. Предпочтительнее использовать только Delay2, так как эта линия должна содержать инвертор и функции инверсии и задержки могут быть совмещены. Один элемент задержки необходим, если порты работают в режиме 1, и два элемента необходимы, если порты работают в режиме 2.

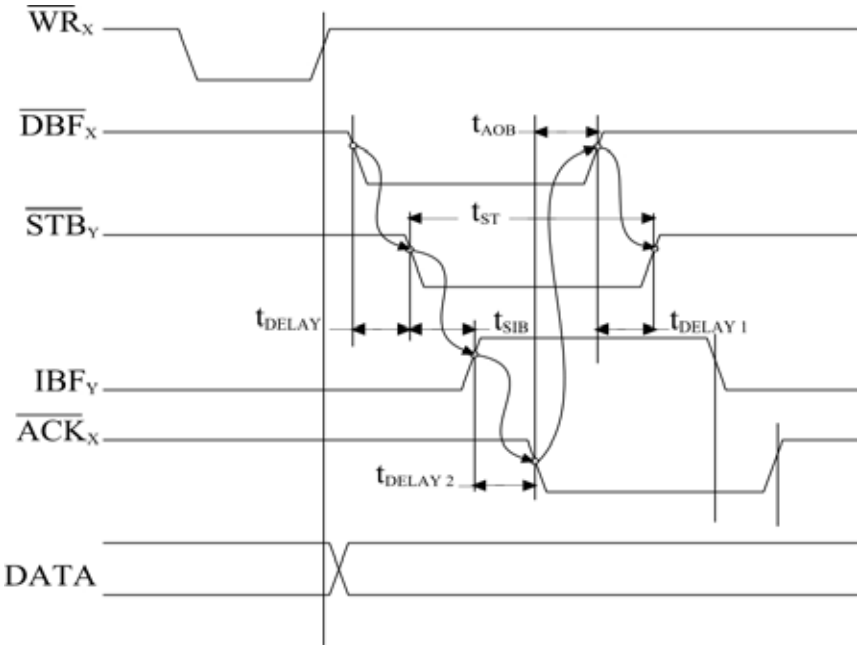


Рис. 7.9. Временная диаграмма

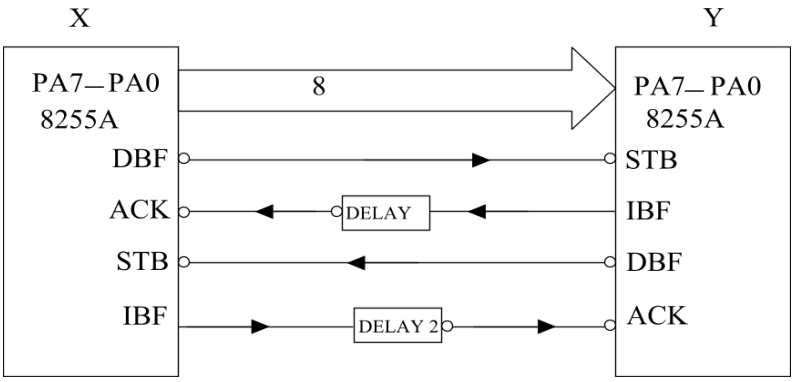


Рис. 7.10. Соединение двух адаптеров 8255

Задачи

Пользовательские интерфейсы

7.1. Что такое протокол обмена? Как делятся интерфейсы по протоколам обмена?

7.2. Что такое адаптер параллельного интерфейса? Как его можно применить при разработке пользовательского параллельного интерфейса?

7.3. Что такое адаптер последовательного интерфейса? Как его использовать для реализации последовательного интерфейса?
7.4. Что такое программируемый интервальный таймер? Как его можно использовать при проектировании интерфейсов?
7.5. Подключите к микропроцессорной системе 9 внешних устройств: 5 входных и 4 выходных. Адреса портов входных устройств должны лежать в диапазоне 01H...05H, выходных – в диапазоне 06H…09H.
7.6. Нарисуйте схему интерфейса с использованием параллельного адаптера 8255 и напишите программу, которая должна периодически вводить данные из порта A микросхемы 8255 и суммировать их. Когда сумма станет равной или больше некоторого числа *k*, микропроцессор должен вывести это число в порт C микросхемы 8255.
7.7. Нарисуйте принципиальную схему системы, состоящей из микропроцессора 8086, программируемого параллельного адаптера 8255 и интервального таймера 8253. Запрограммируйте эту систему в режим 0 для микросхемы 8255 (порты A и B работают на ввод, порт C – на вывод информации). Каналы 1 и 2 таймера 8253 должны работать в режиме делителя частоты.
7.8. Разработайте принципиальную схему интерфейса аналого-цифрового преобразователя с микропроцессорной системой.

Специальные режимы ввода/вывода

7.9. Что такое режим прерываний? Как его можно реализовать в микропроцессорной системе на основе 8086?
7.10. Что такое режим прямого доступа к памяти? Как его можно реализовать в микропроцессорной системе на основе 8086?
7.11. Напишите программу обработки прерывания, которая должна вводить данные из порта 1AH, определять знак числа и, если число положительное, выводить его в порт 0F0H.
7.12. Нарисуйте схему и напишите программу для передачи 64 байт информации в режиме прямого доступа к памяти из порта 8FH в память с начальным адресом 0100:00F0.

Стандартные интерфейсы

7.13. Разработайте схему интерфейса *Centronics* на основе программируемого параллельного адаптера 8255.
7.14. Разработайте схему интерфейса RS232C на основе программируемого последовательного адаптера 8251.

Глава 8
ОДНОКРИСТАЛЬНЫЕ МИКРОКОНТРОЛЛЕРЫ

Микроконтроллеры представляют собой еще одно важное средство для разработки вычислительных и управляющих систем. Они очень широко используются в различных приборах и оборудовании. В отличие от микропроцессоров микроконтроллеры содержат на одном кристалле законченную вычислительную систему. Это позволяет строить очень компактные, потребляющие мало энергии системы и встраивать их в различные устройства как управляющие системы. Для расширения диапазона применений современные микроконтроллеры часто содержат на кристалле многоканальные АЦП, многофункциональные логические схемы и интерфейсы. Но, с другой стороны, микроконтроллеры по сравнению с микропроцессорами имеют ряд недостатков: они обладают меньшей разрядностью процессора, меньшим объемом адресуемой памяти и, соответственно, меньшей производительностью.

Микроконтроллеры в настоящее время являются одними из самых массовых цифровых БИС, производимых промышленностью. Однако в широкой номенклатуре выпускаемых приборов можно отчетливо проследить несколько семейств микроконтроллеров с определенной базовой архитектурой. Среди 8-разрядных приборов, выпускаемых в настоящее время, наибольшую популярность у производителей имеет архитектура MCS51, разработанная фирмой *Intel*. Архитектура микроконтроллеров имеет значительные особенности по сравнению с рассмотренной нами ранее архитектурой микропроцессоров. Поэтому имеет смысл рассмотреть эти особенности более подробно.

8.1. Особенности аппаратных средств микроконтроллеров

Микроконтроллеры семейства MCS51 построены по гарвардской архитектуре, то есть имеют отдельную независимую память для данных и для программ. Кроме того, в микроконтроллерах имеется возможность расширения как памяти данных, так и памяти программ за счет добавления внешних кристаллов памяти. Память, которая находится непосредственно на кристалле микроконтроллера, мы будем называть резидентной. Основу структурной схемы микроконтроллера (рис. 8.1) обра-

зует внутренняя двунаправленная 8-битная шина, которая связывает между собой все основные узлы и устройства: резидентную память, процессор, блок регистров специальных функций (таймеры, последовательный порт и др.), устройство управления и порты ввода/вывода. Рассмотрим основные элементы структуры и особенности организации вычислительного процесса в микроконтроллере.

8-битное АЛУ может выполнять арифметические операции сложения, вычитания, умножения и деления, логические операции «И», «ИЛИ», «Исключающее ИЛИ», а также операции циклического сдвига, сброса, инвертирования и т. п. В АЛУ имеются программно-недоступные регистры, предназначенные для временного хранения операндов, схема десятичной коррекции и схема формирования признаков. Простейшая операция сложения используется в АЛУ для инкрементирования содержимого регистров, продвижения регистра-указателя данных и автоматического вычисления следующего адреса памяти программ. Простейшая операция вычитания используется в АЛУ для декрементирования регистров и сравнения переменных.

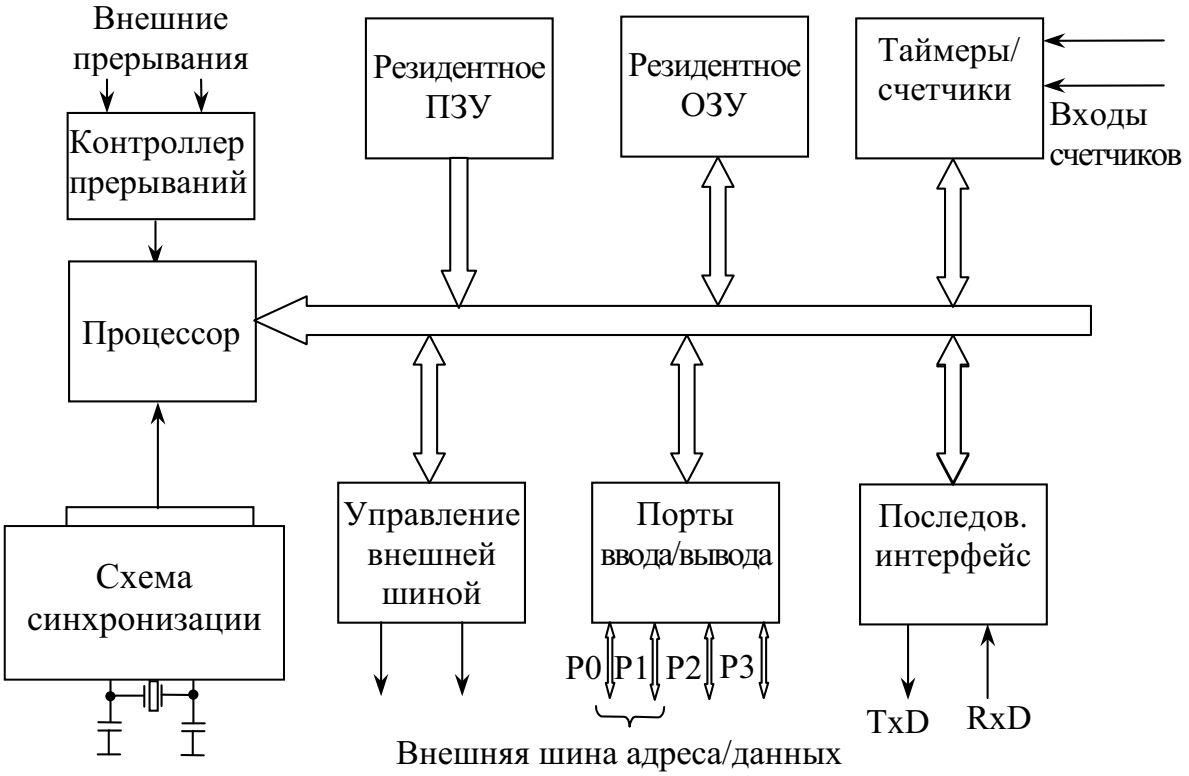


Рис. 8.1. Структурная схема микроконтроллера

Важной особенностью АЛУ является его способность оперировать не только байтами, но и битами. Отдельные программно-доступные биты могут быть установлены, сброшены, инвертированы, переданы, проверены и использованы в логических операциях. Эта способность АЛУ

оперировать битами очень важна. Для управления объектами часто применяются алгоритмы, содержащие операции над входными и выходными булевскими переменными (истина/ложь), реализация которых средствами обычных микропроцессоров сопряжена с трудностями.

Таким образом, АЛУ может оперировать четырьмя типами информационных объектов: булевскими (1 бит), цифровыми (4 бита), байтовыми (8 бит) и адресными (16 бит). В АЛУ выполняется 51 различная операция пересылки или преобразования этих данных.

Память программ и память данных, размещенные на кристалле микроконтроллера, физически и логически разделены, имеют различные механизмы адресации, работают под управлением различных сигналов и выполняют разные функции.

Резидентная память программ имеет емкость 4 кб и предназначена для хранения команд, констант, управляющих слов инициализации, таблиц перекодировки входных и выходных переменных. Память программ имеет 16-битную шину адреса, через которую обеспечивается доступ из счетчика команд или из регистра-указателя данных.

Структура встроенной (резидентной) памяти данных показана на рис. 8.2. Резидентная память данных (РПД) предназначена для хранения переменных в процессе выполнения прикладной программы, адресуется одним байтом и имеет емкость 128 байт. Кроме того, к адресному пространству РПД примыкают адреса регистров специальных функций, перечисленные в табл. 8.1.

Младшие 32 байта резидентной памяти (начиная с адреса 00H) сгруппированы в 4 банка по 8 регистров. В программе к ним можно обратиться по именам R0…R7 (к регистрам можно обратиться и обычным способом – указывая адреса ячеек памяти). Два бита в слове состояния программы (PSW) определяют, какой из банков регистров используется в данный момент. Это позволяет более эффективно использовать память программ, так как команды с регистровой адресацией короче, чем команды с прямой адресацией.

Следующие 16 байт образуют побитно-адресуемую область резидентной памяти. Система команд микроконтроллера включает широкий набор команд работы с отдельными битами, и 128 прямоадресуемых бит могут быть эффективно использованы этими командами. Адреса битов составляют от 00H до 7FH. 16 регистров в области регистров специальных функций также являются побитно-адресуемыми (это регистры с адресами, оканчивающимися на 000B). Адреса битов в этой области составляют от 80H до 0FFH.

Память программ так же, как и память данных, может быть расширена до 64 кб путем подключения внешних БИС.

Аккумулятор является источником операнда и местом фиксации результата при выполнении арифметических, логических операций и ряда операций передачи данных. Кроме того, только с использованием аккумулятора могут быть выполнены операции сдвигов, проверка на нуль, формирование флага паритета и т. п.

Рис. 8.2. Распределение внутренней памяти микроконтроллера

8-битный указатель стека (SP) может адресовать любую область РПД. Его содержимое инкрементируется прежде, чем данные будут запомнены в стеке в ходе выполнения команд PUSH и CALL. Содержимое SP декрементируется после выполнения команд POP и RET. Подобный способ адресации элементов стека называют прединкрементным/постдекрементным. В процессе инициализации после сигнала СБРОС в SP автоматически загружается код 07H. Это значит, что если прикладная программа не переопределяет стек, то первый элемент данных в стеке

будет располагаться в ячейке РПД с адресом 08H. Двухбайтовый регистр-указатель данных (DPTR) обычно используется для фиксации 16-битного адреса в операциях с обращением к внешней памяти. Командами микроконтроллера DPTR может быть использован или как 16-битный регистр, или как два независимых 8-битных регистра (DPH и DPL).

В составе микроконтроллера имеются регистровые пары с символическими именами TH0, TL0 и TH1, TL1, на основе которых функционируют два независимых программно-управляемых 16-битных таймера/счетчика событий. Регистр с символическим именем SBUF представляет собой два независимых регистра – буфер приемника и буфер передатчика данных в последовательном формате. Загрузка байта в SBUF немедленно вызывает начало процесса передачи через последовательный порт. Когда байт считывается из SBUF, это значит, что его источником является приемник последовательного порта.

Таблица 8.1

Блок регистров специальных функций

Символ	Наименование	Адрес
*ACC	Аккумулятор	0E0H
*B	Регистр расширитель аккумулятора	0F0H
* PSW	Слово состояния программы	0D0H
SP	Регистр-указатель стека	81H
DPTR	Регистр-указатель данных (DPH)	83H
	(DPL)	82H
*P0	Порт 0	80H
*P1	Порт 1	90H
*P2	Порт 2	0A0H
*P3	Порт 3	0B0H
*IP	Регистр приоритетов	0B8H
*IE	Регистр маски прерываний	0A8H
TMOD	Регистр режима таймера/счетчика	89H
*TCON	Регистр управления/статуса таймера	88H
TH0	Таймер 0 (старший байт)	8CH
TL0	Таймер 0 (младший байт)	8AH
TH1	Таймер 1 (старший байт)	8DH
TL1	Таймер 1 (младший байт)	8BH
*SCON	Регистр управления приемопередатчиком	98H
SBUF	Буфер приемопередатчика	99H
PCON	Регистр управления мощностью	87H

Примечание. Регистры, имена которых отмечены знаком (*), допускают адресацию отдельных бит.

Регистры с символическими именами IP, IE, TMOD, TCON, SCON и PCON используются для фиксации и программного изменения управляющих битов и битов состояния схемы прерывания, таймера/счетчика, приемопередатчика последовательного порта и для управления мощностью электропитания микроконтроллера.

Все четыре порта микроконтроллера предназначены для ввода или вывода информации побайтно. Каждый порт содержит управляемые регистр-защелку, входной буфер и выходной драйвер.

Выходные драйверы портов 0 и 2, а также входной буфер порта 0 используются при обращении к внешней памяти (ВП). При этом через порт 0 в режиме временного мультиплексирования сначала выводится младший байт адреса ВП, а затем выдается или принимается байт данных. Через порт 2 выводится старший байт адреса в тех случаях, когда разрядность адреса равна 16 бит.

Все выводы порта 3 могут быть использованы для реализации альтернативных функций. Альтернативные функции могут быть задействованы путем записи 1 в соответствующие биты регистра-защелки (P3.0…P3.7) порта 3.

Порт 0 является двунаправленным, а порты 1, 2 и 3 – квазидвунаправленными. Каждая линия портов может быть использована независимо для ввода или вывода информации. Для того чтобы некоторая линия порта использовалась для ввода, в D-триггер регистра-защелки порта должна быть записана 1, которая закрывает МОП-транзистор выходной цепи. При этом линии портов 1, 2, 3 будут установлены в единичное состояние за счет внутренних «подтягивающих» резисторов между шиной питания и линией ввода/вывода. Но их можно перевести в нулевое состояние внешним входным напряжением.

Порт 0 отличается тем, что вместо «подтягивающих» резисторов установлены транзисторы. Эти транзисторы используются только тогда, когда через линии порта выводятся единицы при обращении к внешней памяти (т. е. порт используется как драйвер шины данных). Во всех остальных случаях транзистор выключен, соответственно, линии порта являются выходами с открытым коллектором. Если выходной управляющий транзистор регистра-защелки закрыт, линия порта находится в «плавающем» высокоомном состоянии и может использоваться как вход. Так как порты 1, 2 и 3 имеют фиксированные внутренние «подтягивающие» резисторы, они иногда называются квазидвунаправленными. Когда они сконфигурированы на ввод информации, они находятся в единичном состоянии и внешним сигналом могут быть переведены в нулевое. Порт 0 двунаправленный и, будучи сконфигурирован на ввод, находится в высокоомном состоянии.

Таблица 8.2

Альтернативные функции порта 3

Символ	Позиция	Имя и назначение
RD	P3 7	Чтение. Активный сигнал низкого уровня формируется аппаратно при обращении к ВПД
WR	P3.6	Запись. Активный сигнал низкого уровня формируется аппаратно при обращении к ВПД
T1	P3.5	Вход таймера/счетчика 1 или тест-вход
T0	P3.4	Вход таймера/счетчика 0 или тест-вход
INT1	P3.3	Вход запроса прерывания 1. Воспринимается сигнал низкого уровня или срез
INT0	P3.2	Вход запроса прерывания 0. Воспринимается сигнал низкого уровня или срез
TXD	P3.1	Выход передатчика последовательного порта в режиме УАПП. Выход синхронизации в режиме сдвигающего регистра
RXD	P3.0	Вход приемника последовательного порта в режиме УАПП. Ввод/вывод данных в режиме сдвигающего регистра

Микроконтроллеры MCS51 имеют встроенный генератор тактовых импульсов. Чтобы его использовать, необходимо подключить кварцевый резонатор между выводами XTAL1 и XTAL2 микроконтроллера и конденсаторы, как показано на рис. 8.3.

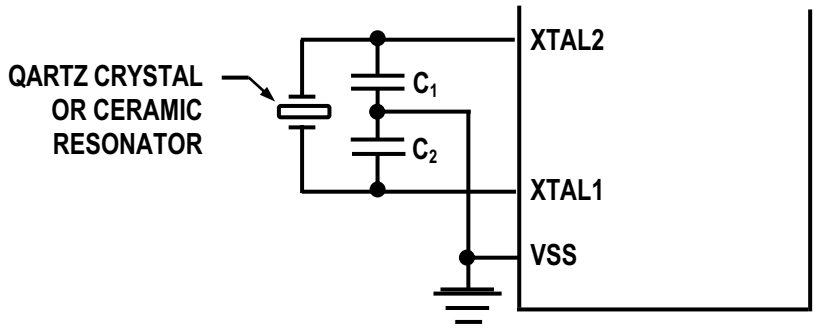


Рис. 8.3. Использование встроенного генератора

Генератор синхронизации определяет последовательность из 6 тактов (S1…S6), образующих машинный цикл микроконтроллера. Каждый такт длится два периода синхронизации. Таким образом, машинный цикл длится 12 тактов синхронизации, или 1 мкс, при частоте генератора 12 МГц. Каждый такт делится на 2 фазы. На рис. 8.4 показаны временные диаграммы выполнения различных типов команд. Обычно в каждом машинном цикле формируется два обращения к памяти программ для выборки команды, даже если это не требуется. В таком слу-

чае второй выбранный байт просто игнорируется, а программный счетчик не инкрементируется.

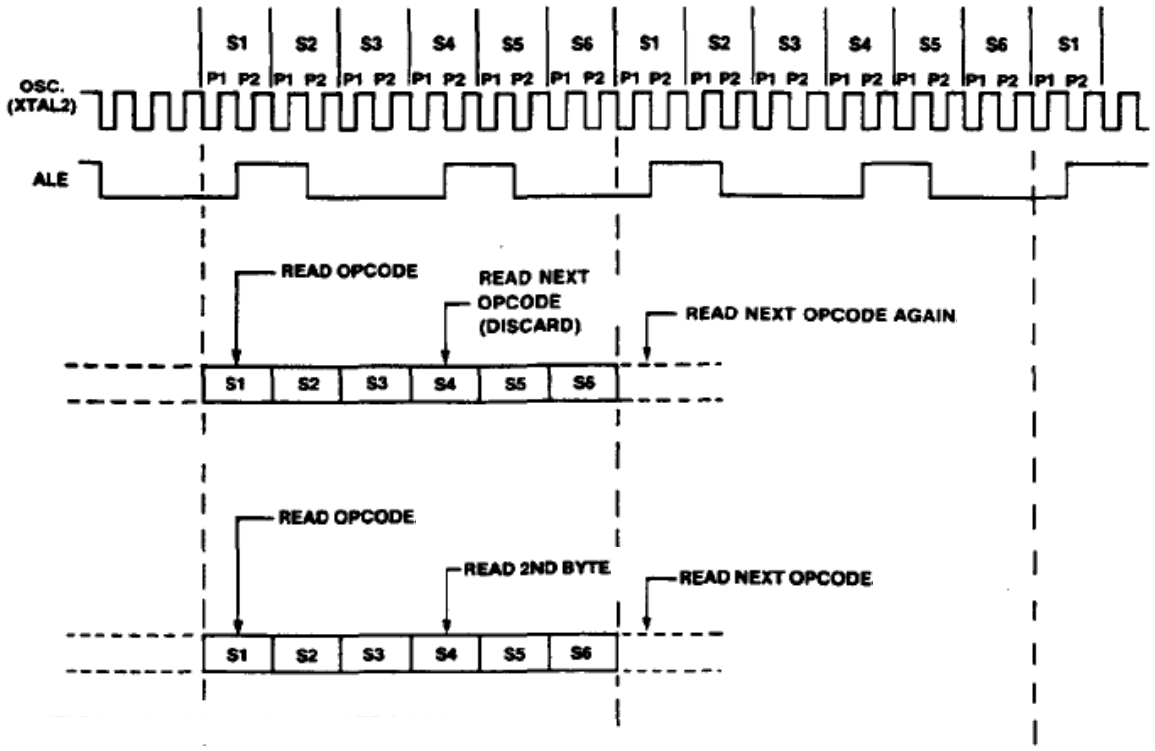


Рис. 8.4. Машинные циклы выполнения команд

Выполнение команд в один цикл начинается в такте S1, в котором код команды помещается в регистр команд. Следующая выборка команды происходит в такте S4. Заканчивается выполнение цикла в такте S6.

Последовательность тактов выборки/выполнения команд и время выполнения не зависит от того, какая память программ – внутренняя (резидентная) или внешняя – используются. На рис. 8.5 показаны временные диаграммы сигналов обращения к внешней памяти программ. В этом случае дважды за машинный цикл активируется строб чтения внешней памяти программ PSEN. Когда происходит обращение к внешней памяти данных, сигнал PSEN не формируется, а активируются сигналы WR или RD. Цикл обращения к внешней памяти данных занимает вдвое большее время, чем обращение к внешней памяти программ.

При выполнении программы из резидентной памяти программ сигнал PSEN не формируется, однако сигнал ALE по-прежнему формируется дважды за машинный цикл, т. е. его можно использовать как сигнал синхронизации для дополнительных устройств системы.

Схема подключения внешней памяти программ приведена на рис. 8.6, 16 линий портов P0 и P2 используются как мультиплексная шина адреса/данных. Порт P0 используется для передачи данных и старших

разрядов адреса памяти, а порт P2 для передачи младшего байта адреса памяти. Сигнал ALE используется для разделения сигналов адреса и данных (ALE сопровождает адресную информацию на линиях порта P0).

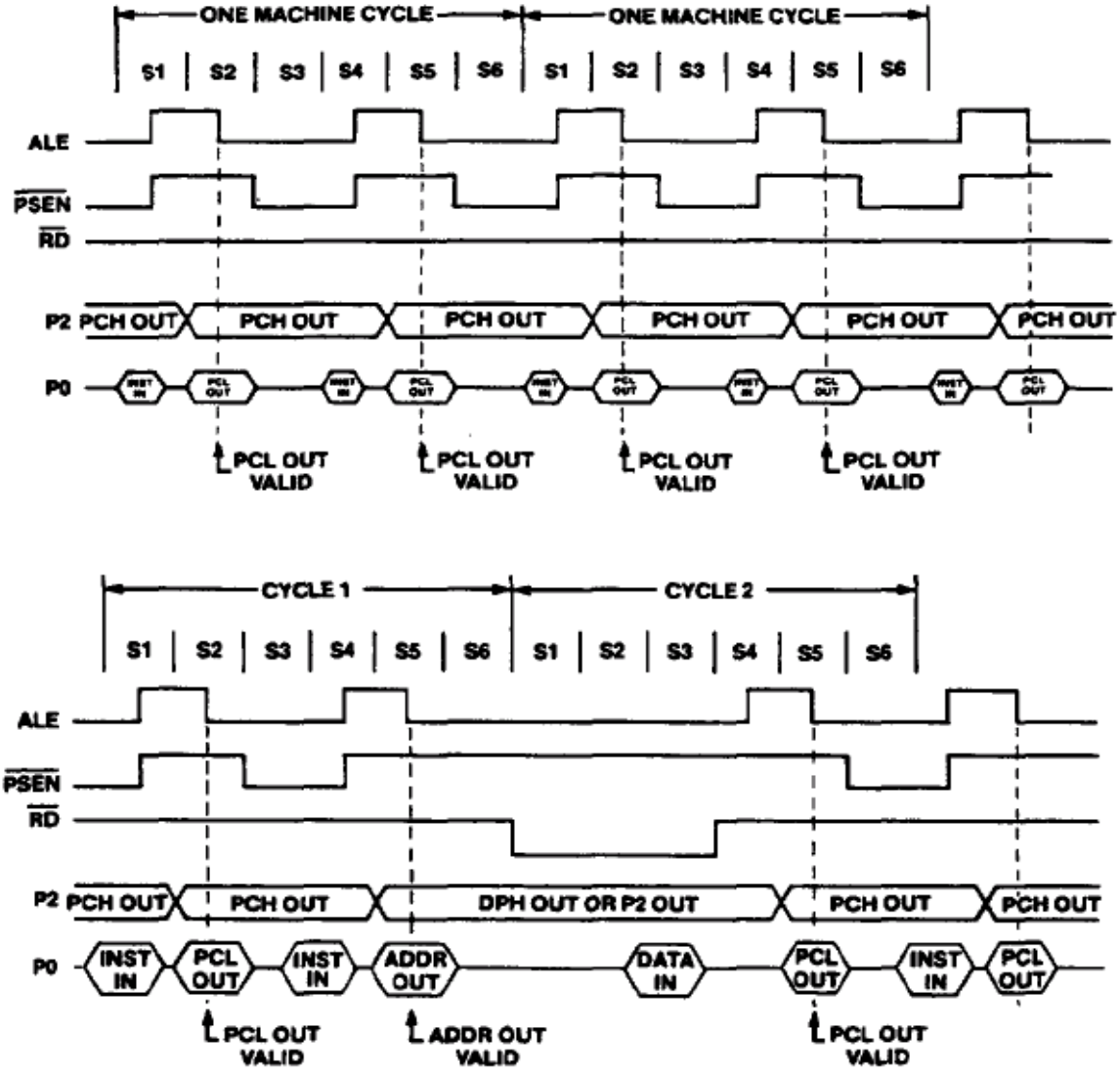


Рис. 8.5. Циклы обращения к внешней памяти

На рис. 8.7 показана схема доступа к внешней памяти данных объемом 2 К. Программа выполняется из внутренней памяти программ.

Микроконтроллер использует 5 источников прерывания: 2 внешних прерывания, 2 прерывания от таймера и прерывание от порта последовательного обмена данными.

Каждый источник прерывания может быть разрешен или запрещен посредством установки или сброса битов в регистре специальных функций IE. Приоритет прерываний может быть переустановлен с использованием регистра IP. Прерывание более низкого приоритета может быть прервано прерыванием более высокого уровня. Если приняты два за-

проса на прерывание одновременно, то активируется прерывание более высокого уровня.

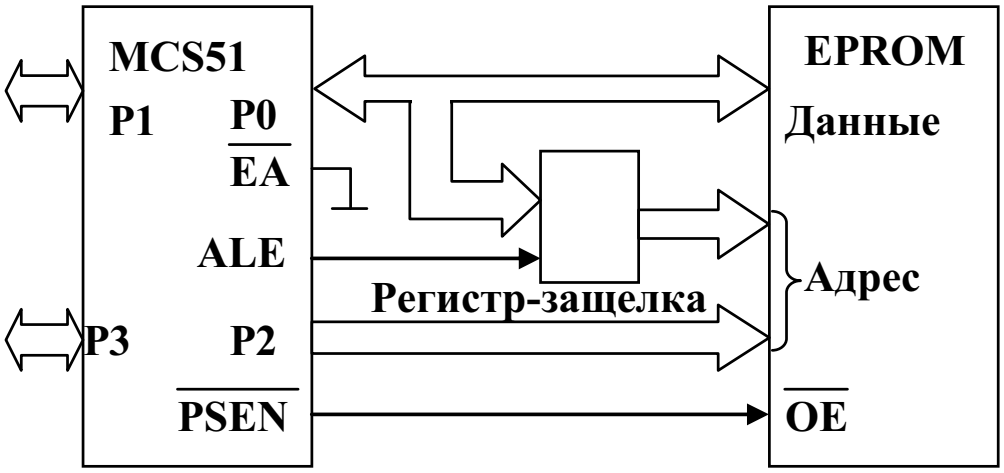


Рис. 8.6. Организация доступа к внешней памяти программ

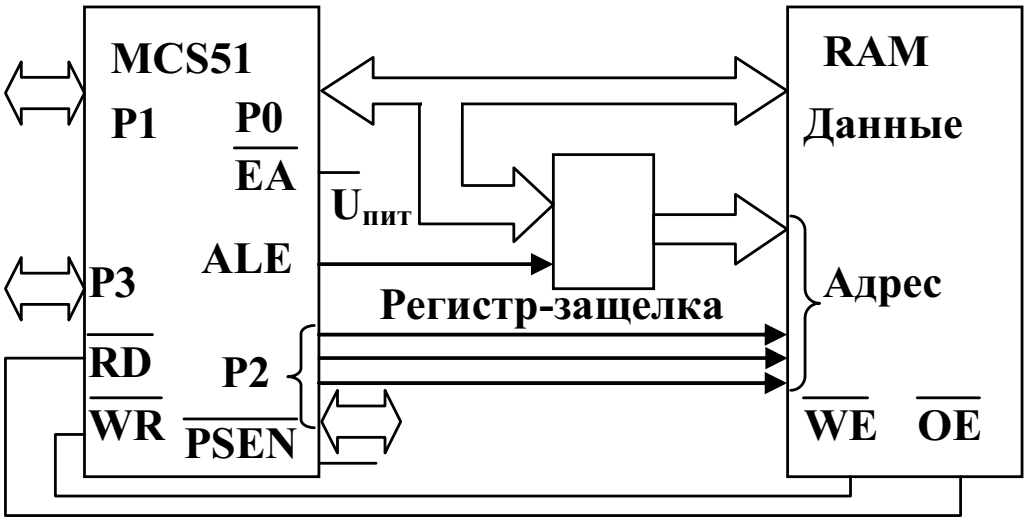


Рис. 8.7. Организация доступа к внешней памяти данных

Рис. 8.8 показывает источники прерываний и функции регистров IE и IP, а также начальную установку приоритетов прерываний от разных источников. При работе микроконтроллера флаги прерываний опрашиваются в такте S5 каждого машинного цикла и обрабатываются в следующем машинном цикле. Если прерывание разрешено, система прерываний формирует команду LCALL к началу подпрограммы обработки прерывания. При этом текущее содержимое программного счетчика записывается в стек, а программный счетчик перегружается стартовым адресом подпрограммы. Для каждого типа прерывания стартовый адрес программы обработки прерывания начинается с определенного фиксированного адреса.

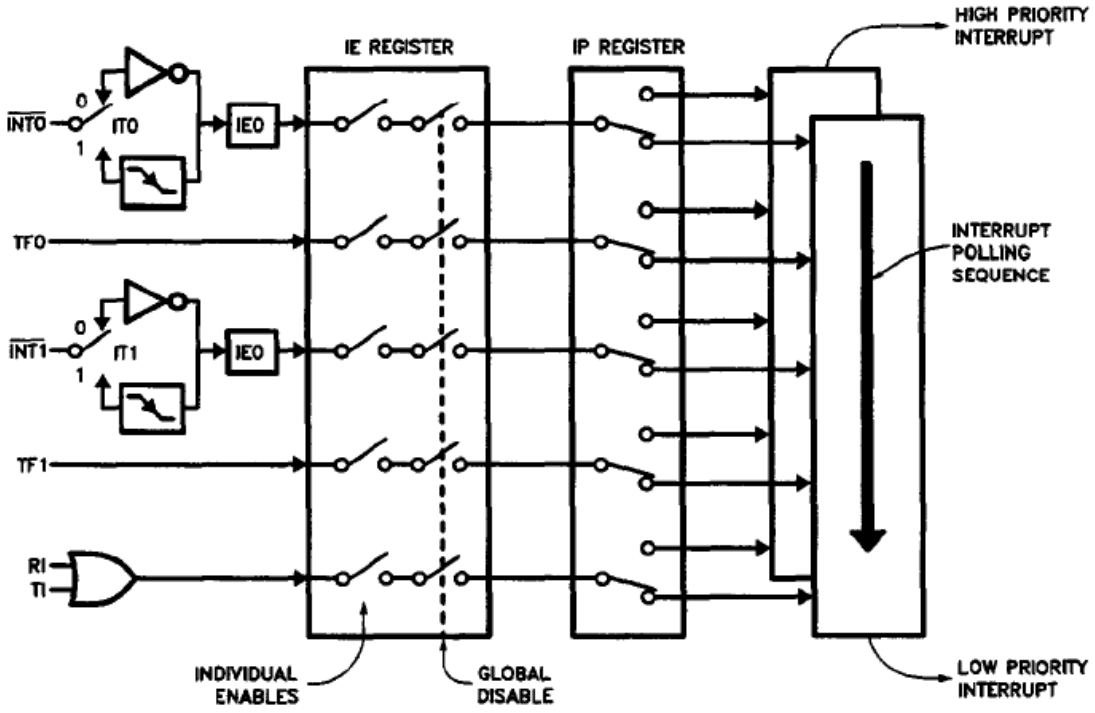


Рис. 8.8. Организация системы прерываний микроконтроллера

8.2. Система команд MCS51

Система команд микроконтроллера содержит 111 базовых команд, которые по функциональному признаку удобно разделить на пять групп: команды передачи данных, арифметических операций, логических операций, передачи управления и операций с битами.

Система команд MCS51 достаточно мощная и широкая, в ее состав входят команды умножения, деления, операций над битами, операций со стеком и расширенный набор команд передачи управления. Большинство команд имеют одно- или двухбайтовый формат и выполняются за один или два машинных цикла. При тактовой частоте 12 МГц длительность машинного цикла составляет 1 мкс.

Операнды в командах микроконтроллера могут быть четырех типов: биты, 4-битные двоично-десятичные цифры, байты и 16-битные слова. Микроконтроллер имеет 128 программно-доступных битов в резидентной памяти данных. Имеется также возможность адресации отдельных битов блока регистров специальных функций и портов. Для адресации битов используется прямой 8-битный адрес. Косвенная адресация бит невозможна.

Слово состояния программы (PSW) содержит несколько бит, отражающих текущее состояние процессора. Назначение битов PSW показано на рис. 8.9.

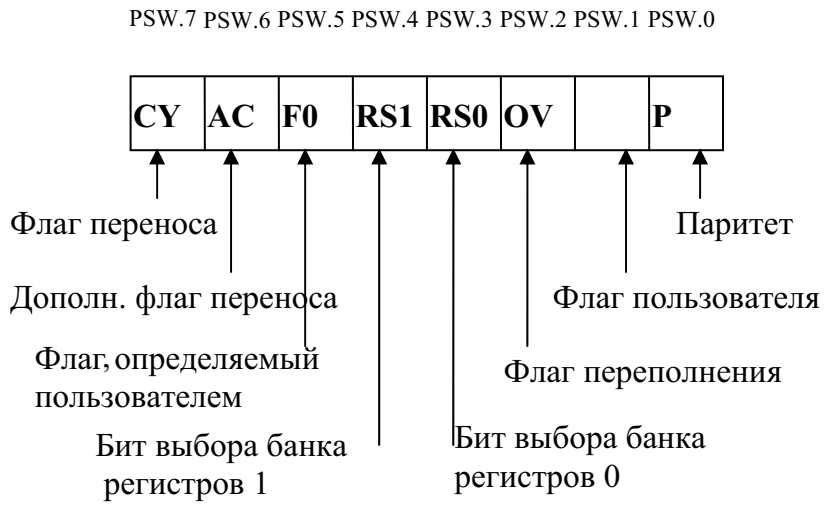


Рис. 8.9. Слово состояния программы

Биты RS0 и RS1 используются для выбора одного из четырех банков регистров в качестве текущего. Два бита статусного регистра могут быть использованы по усмотрению программиста. Слово состояния программы включает в себя четыре флага: CY – перенос, AC – вспомогательный перенос, OV – переполнение и P – паритет. Флаг паритета напрямую зависит от текущего значения аккумулятора. Если число единичных бит аккумулятора нечетное, то флаг P устанавливается, а если четное – сбрасывается. Флаг формируется комбинационной логической схемой и не фиксируется триггером, поэтому изменить флаг P, присваивая ему новое значение, нельзя, если содержимое аккумулятора при этом останется неизменным. Флаг AC устанавливается в случае, если при выполнении операции сложения/вычитания между тетрадами байта возник перенос/заем. Флаг CY устанавливается, если в старшем бите результата возникает перенос или заем. При выполнении операций умножения и деления флаг CY сбрасывается. Флаг OV устанавливается, если результат операции сложения/вычитания не укладывается в семи битах и старший (восьмой) бит результата не может интерпретироваться как знаковый. При выполнении операции деления флаг OV сбрасывается, а в случае деления на нуль устанавливается. При умножении флаг OV устанавливается, если результат больше 255.

Микроконтроллер использует в качестве операндов одно- и двухбайтовые числа, четырехбитные числа и отдельные биты. Четырехбитные операнды используются в операциях обмена и при вычислениях. Однобайтовым операндом может быть ячейка памяти программ или данных (резидентной или внешней), константа (непосредственный операнд), регистры специальных функций, а также порты ввода/вывода.

Двухбайтные операнды – это константы и прямые адреса, для представления которых используются второй и третий байты команды.

Микроконтроллер использует следующие способы адресации операндов: прямая адресация, косвенная адресация, регистровая адресация, непосредственная адресация и индексная адресация.

При прямой адресации 8-битный адрес операнда является вторым байтом команды. Прямым способом могут быть адресованы только операнды, которые находятся в резидентной памяти данных и области регистров специальных функций.

При косвенной адресации в команде указывается регистр, который содержит адрес операнда. В качестве регистров могут быть использованы регистры R0 или R1 из текущего банка регистров или указатель стека. Так можно адресовать операнды, находящиеся в резидентной и внешней памяти данных (адресуется только первые 256 байт). Для адресации всей внешней памяти нужно использовать регистр указатель данных DPTR.

Регистры адресуются 3-битовым кодом в коде команды. Регистровый метод адресации очень эффективен с точки зрения экономии памяти программ.

При непосредственной адресации операнд следует в команде за кодом операции. В ассемблерной записи команды непосредственному операнду предшествует знак #. Например, команда MOV A,#100 загружает в аккумулятор десятичное число 100 (или шестнадцатеричное число 64 H).

Индексный метод адресации используется для адресации памяти программ. В этом случае регистр DPTR используется как базовый, а аккумулятор как индексный регистр.

Группу команд арифметических операций образуют 24 команды, выполняющие операции сложения, десятичной коррекции, инкремента/декремента байтов, а также команды вычитания, умножения и деления байтов. Команды этой группы перечислены в табл. 8.3. Таблица показывает способы адресации, которые могут быть использованы в команде для доступа к операнду. Например, команда ADD A,<byte > в зависимости от метода адресации будет выглядеть следующим образом:

```
ADD   A,7FH      (прямая адресация);
ADD   A,@R0      (косвенная адресация);
ADD   A,R7       (регистровая адресация);
ADD   A, #127    (непосредственная адресация).
```

Таблица 8.3

Арифметические команды

Мнемоника	Операция	Способы адресации			
		Прямой	Косвенный	Регистровый	Непосредственный
ADD A,<byte >	A = A+<byte >	+	+	+	+
ADDC A,<byte >	A = A+<byte > + C	+	+	+	+
SUBB A,<byte >	A = A-<byte > – C	+	+	+	+
INC A	A = A+1	только аккумулятор			
INC <byte >	<byte > = <byte > +1	+	+	+	
INC DPTR	DPTR = DPTR+1	только DPTR			
DEC A	A = A-1	только аккумулятор			
DEC <byte >	<byte > = <byte > – 1	+	+	+	
MUL AB	B:A = B×A	только ACC и B			
DIV AB	A = Int[A/B] B = Mod[A/B]	только ACC и B			
DA A	Decimal Ajust	только аккумулятор			

Все арифметические команды выполняются за один машинный цикл, за исключением команды INC DPTR (2 цикла) и команд умножения и деления, выполняющихся за 4 цикла.

Операция умножения MUL AB перемножает содержимое аккумулятора и B регистра и помещает результат в эти же регистры. Операция деления делит данное, находящееся в аккумуляторе, на содержимое регистра B. Частное помещается в аккумулятор, а остаток – в регистр B.

Команда DA A используется в арифметических операциях с двоично-десятичными числами. Если используются двоично-кодированные десятичные числа, то за командами сложения должны следовать команды десятичной коррекции.

В табл. 8.4 перечислены команды логических операций. В эту группу входят 25 команд, реализующих логические операции над байтами. Однако в MCS51 значительно расширено число типов операндов, участвующих в операциях. Имеется возможность производить операции «исключающее ИЛИ» с содержимым портов. В командах логических операций также можно использовать различные способы адресации операндов.

Таблица 8.4

Логические команды

Мнемоника	Операция	Способы адресации			
		Прямой	Косвенный	Регистровый	Непосредственный
ANL A,<byte >	A = A .AND. <byte >	+	+	+	+
ANL <byte > , A	<byte > = <byte > .AND. A	+			
ANL <byte > , #data	<byte > = <byte > .AND. #data	+			
ORL A,<byte >	A = A .OR. <byte >	+	+	+	+
ORL <byte > , A	<byte > = <byte > .OR. A	+			
ORL <byte > , #data	<byte > = <byte > .OR. #data	+			
XRL A,<byte >	A = A .XOR. <byte >	+	+	+	+
XRL <byte > , A	<byte > = <byte > .XOR. A	+			
XRL <byte > , #data	<byte > = <byte > .XOR. #data	+			
CRL A	A = 00H	только аккумулятор			
CPL A	A = .NOT. A	только аккумулятор			
RL A	Сдвиг ACC влево на 1 бит	только аккумулятор			
RLC A	Сдвиг влево через перенос	только аккумулятор			
RR A	Сдвиг ACC вправо на 1 бит	только аккумулятор			
RRC A	Сдвиг вправо через перенос	только аккумулятор			
SWAP A	Обмен тетрад в ACC	только аккумулятор			

Команда SWAP A обменивает местами младшую и старшую тетрады аккумулятора. Эта операция полезна при действиях с двоично-десятичными числами. Например, если аккумулятор содержит число, которое заведомо меньше 100, то оно может быть быстро переведено в двоично-десятичный код следующей программой:

```
MOV     B,# 10
DIV     AB
SWAP    A
ADD     A,B
```

Деление числа в аккумуляторе на 10 дает количество десятков в аккумуляторе, а количество единиц – в регистре B. Команды SWAP и ADD формируют число в упакованном BCD-формате.

Команды передачи данных показаны в табл. 8.5. Большую часть команд передачи данных составляют команды передачи и обмена байтов. Все команды данной группы не модифицируют флаги результата, за исключением команд загрузки PSW и аккумулятора (флаг паритета). Команды пересылки бит представлены в группе команд битовых операций (табл. 8.8). Все команды выполняются за 1 или 2 машинных цикла.

Таблица 8.5

Команды передачи данных, использующие РПД

Мнемоника	Операция	Способы адресации			
		Прямой	Косвенный	Регистровый	Непосредственный
MOV A,<src >	A = <src >	+	+	+	+
MOV <dest > ,A	<dest > = A	+	+	+	
MOV <dest > ,<src >	<dest > = < >	+	+	+	+
MOV DPTR,#data16	DPTR = 16-битовая константа				+
PUSH <src >	INC SP: MOV @SP,<src >	+			
POP <dest >	MOV <dest > ,@SP: DEC SP	+			
XCH A,<byte >	обмен данных между ACC и <byte >	+	+	+	
XCHD A,@R_i	ACC и @R_i обменивают младшие тетрады		+		

Стек микроконтроллера расположен в резидентной памяти данных и заполняется в сторону увеличения адресов. Команда записи в стек увеличивает указатель стека, а затем копирует данное в стек. При извлечении из стека порядок обратный. Следует подчеркнуть, что стек не может записан по адресам, расположенным в области регистров специальных функций. Если указатель стека указывает на эту область, то при записи в стек данные теряются, а при чтении из стека будут неопределенными.

Команда XCH A,<byte > обменивает содержимое аккумулятора и адресуемого байта. XCHD A,@R_i подобна предыдущей, но в обмене участвуют только младшие тетрады. Чтобы продемонстрировать, как эти команды могут облегчить обработку данных, рассмотрим пример. Пусть необходимо сдвинуть 8-разрядное двоично-десятичное число вправо на 2 цифры. Программа ниже показывает, как это можно сделать двумя способами:

```
MOV     A,2EH
MOV     2EH,2DH
MOV     2DH,2CH
MOV     2CH,2BH
MOV     2BH,#0
```

(a) Используя команды MOV: 14 байтов, 9 мкс;

```
CLR  A
XCH  A,2BH
XCH  A,2BH
XCH  A,2BH
XCH  A,2BH
```

(b) Используя XCHS: 9 байт, 5 мкс.

Таблица 8.6

Команды для работы с внешней памятью данных

Address width	Mnemonic	Operation
8 bits	MOVX A,@R_i	Read external RAM @R_i
8 bits	MOVX @R_i,A	Write external RAM @R_i
16 bits	MOVX A,@DPTR	Read external RAM @DPTR
16 bits	MOVX @DPTR,A	Write external RAM @DPTR

Для работы с внешней памятью данных используются команды, описанные в табл. 8.6. Используется только косвенная адресация операндов. Можно использовать регистры R0 или R1 и DPTR в качестве источника косвенного адреса. В первом случае можно адресовать только первые 256 байт внешней памяти.

В программной памяти программист может располагать таблицы данных. Для доступа к таким таблицам используются команды из табл. 8.7.

Таблица 8.7

Команды работы с таблицами в программной памяти

Мнемоника	Операция
MOVC A,@A+DPTR	Чтение памяти программ по адресу (A+DPTR)
MOVC A,@A+PC	Чтение памяти программ по адресу (A+PC)

Команда MOVC обеспечивает доступ к таблице, содержащей до 256 входов (обозначаемых от 0 до 255). Номер требуемого входа в таблицу записывается в аккумулятор, а DPTR указывает на базовый (начальный) адрес таблицы. Затем команда копирует данное из требуемой ячейки таблицы в аккумулятор.

Вторая команда работает аналогичным образом, только для доступа к таблице используется программный счетчик, который используется

как источник базового адреса таблицы, а доступ к таблице осуществляется через подпрограмму. Номер требуемой ячейки таблицы помещается в аккумулятор, после чего вызывается подпрограмма:

```
        MOV         A,ENTRY_NUMBER
        CALL        TABLE
```

Подпрограмма “TABLE” должна выглядеть следующим образом:

```
TABLE:  MOVC    A,@A+PC
        RET
```

Собственно таблица должна следовать сразу же за командой RET. Данный способ обеспечивает доступ к 255 ячейкам таблицы. Номер 0 не может быть использован, так как он указывает на код команды RET.

Таблица 8.8

Битовые операции

Mnemonic	Operation
ANL C,bit	C = C .AND. bit
ANL C,/bit	C = C .AND. .NOT. bit
ORL C,bit	C = C .OR. bit
ORL C,/bit	C = C .OR. .NOT. bit
MOV C,bit	C = bit
MOV bit,C	Bit = C
CLR C	C = 0
CLR bit	Bit = 0
SETB C	C = 1
SETB bit	Bit = 1
CPL C	C = .NOT. C
CPL bit	Bit = .NOT. bit
JC rel	Переход, если C = 1
JNC rel	Jump if C = 0
JB bit,rel	Jump if bit = 1
JNB bit,rel	Jump if bit = 0
JBC bit,rel	Jump if bit = 1, CLR bit

Отличительной особенностью системы команд MCS51 является наличие группы команд работы с битовыми операндами (табл. 8.8). В качестве таких операндов могут выступать отдельные биты некоторых регистров специальных функций (РСФ) и портов, а также 128 программно-доступных битов в резидентной памяти данных. Существуют команды сброса, установки и инверсии бит, а также конъюнкции и дизъюнкции бита и флага переноса. Все команды используют прямую адресацию бит.

Используя битовые команды, можно, например, легко скопировать значение флага на линию ввода/вывода порта:

```
MOV     C,FLAG
MOV     P1.0,C
```

В примере FLAG это имя любого прямо адресуемого бита. Линия 0 порта 1 будет установлена в 1 или 0 в зависимости от значения бита FLAG.

Следующий пример показывает как можно выполнить операцию исключающего «ИЛИ» над битами (C = bit l.XRL.bit 2):

```
MOV     C,bit 1
JNB     bit 2,OVER
CPL     C
OVER: (continue)
```

Сначала бит 1 копируется в бит переноса. Если бит 2 равен 0, то бит переноса содержит правильный результат. В противном случае бит переноса должен быть инвертирован.

В примере использована команда проверки битов и переходов по результатам проверки. Команда JBC выполняет переход и очищает бит за одну операцию.

Биты слова состояния программы PSW являются прямоадресуемыми, т. е. они могут участвовать в битовых операциях.

К группе команд передачи управления относятся команды, обеспечивающие условное и безусловное ветвление, вызов подпрограмм и возврат из них, а также команда пустой операции. В большинстве команд используется прямая адресация, т. е. адрес перехода целиком (или его часть) содержится в самой команде передачи управления. Можно выделить три разновидности команд ветвления по разрядности указываемого адреса перехода:

- Длинный переход (L). Переход по всему адресному пространству памяти программ. В команде содержится полный 16-битный адрес перехода.
- Относительный переход (S). Короткий относительный переход позволяет передать управление в пределах –128…+127 байт относительно адреса следующей команды (команды, следующей по порядку за командой относительного перехода).
- Абсолютный переход (A). Адрес перехода является 11-разрядным числом, что сокращает длину команды до двух байт.

Команды безусловных переходов представлены в табл. 8.9. Хотя в таблице показана только одна команда JMP, реально микроконтроллер имеет три команды AJMP, SJMP и LJMP.

Таблица 8.9

Безусловные переходы

Мнемоника	Операция
JMP addr	Переход по адресу
JMP @A+DPTR	Переход по косвенному адресу @A+DPTR
CALL addr	Вызов подпрограммы
RET	Возврат из подпрограммы
RETI	Возврат из прерывания
NOP	Нет операции

Команда косвенного перехода JMP @A+DPTR позволяет передавать управление по косвенному адресу. Эта команда удобна тем, что предоставляет возможность организации перехода по адресу, вычисляемому самой программой и неизвестному при написании исходного текста программы. Обычно DPTR указывает на начальный адрес таблицы переходов, а аккумулятор на номер входа в таблице. Например, таблица переходов может быть реализована следующим образом:

```
MOV       DPTR, #JUMP_TABLE
MOV       A,INDEX_NUMBER
RL        A
JMP       @A+DPTR
```

Команда RL A преобразует номер входа в таблице (от 0 до 4) в четное число в диапазоне от 0 до 8, так как каждый вход в таблице переходов занимает 2 байта:

```
JUMP_TABLE:
AJMP  CASE_0
AJMP  CASE_1
AJMP  CASE_2
AJMP  CASE_3
AJMP  CASE_4
```

Развитая система условных переходов (табл. 8.10) предоставляет возможность осуществлять ветвление по следующим условиям: аккумулятор содержит нуль; содержимое аккумулятора не равно нулю; перенос равен единице; перенос равен нулю, адресуемый бит равен единице; адресуемый бит равен нулю. PSW не содержит бита нулевого результата, поэтому команды проверяют непосредственно аккумулятор.

Таблица 8.10

Условные переходы

Мнемоника	Операция	Способы адресации			
		Прямой	Косвенный	Регистровый	Непосредственный
JZ rel	Переход, если A = 0	только аккумулятор			
JNZ rel	Переход, если A ≠ 0	только аккумулятор			
DJNZ <byte > ,rel	Декремент и переход, если не нуль	+		+	
CJNE A,<byte > ,rel	Переход, если A ≠ < byte >	+			+
CJNE <byte > ,#data,rel	Переход, если <byte > ≠ # data		+	+	

Для организации программных циклов удобно пользоваться специальными командами, которые одновременно осуществляют декремент регистра, проверку результата и переход, если результат равен нулю. Например, команда DJNZ (Decrement and Jump if Not Zero). Чтобы выполнить цикл раз, нужно загрузить счетчик числом N и в конце цикла использовать DJNZ. Пример показывает цикл, который выполняется 10 раз:

```
    MOV   COUNTER,#10
LOOP: (begin loop)
      .
      .
      (end loop)
      DJNZ        COUNTER,LOOP
      (continue).
```

Команда CJNE (Compare and Jump if Not Equal) также может быть использована для организации циклов. Команда сравнивает первый и второй операнд и осуществляет переход, если они не равны.

Вопросы для повторения

1. Что такое микроконтроллер? Чем он отличается от микропроцессора?
2. Какова основная особенность архитектуры микроконтроллеров?
3. Какие типы памяти можно выделить в микроконтроллере?
4. Каковы основные особенности системы команд микроконтроллера?
5. Какие дополнительные устройства находятся на кристалле микроконтроллера? Как осуществляется управление и обмен данными с этими устройствами?
6. Какие способы адресации используются в микроконтроллерах?

Глава 9
ПРИНЦИПЫ ИСПОЛЬЗОВАНИЯ МИКРОПРОЦЕССОРОВ В ИНФОРМАЦИОННО-ИЗМЕРИТЕЛЬНЫХ СИСТЕМАХ

9.1. Проектирование микропроцессорных автоматизированных измерительных систем

Автоматизированные измерительные системы представляют собой сложный объект проектирования. Для проектирования таких систем общепринят **блочно-иерархический подход**, при котором представления об объекте разбиваются на уровни, начиная с наименее детализированных и кончая наиболее детализированными. Процесс проектирования носит итеративный характер и заключается в движении от решения более общих допросов к решению частных с возможными возвратами на более общие уровни для коррекции принятых решений.

Другой особенностью создания систем является раздвоенность процесса проектирования на программную и аппаратную части.

Упрощенная схема процесса проектирования автоматизированной системы измерений и краткая расшифровка содержания этапов проектирования приведена в табл. 9.1. В этой схеме допущены два основных упрощения. Во-первых, на каждом этапе проектирования зафиксирован только его основной результат, за которым скрыт свой внутренний цикл проектирования и в котором могут быть этапы анализа, проверки условий работоспособности или оценка качества, поиск оптимальных параметров, уточнение технического решение и возвращение к анализу. Такой цикл может повторяться неоднократно. Во-вторых, в схеме на рис. 9.1, ориентированной на проектирование не слишком сложных систем, этап анализа и корректировки приведен только для уровня разработки функциональной схемы и программы. Для более сложных систем анализ и корректировка могут производиться на 3-м, 4-м и на 7-м уровнях, да и само число уровней может быть увеличено. Так, для развитой цифровой аппаратуры с жесткими связями возможно выделение логического, регистрового подуровней и подуровней макроблоков различного масштаба.

Первый и второй этапы проектирования существенно связаны с предварительно проведенными научно-исследовательскими работами (НИР) и эскизным проектированием, в которых должны быть определены физические принципы измерений и системное взаимодействие разрабатываемой системы с окружающей средой.

В ходе выполнения этих этапов строится математическая модель измерения и ставится математическая задача компенсации искажений, вносимых измерительным прибором. Выбирается метод решения этой задачи.

Таблица 9.1

<table>
<tr><th>№ п/п</th><th>Этап</th><th colspan="2">Сущность процесса проектирования</th></tr>
<tr><td>1</td><td>Задача</td><td colspan="2">Строгая формулировка задачи. Определение входных и выходных требований для системы</td></tr>
<tr><td>2</td><td>Метод</td><td colspan="2">Выбор математического метода обработки информации. Обоснование метода путем сравнения вариантов обработки. Расчет и оптимизация неизвестных параметров</td></tr>
<tr><td>3</td><td>Алгоритм</td><td colspan="2">Разработка схем алгоритма. Определение состава и количества операций, операндов и констант, допустимого времени вычислений, характера обмена данными</td></tr>
<tr><td rowspan="3">4</td><td rowspan="3">Структура</td><td colspan="2">Разделение задачи на аппаратную и программную части</td></tr>
<tr><td>Аппаратная часть</td><td>Программная часть</td></tr>
<tr><td>Разработка структурной схемы средств сопряжения. Определение состава устройств</td><td>Выбор микропроцессора и анализ его структуры с точки зрения выполнения требуемых программных операций</td></tr>
<tr><td>5</td><td>Детализация</td><td>Разработка функциональной схемы с возможными натурными или имитационными экспериментами</td><td>Разработка рабочей программы на языке ассемблера. Моделирование и отладка программы на ЭВМ</td></tr>
<tr><td>6</td><td>Анализ и корректировка</td><td>Определение аппаратурных затрат. Анализ технических характеристик. Принятие решения о корректировке предыдущих этапов</td><td>Определение вычислительных затрат. Принятие решения о корректировке предыдущих этапов</td></tr>
<tr><td>7</td><td>Реализация</td><td>Разработка принципиальной схемы, конструкции и технической документации средств сопряжения</td><td>Трансляция рабочей программы с языка ассемблера в коды МП</td></tr>
<tr><td rowspan="2">8</td><td rowspan="2">Воплощение</td><td>Изготовление разработанных средств сопряжения</td><td>Занесение программы в ПЗУ</td></tr>
<tr><td colspan="2">Испытания и окончательная отладка в процессе совместной работы аппаратной и программной частей</td></tr>
</table>

Отличительная особенность **третьего этапа** проектирования состоит в разработке и анализе алгоритмов на уровне арифметических и логических операций, а не на уровне аппаратных операций, которые выделялись бы по реализующим их цифровым узлам. Уже на алгоритмическом этапе первоначально намечается разделение решаемой задачи на аппаратную и программную части: к аппаратной тяготеют участки алгоритма, содержащие циклы, в которых многократно повторяются одни и те же операции и которые «съедают» существенную часть общего времени вычислений.

Четвертый этап – выбор структуры – наиболее критичный этап проектирования. Здесь происходит раздвоение процесса проектирования, здесь обе ветви еще связаны самым тесным образом, здесь разработчику приходится решать наибольшее число творческих вопросов и, наконец, здесь удача или промах разработчика сильнее, чем на всех оставшихся этапах, влияет на качество разработанной системы. Требования приема и предварительной обработки физической информации аппаратной части создаются и уточняются одновременно с требованиями к используемому микропроцессорному комплекту, структура которого определяет характер обмена данными, устройства ввода/вывода, а также возможность аппаратной реализации некоторых функций с помощью входящих в комплект вспомогательных микросхем.

В начале структурного этапа следует разделить решаемую задачу на программную и аппаратную части.

До возникновения микропроцессорной техники и ее применений в измерительных системах подобные вопросы ставились и решались только при построении гибридных вычислительных систем, соединяющих ЭВМ с аналоговой техникой. Однако, несмотря на многолетний опыт развития гибридных вычислительных средств в решении вопросов разделения задачи на цифровую (программную) и аналоговую (аппаратную) части, не было выработано формализованных методик, позволяющих пользователю или разработчику специализированной системы, не задумываясь, разделить задачу. Напротив, разделение не слишком простой задачи было и всегда будет проблемой, которую приходится решать каждый раз вновь в условиях недостаточных априорных данных.

Замена микропроцессорами или микроЭВМ схем с жесткими связями может дать следующие преимущества:

- достижение большей гибкости в изменении программы обработки информации без каких-либо переделок монтажа, печатных плат и т. п. Изменения могут вноситься только в память ЭВМ;

- системы на основе микропроцессоров имеют меньшую стоимость, уменьшается количество печатных плат, межсоединений, упрощается настройка, переделка, снижается стоимость проектирования системы;
- время разработки на основе микропроцессора может быть значительно меньше времени проектирования схемы с жесткими связями, особенно это влияет на возможность перестройки, переделки, модификации характеристик;
- надежность микроЭВМ выше надежности схем с жесткими связями, благодаря разумному сокращению числа корпусов и межсоединений.

Вместе с тем достоинства схем с жесткими соединениями заключаются в следующем:

- если структура устройства адекватна структуре требуемого вычислительного процесса и характеру обрабатываемых данных, то это может дать выигрыш: в скорости обработки информации, в помехоустойчивости, надежности, контролепригодности;
- применение очевидных или простых принципов проектирования устройств с жесткими схемами может предельно упрощать процесс проектирования (т. е. и программирования);
- специализированные блоки обработки информации могут быть в пределах их принципиальных возможностей сделаны многофункциональными или составлять универсальный функциональный базис, на котором можно собирать также и перестраиваемые структуры различного назначения.

Этого несколько «обуженного» запаса универсальности может быть вполне достаточно для различных проблемно-ориентированных аппаратных средств обработки информации.

Одним из первых и главных критериев при выборе микропроцессора является его быстродействие. В справочной литературе приведены сравнительные данные о тактовой частоте различных типов микропроцессоров. Однако эта характеристика (как и время выполнения операций сложения) может служить лишь для первоначальной, грубой оценки быстродействия. Наибольшая точность оценки может быть получена при анализе использования микропроцессора для конкретного применения. Самым распространенным методом предварительной оценки быстродействия является использование бенчмарковских программ.

Бенчмарковская программа – программа решения на анализируемом микропроцессоре такой задачи, которая по составу операций соответствует классу задач предполагаемого применения. Обычная ее длина

100–200 команд. В ее состав обязательно должны входить операции по вводу и выводу информации.

На **пятом этапе** – детализации – аппаратная и программная ветви развиваются уже независимо. Аппаратная часть заключается в разработке функциональных схем устройств, доведенной до уровня, позволяющего выбрать элементную базу, оценить реальные аппаратные затраты и выполнение требований задания по скорости вычислений и другим показателям. Программная часть этого этапа заключается в составлении рабочей программы на языке ассемблера выбранного микропроцессора. За основу берется установленный в ходе предыдущего этапа порядок следования арифметических и обменных операций. На этом этапе осуществляется отладка программы, то есть выявление ошибок и редактирование на основе имеющегося программного обеспечения микропроцессора. Составленная на этом этапе рабочая программа позволяет оценить требуемый объем вычислительных затрат по времени вычислений, объемам памяти ОЗУ, ПЗУ и т. п.

Шестой этап – анализа и корректировки – заключается в выполнении всех необходимых проверок работоспособности устройства и соответствия его требованиям технического задания. Этот этап придает итеративный характер всему процессу проектирования. На этом этапе разработчик получает необходимую информацию для организации поиска новых вариантов технических решений и контроля их приближения к выполнению всех требований задания.

Рассмотрим вкратце, в каких случаях на этапе корректировки приходится возвращаться на тот или иной предыдущий уровень проектирования.

Корректировка функциональной схемы или (и) рабочей программы выполняется при необходимости незначительного сокращения аппаратурных или вычислительных затрат. В аппаратной части используются, как правило, всегда имеющиеся, хотя и ограниченные, возможности сократить оборудование путем минимизации логических схем в укрупненных постановках этой задачи, а также возможности выполнения совмещенных функций на одном и том же оборудовании с меньшим числом микросхем (просмотреть возможность замены дешифраторов или других логических схем на элементах меньшей степени интеграции, например, мультиплексорами или программируемыми логическими матрицами с более высокой степенью интеграции и т. п.). Не исключены и оригинальные инженерные решения. В программной части можно отыскать избыточные операции, операнды или константы, которые могут быть устранены при более полном и рациональном использовании математического обеспечения микропроцессора. При этом может быть достигнута небольшая экономия объема памяти и времени вычислений.

Корректировка структуры позволяет получить более существенные отклонения от исходного варианта:

- при недостаточном быстродействии микропроцессора вынести некоторые вычислительные операции из программной части в аппаратную;
- при избытке быстродействия микропроцессора сократить оборудование аппаратной части за счет программной реализации некоторых ее функций;
- при недопустимых затратах времени на обмен между микропроцессором и внешними устройствами или объема памяти попытаться изменить способ обмена, если не удается, то выбрать другой комплект, более подходящий по быстродействию, набору команд, принципу организации обмена данными или выбрать другую элементную базу для аппаратной части, изменить ее структуру или принципы построения отдельных узлов.

Если корректировка на структурном этапе не приводит к желаемому результату, приходится возвращаться к более ранним этапам.

Корректировка алгоритма производится, если обнаруживается, что на его основе нельзя удовлетворить техническим требованиям из-за наличия принципиальных просчетов или его нереализуемости на современной элементной базе. В последнем случае анализируются все возможности упрощения алгоритма с сохранением его качества: изменение параметров, уменьшение разрядности некоторых чисел, отбрасывание некоторых операций, изменение состава внешних устройств или аппаратной части и принципов их взаимодействия с микропроцессором.

Если же все эти попытки приводят к недопустимому снижению качества алгоритма или не дают требуемых аппаратурных выигрышей, то это означает, что возможности корректировки алгоритма исчерпаны и остается пересматривать методическую основу и постановку задачи создания измерительной системы.

9.2. Современный арсенал средств разработчика автоматизированных средств и систем контроля

Применение в измерительной аппаратуре интегральных микросхем для аналоговой и цифровой обработки сигналов существенно улучшили эксплуатационные характеристики (надежность, габариты, масса, энергопотребление и др.), а также открыли новые богатые возможности усложнения алгоритмов обработки физической информации. До появления микропроцессорной техники компетенция инженера, как правило, ограничивалась задачами создания схем с жесткими связями. Если же

строились специализированные ЭВМ для обработки результатов физических экспериментов, то разработка структуры ЭВМ, ее архитектуры и значительной части ее программного обеспечения относилась к компетенции специалистов вычислительной техники. С появлением микропроцессоров и микроЭВМ начинается их органическое слияние с измерительной аппаратурой и установками, и у инженера возникает потребность более глубокого освоения арсенала программных и аппаратных средств вычислительной техники, необходимого для проектирования автоматизированных систем измерений.

Системы измерения сами по себе нельзя считать идеально подготовленным полем для внедрения микропроцессоров. Вспомним хотя бы один из принципов Неймана, заложенных в основу ЭВМ, – информация обрабатываемая и информация командная представляются словами одинаковой формы. У наших систем командная информация резко отличается от исходной обрабатываемой информации. Последняя, как правило, носит непрерывный характер и возникает как результат преобразования интересующей нас физической величины в более удобный электрический сигнал. Формы представления информации от датчиков могут быть различными: напряжение, ток, сопротивление, параметры импульсов (амплитуда, частота, фаза, длительность импульсов или временной интервал между ними).

Естественно, что измерительная система в общем случае включает в себя также и схемы с жесткими связями для обработки сигналов аналоговыми устройствами, дискретизации, аналого-цифрового преобразования и (или) возможные цифровые функциональные преобразования с целью укрупнения информации перед вводом ее в процессор. Человек общается с системой через внешние устройства – дисплей (на экране которого можно получить «мягкие» копии результатов обработки измерения) и устройства печати и графопостроители (выдающие «жесткие» копии результатов).

С точки зрения системотехники, применяемые новые элементы – микропроцессоры, ОЗУ, ПЗУ, УВВ – являются стандартными микросхемами повышенной степени интеграции, параметры которых так же, как и у счетчиков, регистров, дешифраторов, можно найти в справочниках. Однако для проектирования систем с микропроцессорами необходимо владеть программированием и обладать новым уровнем знания основ организации вычислительных процессов в системах, состоящих из вычислительных средств и дополнительных электронных схем. Поэтому разработчик автоматизированных измерительных систем должен владеть вопросами цифровой и аналоговой схемотехники, программи-

рования и современной элементной базы микропроцессоров и микроконтроллеров.

Несмотря на непрерывное развитие и появление все новых и новых 16- и 32-разрядных микроконтроллеров и микропроцессоров, наибольшая доля мирового микропроцессорного рынка остается за 8-разрядными устройствами. По всем прогнозам аналитических компаний на ближайшие 5 лет лидирующее положение 8-разрядных микроконтроллеров на мировом рынке сохранится.

Среди всех 8-разрядных микроконтроллеров – семейство 8051 является несомненным чемпионом по количеству разновидностей и количеству компаний, выпускающих его модификации. Важную роль в достижении такой высокой популярности семейства 8051 сыграла открытая политика фирмы *Intel*, родоначальницы архитектуры, направленная на широкое распространение лицензий на ядро 8051 среди большого количества ведущих полупроводниковых компаний мира.

В результате, на сегодняшний день существует более 200 модификаций микроконтроллеров семейства 8051, выпускаемых почти 20 компаниями. Эти модификации включают в себя кристаллы с широчайшим спектром периферии: от простых 20-выводных устройств с одним таймером и 1К программной памяти до сложнейших 100-выводных кристаллов с 10-разрядными АЦП, массивами таймеров-счетчиков, аппаратными 16-разрядными умножителями и 64К программной памяти на кристалле. Каждый год появляются все новые варианты представителей этого семейства. Основными направлениями развития являются: увеличение быстродействия (повышение тактовой частоты и переработка архитектуры), снижение напряжения питания и потребления, увеличение объема ОЗУ и FLASH памяти на кристалле с возможностью внутрисхемного программирования, введение в состав периферии микроконтроллера сложных устройств типа системы управления приводами, CAN и USB интерфейсов и т. п.

Основными производителями клонов 51-го семейства в мире являются фирмы *Philips*, *Siemens*, *Intel*, *Atmel*, *Dallas*, *Temic*, *Oki*, *AMD*, *MHS*, *Gold Star*, *Winbond*, *Silicon Systems* и ряд других. В рамках СССР производство микроконтроллера 8051 осуществлялось в Киеве, Воронеже (1816ВЕ31/51, 1830ВЕ31/51), Минске(1834ВЕ31) и Новосибирске (1850ВЕ31).

Микроконтроллеры семейства MCS-51 Фирма *Intel* является родоначальницей архитектуры MCS-51, которая получила свое название от первого представителя этого семейства – микроконтроллера 8051, выпущенного в 1980 году на базе технологии HMOS. Удачный набор периферийных устройств, возможность гибкого выбора внешней или

внутренней программной памяти и приемлемая цена обеспечили этому микроконтроллеру успех на рынке. С точки зрения технологии, микроконтроллер 8051 являлся для своего времени очень сложным изделием – в кристалле было использовано 128 тыс. транзисторов, что в 4 раза превышало количество транзисторов в микропроцессоре 8086.

Основными элементами базовой архитектуры являются:

- 8-разрядное АЛУ на основе аккумуляторной архитектуры;
- 4 банка регистров, по 8 в каждом;
- встроенная память программ 4 кб;
- внутреннее ОЗУ 128 байт;
- булевый процессор;
- 2 шестнадцатиразрядных таймера;
- контроллер последовательного канала (UART);
- контроллер обработки прерываний с двумя уровнями приоритетов;
- четыре 8-разрядных порта ввода/вывода, два из которых используются в качестве шины адреса/данных для доступа к внешней памяти программ и данных;
- встроенный тактовый генератор.

Затем был выпущен микроконтроллер 80C52, который отличался увеличенным объемом памяти программ и данных на кристалле, был введен третий таймер с функциями выборки и сравнения и соответственно расширен контроллер прерывания.

Следующим принципиальным шагом в развитии MCS-51 стал перевод технологии изготовления на CHMOS. Это позволило реализовать режимы Idle и Power Down, позволившие резко снизить энергопотребление кристалла и открывшие дорогу к применению микроконтроллера в энергозависимых приложениях, например в автономных приборах с батарейным питанием.

И последним принципиальным этапом развития этого направления фирмой *Intel* в рамках 8-битной архитектуры стал выпуск микроконтроллеров 8xC51FA/FB/FC, которые для краткости часто обозначаются как 8xC51FX. Главной отличительной особенностью этой группы кристаллов является наличие у них массива программируемых счетчиков (PCA).

В состав PCA входят:

- 16-разрядный таймер-счетчик;
- 5 шестнадцатиразрядных модуля выборки и сравнения, каждый из которых связан со своей линией порта ввода-вывода микроконтроллера.

Таймер-счетчик обслуживает все пять модулей выборки и сравнения, которые могут быть запрограммированы на выполнение одной из следующих функций:

- 16-битная выборка значения таймера по положительному фронту внешнего сигнала;
- 16-битная выборка значения таймера по отрицательному фронту внешнего сигнала;
- 16-битная выборка значения таймера по любому фронту внешнего сигнала;
- 16-битный программный таймер;
- 16-битное устройство скоростного вывода (HSO);
- 8-битный ШИМ.

Выполнение всех перечисленных функций происходит в PCA на аппаратном уровне и не загружает центральный процессор, что позволяет повысить общую пропускную способность системы, повысить точность измерений и отработки сигналов и снизить время реакции микроконтроллера на внешние события, что особенно важно для систем реального времени. Реализованный в 8xC51FX PCA оказался настолько удачным, что архитектура микроконтроллеров FX стала промышленным стандартом де-факто, а сам PCA многократно воспроизводился в различных модификациях микроконтроллеров разных фирм.

Микроконтроллеры семейства MCS-251 Изначально наиболее «узкими» местами архитектуры MCS-51 были 8-разрядное АЛУ на базе аккумулятора и относительно медленное выполнение инструкций (для выполнения самых быстрых инструкций требуется 12 периодов тактовой частоты). Это ограничивало применение микроконтроллеров семейства в приложениях, требующих повышенного быстродействия и сложных вычислений (16- и 32-битовых). Насущным стал вопрос принципиальной модернизации старой архитектуры. Для решения этой задачи была создана совместная группа из специалистов компаний *Intel* и *Philips*, но позднее пути этих двух фирм разошлись. В результате в 1995 г. появилось 2 существенно отличающихся семейства: MCS-251/151 у *Intel* и 51XA – у *Philips*.

Основные характеристики архитектуры MSC-251:

- 24-разрядное линейное адресное пространство, обеспечивающее адресацию до 16M-ячеек памяти;
- регистровая архитектура, допускающая обращение к регистрам как к байтам, словам и двойным словам;
- страничный режим адресации для ускорения выборки инструкций из внешней программной памяти;

- очередь инструкций;
- расширенный набор команд, включающий 16-битовые арифметические и логические инструкции;
- расширенное адресное пространство стека до 64 К;
- выполнение самой быстрой инструкции за 2 такта;
- совместимость на уровне кода с программами для MCS-51.

Таблица 9.2

Микроконтроллеры семейства 8051 фирмы Intel

Обозначение	Частота, МГц	ROM/ EPROM, байт	RAM, байт	Таймеры/ счетчики	Линии ввода/ вывода	Послед. каналы	АЦП, входы/ разр.
8xC51BH (80C31BH)	24	4K	128	2	32	UART	–
8xC52(80C32)	24	8K	256	3	32	UART	–
8xC54	33	16K	256	3	32	UART	–
8xC58	33	32K	256	3	32	UART	–
8xC5x-L	24	8…32K	256	3	32	UART	–
8xC51FA	24	8K	256	3+PCA	32	UART	–
8xC51FB	24	16K	256	3+PCA	32	UART	–
8xC51FC	24	32K	256	3+PCA	32	UART	–
8xL51Fx	16	8…32K	256	3+PCA	32	UART	–
8xC51RA	24	8K	512	3+PCA+ WDT	32	UART	–
8xC51RB	24	16K	512	3+PCA+ WDT	32	UART	–
8xC51RC	24	32K	512	3+PCA+ WDT	32	UART	–
8xC51GB	16	8K	256	3+PCAx2+ WDT	48	UART	8x8
8xC152JA/JC	16,5	8K	256	2	40	UART, GSC	–
80C152JB/JD	16,5	–	256	2	56	UART, GSC	–

Система команд MCS-251 построена на базе двух наборов инструкций. Первый набор является копией системы команд MCS-51, а второй набор состоит из расширенных инструкций, реализующих преимущества архитектуры MSC-251. Перед использованием микроконтроллера его необходимо сконфигурировать, т. е. с помощью программатора «прожечь» конфигурационные байты, определяющие, какой из наборов

инструкций станет активным после включения питания. Если установить набор инструкций MCS-51, то в этом случае MSC-251 будет совместим с MCS-51 на уровне двоичного кода. Такой режим называется Binary Mode. Однако расширенные инструкции в этом режиме также доступны через «форточку» – зарезервированный код инструкции 0A5h. Естественно, длина каждой расширенной инструкции увеличивается в таком случае на 1 байт. Если же изначально установить набор расширенных инструкций, то в этом случае программы, написанные для MCS-51, потребуют перекомпиляции на кросс-средствах для MCS-51, т. к. теперь уже стандартные инструкции будут доступны через ту же «форточку» 0A5h и длина их также увеличится на 1 байт. Такой режим называется Source Mode. Он позволяет с максимальной эффективностью использовать расширенные инструкции и достигнуть наибольшего быстродействия, но требует переработки программного обеспечения.

Таблица 9.3

Микроконтроллеры семейства MCS-251

Обозначение	Макс. частота, МГц	ROM/ EPROM, байт	RAM, байт	Таймеры/ счетчики	Линии ввода/ вывода	Послед. каналы	АЦП, входы/ разр.
8xC251SA	16	8K	1K	3+PCA+ WDT	32	UART	–
8xC251SB	16	16K	1K	3+PCA+ WDT	32	UART	–
8xC251SP	16	8K	512	3+PCA+ WDT	32	UART	–
8xC251SQ	16	16K	512	3+PCA+ WDT	32	UART	–
TSC8xC251G1	16	16K	1K	3+ WDT	32	UART, I2C, SPI	–
TSC8xC251A1	16	24K	1K	2+ WDT	32	UART	4x8 bit
8Xc151SA	16	8K	256	3+PCA+ WDT	32	UART	–
8xC151SB	16	16K	256	3+PCA+ WDT	32	UART	–

Для пользователей, ориентированных на применение микроконтроллеров MCS-251 в качестве механической замены MCS-51, фирма *Intel* выпускает микроконтроллеры MCS-251 с уже запрограммированными битами конфигурации в состоянии Binary Mode. Такие микроконтроллеры получили индекс MCS-151.

В настоящее время *Intel*, устремленная на рынок Pentium-процессоров, сворачивает производство кристаллов MCS-51.

Микроконтроллеры фирмы PHILIPS. Фирму Philips назвать чемпионом по количеству выпускаемых ею модификаций семейства 8051 – их более 100. В состав семейства 8051 от *Philips* входят микроконтроллеры в корпусах от 24 до 80 выводов, тактовыми частотами до 40 МГц и напряжением питания от 1,8 В. Во всех микроконтроллерах *Philips* используется стандартное ядро MCS-51, поэтому все временные и функциональные характеристики полностью соответствуют характеристикам микроконтроллеров фирмы *Intel*. Фирма *Philips* значительные усилия направила на интегрирование широкого спектра периферийных устройств на базе ядра 8051.

Основные элементы периферии *Philips*:

- АЦП с точностью преобразования 10 разрядов;
- широтно-импульсные модуляторы;
- массивы программируемых счетчиков-таймеров;
- интерфейсы I2C, CAN;
- интерфейсы с процессорными шинами;
- EEPROM и FLASH на кристалле;
- специализированная периферия для телевизионной, видео- и аудиотехники.

Новыми возможностями кристаллов фирмы являются:

- максимальная тактовая частота кристаллов увеличена до 33 МГц;
- расширен диапазон напряжения питания от 2,7 до 5,5 В;
- количество аппаратных уровней прерываний увеличено до 4-х;
- во все кристаллы введена функция программируемого выхода синхронизации;
- UART заменен на улучшенный;
- добавлена функция снижения электромагнитных помех;
- добавлен второй DPTR;
- потребление энергии для питания микроконтроллера снижено на 50 %. В сочетании с 3-вольтовым питанием это может дать экономию до 75 %, по сравнению с предыдущими образцами;
- снижена цена на 30 %.

Фактически такие новые возможности дают второе рождение старым кристаллам. Проблема для разработчика, однако, состоит в том, что маркировка микроконтроллеров после модернизации не изменилась, из-за чего возможна путаница между старыми и новыми модификациями. Кроме того, фирма *Philips* выпустила группу микроконтроллеров, названную RX+. По сути, это дальнейшее развитие группы FX, в которой расширен объем внутреннего ОЗУ (512 байт, 1 кб) и программной памя-

ти (до 64 К). Группа RX+ обладает также всеми возможностями, предоставляемыми технологией.

В 1997 годы фирма *Philips* взяла курс на развитие FLASH-технологии в производстве микроконтроллеров. Отчасти это вызвано высокими технологическими возможностями *Philips*, отчасти успехами конкурентов, в первую очередь *Atmel*. Несмотря на то, что FLASH-память дороже в производстве, чем EPROM, в конечном итоге для фирмы дешевле будет поддерживать единый технологический процесс.

Микроконтроллеры фирмы *Atmel*. Основанная в 1984 году, фирма *Atmel* Corp. (США) – прогрессивная компания, выпускающая сложные изделия современной микроэлектроники; один из признанных мировых лидеров в производстве широкого спектра устройств энергонезависимой памяти высокого быстродействия и минимального удельного энергопотребления, микроконтроллеров общего назначения и микросхем программируемой логики от простейших устройств PAL и GAL до микросхем СБИС CPLD и FPGA. Практически все базовые кристаллы промышленного стандарта MCS51 фирмы *Intel* успешно заменены прямыми аналогами семейства AT89 фирмы *Atmel*. Эти скоростные, полностью статические 8-разрядные КМОП-микроконтроллеры с многократно модифицируемой Flash-памятью программ, низким энергопотреблением и широким диапазоном допустимых напряжений питания, аппаратно и программно совместимы с соответствующими микроконтроллерами *Intel* и пользуются популярностью у разработчиков и производителей электронной аппаратуры.

Однако заслугой фирмы является создание нового семейства высокопроизводительных 8-разрядных **RISC** (**R**educed **I**nstruction **S**et **C**omputers) микроконтроллеров общего назначения, объединенных общей маркой ***AVR***.

Замысел создания ***AVR*** родился в исследовательском центре *Atmel* в Норвегии. Группа разработчиков (инициалы некоторых из них сформировали марку "AVR": **A**lf Bogen/**V**ergard Wollan/**R**isc architecture) предложила ряд идей, которые легли в основу концепции ***AVR***-микроконтроллеров:

1. Использовать новейшую, наиболее скоростную и экономичную КМОП-технологию фирмы *Atmel* в сочетании с RISC-архитектурой для разработки и производства быстрых 8-разрядных микроконтроллеров, сравнимых с 16-разрядными микропроцессорами и микроконтроллерами по производительности и превосходящих микросхемы стандартной КМОП-логики по скорости. Ожидаемая производительность – до 20 MIPS на частоте 20 МГц, что всего на 30 % меньше, чем у *Intel*

KU80386EXTC-25 при операциях типа *регистр–регистр*. Время выполнения короткой команды на такой частоте составляет 50 нс.

2. Разрабатывать архитектуру и систему команд AVR в теснейшем согласии с принципами языка Си так, чтобы аппаратная часть нового микроконтроллера и его система команд были неотъемлемыми частями одного целого и использовались с максимальным КПД. В 1990-е годы языки программирования высокого уровня стали стандартным инструментом при создании программного обеспечения для встраиваемых микроконтроллеров. Существенно сокращается время разработки проектов и, соответственно, снижается их стоимость, а также облегчается создание универсальных средств поддержки разработок. Система команд AVR разрабатывалась при непосредственном участии экспертов по языку Си и учитывает все основные особенности стандарта ANSI C.

3. Функционально расширить микроконтроллер возможностью программирования в системе (ISP) путем объединения Flash-технологии фирмы *Atmel* со стандартным скоростным последовательным интерфейсом (SPI). Это позволяет многократно модифицировать программу не только с помощью обычного программатора, но и непосредственно в системе, в конечном устройстве пользователя. При этом не требуется вводить никаких дополнительных аппаратных узлов и вспомогательных источников питания.

Результатом явилось появление нового, очень дешевого, скоростного, легкого в освоении и использовании семейства AT90S 8-разрядных микроконтроллеров марки ***AVR***. Они представляют собой мощный инструмент, базу для создания современных высокопроизводительных и экономичных контроллеров многоцелевого назначения. Так, например, ***AVR*** используются в изделиях класса Smart Card для персональных компьютеров, в спутниковых навигационных системах для определения местоположения автомобилей на трассе, в миниатюрных автомобильных пультах дистанционного управления, в сетевых картах и на материнских платах компьютеров, в сотовых телефонах нового поколения и т. д. В табл. 9.4 представлены существующие кристаллы ***AVR*** микроконтроллеров.

Все ***AVR*** имеют Flash-память программ ROM объемом 1...8 К, которая может быть загружена как с помощью обычного программатора, так и посредством SPI интерфейса, и внутреннюю оперативную память SRAM (кроме AT90S1200) объемом 128...512 байт. Число циклов перезаписи ROM – не менее 1000. Два программируемых бита секретности позволяют защитить память программ от несанкционированного считывания. Все ***AVR*** имеют также блок энергонезависимой электрически стираемой памяти данных EEPROM объемом 64...512 байт. Этот тип

памяти, доступный программе микроконтроллера непосредственно в ходе ее выполнения, удобен для хранения промежуточных данных, констант, таблиц перекодировок, калибровочных коэффициентов и т. п. EEPROM может быть загружена извне как через SPI интерфейс, так и с помощью обычного программатора.

Таблица 9.4

AVR-микроконтроллеры фирмы ATMEL

AT90S	**1200**	**2313**	**4414**	**8515**	**2323**	**4433**
Диапазон напр. пит., В	2,7–6,0					
Тактовая частота, МГц *)	0–16		0–20		0–16	0–20
Кол. линий ввода/вывода	15		32		5	18
Количество инструкций	89	120				
Объем Flash ROM, байт	1K	2K	4K	8K	2K	4K
Объем EEPROM, байт	64	128	256	512	128	256
Объем внутр. SRAM, байт	–	128	256	512	128	128
Объем внешн. SRAM, байт	–	–	64K	64K	–	–
Объем регистрового файла, байт	32					
Кол. таймеров/счетчиков	1	2	2	2	1	2
ШИМ: каналов/разрядн.	–	1/8–10	2/8–10	2/8–10	–	2/8–10
Количество модулей захвата/сравнения	–	1	2	2	–	2
Аналоговый компаратор	+	+	+	+	–	+
SPI (загрузка ROM и EEPROM)	+	+	+	+	+	+
SPI-интерфейс (Master/Slave port)	–	–	+	+	–	+
Сторожевой таймер	+	+	+	+	+	+
Асинхронный последовательный порт	–	+	+	+	–	+
Аналого-цифровой преобразователь	–	–	–	–	–	+
Количество битов защиты	2					
Число режимов энергосбережения	2					
Число источников прерывания: внутр./внешн.	2/1	8/2	10/2	10/2	2/1	11/2

Микроконтроллеры фирмы Microchip. Наиболее «узкими» местами архитектуры MCS-51 являются медленное АЛУ на базе аккумулятора и долгое время выполнения инструкций (12 машинных тактов). Кроме этого, стандартный микроконтроллер MCS-51 позволял себе даже та-

кую роскошь, как холостые командные циклы. Были изобретены различные способы повышения производительности, но за ускорение приходилось платить повышенными энергопотреблением и стоимостью, что совершенно лишало семейство MCS-51 всех преимуществ в низко стоимостных и критичных к потреблению применениях.

Еще в 1975 году фирма GI разработала периферийный контроллер (Peripheral Interface Controller или PIC), предназначенный для поддержки ввода/вывода 16-разрядного процессора. В нем не требовалась сложная обработка, поэтому набор его команд был сильно ограничен, но почти все команды выполнялись в нем за один машинный цикл. Этот контроллер с RISC-архитектурой и стал прообразом современной архитектуры микроконтроллеров PIC, выпускаемых с конца 80-х годов дочерней GI-компанией Arizona Microchip Technology Ltd.

Первые промышленные микроконтроллеры семейства PIC16C5X были простыми, но быстрыми. Основной представитель семейства PIC16C54A-20 выпускался в 18-выводном корпусе, имел память программ объемом 512 байт и память данных 25 байт, всего 33 команды со временем исполнения инструкции 200 нс и одноуровневым конвейером команд (тактовая частота 20 МГц), причем потреблял он при этом всего 10 ма. На частоте 1 МГц напряжение питание можно понизить до 2 В, с током потребления ниже 1 ма. Вкупе с низкой стоимостью, в среднем, меньше $ 1 в США, все эти качества сделали PIC16C54A и его вариации весьма популярными. В итоге, новое семейство PIC-контроллеров несколько потеснило со своих позиций микроконтроллеры MC68C05 компании MOTOROLA и ряда других производителей, обосновавшихся в нише низкостоимостных применений.

Результатом дальнейших усилий в области миниатюризации и удешевления стало появление таких необычных контроллеров, как 12C508 и 12C509, имевших всего по 512 байт и 1 кб памяти программ, соответственно, и всего по восемь выводов, шесть из которых являются портами ввода/вывода. Не так давно был анонсирован очередной такой «малыш», но уже со встроенным АЦП .

В настоящее время MICROCHIP выпускает три основных серии PIC-контроллеров:

1) PIC16C5X – базовое семейство с 12-разрядными командами;
2) PIC16C6X/7X/8X – расширенное семейство средней производительности с 14-разрядными командами;
3) PIC17CXX – высокопроизводительные микроконтроллеры с 16-разрядными командами.

Большинство PIC-контроллеров сделано по OTP-технологии (однократно программируемые микросхемы) – тяжелое наследие 80-х годов.

Для целей отладки предлагается использование микросхем с ультрафиолетовым стиранием и довольно высокой стоимостью. Все PIC-контроллеры оборудованы внутренними схемами сброса по питанию и сторожевыми таймерами, многие модели имеют возможность внутрисхемного программирования.

Все последовательно произведенные серии PIC-контроллеров являются логическими продолжениями единого базового ядра и, как заявляют представители компании MICROCHIP, перекрывают весь диапазон применений 8-разрядных микроконтроллеров. Делая упор на низкую стоимость своих изделий, MICROCHIP пришлось отказаться от универсальных микроконтроллеров с разнообразной и развитой периферией (одно из исключений – PIC–14000 , правда он является полузаказной микросхемой) и распределить периферийные устройства по всем выпускаемым семействам. Иными словами, если разработчику в контроллере требуются компараторы, то как раз для этого выпускается серия PIC16CX , если необходимо АЦП – используется серия PIC16C7X и т. д. Но то, что является благом для инженера в Америке, оборачивается проблемой для российского разработчика. К сожалению, наши поставщики микросхем не в состоянии держать на складе всю номенклатуру PIC-контроллеров (это более двухсот наименований) и ограничиваются самыми распространенными изделиями.

Глава 10
ПРИМЕР РАЗРАБОТКИ ОММЕТРА НА ОСНОВЕ МИКРОКОНТРОЛЛЕРА PIC16F84A

10.1. Проектирование аппаратной части прибора

Использование микроконтроллеров позволяет проектировать очень простые, дешевые, но достаточно точные измерительные приборы. В качестве примера рассмотрим проект реализации омметра на базе микроконтроллера PIC16F84A, который измеряет сопротивления от 1 до 150 кОм, и в связи с наличием свободных портов данный диапазон может быть существенно увеличен. Себестоимость такого прибора составляет около 150 рублей, но при использовании однократно программируемых микроконтроллеров может быть снижена на 30–40 %. Погрешность измерения не превышает 2 %.

Прибор работает на основе измерения времени зарядки конденсатора через измеряемый резистор и сравнения со временем зарядки через эталонный резистор. Неизвестное сопротивление Rm вычисляется в соответствии с формулой $Rm = \frac{Tm}{Tc} \cdot Rc$, где Tm и Tc – временные интервалы заряда эталонного конденсатора через измеряемое и эталонное сопротивление, соответственно, Rc – эталонное сопротивление. Так как процессы заряда одной и той же эталонной емкости через измеряемое и эталонное сопротивления происходят последовательно и в короткий промежуток времени, то точность измерения не зависит от изменения параметров измерительной схемы и параметров конденсатора и зависит только от точности образцового сопротивления и точности измерения временных интервалов. Образцовые сопротивления с точностью 0,5 % и точнее найти просто и стоят они недорого. Используя микроконтроллер, измерить временной интервал можно с высокой точностью (практически с любой наперед заданной). Таким образом получить точность измерения порядка 1 % довольно просто.

Принципиальная схема прибора представлена на рис. 10.1, а перечень используемых элементов приведен в табл. 10.1. На рис. 10.2–10.4 приведены схемные обозначения микросхем, используемых в проекте.

В соответствии с программой, приведенной далее, каждая часть схемы выполняет определенную функцию.

Кварцевый резонатор (ZQ1на 4 МГц) и конденсаторы C_1, C_2 (по 15 пФ) обеспечивают работу встроенного тактового генератора.

Резисторы R2 (9,4 кОм) и R3 (100 кОм), подключенные, соответственно, к выводам 7 и 6 порта B, являются образцовыми резисторами для первого и второго измерительных поддиапазонов, что увеличивает точность измерения в широком диапазоне сопротивлений.

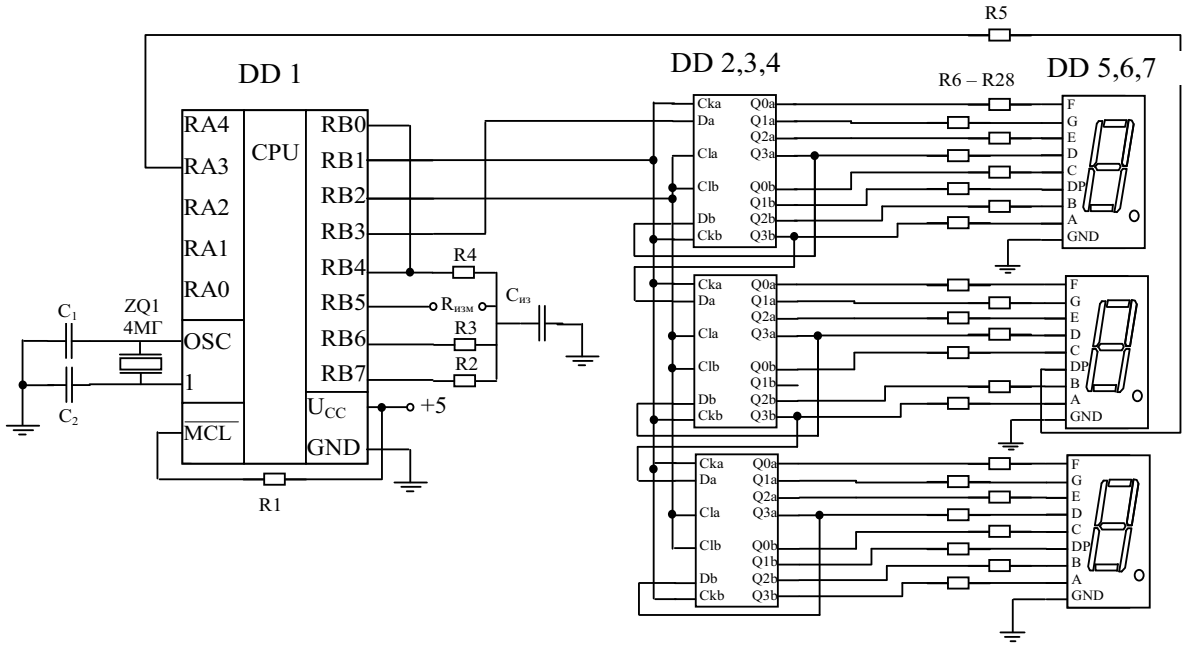


Рис. 10.1. Принципиальная схема прибора

Резистор R4 (510 Ом) предотвращает ток короткого замыкания при разрядке конденсатора и снятии данных с конденсатора.

Таблица 10.1

Перечень элементов, представленных на схеме

DD1	Микроконтроллер PIC16F84
DD 2, 3, 4	Два 4-разрядных регистра сдвига561ИР2 (CD4015A)
DD 5, 6, 7	Семисегментный цифробуквенный индикатор SC04-11GWA
ZQ1	Кварцевый резонатор на 4 МГц
C_1, C_2	Конденсатор 15 пФ
$C_{изм}$	Конденсатор, накапливающий заряд в процессе измерения, 68 пФ
$R_{изм}$	Клеммы для подключения измеряемого сопротивления
R1	Резистор 100 кОм
R2	Резистор 9,4 кОм
R3	Резистор 100 кОм
R4	Резистор 510 Ом
R5, R6–R28	Резистор 510 Ом

Конденсатор $C_{изм}$ (68 пФ) заряжается в процессе измерения сопротивления.

Три микросхемы 4-разрядных регистров сдвига (561ИР2) позволяют выводить полученный в результате измерения результат на цифробуквенные семисегментные индикаторы с использованием только трех выводов микроконтроллера RB3, RB2, RB1 (для передачи данных, сброса и синхронизации, соответственно).

Цифробуквенные семисегментные индикаторы предназначены для вывода цифровой информации в шестнадцатеричном коде от 0 до F или в десятичном коде. Принцип действия основан на свечении расположенного в определенном месте светодиода (сегмента) при подаче напряжения высокого уровня на соответствующий вход. Обозначение сегментов и номера соответствующих выводов представлены на рис. 10.2 – 10.4.

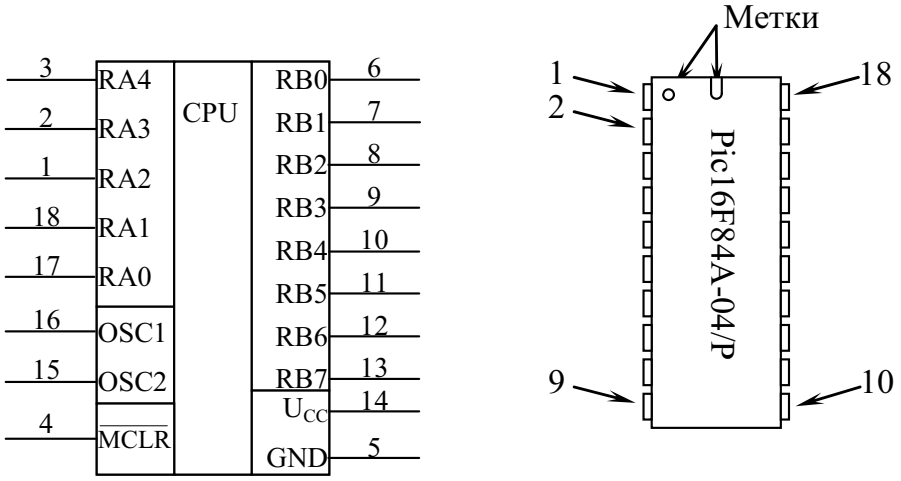


Рис. 10.2. Микроконтроллер Pic16F84:
общий вид, назначение и нумерация выводов
(RA0 – RA4 – порт A; RB0 – RB7 – порт B; Ucc – питание (+5B); GND – земля;
OSC1, 2 – ножки синхронизации тактового генератора)

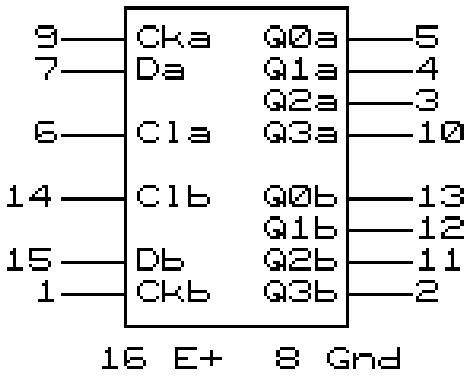


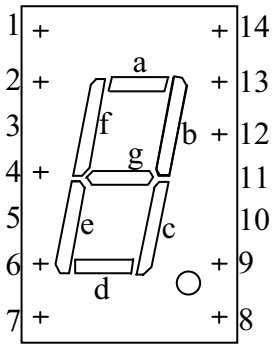
Рис. 10.3. 4-разрядный регистр сдвига 561ИР2 CD4015A:
Ck – тактовый вход; Cl – сброс в 0 (активный потенциал H);
D – последовательный вход данных; Q – параллельные выходы

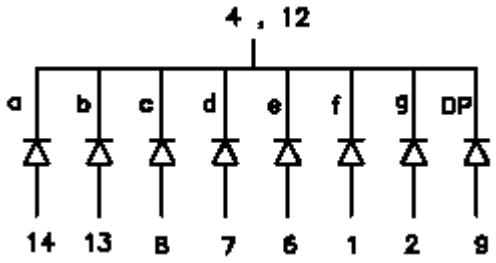
Таблица 10.2

Таблица истинности регистра сдвига 561ИР2

	Входы			Выходы			
n	Ck	D	Cl	Q0	Q1	Q2	Q3
1	_/~	D1	L	D1	X	X	X
2	_/~	D2	L	D2	D1	X	X
3	_/~	D3	L	D3	D2	D1	X
4	_/~	D4	L	D4	D3	D2	D1
-	⌥	X	L	без изменений			
-	X	X	H	L	L	L	L

Примечания: Dn – потенциал либо H, либо L; n – номер тактового импульса





Выводов 3,5,10,11 нет

Рис. 10.4. Семисегментный цифробуквенный индикатор SC04-11GWA

10.2. Разработка программы измерений

Блок-схема программы приведена на рис. 10.5. В программе использованы следующие переменные:

- X1, X2, X3, X4 – переменные, применяются в процедуре disch (X1, X2) и Outled (X3, X4);
- Tm – переменная, в которую помещается значение времени измерения «измеряемого» резистора;
- Tc – переменная, в которую помещается значение времени измерения «эталонного» резистора;
- tmpl, tmph;
- cnt1;
- resl, resh;
- d0, d1, d2;
- ETALON; переменная для значения эталонного резистора;
- N ; счетчик (таймер);
- PES; переменная.

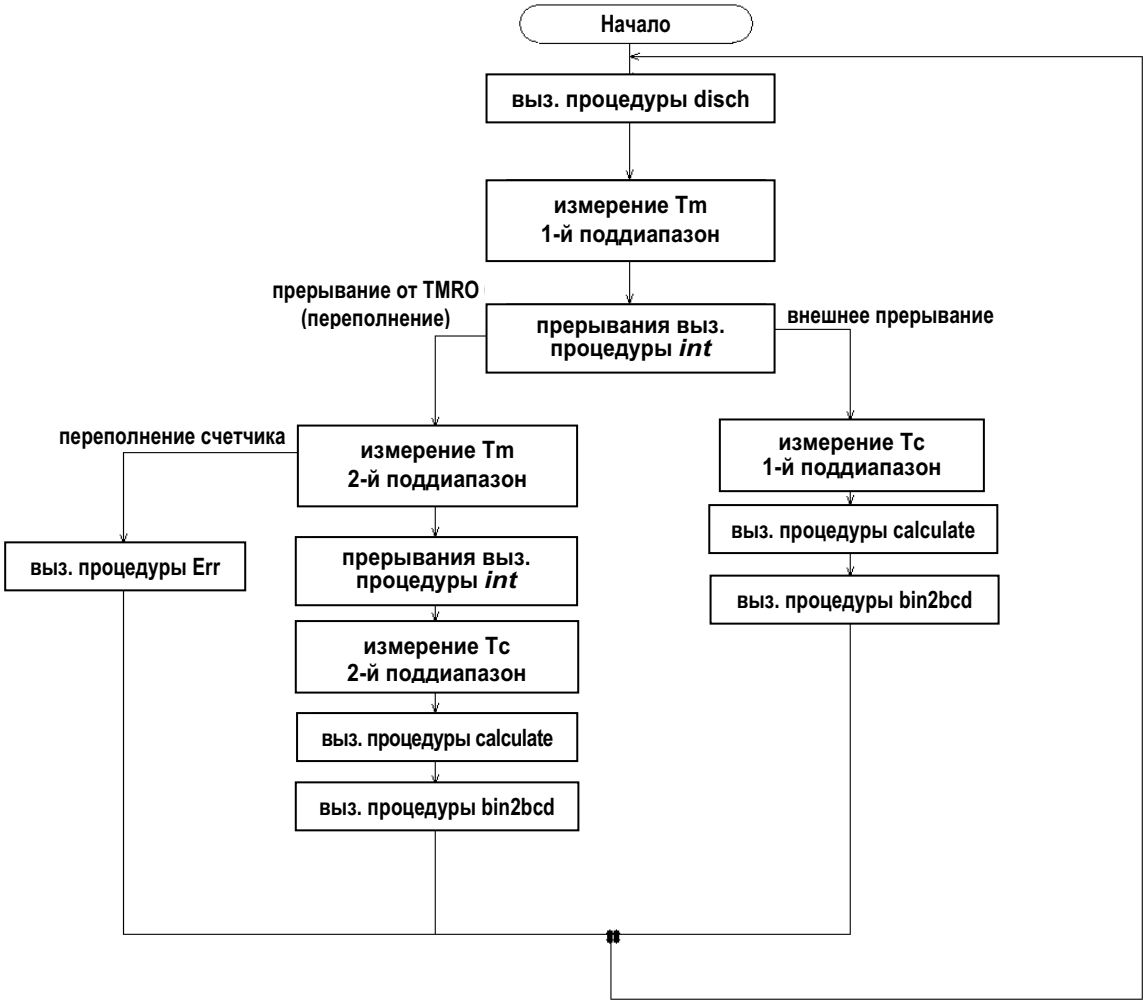


Рис. 10.5. Блок-схема программы

Обработка прерываний INT

При обработке прерывания происходит запись значения из регистра TMR0 в аккумулятор. Затем проверяется причина прерывания. Если прерывание произошло по причине переполнения регистра TMR0, то происходит переключение на другой поддиапазон, т. е. переключаемся на другой эталонный резистор и проводим измерения заново.

Блок-схема процедуры представлена на рис. 10.6.

Текст процедуры:

```
int
    movf TMR0, w
    BTFSS INTCON, 1  ; проверяем внешнее прерывание
    goto int_no      ; если нет, то переходим на дру-
                       гой уровень
    goto   int_yes   ; если да, то заряжаем конденса-
                       тор через эталонный резистор
```

```
int_no
    CALL disch
    goto charging_m100

int_yes
    BTFSS INTCON, 5 ;  смотрим, разрешено ли прерыва-
                       ние по переполнению таймера
    goto c100       ;  если нет, то переходим на метку
                       c100
    goto c10»       ;  если да, то переходим на метку
                       c10
```

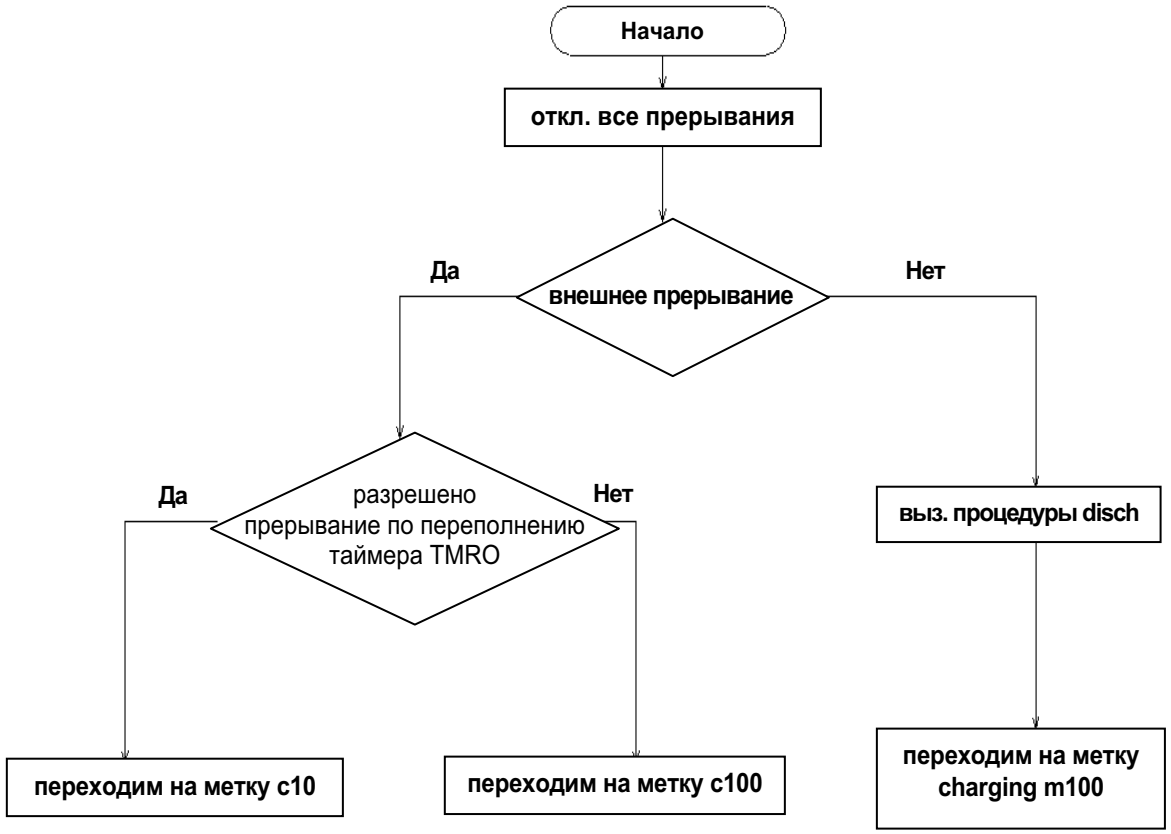


*Рис. 10.6. Блок-схема процедуры обработки прерывания **int***

Начало выполнения программы (метка reset)

Очистка переменных и регистров, разряд конденсатора. Затем подготавливаем порт B, для того чтобы начать заряжать конденсатор через измеряемый резистор, и ждем прерывание либо по переполнению, если резистор имеет большое сопротивление, либо внешнее прерывание.

В регистре OPTION_REG устанавливаем: предделитель 1:8, прерывание по переднему фронту, подтягивающие резисторы отключены, приращение по переднему фронту сигнала, предделитель включен перед TMR0.

С помощью регистра INTCON разрешаем внешние прерывания и прерывания по переполнению TMR0.

Блок-схема представлена на рис. 10.7.

Рис. 10.7. Блок-схема

Текст процедуры:

```
    ORG       0x020
reset
    clrf      PORTA
    clrf      TMR0
    clrf      PES
```

```
char_m10
    ; зарядка конденсатора через Rm
    CALL     disch        ; разряжаем конденсатор
    clrf     TMR0
    ; зарядка через измеряемый резистор не более 10 кОм
    movlw    b'11000010'  ; настраиваем таймер
    banksel  OPTION_REG
    movwf    OPTION_REG
    banksel  PORTB        ; переключаемся на банк,
    ; в котором находится регистр PORTB
    movlw    b'10110000'  ; разрешаем прерывания
    movwf    INTCON
    movlw    b'11011111'  ; настраиваем пятый вывод
    ; порта B на выход
    banksel  TRISB
    movwf    TRISB
    banksel  PORTB
    clrf     TMR0
    bsf      PORTB, 5     ; устанавливаем единицу на
    ; пятом выводе порта B
char_m10_L
    goto     char_m10_L   ; бесконечный цикл, ждем
    ; прерывания
```

Зарядка через эталонный резистор первого поддиапазона, расчет и вывод значения сопротивления на экран

Время Tm уже измерено и остается измерить время Tc, рассчитать сопротивление по формуле и вывести на индикаторы.

Заряд конденсатора через эталонный резистор происходит аналогично тому, как и при заряде через измеряемый, только используется 7 вывод порта B (процедура char_c10).

Переменная PES необходима, чтобы знать произошла ли процедура зарядки конденсаторы через эталонный резистор, т. е. провели ли измерение времени Tc. Если да, то произвести вычисление сопротивления при помощи процедуры calculate, а затем вывести на индикаторы при помощи процедуры bin2bcd и outled (эти процедуры будут описаны ниже). Но перед этим необходимо зажечь точку на среднем индикаторе, чтобы знать, что измерения были проведены на первом поддиапазоне.

Блок-схема представлена на рис. 10.8.

Текст программы:

```
c10
    movwf    Tc          ; помещаем значение TMR0 в
    ; переменную Tc
    CALL     disch       ; разряжаем конденсатор
    BTFSS    PES, 0      ; проверяем, есть ли
    ; единица в нулевом разряде переменной PES
    goto     char_c10    ; если нет, то производим
    ; зарядку конденсатора через Rc
    movlw    .94         ;записываем значение Rc*10
    ; в переменную ETALON
    movwf    ETALON
    movlw    b'11110111'
    banksel  TRISA
    movwf    TRISA
    banksel  PORTB
    bsf      PORTA, 3    ;зажигаем точку на среднем
    ; индикаторе
    CALL     calculate
    CALL     bin2bcd
    goto     vse
char_c10
    ;заряжаем конденсатор через эталонный резистор
    movf     Tc, w
    movwf    Tm
    bsf      PES, 0
    clrf     TMR0
    ; зарядка через эталонный резистор 10 кОм
    movlw    b'11000010'
    banksel  OPTION_REG
    movwf    OPTION_REG
    banksel  PORTB
    movlw    b'10110000'
    movwf    INTCON
    movlw    b'01111111'
    banksel  TRISB
    movwf    TRISB
    banksel  PORTB
    clrf     TMR0
    bsf      PORTB, 7
char_c10_L
    goto     char_c10_L
```

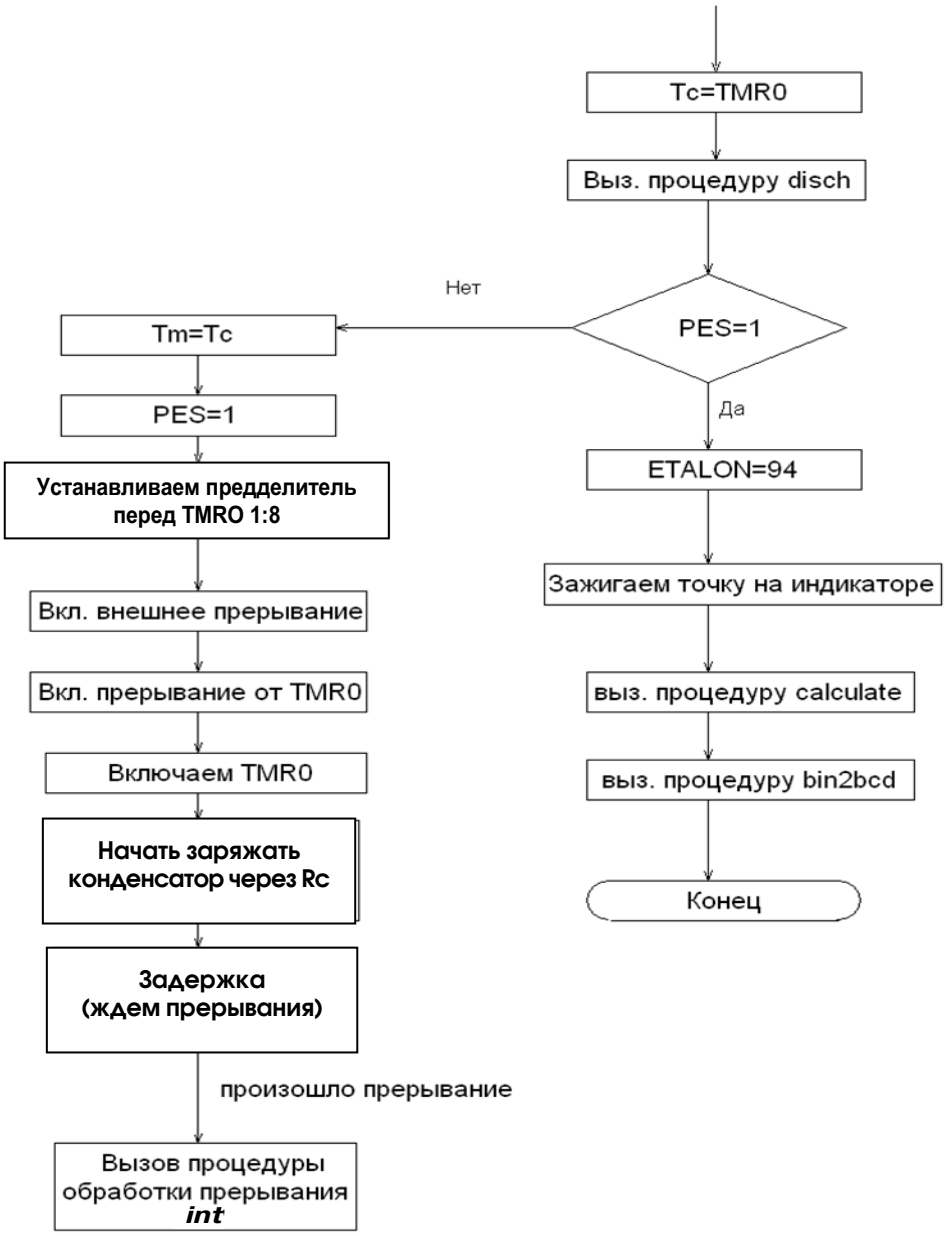


Рис. 10.8. Блок-схема процедуры измерения

Измерение во втором поддиапазоне

Измерение Tm

Сначала измеряем Tm. Заряжаем конденсатор через измеряемый резистор только уже в другом поддиапазоне, т. е. увеличиваем время измерения. Для этого устанавливаем коэффициент деления предделителя для TMR0 1:2, а TMR0 используем как делитель (записываем в TMR0 расчетное значение 230). Таймером теперь будет переменная Tm, т. е. приращение времени будет записываться непосредственно в переменную Tm. Для этого мы должны запретить прерывание по переполнению TMR0.

Блок-схема представлена на рис. 10.9.

```
charging_m100
    clrf      Tm
    clrf      TMR0
    ; зарядка через измеряемый резистор от 10 до 100 кОм
    movlw     b'11000000'  ; устанавливаем делитель
    ; предделителя 1:2
    banksel   OPTION_REG
    movwf     OPTION_REG
    banksel   PORTB
    movlw     b'10010000'  ;запрещаем прерывания по
    ; переполнению
    movwf INTCON
    movlw     b'11011111'  ; настраиваем порт B
    banksel   TRISB
    movwf     TRISB
    banksel   PORTB
    bsf       PORTB, 5
cha_m100_L1
    movlw     .230         ; записываем в TMR0
    ; расчетное значение 230
    movwf     TMR0
cha_m100_L
;увеличиваем значение Tm на 1 по переполнению TMR0,
; пока не произойдет прерывание, если прерывание не
; произошло, то Tm переполняется. Если Tm
; переполняется – выводим на экран надпись Err.
    BTFSS     INTCON,  2
    goto      cha_m100_L
    bcf       INTCON, 2
    movf      Tm, W
    addlw     .1
    movwf     Tm
    btfss     STATUS, C
    goto      cha_m100_L1
    bcf       INTCON, 4
    goto      Err
```

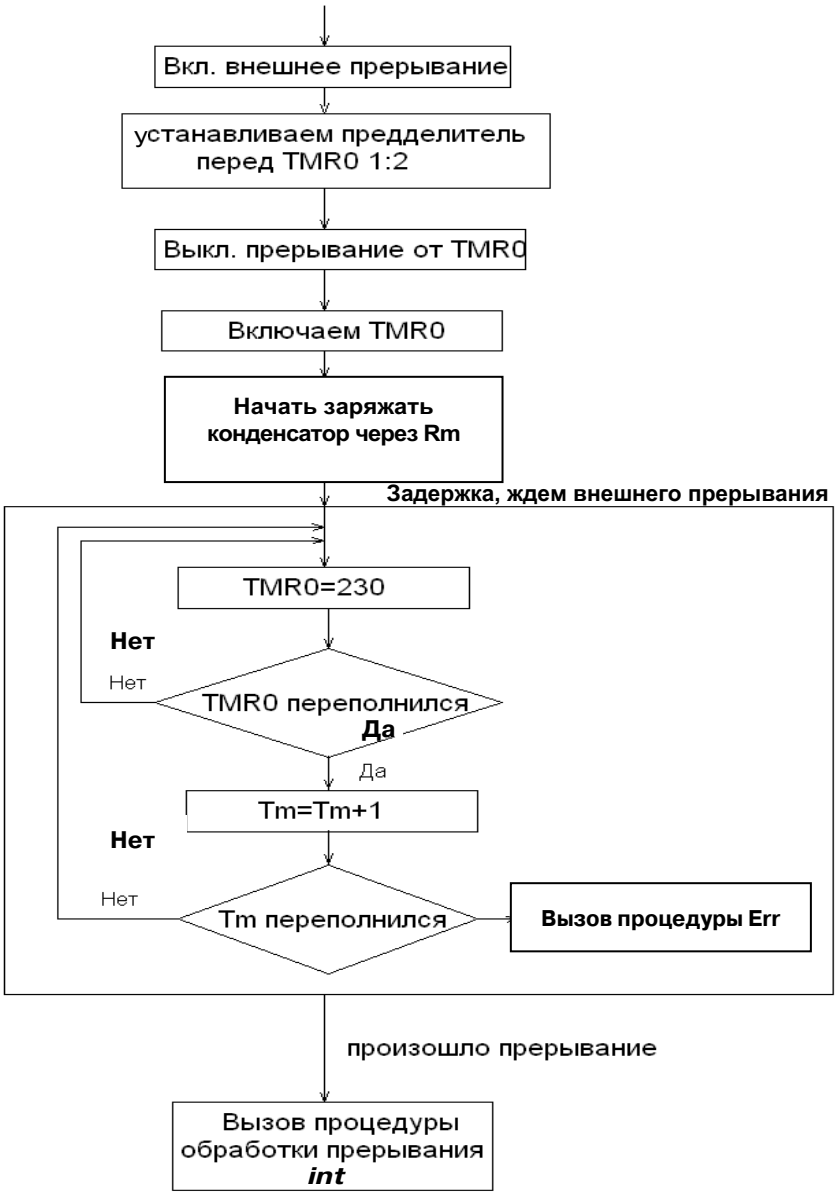


Рис. 10.9

Зарядка через эталонный резистор второго поддиапазона, расчет и вывод значения сопротивления на экран

Разряжаем конденсатор. Производим измерение Tc, если оно еще не производилось. Если измерение Tc уже произошло, то производим расчет Rm и выводим его на индикаторы, при этом точку на среднем индикаторе не зажигаем, т. к. измерения проводились в другом диапазоне.

Зарядка конденсатора через Rc происходит аналогичным образом, что и при Rm, только через 6-й вывод порта B.

Блок-схема представлена на рис. 10.8.

```
c100
    CALL      disch
    BTFSS     PES, 1
    goto      charging_c100
    movlw     .100
    movwf     ETALON
    CALL      calculate
    CALL      bin2bcd
    goto      vse
charging_c100
    clrf      Tc
    bsf       PES, 1
    ; зарядка через эталонный резистор 100 кОм
    movlw     b'11000000'
    banksel   OPTION_REG
    movwf     OPTION_REG
    banksel   PORTB
    movlw     b'10010000'
    movwf     INTCON
    movlw     b'10111111'
    banksel   TRISB
    movwf     TRISB
    banksel   PORTB
    bsf       PORTB, 6
cha_c100_L1
    movlw     .230
    movwf     TMR0
cha_c100_L
    BTFSS     INTCON, 2
    goto      cha_c100_L
    bcf       INTCON, 2
    movf      Tc, W
    addlw     .1
    movwf     Tc
    btfss     STATUS, C
    goto      cha_c100_L1
    bcf       INTCON, 4
    goto      Err
```

Процедура disch

Процедура disch разряжает конденсатор. Чтобы разрядить конденсатор, необходимо отключить все прерывания и настроить 4-й выход порта B на вход и поставить небольшую задержку.

Блок-схема представлена на рис. 10.10.

```
disch
    bcf         INTCON, 7
    movlw       b'11101111'
    banksel     TRISB
    movwf       TRISB
    banksel     PORTB
    bcf         PORTB, 4
disch_delay  ; задержка
    movlw       h'3f'
    movwf       X1
    clrf        X2
disch_delay_1
    decfsz      X2, F
    goto        disch_delay_1
    decfsz      X1, F
    goto        disch_delay_1
    movlw       b'11111111'
    banksel     TRISB
    movwf       TRISB
    banksel     PORTB
    return
```

Рис. 10.10

Процедура Outled

Процедура Outled выводит число на семисегментные индикатор через сдвиговые регистры, т. е. используя всего два-три выхода микроконтроллера (что экономит количество используемых выводов микро-

контроллера). Чтобы вывести результат на индикаторы, необходимо составить таблицу, в зависимости от того, как подключены сдвиговые регистры к семисегментным индикаторам.

Записываем двоичный код, который соответствует числу, в переменную X3, при помощи процедуры DES. Затем мы проверяем 7-й бит переменной X3 и выводим 1 или 0 в зависимости от того, что там находится. Потом осуществляем циклический сдвиг влево командой RLF и опять проверяем 7-й бит, и так восемь раз.

Таблица 10.3

Двоичный код	Число
b'11011101	**0**
b'01010000'	**1**
b'11001110'	**2**
b'11011010'	**3**
b'01010011'	**4**
b'10011011'	**5**
b'10011111'	**6**
b'11010000'	**7**
b'11011111'	**8**
b'11011011'	**9**
b'00000000'	**-**
b'10001111'	**E**
b'10000101'	**r**

```
outled
    CALL        DES
    movwf       X3
    movlw       h'8'
    movwf       X4
    movlw       b'11110001'
    banksel     TRISB
    movwf       TRISB
    banksel     PORTB
    bcf         PORTB, 2
    bcf         PORTB, 1
outled_out
    BTFSS       X3, 7
    GOTO        outled_no
    GOTO        outled_yes
```

```
outled_yes
    BSF         PORTB, 3
    BSF         PORTB, 1
    BCF         PORTB, 1
    RLF         X3, f
    DECFSZ      X4, f
    goto        outled_out
    goto        outled_kon
outled_no
    BCF         PORTB, 3
    BSF         PORTB, 1
    BCF         PORTB, 1
    RLF         X3, f
    DECFSZ      X4, f
    goto        outled_out
    goto        outled_kon
outled_kon
    RLF         X3, f
    return
DES
    addwf       PCL, f
    retlw       b'11011101'
    retlw       b'01010000'
    retlw       b'11001110'
    retlw       b'11011010'
    retlw       b'01010011'
    retlw       b'10011011'
    retlw       b'10011111'
    retlw       b'11010000'
    retlw       b'11011111'
    retlw       b'11011011'
    retlw       b'00000000'
    retlw       b'10001111'
    retlw       b'10000101'
```

Процедура calculate

Процедура *calculate* вычисляет Rm в соответствии с формулой $Rm = \frac{Tm}{Tc} \cdot Rc.$

Результат вычисления записывает в двух регистрах resl и resh. Вычисления проводим по действиям, сначала умножаем Tm на Rc в подпрограмме clc_1 (Rc = ETALON) (т. е. мы складываем Tm + Tm Rc раз) и записываем результат умножения в регистрах Tmpl и Tmph. Затем делим результат умножения (Tm и Rc) на Tc и результат деления записываем в двух регистрах – resl и resh.

Блок-схема представлена на рис. 10.11.

```
calculate
    movlw       b'11111111'
    banksel     TRISB
    movwf       TRISB
    banksel     PORTB
    bcf         INTCON, 7
    clrf        tmph
    clrf        tmpl
    movf        ETALON, w     ;100 кОм ЭТАЛОН
    movwf       cnt1
clc_1:
    ; Умножаем Тм на Rc, результат получаем в двух
    ; регистрах Tmpl и Tmph
    movf        tmpl, W
    addwf       Tm, W
    movwf       tmpl
    btfsc       STATUS, C
    incf        tmph, F
    decfsz      cnt1, F
    goto        clc_1
    movlw       .255
    movwf       resl
    movwf       resh
clc_2:
    ;Инкремент res
    movf        resl, W
    addlw       .1
    movwf       resl
    btfsc       STATUS, C
    incf        resh, F
    ;tmp = tmp - Tc
    movf        Tc, W
    subwf       tmpl, W
    movwf       tmpl
    btfsc       STATUS, C
    goto        clc_2
    movlw       .1
    subwf       tmph, W
    movwf       tmph
    btfsc       STATUS, C
    goto        clc_2
    return
```

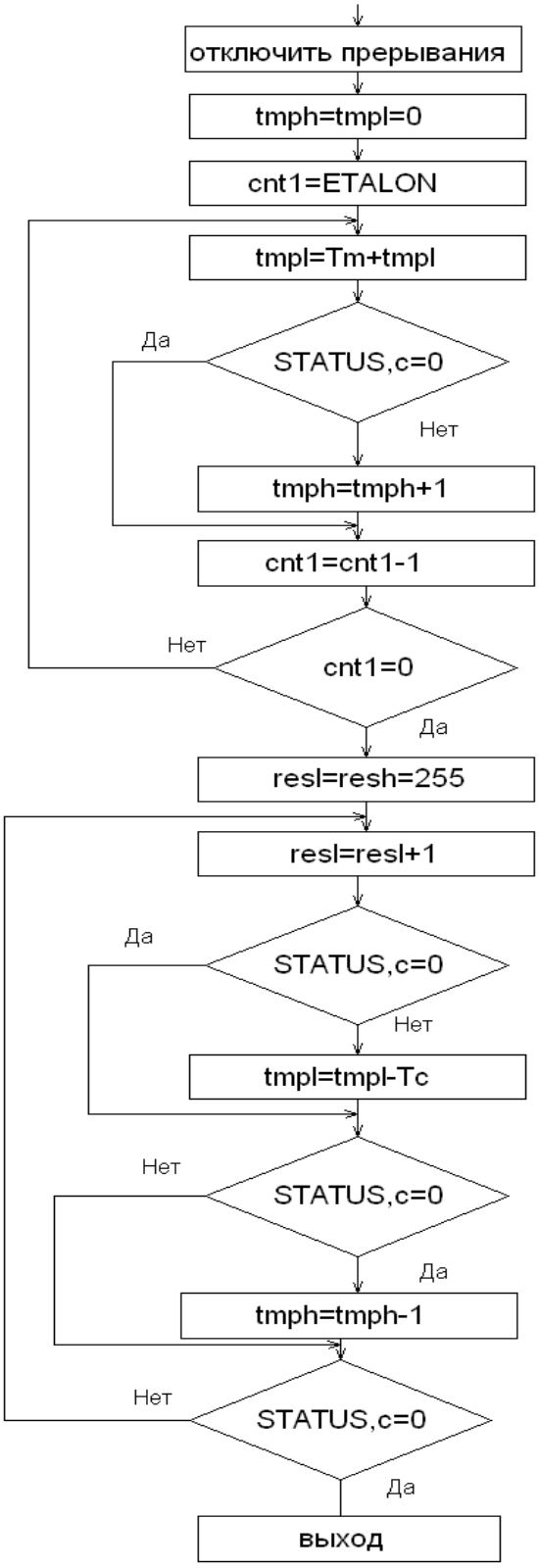

отключить прерывания
tmph=tmpl=0
cnt1=ETALON
tmpl=Tm+tmpl
Да
STATUS,c=0
Нет
tmph=tmph+1
cnt1=cnt1-1
Нет
cnt1=0
Да
resl=resh=255
resl=resl+1
Да
STATUS,c=0
Нет
tmpl=tmpl-Tc
Нет
STATUS,c=0
Да
tmph=tmph-1
Нет
STATUS,c=0
Да
выход

Рис. 10.11

Процедура bin2bcd

Эта процедура преобразует двоичное число, записанное в двух регистрах resl и resh, в десятичное и записывает в трех регистрах – d2, d1 и d0, т. е. в них находятся сотни, десятки и единицы, соответственно. Затем вызывает процедуру вывода этих чисел на семисегментные индикаторы. Также проверяет, чтобы в регистре (переменной) d2 число сотен не превышало 9, т. к. на один семисегментный индикатор нельзя вывести двухразрядное число, т. е. больше 9.

Блок-схема представлена на рис. 10.12.

```
bin2bcd:
    ;Находим сотни
    movlw       .255
    movwf       d2
    b2b_1:
    incf        d2, F
    movlw       .100
    subwf       resl, W
    movwf       resl
    btfsc       STATUS, C
    goto        b2b_1
    movlw       .1
    subwf       resh, W
    movwf       resh
    btfsc       STATUS, C
    goto        b2b_1
Prov
    ; проверка числа в d2, чтобы не превышало 9
    BTFSC       d2, 7
    goto        Err
    BTFSC       d2, 6
    goto        Err
    BTFSC       d2, 5
    goto        Err
    BTFSC       d2, 4
    goto        Err
    BTFSS       d2, 3
```

```
    goto        prov_1
    BTFSC       d2, 2
    goto        Err
    BTFSC       d2, 1
    goto        Err
prov_1
    movf        d2, w       ; вывод старшего числа на
    call        outled      ; индикаторы
    movlw       .100
    addwf       resl, W
    movwf       resl
    ;Находим десятки
    movlw       .255
    movwf       d1
b2b_2:
    incf        d1, F
    movlw       .10
    subwf       resl, W
    movwf       resl
    btfsc       STATUS, C
    goto        b2b_2
    movf        d1, w       ; вывод среднего числа на
    call        outled      ; индикаторы
    movlw       .10
    addwf       resl, W
    movwf       resl
    ;Находим единицы
    movf        resl, W
    movwf       d0
    movf        d0, w       ; вывод старшего числа на
    call        outled      ; индикаторы
    return
```

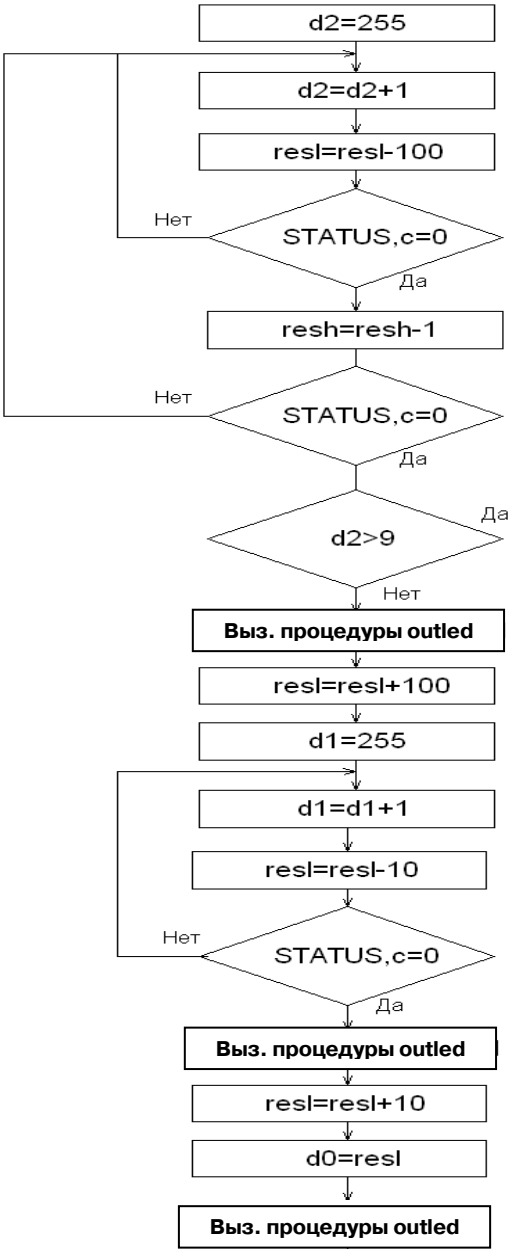


Рис. 10.12

Процедура Err

Эта процедура выводит на индикаторы надпись **Err**

```
Err
    movlw       .11
    CALL        outled
    movlw       .12
    CALL        outled
    movlw       .12
    CALL        outled
    goto        vse
```

СПИСОК ЛИТЕРАТУРЫ

1. R.L. Tokheim Theory and problems of microprocessor fundamentals. – Mc Graw-Hill, 1983.
2. P. Horowitz, W. Hill. The art of electronics. – Cambridge University Press, 1980.
3. The industrial electronic handbook / edited by D. Irwin. – CRC Press, 1996.
4. Вычислительная и микропроцессорная техника: учебник для вузов / под ред. Э.В. Евреинова. – М.: Радио и связь, 1991.
5. Самохвалов К.Г. Микропроцессоры. – К.: Техника, 1989.
6. Григорьев В.Л. Программное обеспечение микропроцессорных систем. – М.: Энергоатомиздат, 1989.
7. Гутников В.С. Интегральная электроника в измерительных устройствах. – Л.: Энергоатомиздат, 1988.
8. Балашов Е.П., Пузанков Д.В. Микропроцессоры и микропроцессорные системы. – М.: Радио и связь, 1981.
9. Михальчук В.М. и др. Микропроцессоры i80x86. Архитектура, функционирование, программирование, оптимизация кода. – Мн.: Битрикс, 1994.
10. Гольденберг Л.М., Малев В.А., Малько Г.Б. Цифровые устройства и микропроцессорные системы. Задачи и упражнения. – М.: Радио и связь, 1992.
11. Гольденберг Л.М. и др. Цифровые устройства и микропроцессорные системы. – М.: Радио и связь, 1992.
12. Микропроцессоры и микропроцессорные комплекты интегральных схем: в 2-х томах / под ред. В.А. Шахнова. – М.: Радио и связь, 1989.
13. Проектирование микропроцессорной электронно-вычислительной аппаратуры: справочник / В.Г. Артюхов и др. – К.: Техника, 1988.
14. Мячев А.А., Степанов В.Н. Персональные ЭВМ и микроЭВМ. Основы организации. – М.: Радио и связь, 1991.
15. Большие интегральные схемы запоминающих устройств / под ред. А.Ю. Гордонова. – М.: Радио и связь, 1990.
16. Петросян О.А. и др. Схемотехника БИС постоянных запоминающих устройств. – М.: Радио и связь, 1987.
17. Применение интегральных микросхем памяти: справочник / под ред. А.Ю. Гордонова. – М.: Радио и связь, 1994.
18. Мирский Г.Я. Микропроцессоры в измерительных приборах. – М.: Радио и связь, 1984.
19. Алексенко А.Г. и др. Проектирование радиоэлектронной аппаратуры на микропроцессорах. – М.: Радио и связь, 1984.
20. Microprocessors and chipsets. Materials of *Intel* Corporation.

ОГЛАВЛЕНИЕ

Введение ... 1
Глава 1. Информация и ее представление в компьютерных системах ... 7
1.1. Что такое информация ... 7
1.2. Измерение информации ... 13
1.3. Системы счисления, применяемые в цифровых системах ... 17
1.4. Кодирование информации в цифровых системах ... 23
Примеры решения задач ... 28
Задачи и упражнения ... 30
Глава 2. Вычислительная система и алгоритм ее работы ... 33
2.1. Структура вычислительной системы и назначение ее основных частей ... 33
2.2. Архитектура однокристальных микропроцессоров ... 45
2.2.1. Параметры микропроцессоров и общая характеристика их архитектуры ... 45
2.2.2. Функциональная схема и набор интерфейсных сигналов ... 47
2.2.3. Программная модель и система команд ... 51
2.2.4. Формат команды, способы адресации операндов ... 55
Примеры решения задач ... 57
Вопросы и задания для повторения ... 62
Глава 3. Архитектура 16-разрядного универсального микропроцессора ... 65
3.1. Функциональная схема микропроцессора 8086 ... 66
3.2. Интерфейсные сигналы микропроцессора, циклы обмена с шиной ... 68
3.3. Программная модель микропроцессора ... 73
3.4. Способы адресации операндов ... 79
Примеры решения задач ... 88
Вопросы и задания для повторения ... 89
Глава 4. Система команд 16-разрядного универсального микропроцессора ... 91
4.1. Команды передачи данных ... 92
4.2. Команды арифметических операций ... 96
4.3. Команды логических операций и команды сдвигов ... 103
4.4. Команды передачи управления ... 106
4.5. Цепочечные команды ... 113
4.6. Команды управления микропроцессором ... 116
Примеры решения задач ... 118
Вопросы и задания для повторения ... 119
Глава 5. Проектирование систем на базе микропроцессоров *Intel* ... 121
5.1. Микропроцессорная система на основе микропроцессора 8085 ... 121
5.2. Микропроцессорная система на основе микропроцессора 8086 в минимальном режиме ... 129
5.3. Многопроцессорные и многопользовательские системы ... 133
Вопросы и задания для повторения ... 139
Примеры решения задач ... 139
Задачи и упражнения ... 142

Глава 6. Организация памяти микропроцессорных систем 144
6.1. Основные принципы организации памяти 145
6.2. Элементная база запоминающих устройств 148
6.3. Принципы построения ЗУ на микросхемах памяти 154
Вопросы и задания для повторения 158
Примеры решения задач 158
Задачи 159
Глава 7. Организация ввода/вывода в микропроцессорной системе 163
7.1. Схемотехника и программирование пользовательских интерфейсов .. 163
7.2. Специальные режимы ввода/вывода 170
7.2.1. Прерывания 170
7.2.2. Прямой доступ к памяти 173
7.3. Стандартные интерфейсы 175
Вопросы и задания для повторения 178
Задачи 180
Глава 8. Однокристальные микроконтроллеры 182
8.1. Особенности аппаратных средств микроконтроллеров 182
8.2. Система команд MCS51 192
Вопросы для повторения 202
Глава 9. Принципы использования микропроцессоров
в информационно-измерительных системах 203
9.1. Проектирование микропроцессорных автоматизированных
измерительных систем 203
9.2. Современный арсенал средств разработчика автоматизированных
средств и систем контроля 208
Глава 10. Пример разработки омметра на основе микроконтроллера PIC16F84A 221
10.1. Проектирование аппаратной части прибора 221
10.2. Разработка программы измерений 224
Список литературы 242

Учебное издание

Алхимов Юрий Васильевич

МИКРОПРОЦЕССОРЫ И ЦИФРОВЫЕ СИСТЕМЫ В НЕРАЗРУШАЮЩЕМ КОНТРОЛЕ

Учебное пособие

Научный редактор
доктор технических наук,
профессор — *Б.И. Капранов*

Редактор — *М.В. Пересторонина*

Верстка — *В.П. Аршинова*

Дизайн обложки — *О.Ю. Аршинова*
О.А. Дмитриев

Подписано к печати 27.07.2008. Формат 60x84/16. Бумага «Снегурочка».
Печать XEROX. Усл. печ. л. 14,31. Уч.-изд. л. 12,94.
Заказ 736. Тираж 500 экз.

Томский политехнический университет
Система менеджмента качества
Томского политехнического университета сертифицирована
NATIONAL QUALITY ASSURANCE по стандарту ISO 9001:2000

ИЗДАТЕЛЬСТВО ТПУ. 634050, г. Томск, пр. Ленина, 30.