

РІС-микроконтроллеры

Полное руководство



COMPUTER COMMUNICATION AND NETWORKS
SERIES

Sid Katzen

THE QUITESSENTIAL PIC® MICROCONTROLLER

Second Edition

 Springer

СЕРИЯ
ПРОГРАММИРУЕМЫЕ СИСТЕМЫ

Сид Катцен

РІС-МИКРОКОНТРОЛЛЕРЫ ПОЛНОЕ РУКОВОДСТВО

*Перевод с английского
Евстифеева А. В.*



Москва
Издательский дом «Додэка-XXI»
2010

Катцен, Сид

К29 PIC-микроконтроллеры. Полное руководство / Сид Катцен; пер. с англ. Евстифеева А. В. — М. : Додэка-XXI, 2010. — 656 с. : ил. (Серия «Программируемые системы»). — Доп. тит. л. англ. — ISBN 978-5-94120-218-8.

Данная книга представляет собой исчерпывающее руководство по микроконтроллерам семейства PIC компании Microchip, являющегося промышленным стандартом в области встраиваемых цифровых устройств. В книге подробно описывается архитектура и система команд 8-битных микроконтроллеров PIC, на конкретных примерах изучается работа их периферийных модулей.

В первой части излагаются основы цифровой схемотехники, математической логики и архитектуры вычислительных систем. Вторая часть посвящена различным аспектам программирования PIC-микроконтроллеров среднего уровня: описывается набор команд, рассматривается написание программ на ассемблере и языке высокого уровня (Си), а также поддержка подпрограмм и прерываний. В третьей части изучаются аппаратные аспекты взаимодействия микроконтроллера с окружающим миром и обработки прерываний. Рассматриваются такие вопросы, как параллельный и последовательный ввод/вывод данных, временные соотношения, обработка аналоговых сигналов и использование EEPROM. В заключение приводится пример разработки реального устройства. На этом примере также демонстрируются простейшие методики отладки и тестирования, применяемые при разработке реальных устройств.

Книга рассчитана на самый широкий круг читателей — от любителей до инженеров, при этом для понимания содержащегося в ней материала вовсе не требуется каких-то специальных знаний в области программирования, электроники или цифровой схемотехники. Эта книга будет также полезна студентам, обучающимся по специальностям «Радиоэлектроника» и «Вычислительная техника», которые смогут использовать ее в качестве учебного пособия при прослушивании соответствующих курсов или выполнении курсовых проектов.

УДК 004.312.46
ББК 32.973.26-04

Все права защищены. Никакая часть этого издания не может быть воспроизведена в любой форме или любыми средствами, электронными или механическими, включая фотографирование, ксерокопирование или иные средства копирования или сохранения информации, без письменного разрешения издательства.

Translation from the English language edition:
The Quintessential PIC® Microcontroller By Sid Katzen
Copyright © Springer-Verlag London Ltd, being a part of Springer Science+Business Media
All Rights Reserved

ISBN 978-5-94120-218-8 (рус.)
ISBN 0-7506-7698-1 (англ.)

© Springer-Verlag London Limited, 2005
© Издательский дом «Додэка-XXI», 2010
® Серия «Программируемые системы»

ОГЛАВЛЕНИЕ

Часть I. Основы	13
Глава 1. Цифровое представление	16
Глава 2. Логические схемы	30
Глава 3. Обработка хранимой программы	57
Центральный процессор	60
Память	60
Интерфейсные порты	61
Шина данных	62
Счетчик команд	63
Конвейер	64
Дешифратор команд	65
Регистр адреса	65
Регистр данных	65
Арифметико-логическое устройство	66
Регистр состояния	66
Рабочий регистр	66
Память программ	66
Память данных	66
Прямая адресация регистра данных	68
Операции с константами	69
Примеры	80
Вопросы для самопроверки	84
Часть II. Программное обеспечение	85
Глава 4. Микроконтроллер PIC16F84	87
Блок выборки	90
Исполнительный блок	93
Примеры	108
Вопросы для самопроверки	112
Глава 5. Набор команд	114
Адресация кодом команды	116

Адресация константы	116
Абсолютная адресация памяти программ	117
Прямая адресация памяти данных	118
Косвенная адресация памяти данных	123
Битовая адресация	128
Команды пересылки данных	128
Команды арифметических операций	131
Команды логических операций и операций сдвига	141
Команды передачи управления	152
Примеры	156
Вопросы для самопроверки	165
Глава 6. Подпрограммы и модули	168
Примеры	193
Вопросы для самопроверки	204
Глава 7. Обработка прерываний	207
Примеры	224
Вопросы для самопроверки	235
Глава 8. Инструментальные средства для работы с языком ассемблера	238
Примеры	268
Вопросы для самопроверки	272
Глава 9. Язык высокого уровня	275
Примеры	290
Вопросы для самопроверки	298
Часть III. Окружающий мир	299
Глава 10. Реальное окружение	302
Примеры	322
Вопросы для самопроверки	324
Глава 11. Ничего, кроме байтов	325
Примеры	352
Вопросы для самопроверки	366
Глава 12. Ох уж эти биты!	368
Примеры	435
Глава 13. Главное — время	450
Примеры	479
Вопросы для самопроверки	486
Глава 14. Этот безумный аналоговый мир	488
Примеры	527
Вопросы для самопроверки	540
Глава 15. Хранить вечно!	542
Примеры	559
Вопросы для самопроверки	569

Глава 16. Дальнейшее развитие	571
Блок выборки	572
Исполнительный блок	575
Периферийные устройства	581
Обработка прерываний	583
Система команд	584
Глава 17. Учебный пример	595
Конфигурирование кристалла	607
Выполнение программы	607
Приложение А. Список сокращений, символических имен и аббревиатур	618
1. Русская нотация	618
2. Английская нотация	619
Приложение Б. Регистры специального назначения микроконтроллеров PIC16F87XA ..	632
Приложение В. Элементы языка Си	635
Приложение Г. Набор команд микроконтроллеров с 14-битным ядром	637
Предметный указатель	639

Предисловие ко второму изданию

Поводом к выпуску второго издания данной книги стало большое количество предложений и замечаний от моих студентов и читателей из разных уголков земного шара — от Шотландии до Гавайских островов. Со времени выхода первого издания книги в конце 1990-х микроконтроллеры PIC компании Microchip стали самыми продаваемыми 8-битными микроконтроллерами. Возможности моделей среднего уровня, рассматривавшихся в первом издании, значительно возросли, так что некоторые из использовавшихся ранее моделей безнадежно устарели. Кроме того, значительно увеличилось количество моделей с 16-битным словом команд. В то же время появились новые представители линейки микроконтроллеров младшего (или базового) уровня. Поскольку все выпускаемые линейки микроконтроллеров имеют очень много общего, в новом издании основное внимание будет по-прежнему уделяться микроконтроллерам среднего уровня.

Практически все рисунки были изменены, причем многие довольно существенно; было добавлено множество новых иллюстраций. При переработке книги особое внимание уделялось ясности изложения базовых концепций. По этой причине, а также для улучшения связи с 4-й и 5-й главами третья глава была значительно переработана. К слову сказать, в обеих упомянутых главах от первоначального текста вообще практически ничего не осталось. Также с целью подробного разъяснения сложных для понимания вопросов была существенно переработана глава 7, посвященная обработке прерываний. Третья часть книги была не только обновлена в связи с использованием современных моделей микроконтроллеров, но и расширена, с тем чтобы охватить новые периферийные модули, такие как аналоговый компаратор и встроенный источник опорного напряжения. Кроме того, была добавлена глава, знакомящая читателя с линейкой наиболее развитых микроконтроллеров PIC18XXX.

Все главы книги, за исключением двух первых и последней, снабжены рабочими примерами, а также вопросами для самопроверки. Кроме того, к вашим услугам имеется сайт¹⁾

<http://www.engj.ulst.ac.uk/sidk/quintessential>,

¹⁾ Этот сайт посвящен оригинальному изданию книги на английском языке, и все перечисленные ниже материалы представлены также на английском. — *Примеч. ред.*

на котором вы сможете найти:

- Ответы к вопросам для самопроверки.
- Дополнительные вопросы для самопроверки.
- Дополнительные материалы.
- Исходные тексты всех примеров и задач, встречающихся в книге.
- Ссылки на инструментальные средства разработки, а также на документацию к микросхемам, упоминающимся в книге.
- Список опечаток.
- Отзывы читателей.

Рукопись книги¹⁾ набиралась автором на различных ПК, работающих под управлением Microsoft® Windows™ с использованием среды L^AT_EX2ε (реализация Y&Y) и шрифта Lucida Bright. Векторные иллюстрации были созданы или отредактированы в программе Autocad R13 и внедрены в файл рукописи в виде EPS-файлов. Все фотографии были сделаны самим автором при помощи различных цифровых фотоаппаратов фирмы Olympus, кстати, битком набитых микроконтроллерами!

Надеюсь, что мне удалось изгнать из рукописи всех гремлин, однако, если вы все же найдете ошибки или у вас возникнут какие-либо предложения, я буду рад, если вы свяжетесь со мной через сайт.

Сид Катцен
Ольстерский университет, Джорданстаун
Июль, 2005 г.

¹⁾ Имеется в виду оригинальное издание книги на английском языке. — *Примеч. ред.*

Предисловие к первому изданию

Микропроцессоры и производные от них — микроконтроллеры — являются широко распространенным и при этом незаметным элементом инфраструктуры современного общества, основанного на электронике и коммуникациях. Исследования¹⁾, проведенные в 1998 году, показали, что в каждом доме незаметно для нас «живет» около 100 микроконтроллеров и микропроцессоров. Они присутствуют буквально всюду: в звуковых открытках, стиральных машинах, микроволновых печах, телевизорах, телефонах, персональных компьютерах и разных других устройствах. Даже в самом обыкновенном автомобиле скрывается более двадцати таких элементов, где они, в частности, контролируют состояние беспроводных датчиков давления в шинах и отображают критичные данные, получаемые по сети CAN.

Каждый год продается около четырех миллиардов подобных изделий, предназначенных для реализации «мозгов» разнообразных «умных» устройств, начиная от интеллектуальных таймеров для яйцеварок и заканчивая системами управления самолетом. Эволюция микропроцессоров, первые из которых были выпущены компанией Intel в далеком 1971 году, привела к коренному изменению структуры общества, спровоцировав в начале XXI века вторую промышленную революцию. Несмотря на то что микропроцессоры, являясь основным компонентом вездесущих ПК, известны лучше, объем продаж различных микропроцессоров, таких как Intel Pentium, составляет всего около 2% от общего объема продаж подобных устройств. Подавляющее же большинство продаж приходится на дешевые микроконтроллеры, встраиваемые в специализированные электронные устройства, такие как смарт-карты. Причем если основной задачей микропроцессоров является обеспечение собственно вычислительной мощности, то во втором случае акцент смещается в сторону объединения на одном кристалле центрального процессора, памяти и устройств ввода/вывода. Такая интегрированная вычислительная система называется *микроконтроллером*.

Задумывая книгу по этой тематике, автор ставил перед собой задачу дать читателю базовые знания в области разработки небольших встроенных систем на базе микроконтроллеров, а не просто рассказать об архитектуре ЭВМ в традиционном

¹⁾ *New Scientist*, vol.59, no. 2141, 4 July 1998, p.139.

понимании этого слова на примере микроконтроллеров. Будем надеяться, что подобный подход даст читателю уверенность в том, что даже на таком начальном уровне он сможет разработать, изготовить и запрограммировать полностью готовую рабочую встроенную систему.

Учитывая практический характер излагаемого материала, для его иллюстрации используется реально существующее аппаратное и программное обеспечение. Основную долю на рынке занимают устройства, оперирующие 8-битными данными (хотя имеются как 4-, так и 16-битные устройства), во многом схожие с первыми микропроцессорами и кардинальным образом отличающиеся от современной «тяжелой артиллерии» в лице микропроцессоров Intel Pentium и Power PC. В отличие от последних, сущностью микроконтроллера является высокая степень системной интеграции при низкой стоимости. Суммарная вычислительная мощность системы может быть увеличена путем распределения процессоров по системе. Так, в каждом сочленении манипулятора робота может использоваться свой микроконтроллер, выполняющий простые локальные операции и обменивающийся данными с более мощным процессором, определяющим функционирование всего робота.

При выборе конечной архитектуры принимались во внимание ее популярность на коммерческом рынке, доступность и наличие недорогого ПО для разработки. В итоге выбор был сделан в пользу микроконтроллеров фирмы Microchip — одного из наиболее популярных семейств, использующихся при изучении микроконтроллеров/микропроцессоров на самых разных этапах учебного процесса, начиная со старших классов школы и заканчивая университетом. Освоение микроконтроллеров этой фирмы, в частности, облегчается небольшим набором команд и относительно простой передовой архитектурой. Помимо использования в промышленности и образовательном процессе, микроконтроллеры семейства PIC® применяются в большинстве любительских устройств, в чем можно убедиться, открыв любой журнал, посвященный радиолюбительству.

Компания Microchip Inc. — относительно молодой игрок на рынке микроконтроллеров, на который она вышла в 1989 году после разработки нового семейства микроконтроллеров с гарвардской архитектурой. К концу 1999 года компания Microchip была уже вторым по величине производителем 8-битных микроконтроллеров, уступая только компании Motorola.

Книга, которую вы держите в руках, состоит из трех частей. В первой части излагаются основы цифровой схемотехники, математической логики и архитектуры вычислительных систем. Приведенных сведений будет достаточно для понимания вопросов, рассматриваемых в остальных двух частях книги. Наличие в книге информации такого рода позволяет обойтись без изучения дополнительной литературы.

Вторая часть книги посвящена главным образом различным аспектам программирования PIC-микроконтроллеров среднего уровня: набор команд, написание программ на ассемблере и языке высокого уровня (Си), поддержка подпрограмм и прерываний. Несмотря на то что при изложении материала используется

линейка 14-битных моделей, рассмотренные принципы и архитектура справедливы как для 12-битных, так и для 16-битных¹⁾ представителей семейства.

В третьей части изучаются аппаратные аспекты взаимодействия микроконтроллера с окружающим миром, а также обработки прерываний. Разумеется, параллельно продолжается изучение аппаратных и программных средств микроконтроллера. Рассматриваются такие вопросы, как параллельный и последовательный ввод/вывод данных, формирование сигналов и измерение их временных параметров, обработка аналоговых сигналов и использование EEPROM. В заключение рассматривается процесс разработки реального устройства, позволяющий объединить разрозненные знания, полученные при чтении книги, в одно целое. На этом примере также демонстрируются простейшие методики отладки и тестирования, применяемые при разработке реальных устройств.

Сид Катцен

Ольстерский университет, Джорданстаун
Декабрь, 2000 г.

¹⁾ Здесь имеется в виду не размер данных, которыми оперирует микроконтроллер, а число битов, использующихся для записи слова команды. — *Примеч. пер.*

ЧАСТЬ I

ОСНОВЫ

Глава 1. Цифровое представление

Глава 2. Логические схемы

Глава 3. Обработка хранимой программы

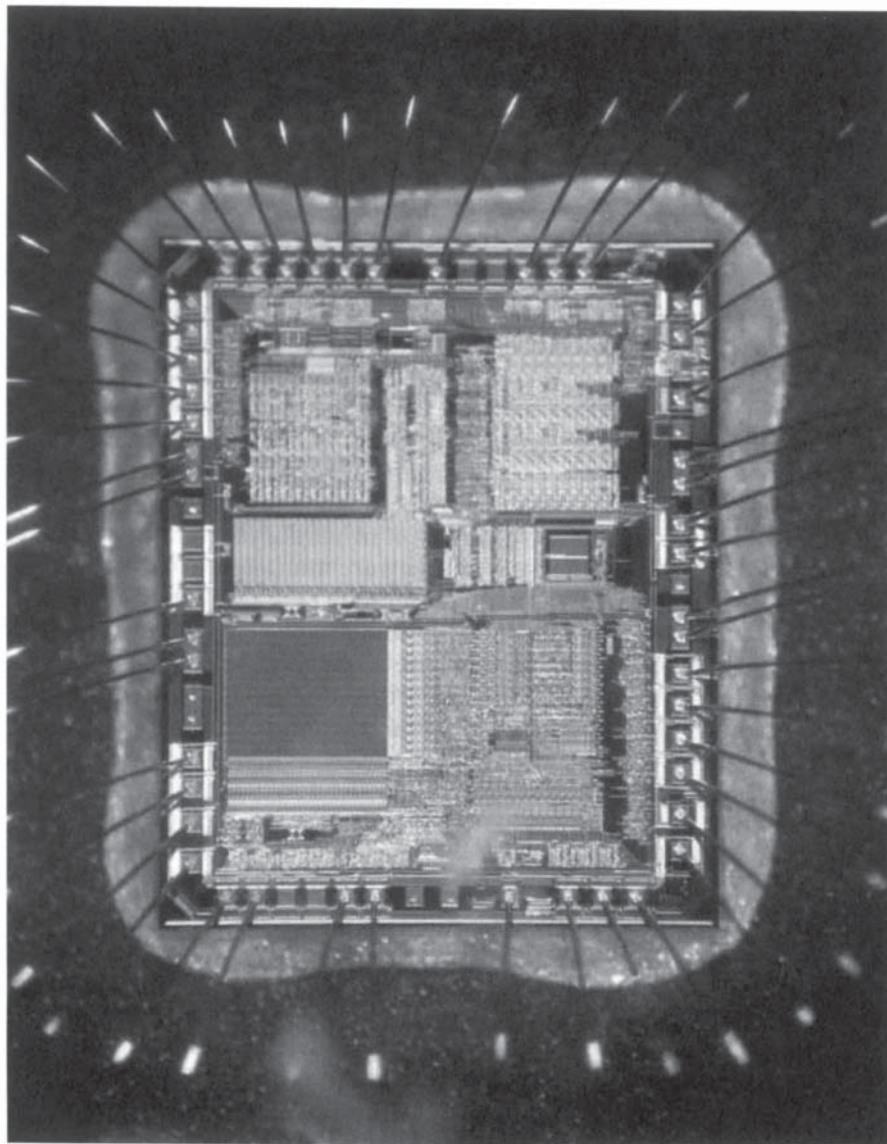
Эта книга посвящена микроконтроллерам. Микроконтроллеры представляют собой цифровые устройства, построенные по образу и подобию ЭВМ с хранимой программой и объединенные вместе со вспомогательными узлами, памятью различного типа и блоками сопряжения в микросхемах сверхвысокой степени интеграции. Хотя, говоря о микроконтроллерах, часто имеют в виду их более известных «двоюродных братьев» — микропроцессоры, которые являются важнейшим узлом персональных компьютеров, подавляющее большинство как микроконтроллеров, так и микропроцессоров, помимо ПК, используются и во многих других электронных устройствах. Первые микропроцессоры, появившиеся на рынке в начале 70-х, позиционировались в качестве альтернативного способа реализации цифровых схем. Выполняемые функции определялись последовательностью инструкций, хранящихся в виде двоичных чисел в постоянном запоминающем устройстве (ПЗУ). Это решение обладало большей гибкостью по сравнению с традиционной схемой соединения различных микросхем. Современный микроконтроллер является одним из воплощений такого интегрированного вычислителя.

Использованию встраиваемых микроконтроллеров в контексте собственно цифровых вычислений посвящены 2-я и 3-я части книги. Пока же нам требуется заложить фундамент для понимания этого материала. Итак, в первой части мы с вами рассмотрим:

- Цифровые коды.
- Двоичную арифметику.
- Основы цифровой схемотехники.
- Архитектуру вычислительных устройств и их программирование.

Разумеется, мы не сможем в полной мере охватить все указанные вопросы, однако существует много других превосходных книг¹⁾ по этой тематике, с помощью которых вы сможете продолжить изучение на более глубоком уровне.

¹⁾ См., например: *Рональд Дж. Точчи, Нил С. Уидмер*. Цифровые системы. Теория и практика: 8-е изд.: Пер. с англ. — М.: Издательский дом «Вильямс», 2004.



Заглядывая внутрь микросхемы

ЦИФРОВОЕ ПРЕДСТАВЛЕНИЕ

Как для компьютера, так и для микроконтроллера окружающий мир представляется в виде различных чисел. В *десятичной* системе счисления числовые величины описываются с помощью десяти цифр: 0, 1,..., 9. Используя при необходимости символы «+», «−» и «.»¹⁾, можно выразить любое число из диапазона $\pm\infty$. На самом деле, с помощью чисел можно выражать даже нечисловые понятия. К примеру, в коде ASCII (американский стандартный код обмена информацией) символу «А» соответствует число 65, символу «В» — 66,..., «Z» — 90, «а» — 97, «b» — 98,..., «z» — 122 и т.д. Соответственно, слово «Microcontroller» можно закодировать в виде последовательности чисел «77, 105, 99, 114, 111, 99, 111, 110, 116, 114, 111, 108, 108, 101, 114». При условии, что нам известен контекст, т.е. какие числа описывают реальные числовые величины, а какие — текст, с их помощью можно закодировать практически любые символы²⁾.

Электронные схемы не очень хорошо подходят для хранения и обработки множества различных значений. Да, первая американская цифровая вычислительная машина ENIAC (электронный цифровой интегратор и калькулятор), созданная в 1964 году, выполняла арифметические операции в десятичном виде³⁾, однако все компьютеры, появившиеся впоследствии, оперировали уже данными в *двоичной* (с основанием 2) системе. В действительности десятичная система счисления удобна только для человека, поскольку у нас на руках 10 пальцев⁴⁾. Так что в этой главе мы будем рассматривать исключительно свойства двоичных разрядов, их группирование, а также операции над двоичными числами. Прочитав главу, вы:

- Поймете, почему двоичное представление данных является наиболее удобным для цифровых схем.
- Узнаете, как одну и ту же величину можно выразить в двоичном, шестнадцатеричном и двоично-десятичном (BCD) виде.

¹⁾ В данной книге для отделения целой части числа от дробной используется точка, а не запятая. — *Примеч. ред.*

²⁾ Разумеется, существует множество других цифровых кодировок, к примеру 6-точечный код Брайля для слепых.

³⁾ Как и механическое вычислительное устройство Бэббиджа, появившееся столетием раньше.

⁴⁾ И десять пальцев на ногах, однако система счисления по основанию 20 используется очень редко (но все-таки она существует).

- Научитесь выполнять сложение и вычитание двоичных чисел.
- Узнаете, как выполнять умножение посредством сдвига влево.
- Узнаете, как выполнять деление посредством сдвига вправо с копированием знакового бита.
- Познакомитесь с логическими операциями НЕ, И, ИЛИ и Исключающее ИЛИ.

В основе информационных технологий лежит обработка, вычисление и передача информации, представленной в цифровом виде. Эта информация в подавляющем большинстве случаев представлена в виде множества двоичных разрядов (*битов*¹⁾). Как правило, такая обработка осуществляется с использованием микропроцессоров²⁾ и микроконтроллеров. Интересно отметить, что вычислительная мощность современной звуковой открытки превышает совокупную мощность всех вычислительных устройств, имевшихся на планете в 1950 году!

Двоичная система — это универсальный способ представления данных, поскольку простейшим устройством, которое можно реализовать на одном транзисторе, является электронный ключ. Такие ключи, имеющие только два состояния, очень малы; они способны очень быстро изменять свое состояние и потребляют незначительный ток. Более того, поскольку требуется различать только два состояния, очевидно, что двоичное представление менее подвержено воздействию помех. Из сказанного становится ясно, что и плотность компоновки элементов на кристалле, и скорости переключения этих элементов могут достигать очень больших значений. Хотя сам ключ как таковой не обладает какой-либо вычислительной мощностью, 5 миллионов ключей, переключающихся 100 миллионов раз в секунду, способны продемонстрировать, по крайней мере, видимость интеллекта!

Два состояния бита обычно называются *логическим нулем* (лог. 0) и *логической единицей* (лог. 1) или просто 0 и 1. Один бит может быть представлен двумя состояниями любой физической величины, например напряжения или силы электрического тока, освещенности, давления воздуха. В большинстве микроконтроллеров состоянию лог. 0 соответствует напряжение 0 В (или «земля»), а состоянию лог. 1 — напряжение +3...5 В, хотя это правило и не универсально. Например, в последовательном порту RS-232 вашего ПК для индикации состояния лог. 0 используется напряжение +12 В, а для индикации состояния лог. 1 — напряжение –12 В.

Итак, один бит может представлять только два состояния. Более сложные элементы можно выразить с помощью комбинаций битов. Например, обычные алфавитно-цифровые символы³⁾ можно представить с помощью 7-битных групп

¹⁾ Не думайте, что двоичная система была придумана специально для цифровых вычислительных машин! Многие древние культуры пользовались двоичным счетом, например хараппская цивилизация, существовавшая более 4000 лет назад в бассейне реки Инд. В развалинах одного из кварталов хараппского города Мохенджо-Даро был найден набор каменных гирь, веса которых подчинялись соотношению 1, 1, 2, 4, 8, 16, ..., т.е. вес каждой гири был равен удвоенному весу предыдущей (вес самой маленькой гири был равен примерно 25 г, или одной унции). Таким образом, веса этих камней выражались числами, являющимися степенями двойки, т.е. в двоичном коде.

²⁾ Микропроцессоры и микроконтроллеры очень тесно связаны друг с другом (см. **Рис. 3.8** на стр. 78), поэтому мы попеременно будем использовать оба термина.

³⁾ Имеется в виду английский алфавит. — *Примеч. пер.*

двоичных разрядов, как показано в **Табл. 1.1**. Таким образом, ASCII-представление строки «Microcontroller» будет иметь вид

```
1001101 1101001 1100011 1110010 1101111 1100011 1101111 1101110
1110100 1110010 1101111 1101100 1101100 1100101 1110010
```

В кодировке Юникод (Unicode), являющейся дальнейшим развитием кодировки ASCII, используются уже 16-битные группы, поэтому с ее помощью можно выразить символы всех существующих языков, а также различные математические и прочие специальные символы.

Таблица 1.1. 7-битные символы ASCII

Ст. полубайт – Мл. полубайт	h'00' b'000'	h'01' b'001'	h'02' b'010'	h'03' b'011'	h'04' b'100'	h'05' b'101'	h'06' b'110'	h'07' b'111'
h'00' b'0000'	NUL	DLE	SP	0	@	P	`	p
h'01' b'0001'	SOH	XON	!	1	A	Q	a	q
h'02' b'0010'	STX	DC2	“	2	B	R	b	r
h'03' b'0011'	ETX	XOFF	#	3	C	S	c	s
h'04' b'0100'	EOT	DC4	\$	4	D	T	d	t
h'05' b'0101'	ENQ	NAK	%	5	E	U	e	u
h'06' b'0110'	ACK	SYN	&	6	F	V	f	v
h'07' b'0111'	BEL	ETB	'	7	G	W	g	w
h'08' b'1000'	BS	CAN	(8	H	X	h	x
h'09' b'1001'	HT	EM)	9	I	Y	i	y
h'0A' b'1010'	LF	SUB	*	:	J	Z	j	z
h'0b' b'1011'	VT	ESC	+	;	K	[k	{
h'0C' b'1100'	FF	FS	,	<	L	\	l	
h'0D' b'1101'	CR	GS	–	=	M]	m	}
h'0E' b'1110'	SO	RS	.	>	N	^	n	~
h'0F' b'1111'	SI	US	/	?	O	_	o	DEL

Код ASCII называется *невзвешенным*, поскольку отдельные биты не несут какого-либо смысла; значение имеет только вся совокупность битов. В качестве других примеров невзвешенных кодов можно отметить код значения на гранях игральной кости и семисегментный код, изображенный на **Рис. 6.8** (стр. 183). Мы же в основном будем работать с *обычным двоичным взвешенным* кодом, в котором позиция бита определяет его величину или, иначе, вес. В целом двоичном числе самый правый бит имеет вес $2^0 = 1$, находящийся слева от него — $2^1 = 2$ и так далее до n -й позиции, бит в которой имеет вес 2^{n-1} . В частности, десятичное число 1998 представляется таким образом:

$$\begin{array}{cccc} 10^3 & 10^2 & 10^1 & 10^0 \\ 1 & 9 & 9 & 8 \end{array}$$

т.е. $1 \times 10^3 + 9 \times 10^2 + 9 \times 10^1 + 8 \times 10^0$, или 1998. В обычном двоичном коде то же самое число представляется следующим образом:

$$\begin{array}{cccccccccccc} 2^{10} & 2^9 & 2^8 & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{array}$$

т.е. $1 \times 2^{10} + 1 \times 2^9 + 1 \times 2^8 + 1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$, или $b'111111001110^{1)}$. Точно так же можно представлять и дробные числа, при этом позициям, расположенным справа от десятичной точки, соответствуют отрицательные степени двойки. Так, двоичное число $b'1101.11'$ эквивалентно десятичному 13.75. Из примера видно, что двоичное представление чисел гораздо длиннее их десятичных эквивалентов — в среднем не менее чем в 3 раза. Однако 2-позиционный ключ гораздо проще 10-позиционного, поэтому двоичное представление предпочтительнее.

Биты любой n -разрядной двоичной последовательности могут образовывать в общей сложности 2^n комбинаций. При этом большинство компьютеров хранят и обрабатывают биты группами. Например, первый микропроцессор Intel 4004 обрабатывал данные по четыре бита (*полубайт*) за раз. Большинство современных процессоров оперируют с 8-битными (*байт*), 16-битными (*слово*), 32-битными (*двойное слово*) и 64-битными (*счетверенное слово*) блоками. Характеристики некоторых из указанных групп перечислены в **Табл. 1.2**. Приведенные названия являются в какой-то мере стандартом де-факто, однако иногда встречаются и другие варианты.

Как и в десятичной системе счисления, большие двоичные числа часто выражаются с использованием приставок К (кило), М (мега) и Г (гига). В двоичной системе приставка «кило» соответствует множителю 2^{10} , например 64 Кбайт (или КБ) памяти. Аналогично, приставка «мега» соответствует множителю $2^{20} = 1\,048\,576$, например дискета объемом 1.44 Мбайт (или МБ). Точно так же емкость 20 Гбайт (или ГБ) винчестера составляет $20 \times 2^{30} = 21\,474\,836\,480$ байт. Естественно, 1-й вариант записи предпочтительнее.

Таблица 1.2. Наиболее распространенные группировки битов

Тип	Число битов	Десятичное значение	Двоичное значение
Бит	1	0,1	0,1
Полубайт	4	0...15	0000...1111
Байт	8	0...255	0000 0000...1111 1111
Слово	16	0...65 535	0000 0000 0000 0000...1111 1111 1111 1111
Двойное слово	32	0...4 294 967 295	0000 0000 0000 0000 0000 0000 0000 0000... ...1111 1111 1111 1111 1111 1111 1111 1111

Длинные двоичные числа очень неудобны для человеческого восприятия. В **Табл. 1.2** двоичные числа специально были разбиты на 4-битные группы, чтобы их удобнее было читать. Предположим теперь, что адрес какого-либо элемента в памяти равен $b'1000\,1100\,0001\,0100\,0000\,1010'$. Если каждой комбинации из четы-

¹⁾ Запись вида $b'...$ не универсальна, часто используются и другие варианты нотации, например $(1111011110)_2$. Если основание очевидно (известно из контекста), признак основания может опускаться.

рех битов сопоставить свой символ (0...9 и A...F, как показано в **Табл. 1.3**), то этот адрес можно будет записать в виде $h'8C140A^{1)}$, что гораздо удобнее. Этот код называется *шестнадцатеричным*, поскольку для обозначения разрядов в нем используется 16 символов. Шестнадцатеричные числа (числа с основанием 16) — это вполне жизнеспособные самостоятельные числа, а не просто какое-то дополнительное представление двоичных чисел. Разряды шестнадцатеричного числа имеют веса соответственно $16^0, 16^1, 16^2, \dots, 16^n$ ²⁾.

Двоично-десятичный код (Binary-Coded Decimal — BCD) является гибридом двоичного и десятичного представлений, широко используемым при работе с портами ввода/вывода цифровых устройств (см. Пример 11.5 на стр. 360). При таком представлении каждый десятичный разряд заменяется своим двоичным эквивалентом. Так, число 1998 записывается в виде (0001 1001 1001 1000)_{BCD}. Это представление очень сильно отличается от эквивалентного обычного двоичного кода, несмотря на то, что при его записи тоже используются только нули и единицы. Как и следовало ожидать, выполнение арифметических операций с числами, записанными таким образом, представляет собой не простую задачу. Поэтому, как правило, на входе системы BCD-числа преобразовываются в обыкновенные двоичные числа, а после обработки преобразовываются обратно (см. Программу 5.7 на стр. 159).

Таблица 1.3. Различные формы записи чисел от 0 до 20

Десятичная система	Двоичная система	Шестнадцатеричная система	Двоично-десятичный код
00	00000	00	0000 0000
01	00001	01	0000 0001
02	00010	02	0000 0010
03	00011	03	0000 0011
04	00100	04	0000 0100
05	00101	05	0000 0101
06	00110	06	0000 0110
07	00111	07	0000 0111
08	01000	08	0000 1000
09	01001	09	0000 1001
10	01010	0A	0001 0000
11	01011	0B	0001 0001
12	01100	0C	0001 0010
13	01101	0D	0001 0011
14	01110	0E	0001 0100
15	01111	0F	0001 0101
16	10000	10	0001 0110
17	10001	11	0001 0111
18	10010	12	0001 1000
19	10011	13	0001 1001
20	10100	14	0010 0000

¹⁾ Это же шестнадцатеричное число можно записать как 8C140Ah, или 0x8C140A.

²⁾ Многие научные калькуляторы, в том числе и программа «Калькулятор» из состава Microsoft Windows, могут производить вычисления в двоичной и шестнадцатеричной системах счисления.

Двоичная арифметика¹⁾ подчиняется тем же правилам, что и более привычная для вас арифметика по основанию 10. Более того, это утверждение справедливо для любой системы счисления. Простейшей арифметической операцией является операция *сложения*, представляющая сокращенную форму записи операции нахождения общего количества чего-либо по сравнению с более примитивным процессом счета или прибавления единицы. Так, запись $2 + 4 = 6$ гораздо удобнее, чем $2 + 1 = 3$, $3 + 1 = 4$, $4 + 1 = 5$, $5 + 1 = 6$. Однако при этом необходимо помнить правила сложения. Для десятичных чисел существует 45 правил, если учесть, что порядок слагаемых не важен, — от $0 + 0 = 0$ до $9 + 9 = 18$. Двоичное сложение гораздо проще, поскольку подчиняется всего трем правилам:

$$\begin{array}{rcl} 0 + 0 & = & 0 \\ 0 + 1 & \} & \\ 1 + 0 & \} & = 1 \\ 1 + 1 & \} & = 10 \quad (0 \text{ и } 1 \text{ в переносе}) \end{array}$$

Сначала эти правила применяются к самым младшим значащим битам (Least Significant Bit — LSB); при возникновении *переноса* он передается в бит, расположенный левее. Процесс вычисления заканчивается старшими значащими битами (Most Significant Bit — MSB). Если из этой позиции происходит перенос, то именно он становится самым старшим битом суммы. Например:

а) Десятичное число

$$\begin{array}{r} 1 \\ 0 \ 1 \\ 0 \ 0 \ 1 \\ 96 \quad \text{1-е слагаемое} \\ + 37 \quad \text{2-е слагаемое} \\ \hline \text{---} \text{---} \text{---} \quad \text{Переносы} \\ 133 \quad \text{Сумма} \end{array}$$

б) Двоичное число

$$\begin{array}{r} 1 \\ 2 \ 6 \ 3 \ 1 \\ 8 \ 4 \ 2 \ 6 \ 8 \ 4 \ 2 \ 1 \\ 1100000 \quad \text{1-е слагаемое} \\ + 0100101 \quad \text{2-е слагаемое} \\ \hline \text{---} \text{---} \quad \text{Переносы} \\ 10000101 \quad \text{Сумма} \end{array}$$

Подобно тому как при сложении осуществляется прямой счет, операция *вычитания* соответствует обратному счету, при котором от исходного значения отнимаются единицы. Так, операция $8 - 5 = 3$ эквивалентна последовательности операций $8 - 1 = 7$, $7 - 1 = 6$, $6 - 1 = 5$, $5 - 1 = 4$, $4 - 1 = 3$.

В соответствии с известной методикой вычитания десятичных чисел правила вычитания применяются и к двоичным числам, начиная с младших битов и заканчивая старшими. Для каждого бита, в котором из меньшего числа вычитается большее, из ближайшего старшего бита *занимается* единица. С учетом заема правила вычитания в двоичной системе имеют вид

¹⁾ Обычный двоичный код иногда называют кодом «8-4-2-1» по значению весов четырех младших разрядов.

$$\begin{aligned}
 0 - 0 &= 0 \\
 {}^10 - 1 &= 1 \quad \text{Из старшего бита занимается 1} \\
 1 - 0 &= 1 \\
 1 - 1 &= 0
 \end{aligned}$$

Например:

а) Десятичное число

$$\begin{array}{r}
 \begin{array}{c} 1 \\ 0 \end{array} 1 \\
 96 \text{ Уменьшаемое} \\
 - 37 \text{ Вычитаемое} \\
 \hline
 \text{Заемы} \\
 59 \text{ Разность}
 \end{array}$$

б) Двоичное число

$$\begin{array}{r}
 \begin{array}{c} 6 \ 3 \ 1 \\ 4 \ 2 \ 6 \ 8 \ 4 \ 2 \ 1 \end{array} \\
 1100000 \text{ Уменьшаемое} \\
 - 0100101 \text{ Вычитаемое} \\
 \hline
 \text{Заемы} \\
 0111011 \text{ Разность}
 \end{array}$$

Несмотря на то что эти знакомые методы прекрасно работают, при реализации их в цифровых схемах возникает ряд проблем:

- Что делать, если вычитаемое меньше уменьшаемого?
- Как нам различать положительные и отрицательные числа?
- Можно ли выполнить вычитание с помощью блока суммирования?

Чтобы понять суть описанных проблем, взгляните на следующий пример:

а) Десятичное число

$$\begin{array}{r}
 37 \text{ Уменьшаемое} \\
 - 96 \text{ Вычитаемое} \\
 \hline
 41 \text{ Разность } (-59)
 \end{array}$$

б) Двоичное число

$$\begin{array}{r}
 0100101 \text{ Уменьшаемое} \\
 - 1100000 \text{ Вычитаемое} \\
 \hline
 1000101 \text{ Разность } (-0111011)
 \end{array}$$

Обычно, если мы знаем, что уменьшаемое меньше вычитаемого, мы меняем операнды местами и добавляем знак минуса к результату, т.е. вычисляем выражение $-(\text{вычитаемое} - \text{уменьшаемое})$. Если мы не выполним такой перестановки, как показано в примере (а), приведенном выше, то результат окажется неверным. На самом деле число 41 является правильным в том смысле, что представляет собой разность между числом 59 (правильный результат) и 100. То есть число 41 представляет собой *дополнительный код* числа 59 в десятичной системе (10 's complement). Более того, сам факт заема из старшего разряда числа указывает на то, что результат операции отрицателен и представлен соответственно в дополнительном коде. Для преобразования числа, представленного в дополнительном коде, в «нормальный» вид достаточно просто проинвертировать каждый десятичный разряд и к полученному значению прибавить единицу. Инвертирование десятичного разряда заключается в вычитании его значения из 9. Таким образом, дополнительный код числа 3941 в десятичной системе равен -6059 :

$$\overline{3941} \rightarrow 6058; +1 = -6059.$$

Как бы там ни было, единственной причиной, по которой мы не оставляем отрицательные числа в дополнительном коде, является непривычность для нас такого представления чисел.

Разумеется, использование дополнительного кода для представления отрицательных значений применимо и к двоичным числам. Причем, простота инвертирования ($0 \rightarrow 1$, $1 \rightarrow 0$) делает этот метод очень привлекательным. Обратимся к приведенному выше примеру:

$$\overline{1000101} \rightarrow 0111010; + 1 = -0111011.$$

И опять же отрицательные числа следует оставлять в *дополнительном коде* (2's complement)¹⁾. Обратите внимание, что операция преобразования в дополнительный код является обратной, т.е.

дополнительный код \Leftrightarrow прямой код.

При работе с десятичными числами для обозначения положительных и отрицательных чисел используются знаки «+» и «-» соответственно. В системе же с двумя состояниями мы можем оперировать только единицами и нулями. Тем не менее, взглянув на последний пример, можно получить ключ к решению этой проблемы. Как уже было сказано, отрицательное значение получается в результате заема в старший разряд числа. Так что мы можем использовать этот разряд в качестве *знакового бита* (sign bit), причем 0 будет эквивалентен знаку «+», а 1 — знаку «-». Таким образом, число $b'11000101'$ будет соответствовать значению -59 , а $b'00111011'$ — значению $+59$ (в примерах знаковый бит выделен полужирным шрифтом). Преимущество такого представления заключается в том, что при любых арифметических операциях с ним можно обращаться так же, как и с обычным битом. При этом результат операции будет иметь верный знак:

а) Уменьшаемое
меньше вычитаемого

$$\begin{array}{r} \mathbf{01100000} \quad (+96) \\ - \mathbf{11011011} \quad (-37) \\ \hline \mathbf{00111011} \quad (+59) \end{array}$$

б) Уменьшаемое
больше вычитаемого

$$\begin{array}{r} \mathbf{00100101} \quad (+37) \\ - \mathbf{10100000} \quad (-96) \\ \hline \mathbf{11000101} \quad (-59) \end{array}$$

Из примера видно, что если отрицательное число представлено в дополнительном коде, то нам не нужно изобретать аппаратный «вычитатель», поскольку прибавление отрицательного числа эквивалентно вычитанию положительного. Другими словами, $A - B = A + (-B)$. Более того, если числа будут записаны в дополнительном коде, результаты всех последующих арифметических операций также будут в дополнительном коде.

¹⁾ Если вы введете в программе «Калькулятор» Microsoft Windows десятичное отрицательное число и переключитесь в двоичную систему, то это число будет отображено в дополнительном коде.

С арифметическими операциями над отрицательными числами, представленными в дополнительном коде, связаны две проблемы. Первая из этих проблем — *переполнение* (overflow). Она заключается в том, что при сложении двух положительных или двух отрицательных чисел может возникнуть переполнение в знаковом бите, например:

- а) Сумма двух положительных чисел получается отрицательной б) Сумма двух отрицательных чисел получается положительной

$$\begin{array}{r}
 01000 \quad (+8) \\
 + 01011 \quad (+11) \\
 \hline
 10011 \quad (-13!!!)
 \end{array}$$

$$\begin{array}{r}
 11000 \quad (-8) \\
 + 10101 \quad (-11) \\
 \hline
 01101 \quad (+13!!!)
 \end{array}$$

В примере (а) результат сложения $(+8) + (+11)$ равен -13 . В данном случае произошло переполнение из четвертого значащего бита в знаковый (в действительности число $10011b = 19$ является корректным результатом). В примере (б) показана та же ситуация при сложении двух отрицательных чисел. Переполнение может возникнуть только в том случае, если оба операнда имеют **одинаковые** знаковые биты. Поэтому для обнаружения переполнения следует отслеживать значение знакового бита результата, отличающееся от значения знаковых битов операндов. Логическая схема, реализующая обнаружение переполнения, показана на Рис. 1.5.

Вторая проблема касается выполнения арифметических операций над знаковыми операндами разной разрядности, например:

- а) Расширение положительного числа б) Расширение отрицательного числа

$$\begin{array}{r}
 00011001 \quad (+25) \\
 + \quad 0011 \quad (+03) \\
 \hline
 \text{????}
 \end{array}$$

↓

$$\begin{array}{r}
 00011001 \quad (+25) \\
 + 00000011 \quad (+03) \\
 \hline
 00011100 \quad (+28)
 \end{array}$$

$$\begin{array}{r}
 00011001 \quad (+25) \\
 + \quad 1101 \quad (-03) \\
 \hline
 \text{????}
 \end{array}$$

↓

$$\begin{array}{r}
 00011001 \quad (+25) \\
 + 11111101 \quad (-03) \\
 \hline
 00010110 \quad (+22)
 \end{array}$$

В обоих примерах показано сложение 8-битного числа с 16-битным. Если первый операнд положителен, его разрядность можно увеличить до 16 бит, заполнив свободные позиции нулями. Если же требуется расширить отрицательное число, то решение уже не так очевидно. В этом случае расширение числа произ-

водится путем заполнения пустых разрядов единицами. Общее правило звучит так: при расширении данных дополнительные разряды слева следует заполнять знаковым битом. Этот метод называется *расширением знака* (sign extension).

Умножение числа на n -ю степень двойки реализуется сдвигом исходного значения на n позиций влево. Таким образом, последовательность операций 00110 (6) << 01100 (12) << 11000 (24) эквивалентна умножению числа 6 на 2^2 ; оператор «<<» используется для обозначения сдвига влево. Это же правило применимо и к отрицательным числам:

$$\text{а) } +3 \times 8 = +24$$

$$000000011 \text{ (3)}$$

<<

$$000000110 \text{ (6)}$$

<<

$$000001100 \text{ (12)}$$

<<

$$000011000 \text{ (24)}$$

$$\text{б) } -3 \times 8 = -24$$

$$111111101 \text{ (-3)}$$

<<

$$111111010 \text{ (-6)}$$

<<

$$111110100 \text{ (-12)}$$

<<

$$111101000 \text{ (-24)}$$

$$\text{в) } +3 \times 10 = 30$$

$$000000110 \text{ (3} \times 2\text{)}$$

$$+ 000011000 \text{ (3} \times 8\text{)}$$

$$000011110 \text{ (3} \times 10 = 30\text{)}$$

Смена значения знакового бита означает переполнение в старшем бите модуля числа. Некоторые компьютеры (микропроцессоры) поддерживают операцию *арифметического сдвига влево*, которая сигнализирует о такой ситуации в отличие от обычной операции *логического сдвига влево*, используемой для сдвига беззнаковых чисел.

Умножение на число, не являющееся степенью двойки, можно реализовать, комбинируя операции сдвига и суммирования. В частности, как показано в предыдущем примере (в), выражение 3×10 вычисляется следующим образом:

$$(3 \times 8) + (3 \times 2) = (3 \times 10) \text{ или } (3 \ll 3) + (3 \ll 1).$$

Аналогичным образом *деление* числа на n -ю степень двойки реализуется сдвигом значения на n позиций вправо, т.е. 1100 (12) >> 0110 (6) >> 0011 (3) >> 0001.1 (1.5). Этот же способ применим к знаковым числам:

$$\text{а) } +15/8 = 1.875$$

$$01111.000 \text{ (+15)}$$

>>

$$00111.100 \text{ (+7.5)}$$

>>

$$00011.110 \text{ (+3.75)}$$

>>

$$00001.111 \text{ (+1.875)}$$

$$\text{б) } -15/8 = -1.875$$

$$10001.000 \text{ (-15)}$$

>>

$$11000.100 \text{ (-7.5)}$$

>>

$$11100.010 \text{ (-3.75)}$$

>>

$$11110.001 \text{ (-1.875)}$$

$$\text{в) } 15/10 = 1.5$$

$$\begin{array}{r} 0001.1 \\ 1010 \overline{) 1111.0} \\ \underline{-1010} \\ 0101 \\ \underline{-101.0} \\ 000.0 \end{array}$$

Обратите внимание, что освободившиеся при сдвиге влево позиции заполняются не нулями, а содержимым знакового бита. Таким образом, при сдвиге положительных чисел слева вдвигаются нули, а при сдвиге отрицательных чисел — единицы. Данная операция известна как *арифметический сдвиг вправо*, в отличие от *логического сдвига вправо*, при котором всегда вдвигаются нули.

Деление на число, не являющееся степенью двойки, показано в примере (в). Эта операция осуществляется аналогично операции деления столбиком в десятичной системе. При ее выполнении по аналогии с умножением используется комбинирование операций сдвига и вычитания.

Арифметические действия — не единственные операции, которые можно осуществлять над двоичными числами. Английский математик Джордж Буль¹⁾ (George Boole) в середине 19-го столетия создал раздел алгебры, касающийся символической обработки логических отношений. Этот раздел алгебры, называемый *Булевой алгеброй*, оперирует величинами, которые могут иметь только два состояния: истина или ложь. В 30-х годах стало понятно, что этот раздел математики может быть с успехом использован для анализа коммутационных схем и, соответственно, устройств двоичной логики. Мы ограничимся рассмотрением базовых логических операций этой алгебры переключательных схем.

Инверсия, или операция НЕ (NOT), обозначается символом надчеркивания. Таким образом, выражение $f = \bar{A}$ означает, что переменная f является обратной величиной переменной A . То есть если $A = 0$, то $f = 1$, и, наоборот, если $A = 1$, то $f = 0$. На **Рис. 1.1, а** эта зависимость представлена в виде *таблицы истинности* (truth table). По определению двойная инверсия переводит переменную в первоначальное состояние: $\bar{\bar{f}} = f$ ²⁾.

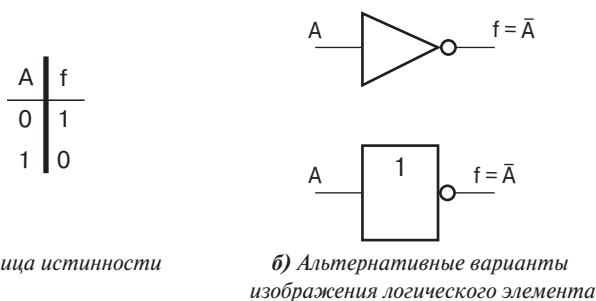


Рис. 1.1. Операция НЕ (NOT)

¹⁾ Джордж Буль — первый профессор математики Куинз-колледжа (Queen's College) в графстве Корк.

²⁾ Давным-давно, когда логические схемы реализовывались на дискретных компонентах, таких как диоды, резисторы и транзисторы, часто возникала проблема паразитных токов. При выполнении одной из лабораторных работ свечение выходной лампы получилось довольно тусклым, и преподаватель предположил, что два элемента НЕ, последовательно включенных в подозрительную линию, смогут предотвратить нежелательную утечку тока, не нарушив при этом логику работы схемы. Позже студенты пожаловались, что рекомендуемая мера не возымела никакого эффекта. При исследовании схемы преподаватель обнаружил два узелка на проблемном проводе, специально затянутых не до конца!

Как правило, реализации логических функций представляются с помощью абстрактных символов, а не подробных электрических схем. Общепринятое изображение элемента НЕ приведено на **Рис. 1.1, б**¹⁾. Кругик на изображении логических схем **всегда** означает инверсию и очень часто используется в сочетании с другими логическими элементами (см., например, **Рис. 1.2, в**).

Оператор И (AND) реализует функцию «все или ничего». Результат операции будет истинным только в том случае, если **все** n входов истинны. На **Рис. 1.2** имеются две входные переменные, и выражение для выходного значения записывается как $f = B \cdot A$, где символ « \cdot » — булевый оператор И²⁾. Количество входных переменных может быть любым, и в общем случае $f = A(0) \cdot A(1) \cdot A(2) \cdot \dots \cdot A(n)$. Операцию И иногда называют операцией логического умножения, поскольку (по аналогии с обычным умножением) результат этой операции между любым битом и 0 всегда будет равен 0.

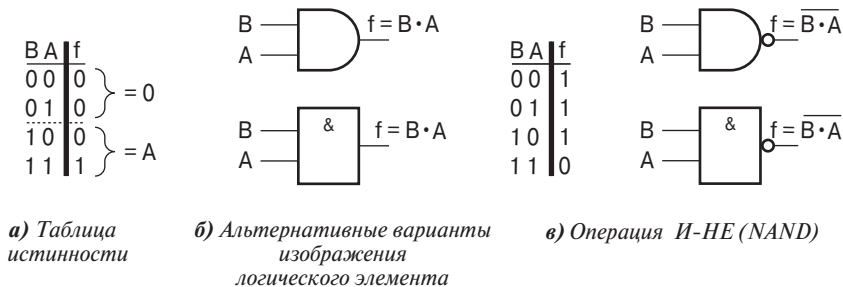


Рис. 1.2. Операция И (AND)

Если предположить, что вход В является управляющим входом, а вход А — входом данных, то, обратившись к таблице истинности, мы увидим, что при $B = 1$ на выходе будут присутствовать входные данные, а при $B = 0$ на выходе постоянно будет 0. Таким образом, эту схему можно рассматривать как управляемый вентиль. В общем случае термин *вентиль* применим к любой логической схеме, реализующей базовые логические операции.

В большинстве практических реализаций вентиля И используется инвертированный выход. Логическая функция такого элемента называется И-НЕ (NOT AND, или NAND), а ее изображение приведено на **Рис. 1.2, в**.

Действие оператора ИЛИ (OR) можно описать словом «что-нибудь». Результат этой операции будет истинным, если истинно хотя бы одно из входных значений (поэтому на символе изображено « ≥ 1 »). Хотя элемент, показанный на **Рис. 1.3**, имеет только два входа, операция ИЛИ применима к любому числу входных переменных. Часто операцию ИЛИ называют логическим сложением, соответственно в качестве математического оператора используется знак «+»³⁾:

¹⁾ Верхний символ используется в зарубежной литературе, а нижний — в отечественной. — *Примеч. пер.*

²⁾ Иногда для обозначения оператора И используется знак « \wedge ». — *Примеч. пер.*

³⁾ В отечественной литературе оператор ИЛИ часто обозначается также знаком « \vee ». — *Примеч. пер.*

$f = B + A$. Подобно тому как вентиль И позволяет обнаружить ситуацию, когда на всех входах присутствуют единицы, вентиль ИЛИ может использоваться для обнаружения ситуации «все нули». Использование его в этом качестве показано на **Рис. 2.20** (стр. 49), где 8-битное нулевое значение вызывает появление лог. 1 на выходе элемента ИЛИ-НЕ. Результат логического сложения любого бита с лог. 1 всегда будет равен лог. 1.

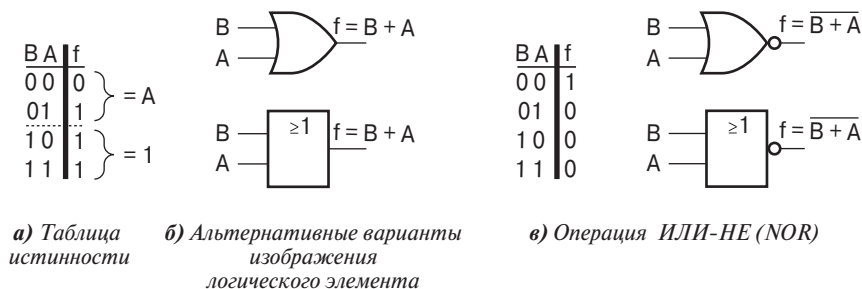


Рис. 1.3. Операция ИЛИ (OR)

Если предположить, что вход В является управляющим входом, а вход А — входом данных (или наоборот), то из **Рис. 1.3, а** видно, что данные проходят через вентиль при $B = 0$ и задерживаются (на выходе постоянно присутствует 1) при $B = 1$. Такое поведение отчасти похоже на инверсное действие функции И. В самом деле, функция ИЛИ может быть выражена через функцию И посредством двойственного соотношения $\overline{A+B} = \overline{B} + \overline{A}$. Из этого соотношения следует, что функцию ИЛИ-НЕ можно реализовать инвертированием сигналов, подаваемых на вход элемента И.

Мы познакомились с тремя основными логическими операторами: И, ИЛИ и НЕ. Однако существует еще одна операция, часто используемая в электронике, — операция Исключающее ИЛИ (eXclusive OR — XOR). Функция XOR истинна, если истинен только один из входов (поэтому на символе изображено «=1», см. **Рис. 1.4, б**). В отличие от обычной операции ИЛИ, при 1 на обоих входах на выходе будет 0.

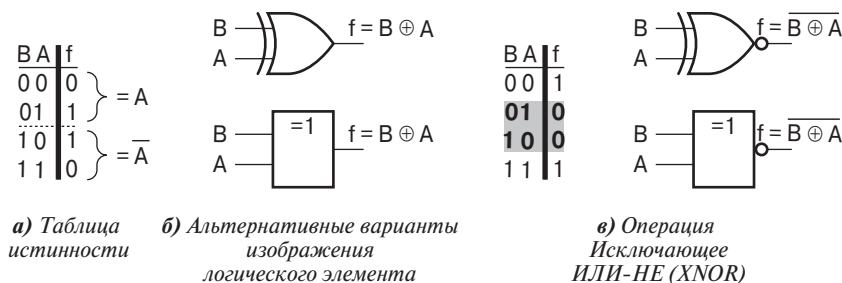


Рис. 1.4. Операция Исключающее ИЛИ (XOR)

Если предположить, что вход В — управляющий, а вход А — вход данных (или наоборот), тогда

- Если $V = 0$, то $f = A$ — данные с входа передаются на выход.
- Если $V = 1$, то $f = \bar{A}$ — выходной сигнал представляет собой инвертированный входной сигнал.

Таким образом, вентиль Иключающее ИЛИ может использоваться в качестве *программируемого инвертора*.

Другим полезным применением функции Иключающее ИЛИ можно назвать использование ее в качестве логического дифференциатора. Из таблицы истинности (Рис. 1.4, а) видно, что выход элемента Иключающее ИЛИ истинен только тогда, когда состояния обоих входов различны. Аналогично, из таблицы истинности оператора Иключающее ИЛИ-НЕ (XNOR), показанной на Рис. 1.4, в, видно, что выход такого элемента истинен при одинаковых сигналах на обоих входах. Таким образом, вентиль Иключающее ИЛИ-НЕ можно рассматривать в качестве 1-битного компаратора. Равенство двух n -битных значений можно проверить, объединив по И набор вентилях Иключающее ИЛИ-НЕ (см. Рис. 2.7 на стр. 37), каждый из которых реализует функцию $\overline{B_k \oplus A_k}$, т.е.

$$f_{A=B} = \sum_{k=0}^{n-1} \overline{B_k \oplus A_k}.$$

В качестве простого примера использования элементов Иключающее ИЛИ и Иключающее ИЛИ-НЕ рассмотрим задачу определения *переполнения* в знаковом бите (см. стр. 24). Эта ситуация возникает, если знаковые биты обоих операндов одинаковы ($S_B \oplus S_A$), а знаковый бит С результата отличается от них, скажем $S_B \oplus S_C$. Схема такого детектора, показанная на Рис. 1.5, описывается логической функцией:

$$\overline{(S_B \oplus S_A)} \cdot (S_B \oplus S_C).$$

И наконец, функцию Иключающее ИЛИ можно использовать для определения четного количества истинных входов. При каскадном соединении $n + 1$ вентилях Иключающее ИЛИ выходной сигнал будет равен 1, если входное n -битное число содержит четное число единичных битов. Добавляя к слову данных дополнительный бит, так чтобы общее число битов было четным, можно реализовать простейшую защиту от ошибок. Приемное устройство будет контролировать четность принимаемых данных, и любое несоответствие будет означать их повреждение.

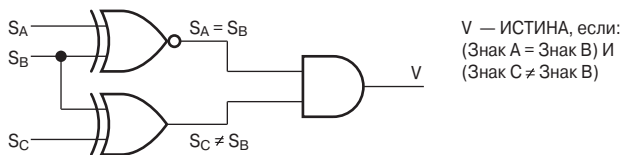


Рис. 1.5. Обнаружение переполнения в знаковом бите

Итак, мы с вами выяснили, что цифровая обработка данных заключается в пересылке, обработке и хранении двоичных значений. В этой главе мы несколько расширим представления, введенные в предыдущей главе, чтобы можно было приступить к рассмотрению собственно архитектуры компьютеров и микроконтроллеров. Мы познакомимся с несколькими важными логическими функциями, рассмотрим выпускаемые микросхемы, которые реализуют эти функции, а также их практическое применение.

Прочитав эту главу, вы:

- Познакомитесь с областями применения и характеристиками выходных каскадов с активной подтяжкой (двухтактный выход), с открытым коллектором и с тремя состояниями.
- Поймете логическую структуру и назначение дешифратора.
- Познакомитесь с интегральной микросхемой, представляющей собой набор элементов Исключающее ИЛИ-НЕ и использующейся для определения равенства двух значений.
- Поймете, как можно реализовать на логических элементах 1-битный сумматор и как его можно доработать для сложения двух n -битных чисел.
- Разберетесь, почему АЛУ имеет такое большое значение для программируемых систем.
- Ознакомитесь со структурой и областями применения постоянных запоминающих устройств (ПЗУ).
- Поймете, как из двух логических элементов, объединенных перекрестными связями, можно создать RS-триггер.
- Разберетесь, чем отличается D-защелка от D-триггера.
- Поймете, как из набора D-триггеров или защелок можно реализовать регистр.
- Узнаете, как с помощью каскадного соединения D-триггеров можно реализовать сдвиговый регистр.
- Поймете, как можно использовать D-триггер в качестве делителя на 2 и как посредством каскадного соединения D-триггеров можно реализовать двоичный счетчик.

- Узнаете, как с помощью связки АЛУ/регистр можно реализовать блок аккумулятора процессора.
- Разберетесь в принципах работы оперативного запоминающего устройства (ОЗУ).

В первых интегральных микросхемах, появившихся в конце 60-х годов, реализовывались главным образом логические элементы И-НЕ, ИЛИ-НЕ и НЕ. Наиболее популярным семейством логических микросхем тогда были, да и сейчас в какой-то мере остаются микросхемы 74-й серии, построенные по технологии ТТЛ (транзисторно-транзисторная логика). Эта серия была разработана фирмой Texas Instruments и впоследствии скопирована всеми ведущими производителями микросхем.

Микросхема 74LS00^{1),2)} содержит четыре двухвходовых элемента И-НЕ, объединенные в 14-выводном корпусе. Для питания микросхемы используется напряжение 5 ± 0.25 В, прикладываемое между выводами V_{CC} ³⁾ (обычно около 5 В) и GND. Напряжения логических уровней для этой серии составляют: 2.4...5 В — для ВЫСОКОГО уровня и 0...0.4 В — для НИЗКОГО. Для большинства семейств логических микросхем требуется напряжение питания 5 В, однако существуют и 3-вольтовые версии. При этом большинство КМОП-микросхем могут работать в диапазоне питающих напряжений от 3 до 15 В.

Цоколевка микросхемы 74LS00 в корпусе DIP показана на **Рис. 2.1, а**. Функция этой микросхемы полностью описывается четырьмя двухвходовыми элементами И-НЕ в положительной логике, поскольку НИЗКИЙ и ВЫСОКИЙ логические уровни эквивалентны логическим значениям 0 и 1. Если же принять, что 0 соответствует ВЫСОКОМУ уровню, а 1 — НИЗКОМУ (отрицательная логика), то микросхема будет выполнять функцию четырех двухвходовых элементов ИЛИ-НЕ. На изображениях логических элементов по стандарту ANSI/IEC⁴⁾ НИЗКИЙ уровень обозначается символом полярности ∇ (см. **Рис. 2.1, б**). Таким образом, изображение символа И-НЕ по стандарту ANSI/IEC основано на **реальном** функционировании схемы. В данном случае логика работы схемы совпадает с функцией И-НЕ в терминах положительной логики. Оператор & (И), изображенный в верхнем прямоугольнике, относится и к остальным трем элементам.

¹⁾ Символы «LS» означают «low-power shottky transistor» (маломощные ТТЛ ИС с диодами Шоттки). Существуют и другие разновидности этой серии, такие как ALS (усовершенствованные маломощные ТТЛ ИС с диодами Шоттки), AS (усовершенствованные ТТЛ ИС с диодами Шоттки) и HC (быстродействующие КМОП ИС). Эти микросхемы отличаются быстродействием и потреблением, однако все микросхемы с одинаковым номером выполняют одни и те же функции и имеют одинаковую цоколевку.

²⁾ Отечественный аналог — микросхема К555ЛА3. — *Примеч. пер.*

³⁾ Исторически сложилось так, что положительный вывод источника питания в цифровых ИС обозначается как V_{CC} (символ «С» взят потому, что питание подается на коллектор биполярного транзистора). Аналогичным образом ИС, построенные по технологии КМОП, используют обозначение V_{DD} (символ «D» указывает на напряжение, подаваемое на сток). Вывод общего провода обычно обозначается как «GND», однако иногда используются обозначения V_{EE} (для эмиттера) или V_{SS} (для истока).

⁴⁾ Национальный Институт Стандартизации США/Международная Электротехническая Комиссия.

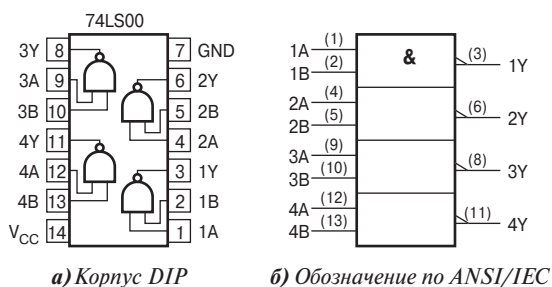


Рис. 2.1. Микросхема 74LS00 (К555ЛА3)

Выходы логических элементов микросхемы 74LS00 построены по *двухтактной* схеме. При такой структуре выходного каскада каждый из уровней формируется путем подключения выхода через низкоомный ключ к линии V_{CC} или GND соответственно. На Рис. 2.2, а эти ключи изображены в виде обычных переключателей, хотя на самом деле они, разумеется, выполнены на транзисторах.

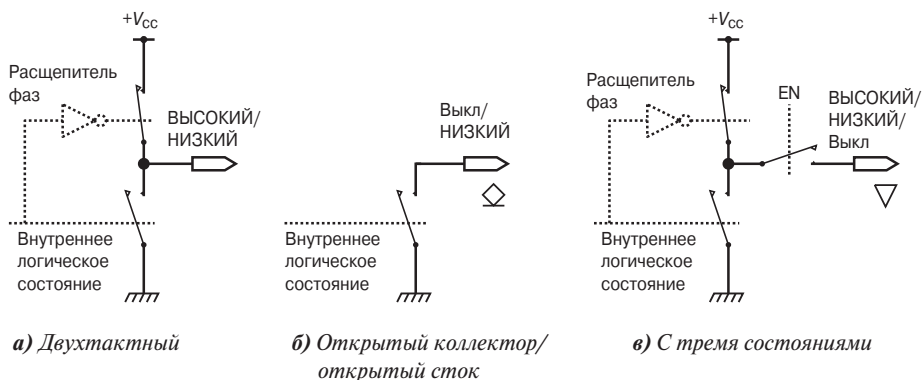


Рис. 2.2. Типы выходных каскадов

В логических микросхемах (например, таких как 74LS00) изменение состояния выхода происходит за время около $10\text{ нс}^{1)}$. Чтобы получить такие значения, емкости всех соединительных проводников и входов других микросхем должны быстро разряжаться. Главным образом именно по этой причине в большинстве цифровых микросхем используется двухтактный выход (называемый также выходом с активной подтяжкой — *active pull-up*). Однако в некоторых ситуациях преимущество имеют выходные каскады других типов. Конфигурация *открытый коллектор* (или открытый сток), показанная на Рис. 2.2, б, обеспечивает «жесткий» НИЗКИЙ уровень, при этом состояние ВЫСОКОГО уровня соответствует разомкнутой цепи. Напряжение ВЫСОКОГО уровня может формироваться под-

¹⁾ Одна наносекунда равна 10^{-9} с, так что за одну секунду может произойти 100 000 000 переключений.

ключением внешнего резистора либо к линии V_{CC} , либо к отдельной шине питания. Роль подобного резистора могут выполнять некоторые устройства, такие как реле, лампы накаливания или светодиоды. Выходной транзистор таких каскадов часто имеет большую, чем обычно, нагрузочную способность по напряжению и/или току.

Один из наиболее интересных для нас вариантов применения выхода с *открытым коллектором* показан на **Рис. 2.3**. В этой схеме четыре элемента с выходом типа «открытый коллектор» подключены к **одному и тому же** подтягивающему резистору. Обратите внимание на символ \diamond , используемый для обозначения выхода с открытым коллектором. Предположим, что на рисунке изображены четыре периферийных устройства, любое из которых может обращаться к процессору (компьютеру или микроконтроллеру). Если этот процессор имеет только один вход для внешнего сигнала прерывания, то четыре сигнальные линии от устройств должны быть объединены вместе по схеме *монтажное ИЛИ*, как показано на рисунке. Когда все сигнальные линии находятся в неактивном состоянии (лог. 0), выходы всех буферных элементов НЕ выключены (**ВЫСОКИЙ** уровень) и общая линия подтянута к V_{CC} резистором R_L . Если **какая-либо** из сигнальных линий становится активной (лог. 1), скажем, линия Sig_1, то на выходе соответствующего буфера появляется **НИЗКИЙ** уровень. В результате, независимо от состояния остальных сигнальных линий, общая линия переключается в состояние **НИЗКОГО** уровня, прерывая таким образом работу процессора.

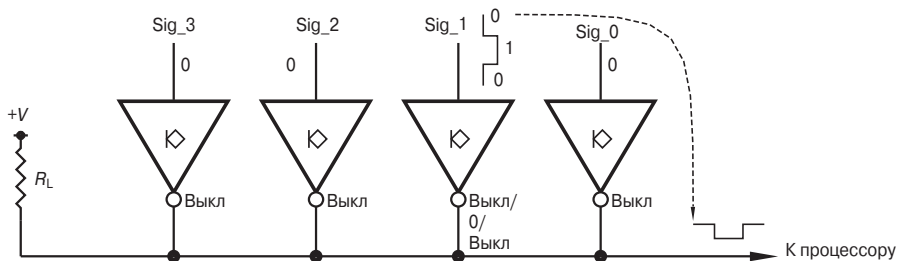


Рис. 2.3. Буферы с открытым коллектором управляют общей линией

Выходной каскад третьего типа (*с тремя состояниями*), приведенный на **Рис. 2.2, в**, обладает свойствами выходов обоих рассмотренных типов. При разрешенном выходе логические состояния формируются обычным образом, т.е. выдачей **ВЫСОКОГО** и **НИЗКОГО** напряжения. При запрещении выхода он становится разомкнутой цепью, независимо от функционирования внутренней логической схемы и любых изменений на ее входах. Выход с тремя состояниями обозначается символом ∇ .

В качестве примера использования выхода указанного типа рассмотрим ситуацию, показанную на **Рис. 2.4**. В данном случае основному контроллеру требуется прочитать данные с одного из нескольких устройств, подключенных к нему группой общих линий. Поскольку эта магистраль, или, иначе, *шина данных*, является

общим ресурсом, в любой момент времени доступ к шине предоставляется только выбранному устройству. Доступ должен быть закрыт сразу же после считывания данных, с тем чтобы шиной могло воспользоваться другое устройство. Как показано на рисунке, все выходы, подключаемые к шине, обозначаются символом ∇ . После выбора устройства управление линиями шины будет осуществляться **только** активными логическими уровнями. Микросхема двойного 4-битного буфера с тремя состояниями 74LS244¹⁾ имеет выходы с повышенной нагрузочной способностью (обозначаемые символом \triangleright), специально предназначенные для работы на длинных линиях, имеющих большую емкость.

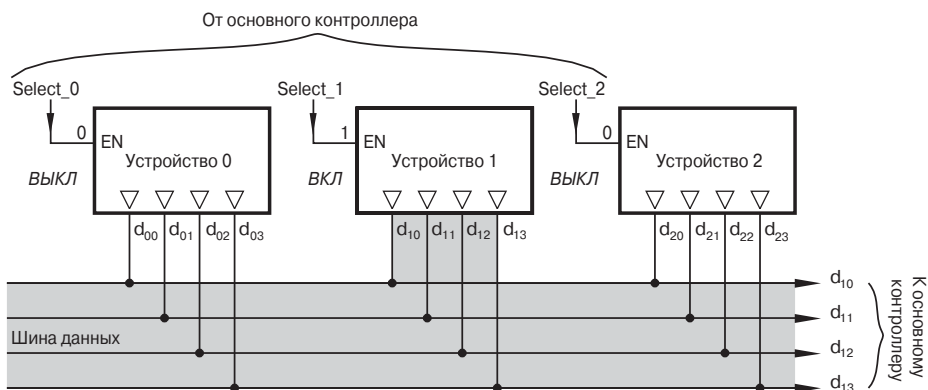


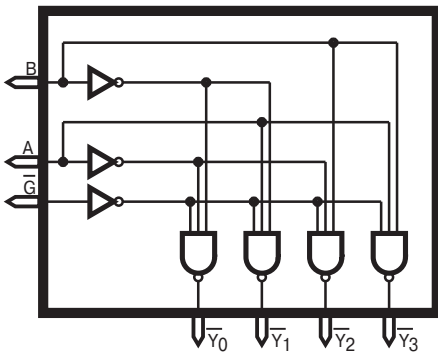
Рис. 2.4. Совместное использование шины

Интегральные микросхемы, содержащие до 12 логических элементов, относятся к микросхемам малой степени интеграции. Если в корпусе микросхемы содержится до 100 логических элементов, то она относится к классу микросхем средней степени интеграции; до 1000 — к классу больших интегральных схем или, сокращенно, БИС. Все микросхемы, имеющие более 1000 логических элементов, относятся к классу сверхбольших интегральных схем (СБИС). К последнему классу, в частности, относятся микросхемы памяти и микроконтроллеры.

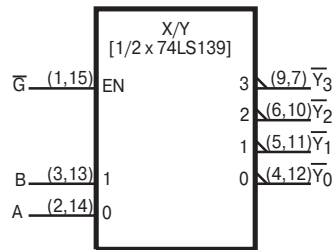
Изображенные на Рис. 2.5 микросхемы, содержащие определенным образом соединенные элементы И-НЕ, являются типичным примером интегральных микросхем средней степени интеграции. Если вспомнить, что на выходе элемента И-НЕ лог. 0 присутствует только в том случае, если на всех его входах присутствует лог. 1 (см. Рис. 1.2, в на стр. 27), то можно увидеть, что при любых сочетаниях сигналов на входах **выборки** $B \ A \ (2^1 \ 2^0)$ (Рис. 2.5, а) сигнал лог. 0 будет присутствовать на выходе только **одного** вентиля. Так, выход \bar{Y}_2 будет активным при $B \ A = 10$. После рассмотрения таблицы истинности становится понятно, что данная схема декодирует двоичный адрес $B \ A$ таким образом, что при подаче адреса n становится активным выход \bar{Y}_n . Полностью название микросхемы

¹⁾ Отечественный аналог — микросхема К555АП5. — Примеч. пер.

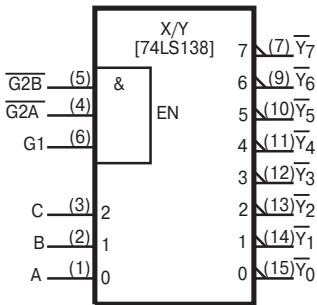
74LS139¹⁾ звучит так: *сдвоенный натуральный дешифратор 2 на 4*. Сдвоенным он называется потому, что в одном корпусе расположены две такие схемы. Символ X/Y обозначает преобразование кода X (натуральное двоичное число) в код Y (унарный — один из n). Вход разрешения \overline{G} подключен параллельно ко всем элементам. Таким образом, дешифратор выполняет свои функции только в том случае, если на входе \overline{G} присутствует НИЗКИЙ уровень (лог. 0). Если на входе \overline{G} присутствует ВЫСОКИЙ уровень, то независимо от состояния входов B и A (в таблице истинности эта ситуация обозначается символом «X» — безразличное состояние) все выходы устанавливаются в неактивное состояние (лог. 1). Пример использования микросхемы 74LS139 приведен на **Рис. 2.25** (стр. 54).



а) 74LS139 — сдвоенный дешифратор 2 на 4



G	B	A	Y ₀	Y ₁	Y ₂	Y ₃
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	X	X	1	1	1	1



б) 74LS138 — дешифратор 3 на 8

EN	C	B	A	Y ₀	Y ₁	Y ₂	Y ₃	Y ₄	Y ₅	Y ₆	Y ₇
0	0	0	0	0	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1
0	1	1	0	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	0
1	X	X	X	1	1	1	1	1	1	1	1

Рис. 2.5. Микросхемы дешифраторов 74LS138 (K555ИД7) и 74LS139 (K531ИД14)

Микросхема 74LS138²⁾, показанная на **Рис. 2.5, б**, похожа на только что рассмотренную, однако выполняет функцию дешифратора 3 на 8. При n -м значении на линиях адреса C B A ($2^2 2^1 2^0$) активным становится только один из восьми выходов \overline{Y}_n . Микросхема 74LS138 имеет три входа стробирования, формирующие

¹⁾ Отечественный аналог — микросхема K531ИД14. — *Примеч. пер.*

²⁾ Отечественный аналог — микросхема K555ИД7. — *Примеч. пер.*

внутренний сигнал разрешения $\overline{G2B} \cdot \overline{G2A} \cdot G1$. То есть функционирование микросхемы разрешено только в том случае, если на обоих входах $\overline{G2B}$ и $\overline{G2A}$ присутствует НИЗКИЙ уровень, а на входе $G1$ — ВЫСОКИЙ. Микросхема 74LS138 используется в схеме на **Рис. 11.12** (стр. 350) в качестве дешифратора линий порта микроконтроллера для подключения к одному порту нескольких устройств.

Приоритетный шифратор 74LS148¹⁾, показанный на **Рис. 2.6**, выполняет обратное преобразование. Подача на один из входов НИЗКОГО уровня вызывает появление на выходе эквивалентного 3-битного значения. Так, если вход $\overline{5} = 0$, то $\overline{a_2}\overline{a_1}\overline{a_0} = 010$ (число 101 в инверсной логике).

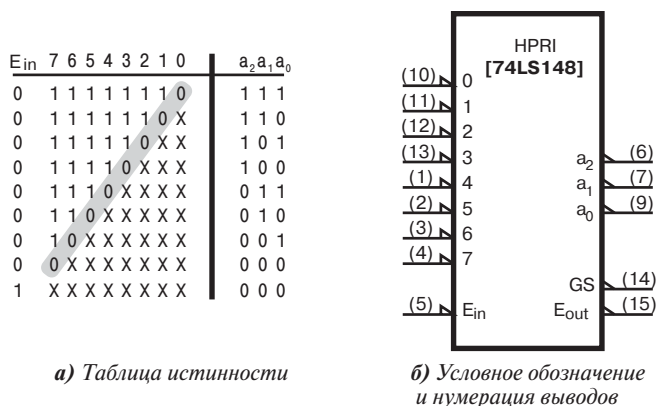


Рис. 2.6. Микросхема приоритетного шифратора 74LS148

Если активный сигнал присутствует на нескольких входах, то выходное значение соответствует входу с наибольшим номером. Так, если НИЗКИЙ уровень присутствует на обоих входах $\overline{5}$ и $\overline{3}$, то выходное значение все равно будет составлять 010. Символы HPRI на условном обозначении микросхемы, приведенной на **Рис. 2.6**, означают «наивысший приоритет» (Higest PRIority). Работа микросхемы разрешается при НИЗКОМ уровне на входе \overline{E}_{in} . Выходы \overline{E}_{out} и \overline{GS} используются при каскадном соединении микросхем для увеличения количества линий.

Большой класс ИС реализует различные арифметические операции. Матрица логических элементов, показанная на **Рис. 2.7**, используется для обнаружения равенства между двумя 8-битными числами P и Q . Каждый из восьми элементов Иключающее ИЛИ-НЕ формирует лог. 1, если оба входных бита P_n и Q_n одинаковы (мы уже встречались с этим элементом на стр. 28). Соответственно, НИЗКИЙ уровень на выходе элемента И-НЕ появится только в том случае, если **все** 8 пар битов одинаковы. Микросхема *компаратора* 74LS688 имеет также вход \overline{G} , сигнал с которого подается на один из входов элемента И-НЕ и выполняет функцию глобального разрешения.

На условном обозначении микросхемы по стандарту ANSI/IEC, приведенном на **Рис. 2.7, б**, функция сравнения указывается аббревиатурой COMP. Префикс

¹⁾ Отечественный аналог — микросхема К555ИБ1. — *Примеч. пер.*

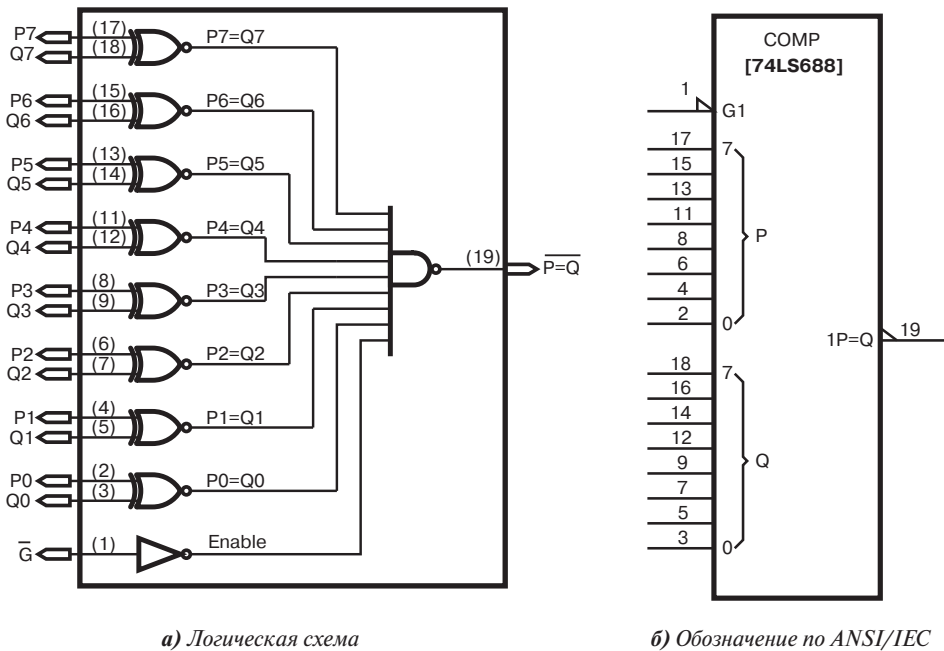


Рис. 2.7. Микросхема 8-битного компаратора 74LS688

«1» в обозначении выхода указывает на то, что выполнение операции « $P = Q$ » зависит от входа, обозначенного тем же номером, т.е. G1. Таким образом, вход разрешения G1 управляет выходом $1P = Q$ (и вход, и выход — с активным НИЗКИМ уровнем).

Одной из первых функций, реализованных в ИС помимо обычных логических элементов, было сложение. В таблице истинности, показанной на Рис. 2.8, а, приведены значения бита суммы S и флага переноса C_1 , образующихся при сложении двух битов A и B и бита переноса из предыдущего разряда C_0 . Например, из 6-й строки таблицы следует, что при сложении двух единиц и 0-го переноса сумма будет равна 0, а перенос — 1 ($1 + 1 + 0 = {}^10$). Для реализации этой строки таблицы нам нужно распознать комбинацию битов 110, описываемую уравнением $A \cdot B \cdot \bar{C}_0$. Эту операцию выполняет 6-й элемент схемы. Таким образом, мы просто объединяем по ИЛИ все возможные комбинации входных переменных:

$$S = (\bar{A} \cdot \bar{B} \cdot C_0) + (\bar{A} \cdot B \cdot \bar{C}_0) + (A \cdot \bar{B} \cdot \bar{C}_0) + (A \cdot B \cdot C_0)$$

$$C_1 = (\bar{A} \cdot B \cdot C_0) + (A \cdot \bar{B} \cdot C_0) + (A \cdot B \cdot \bar{C}_0) + (A \cdot B \cdot C_0)$$

Применяя такую схему для **каждого** разряда и подключая при этом выход переноса разряда с номером $k - 1$ к входу переноса разряда с номером k , мы сможем выполнять сложение любых n -битных чисел.

На **Рис. 2.8, б** показана структурная схема микросхемы 74LS283¹⁾, которая складывает два 4-битных числа за 25 нс. На практике для формирования итогового бита переноса C_4 используется дополнительная схема, чтобы избежать задержек, вызванных прохождением битов переноса через все стадии суммирования, от младшего бита к старшему. Несколько (n) микросхем 74LS283 можно каскадировать для реализации функции сложения слов разрядностью $4 \times n$. Таким образом, две микросхемы 74LS283 выполняют 16-битное сложение за 45 нс (учитывая дополнительную задержку распространения переноса между двумя микросхемами).

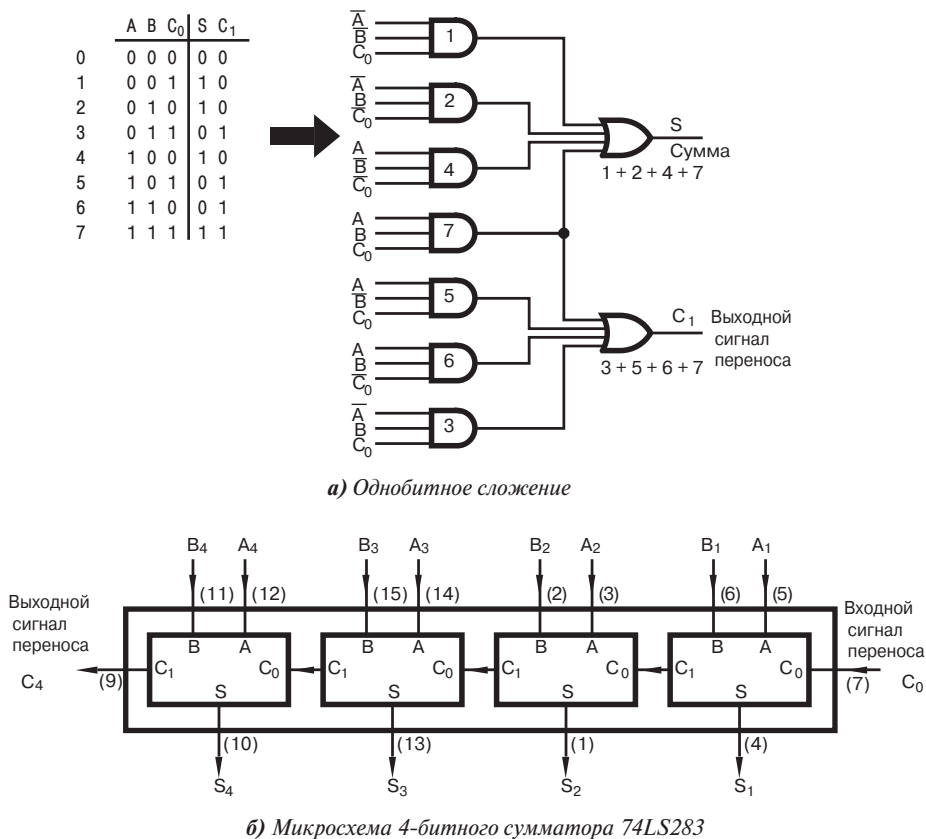


Рис. 2.8. Сложение

Разумеется, сумматоры можно использовать и для вычитания, если перевести операнды в дополнительный код. Схему сумматора/вычитателя можно реализовать при помощи набора логических элементов Исключающее ИЛИ, выступающих в роли *программируемых инверторов* (см. стр. 28). Вход выбора режима \overline{ADD}/SUB , управляющий этими инверторами в схеме на **Рис. 2.9**, подключен также к входу переноса, что вызывает добавление единицы в режиме вычитания.

¹⁾ Отечественный аналог — микросхема К555ИМ6. — *Примеч. пер.*

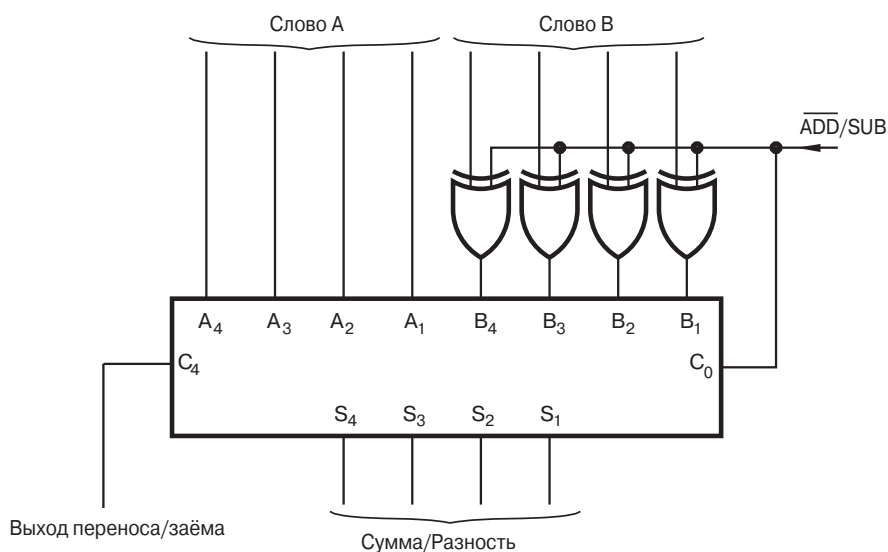
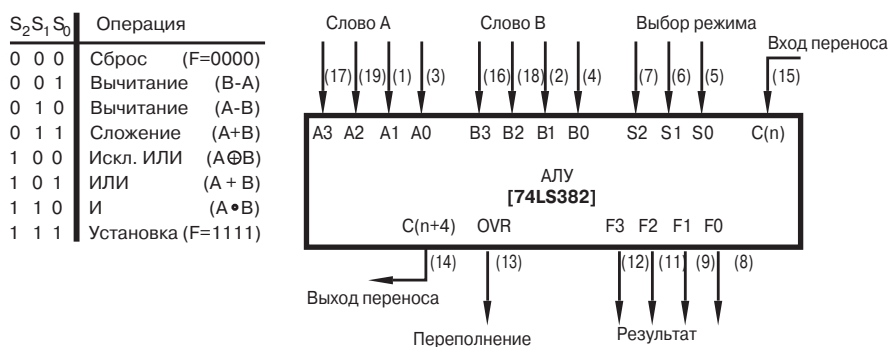


Рис. 2.9. Реализация программируемого сумматора/вычитателя

Расширяя набор аргументов, мы постепенно придем к *арифметико-логическому устройству* (АЛУ). АЛУ представляет собой схему, выполняющую определенный набор арифметических и логических операций над входными данными в соответствии со значением на входах выбора режима. Микросхема 74LS382, показанная на **Рис. 2.10**, выполняет 8 операций над двумя 4-битными числами. Выполняемая операция задается тремя битами выбора режима $S_0S_1S_2$ (**Рис. 2.10, а**). Кроме сложения и вычитания, это АЛУ выполняет также операции И, ИЛИ и Исключающее ИЛИ. Микросхема формирует даже признак *переполнения дополнительного кода* (см. стр. 24).



а) Таблица функций

б) Условное обозначение и нумерация выводов

Рис. 2.10. Микросхема АЛУ 74LS382

Как мы увидим чуть позже, АЛУ является «сердцем» любого компьютера или микропроцессора. Подавая на входы выбора режима некоторую последовательность двоичных значений, можно заставить АЛУ выполнить соответствующую последовательность операций. Эти *коды операций* хранятся во внешней памяти и последовательно считываются схемами управления.

Обычно последовательность кодов операций, составляющих программу, хранится в какой-либо БИС ПЗУ. Обратимся к структуре, показанной на **Рис. 2.11**. На этом рисунке изображен дешифратор 3 на 8, управляющий матрицей диодов 8×2 . Для каждой n -й комбинации сигналов, подаваемых на вход адреса, выбирается n -я строка. Если к этой строке подключен диод, то он открывается и на линии соответствующего столбца появляется НИЗКИЙ уровень. Соответственно, инвертирующий буфер с тремя состояниями формирует ВЫСОКИЙ уровень для каждого подключенного диода и НИЗКИЙ уровень для разомкнутой цепи. Таким образом, для каждого входного кода совокупность подключенных диодов определяет выходной код. Для наглядности матрица запрограммирована на реализацию полного 1-битного сумматора, изображенного на **Рис. 2.8, а**, однако может быть задана и *любая* другая функция трех переменных.

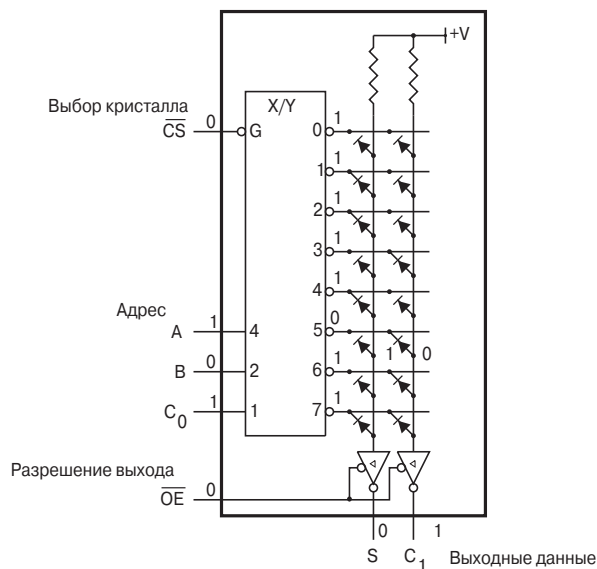


Рис. 2.11. Реализация 1-битного сумматора на ПЗУ

Диодная матрица, показанная на **Рис. 2.11**, называется *постоянным запоминающим устройством* (ПЗУ), поскольку «память» представляет собой комбинацию диодов, формируемую на этапе изготовления микросхемы. Старые устройства, имевшие, как правило, дешифратор и матрицу 32×8 , обычно выпускались в версиях, программируемых пользователем, в которых связи формировались плавкими перемычками. Требуемые диоды можно было исключить из матрицы при по-

мощи высокого напряжения. Такие устройства называются *программируемыми ПЗУ* (ППЗУ).

При реализации СБИС ППЗУ больших объемов, необходимых для хранения программ, плавкие перемычки очень неудобны. Например, небольшое ППЗУ 27С64¹⁾, показанное на **Рис. 2.12**, имеет объем, для формирования которого потребовалось бы 65 536 пар «перемычка — диод». То есть это относительно небольшое устройство способно хранить 8192 байта данных. В микросхеме 27С64 в качестве программируемой перемычки используется электрический заряд на плавающем затворе МОП-транзистора. Второй МОП-транзистор выполняет роль диода. Как и в варианте с плавкими перемычками, инжекция заряда в изолированный затвор осуществляется с помощью высокого напряжения. Образующееся электрическое поле удерживает МОП-транзистор в состоянии проводимости. Для полного рассасывания этого заряда требуется достаточно длительный срок в несколько десятков лет, однако это значение можно уменьшить до 20 мин, подвергая затвор интенсивному ультрафиолетовому излучению. Поэтому устройства, подобные 27С64, называют *стираемым ППЗУ* (СППЗУ). В корпусе микросхем, предусматривающих многократное использование, напротив кристалла размещается кварцевое окошко (см. **Рис. 2.12**), которое можно увидеть на фотографии, приведенной на стр. 15.

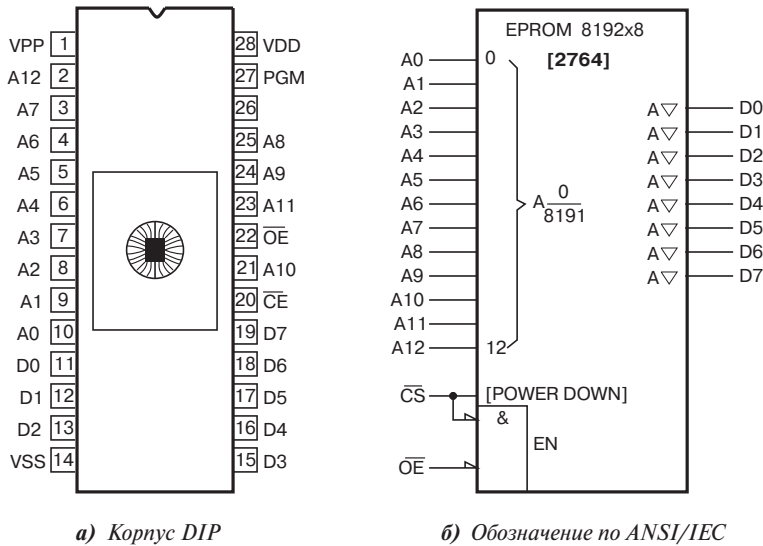


Рис. 2.12. Микросхема стираемого ППЗУ (СППЗУ) 27С64 (К573РФ4/6)

Программирование таких микросхем осуществляется специальными устройствами — программаторами. Версии микросхем без окошка называются однократно-программируемыми ПЗУ, поскольку их нельзя стереть после программи-

¹⁾ Отечественный аналог — микросхемы К573РФ4 и К573РФ6. — *Примеч. пер.*

две разновидности структур — электрически стираемые ППЗУ (ЭСППЗУ, или EEPROM) и FLASH-ППЗУ. В первом случае импульс отрицательного напряжения V_{pp} большой амплитуды приводит к просачиванию электронов из плавающего затвора. Обычно отрицательное напряжение формируется схемами, расположенными непосредственно на кристалле, что исключает необходимость в дополнительном источнике питания. FLASH-вариант ЭСППЗУ основан на эффекте инжектирования горячих электронов в затвор. Площадь, занимаемая ячейкой, в этом случае почти в 2 раза меньше обычной ячейки ЭСППЗУ, что увеличивает плотность упаковки памяти. Одна из промышленно выпускаемых микросхем EEPROM-памяти показана на **Рис. 12.26** (стр. 439).

Большинство современных ЭППЗУ/ЭСППЗУ довольно быстрые, со временем доступа около 150 нс. Процесс программирования происходит гораздо медленнее, около 10 мс на слово, однако это достаточно редкая операция. Программирование FLASH-памяти осуществляется почти в 1000 раз быстрее (на одну ячейку требуется около 10 мкс).

Все схемы, рассмотренные на данный момент, относились к классу *комбинационных*. Они не обладают «памятью» в том смысле, что значение их выходов зависит только от состояния входов в данный момент времени и совершенно не зависит от предыдущих событий, имевших место на входах. Такие же логические схемы, как защелки, счетчики, регистры и оперативная память (допускающая как чтение, так и запись), относятся к классу *последовательностных* схем. Состояние выходов таких схем зависит не только от текущего состояния входов, но и от предыстории сигналов на этих входах.

Возьмем обыкновенную кнопку, которая используется в дверном звонке. Звонок звонит, когда вы нажимаете на нее, и прекращает звонить, когда вы ее отпускаете. Такой ключ не обладает памятью.

Сравним эту кнопку с не менее обыкновенным выключателем. Вы нажимаете на него и свет загорается. Более того, он продолжает гореть даже тогда, когда вы убираете управляющее воздействие (палец). Чтобы выключить свет, вы должны перевести выключатель в выключенное состояние, и опять же, он останется в этом состоянии даже при отсутствии входного воздействия. Ключи такого типа называются *бистабильными*, поскольку они имеют два устойчивых состояния. Каждый такой ключ ведет себя как 1-битная ячейка памяти, которая может запоминать либо включенное, либо выключенное состояние.

В микросхемах оперативной памяти, таких как 6264 (**Рис. 2.26**), каждая бистабильная ячейка формируется с помощью двух перекрестно включенных транзисторов. Здесь мы не будем касаться конкретной реализации этих ячеек. Вместо этого рассмотрим два логических элемента ИЛИ-НЕ, объединенных перекрестными обратными связями (**Рис. 2.14**). Вспомним, что при появлении лог. 1 на каком-либо входе элемента ИЛИ-НЕ на его выходе появляется лог. 0 независимо от состояния остальных входов. Вооружившись этим знанием, попытаемся проанализировать схему:

- Если на вход S подать 1, то выход \overline{Q} переключится в 0. На обоих входах верхнего элемента появится 0, что приведет к появлению 1 на выходе Q . Если теперь на входе S снова появится 0, то нижний элемент останется в 0 (поскольку на входе обратной связи с вывода Q присутствует 1) и состояние выхода верхнего элемента также не изменится. Таким образом, триггер **устанавливается** при подаче положительного импульса на вход S .
- Если на вход R подать 1, то выход Q переключится в 0. На обоих входах нижнего элемента появится 0, что приведет к появлению 1 на выходе \overline{Q} . Если теперь на входе R снова появится 0, то **верхний** элемент останется в 0 (поскольку на входе обратной связи с вывода \overline{Q} присутствует 1) и состояние выхода нижнего элемента также не изменится. Таким образом, триггер **сбрасывается** при подаче положительного импульса на вход R .

При нормальном функционировании (предполагается, что оба входа не могут быть активными в один и тот же момент времени¹⁾) оба выхода дополняют друг друга, что отражено на условном графическом изображении триггера (Рис. 2.14, б).

Существует много различных реализаций бистабильных ячеек. Например, замена элементов ИЛИ-НЕ на элементы И-НЕ приведет к образованию \overline{RS} -триггера, в котором активным входным сигналом является лог. 0. В схеме, приведенной на Рис. 2.15, такой триггер используется для *подавления дребезга* контактов механического переключателя. Переключатели часто используются для управления входами логических схем. Однако большинство металлических контактов не могут замыкаться мгновенно, и при нажатии происходит их многократное размыкание/замыкание в течение нескольких десятков миллисекунд. То есть при использовании механического ключа, скажем, для прерывания работы компьютера/микроконтроллера результат будет совершенно непредсказуем.

В схеме на Рис. 2.15 установка триггера происходит при переводе ключа в верхнее положение. При размыкании контактов состояние триггера не меняется, благодаря чему пульсации на выходе схемы отсутствуют. При переводе ключа в нижнее положение схема работает аналогичным образом, только триггер при этом сбрасывается.

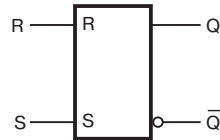
Дальнейшим развитием RS-триггера является *D-защелка*. В этом элементе выходной сигнал (Q) повторяет входной (D), если на входе управления C присутствует активный уровень (в данном случае — **ВЫСОКИЙ**), и сохраняет предыдущее значение при неактивном уровне на входе управления. Таким образом, D-защелку можно рассматривать как 1-битную ячейку памяти, запоминающую значение, которое присутствует на ее входе на момент завершения импульса управления.

На Рис. 2.16, б взаимное влияние входов D и C обозначается символами «C1» и «1D». Префикс «1» у D указывает на то, что этот вход зависит от любого сигнала

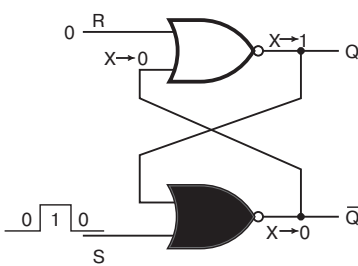
¹⁾ Если это произойдет, то оба выхода Q и \overline{Q} должны будут переключиться в 0. После снятия с входов активных сигналов триггер останется в одном из стабильных состояний, определяемом последовательностью снятия сигналов. Реакция триггера на одновременную подачу сигналов установки и сброса не определена и зависит от его конкретной реализации. Если, скажем, попытаться одновременно включить и выключить выключатель, то он может просто разломиться на две части!

RS	Q
0 0	Q (не изменяется)
0 1	1 (установка)
1 0	0 (сброс)

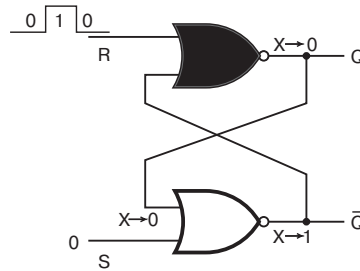
а) Таблица истинности RS-триггера



б) Условное обозначение с прямым и инверсным выходами



в) Установка триггера



г) Сброс триггера

Рис. 2.14. RS-триггер

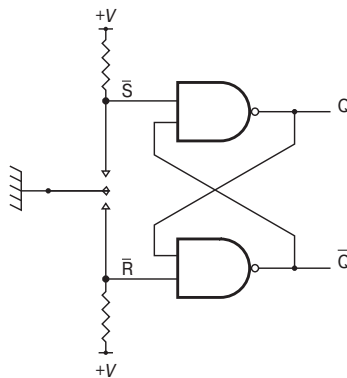


Рис. 2.15. Использование RS-триггера для подавления дребезга контактов

ла, в обозначении которого имеется суффикс «1», в данном случае — от входа С. То есть фиксация значения 1D происходит по сигналу C1.

Триггер тоже представляет собой 1-битную ячейку памяти, однако в нем данные передаются на выход только по активному фронту сигнала на управляющем (тактовом) входе. *D-триггер*, таблица истинности которого приведена на **Рис. 2.16, в**, переключается по нарастающему фронту (в таблице истинности это обозначается символом «↑»), однако часто встречаются и триггеры, переключающиеся по спадающему фронту. Импульсный вход на условном обозначении триггера по стандарту ANSI/IEC обозначается символом >, как показано на **Рис. 2.16, г**.

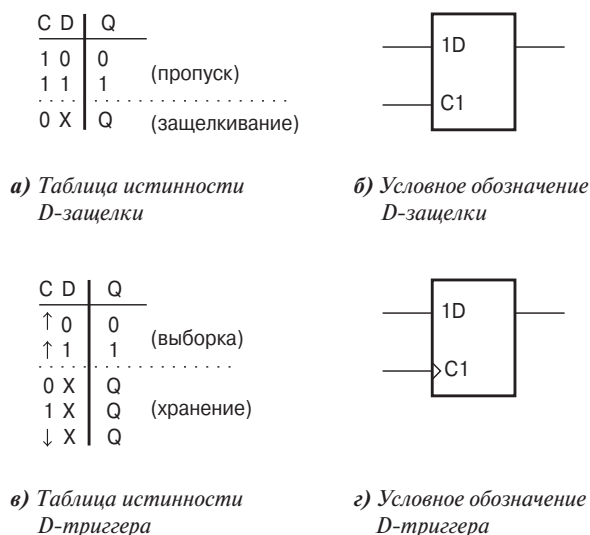


Рис. 2.16. D-защелка и D-триггер

Микросхема малой степени интеграции 74LS74¹⁾, показанная на **Рис. 2.17**, содержит два D-триггера. Каждый триггер имеет входы сброса \overline{R} и установки \overline{S} , которые являются асинхронными, т.е. их функционирование не зависит от тактового сигнала. Среди микросхем средней степени интеграции встречаются наборы из 4, 6 и даже 8 триггеров, имеющих общий тактовый вход.

Микросхема 74LS377²⁾, показанная на **Рис. 2.18**, состоит из восьми D-триггеров, тактируемых одним сигналом C, который, в свою очередь, управляется сигналом \overline{G} . То есть 8 бит данных 8D, ..., 1D защелкиваются по нарастающему фронту на входе C при НИЗКОМ уровне на входе \overline{G} . На условном обозначении микросхемы по стандарту ANSI/IEC, приведенном на **Рис. 2.18, б**, эта зависимость обозначена как $G1 \rightarrow 1C2 \rightarrow 2D$, т.е. вход \overline{G} разрешает работу тактового входа C, который, в свою очередь, воздействует на входы данных.

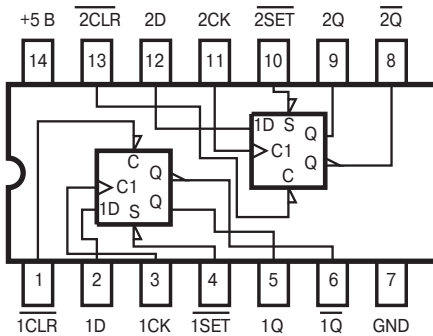
Наборы D-триггеров обычно называются *регистрами*, т.е. устройствами памяти, хранящими одно слово данных. Полное название микросхемы 74LS377 — регистр с параллельным входом и параллельным выходом (РПО-регистр), поскольку данные загружаются в него и считываются из него параллельно (т.е. одновременно).

Выпускаются также микросхемы, содержащие массив D-защелок. В качестве примера можно указать 8-битный регистр-защелку 74LS373³⁾, показанный на **Рис. 2.19**, в котором вместо восьми D-триггеров используется восемь D-защелок. Кроме того, выходы защелок могут устанавливаться в третье состояние. Эта воз-

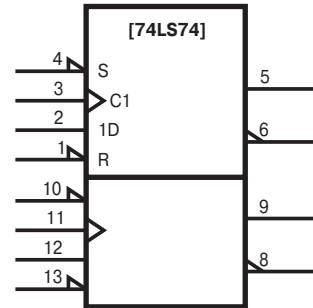
¹⁾ Отечественный аналог — микросхема К555ТМ2. — *Примеч. пер.*

²⁾ Отечественный аналог — микросхема К555ИР27. — *Примеч. пер.*

³⁾ Отечественный аналог — микросхема К555ИР22. — *Примеч. пер.*

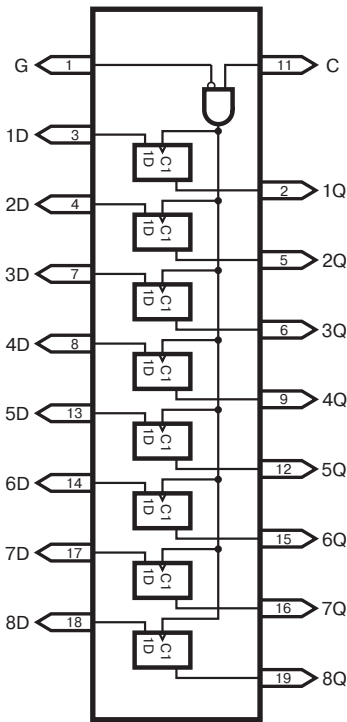


а) Логическая схема

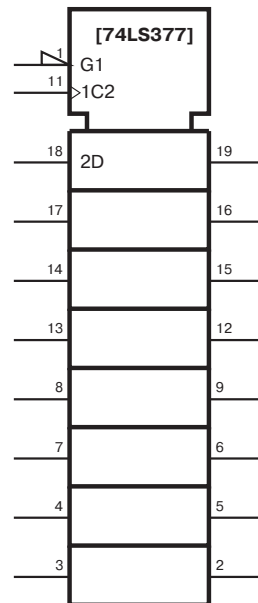


б) Обозначение по ANSI/IEC

Рис. 2.17. Микросхема двоянного D-триггера 74LS74 (K555TM2)



а) Логическая схема



б) Обозначение по ANSI/IEC

Рис. 2.18. Микросхема 8-битного параллельного регистра 74LS377 (K555IP27)

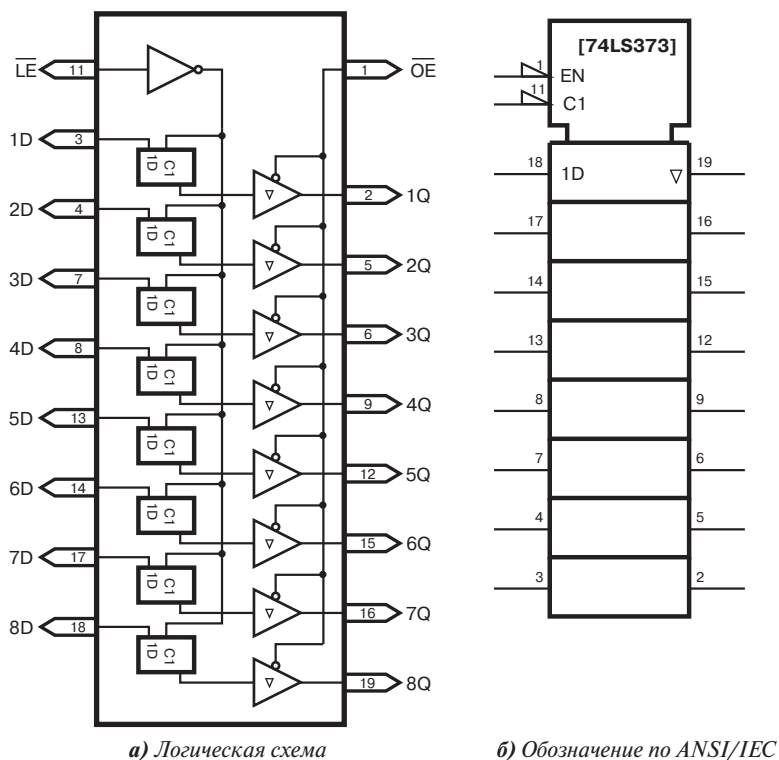


Рис. 2.19. Микросхема 8-битного параллельного регистра-зашелки 74LS373 (K555ИР22)

мощность используется в тех случаях, когда данные сначала зашелкиваются в регистре, а затем выставляются на общую шину для последующего их считывания компьютером.

Подходящий пример использования RPO-регистра приведен на **Рис. 2.20**. На этой схеме к входу 8-битного регистра подключен выход АЛУ. Выходы регистра, в свою очередь, подключены к одному из входов АЛУ. Этот регистр служит для накопления результата последовательных операций и обычно называется *аккумулятором*, или *рабочим регистром*. Чтобы разобраться в работе этой схемы, рассмотрим процесс сложения двух слов — А и В. Если предположить, что АЛУ представляет собой две каскадно-соединенные микросхемы 74LS238, то последовательность операций может быть следующей:

1. Шаг программы

- Режим = 000 (сброс).
- По импульсу на входе «Исполнение» значение с выхода АЛУ (00000000) загружается в регистр.
- Выходные данные — ноль (00000000).

2. Шаг программы

- Значение слова А подается на вход АЛУ.
- Режим = 011 (сложить).
- По импульсу на входе «Исполнение» значение с выхода АЛУ (слово А + ноль) загружается в регистр.
- Выходные данные — слово А.

3. Шаг программы

- Значение слова В подается на вход АЛУ.
- Режим = 011 (сложение).
- По импульсу на входе «Исполнение» значение с выхода АЛУ (слово В + слово А) загружается в регистр.
- Выходные данные — сумма слов В и А.

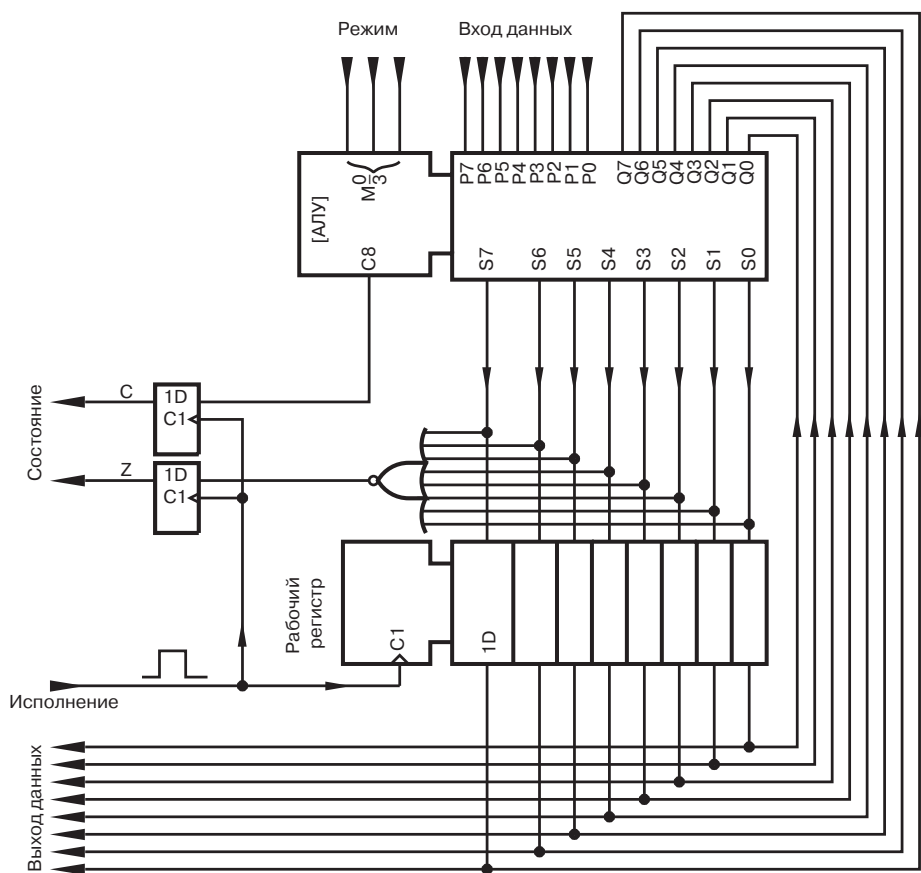



Рис. 2.20. 8-битный блок обработки (АЛУ/рабочий регистр)

Последовательность *кодов операций* (000 — 100 — 100) и составляет программу. На практике каждая команда будет также содержать (при необходимости) адрес обрабатываемых данных; в данном случае — местонахождение слов А и В.

Результат любой операции характеризуется некоторым набором свойств. К примеру, результат может быть равен нулю или же при его вычислении может произойти переполнение. Эти свойства могут потребоваться при дальнейшем выполнении программы. В рассматриваемой схеме для сбора такой информации используются два D-триггера, тактируемые сигналом «Исполнение». В данном контексте состояния этих триггеров называются *флагами* (реже — семафорами). Таким образом, у нас имеются флаг нуля **Z** и флаг переноса из 7-го бита **C**, образующие регистр состояния (STATUS).

Как мы увидим далее, связка АЛУ/рабочий регистр является «сердцем» любого цифрового вычислительного устройства. Причем при использовании сложных систем, таких как компьютер или микроконтроллер, нам совершенно не нужно досконально знать их внутреннее устройство, а процессы, протекающие в системе, скрыты от пользователя. К примеру, на **Рис. 2.21** изображен тот же самый блок, но на более высоком уровне абстракции. В частности, группы линий данных (*шины*) изображены в виде толстых линий, действительная их реализация не имеет никакого значения. Количество линий в шине не показано, но при необходимости оно указывается рядом с коротким штрихом, пересекающим изображенные шины по диагонали, например так .

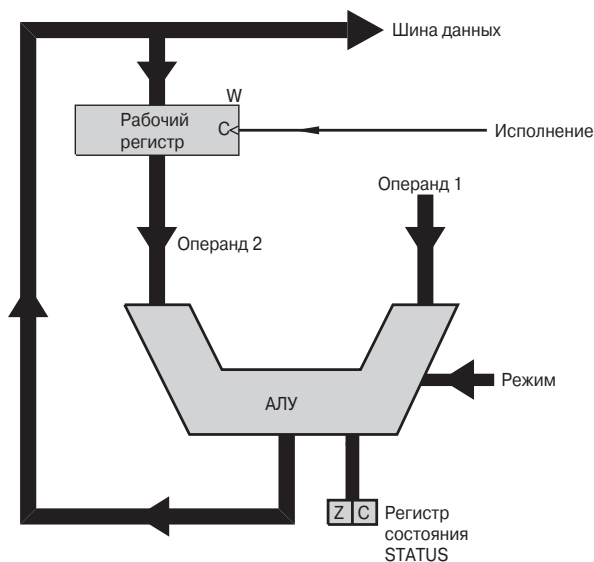


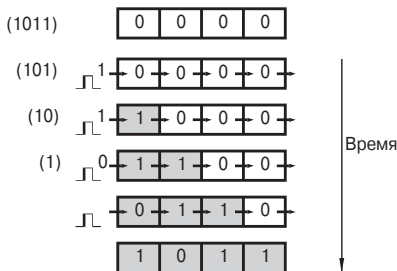
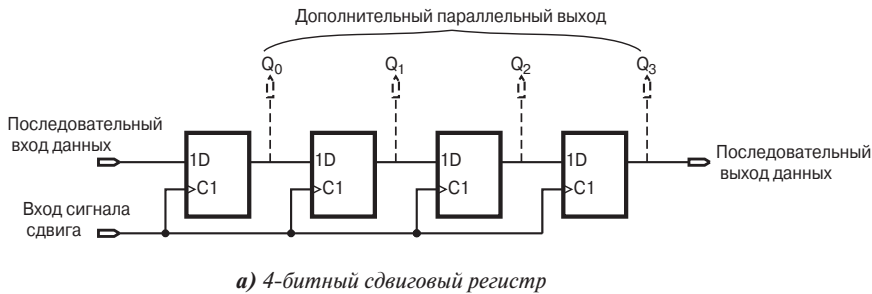
Рис. 2.21. 8-битный блок обработки (АЛУ/рабочий регистр) на системном уровне

Центральным элементом нашей системы является АЛУ, изображение которого имеет сложную форму. Значения на его входах данных (операнды) обрабатываются согласно сигналам на входах режима. Первый операнд поступает извне, тог-

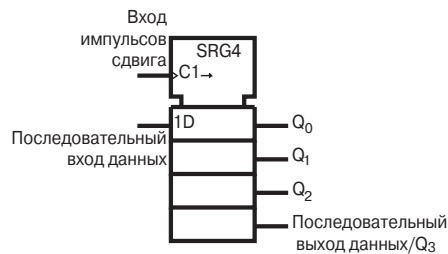
да как 2-й операнд считывается из рабочего регистра. В компьютерах коды, подаваемые на вход режима, обычно считываются из памяти программ, а 1-й операнд — из памяти данных.

Значение с выхода АЛУ может быть загружено обратно в рабочий регистр W по сигналу «Исполнение» либо передано вовне по шине данных. Такая структура показана на **Рис. 3.2** (стр. 60).

Существуют также и другие разновидности регистров. Четырехбитный *сдвиговый регистр*, показанный на **Рис. 2.22, а**, является примером структуры с последовательным вводом и последовательным выводом (SISO). В данном случае бит данных, хранящийся в n -м D-триггере, поступает на вход следующего $((n + 1)$ -го) каскада. При подаче тактового импульса (или, в данном контексте, импульса сдвига) этот бит перегружается в $(n + 1)$ -й триггер, т.е. сдвигается с n -й позиции в позицию $n + 1$. Поскольку все триггеры тактируются одним сигналом, по каждому импульсу сдвига все слово данных сдвигается вправо.



б) Загрузка числа 1011 в регистр



в) Обозначение по ANSI/IEC регистра с последовательным вводом и параллельным выводом

Рис. 2.22. Сдвиговый регистр с последовательным вводом и выводом

В примере, приведенном на **Рис. 2.22, б**, по тактовому сигналу в левую позицию побитно вдвигается 4-битное число. После 4-го импульса новое слово полностью окажется в регистре. Для его считывания потребуется еще четыре импульса, во время которых произойдет побитная выдача содержимого сдвигового регистра. Если обеспечить доступ к выходу каждого триггера, чтобы данные можно было считать за один раз, получим структуру с последовательным входом и параллельным выходом (SIPO).

На **Рис. 2.22**, в символ « \rightarrow » в обозначении тактового входа используется для указания операции сдвига. Аббревиатура SRG4 означает «4-битный сдвиговый регистр». Пример 8-битного сдвигового регистра приведен на **Рис. 12.2** (стр. 370).

Существуют и другие разновидности структур, в том числе структура с параллельным входом и последовательным выходом (PISO), часто применяемая для преобразования параллельного кода в последовательный. Инкрементирование или декрементирование счетных регистров (счетчиков) производится по каждому импульсу тактового сигнала в соответствии с двоичной последовательностью. Обычно n -битный счетчик может отсчитывать 2^n состояний. Некоторые счетчики можно загружать в параллельном режиме, т.е. использовать как память.

Рассмотрим D-триггер, тактируемый по спадающему фронту (**Рис. 2.23**), инверсный выход \bar{Q} которого подключен к входу 1D. По каждому спадающему фронту на входе C1 данные с входа 1D будут защелкиваться и появляться на выходе Q. Поскольку инверсный сигнал этого выхода подается обратно на вход, то в следующий раз триггер переключится в **противоположное** состояние. Это периодическое переключение между двумя состояниями помечено на временной диаграмме символом «Т». В результате при подаче на вход триггера сигнала некоторой частоты на его выходе будет сформирована последовательность импульсов, частота которых в 2 раза ниже. Если частота входного сигнала не изменяется, то выходной сигнал представляет собой точный прямоугольный сигнал (меандр). Иногда такой *T-триггер* называют триггером счетного типа или делителем на два.

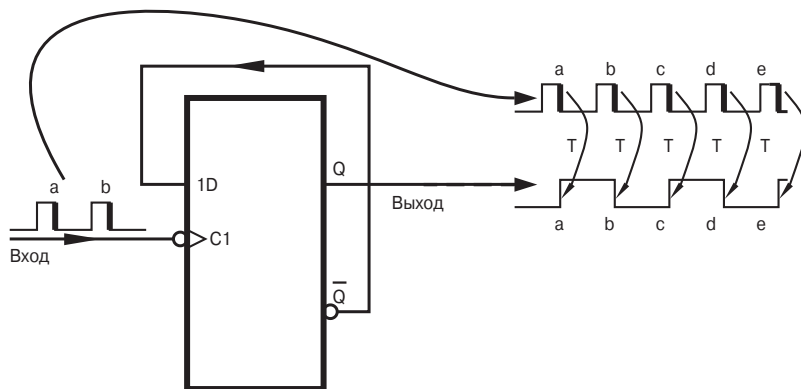
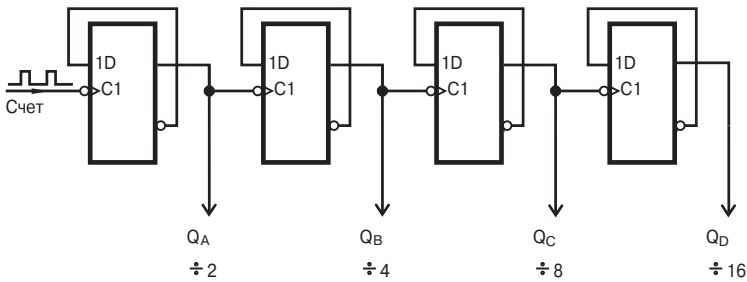


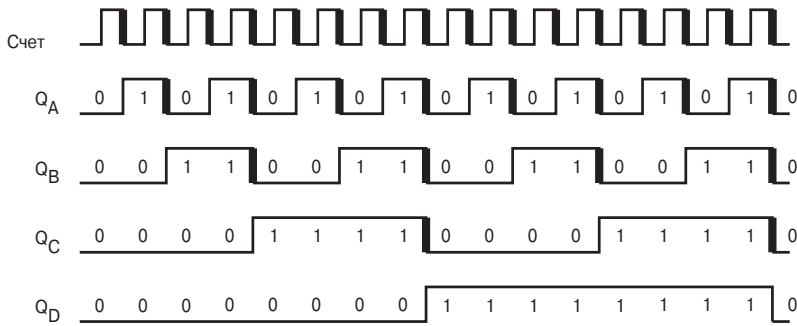
Рис. 2.23. T-триггер

Разумеется, T-триггеры тоже можно каскадировать, как показано на **Рис. 2.24, а**. В данном случае 4 триггера с запуском по спадающему фронту соединены таким образом, чтобы выход n -го разряда управлял тактовым входом разряда $n + 1$. Соответственно, если частота сигнала на входе С равна 8 кГц, то на выходе Q_A будет прямоугольный сигнал частотой 4 кГц, на выходе Q_B — 2 кГц, на Q_C — 1 кГц и на Q_D — 500 Гц. Сигнал Q_A на **Рис. 2.24, б** формируется так же, как и на **Рис. 2.23**. Выход Q_B переключается по каждому спадающему фронту сигнала Q_A .

Аналогично функционируют и остальные выходы. Сопоставив ВЫСОКОМУ уровню лог. 1, а НИЗКОМУ — лог. 0, получим 2^4 (16) двоичных комбинаций в положительной логике, сдвинутых по фазе друг относительно друга. При достижении максимального значения счет начинается с 0 и так до бесконечности. Каждая комбинация остается в регистре до появления активного фронта следующего тактового импульса (в данном случае — спадающего фронта). Если взглянуть на формируемую последовательность, то можно увидеть, что она представляет собой последовательность натуральных двоичных чисел от $b'0000$ до $b'1111$. Вообще говоря, такая схема называется *двоичным счетчиком* по модулю 16. При счете по модулю n используются только первые n формируемых значений¹⁾.



а) Каскадное соединение T-триггеров



б) Формируемые сигналы

Рис. 2.24. Счетчик со сквозным переносом по модулю 16

Теоретически нет никаких ограничений на количество каскадов, соединяемых указанным образом. То есть, используя 8 T-триггеров, мы получим счетчик по модулю 256 (2^8). На практике же каждый триггер переключается с некоторой задержкой, что ограничивает максимально возможную частоту счетчика. К примеру, у сдвоенного D-триггера, показанного на Рис. 2.17, максимальная задержка распространения сигнала от фронта тактового импульса до появления выходного

¹⁾ С математической точки зрения любое число можно преобразовать в его эквивалент по модулю n путем деления этого числа на n . Остаток, или, иначе, модуль, будет представлять собой число от 0 до $n - 1$.

значения составляет 25 нс. Максимальная частота переключения одного каскада, например, такого как показан на **Рис. 2.23**, составляет 25 МГц. Соответственно, максимальная задержка в 8-битном счетчике составит 200 нс. Если такой *счетчик со сквозным переносом* будет тактироваться сигналом с частотой 5 МГц (равной $\frac{1}{200 \text{ нс}}$), то возникнет ситуация, при которой новое значение будет формироваться до установления предыдущего. Это представляет серьезную проблему, если различные состояния счетчика декодируются и используются для управления другими схемами. Схема декодирования, например, такая как приведена на **Рис. 2.25**, может отреагировать на это кратковременное переходное состояние непроизвольным образом, что вызовет сбой в работе устройства. В таких случаях лучше использовать более сложный синхронный счетчик, в котором все триггеры переключаются одновременно.

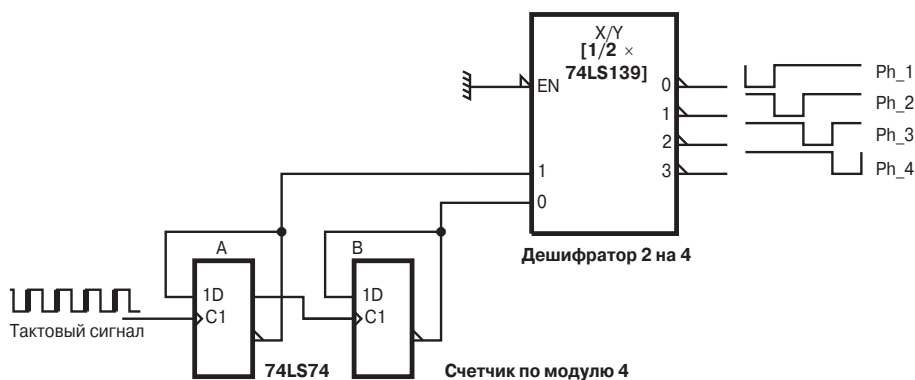


Рис. 2.25. Формирование временных диаграмм

Рассмотренные схемы осуществляли прямой счет. Если в качестве выходов использовать инверсные (\overline{Q}), то счет будет осуществляться в обратном направлении (обратный счет). Того же результата можно достичь, если в качестве элемента памяти использовать триггеры, переключающиеся по нарастающему фронту, такие как сдвоенный триггер 74LS74.

С помощью простой логической схемы можно легко объединить эти две функции и реализовать программируемый реверсивный счетчик. Еще можно добавить логику для параллельной загрузки триггеров любым значением, с последующим счетом от этого значения в заданном направлении. Такие структуры называются счетными регистрами с параллельной загрузкой.

Наряду с наиболее очевидным использованием счетного регистра для накопления числа событий, например, таких как количество консервных банок, прошедших через конвейер, существуют и другие варианты его использования. Одним из таких применений является разнесение во времени некоторых операций. На **Рис. 2.25** счетчик по модулю 4 используется для управления одной из секций дешифратора 2 на 4 в микросхеме 74LS139 (**Рис. 2.5, а**). Этот дешифратор детектирует 4 состояния счетчика и формирует четыре сигнала, сдвинутых во времени друг относительно друга, которые могут использоваться, скажем, для задания

последовательности операций, выполняемых управляющей логикой компьютера. Для адресации дешифратора используется инверсный выход триггеров. Это сделано специально, поскольку в противном случае по нарастающему фронту тактового сигнала осуществлялся бы обратный счет. Счетчики с большей разрядностью могут использоваться для формирования более сложных последовательностей управляющих операций.

Термин «регистр», как правило, используется применительно к элементу оперативной памяти, который может хранить одно двоичное слово, обычно разрядностью от 4 до 64 бит. Память большего объема можно реализовать, группируя n таких регистров и выбирая один из них. Подобная структура обычно называется регистровым файлом. Например, микросхема 74LS670¹⁾ представляет собой регистровый файл 4×4 с отдельными входом и выходом 4-битных данных, а также отдельными входами 2-битного адреса для операций чтения и записи. Это означает, что любой регистр этого файла может быть считан в любой момент времени независимо от одновременно осуществляемой записи.

Память больших объемов называется *оперативной памятью произвольного доступа* или сокращенно *ОЗУ*. Словосочетание «произвольный доступ» означает, что для выбора любого слова памяти требуется одно и то же время, не зависящее от расположения этого слова в матрице²⁾. Этим ОЗУ отличается от памяти на магнитной ленте, в которой бобина должна прокрутиться до требуемого сектора. А если этот сектор находится в конце ленты...

Для примера на **Рис. 2.26** показана микросхема ОЗУ 6264³⁾. Она содержит матрицу из 65 536 (2^{16}) бистабильных ячеек, организованных в виде матрицы из 8192 (2^{13}) 8-битных слов. Слово n выбирается при подаче на линии адреса $A_0...A_{12}$ двоичного числа n .

В режиме чтения ($R/\overline{W} = 1$) на выходы $I/O_7...I/O_0$ выдается n -е слово данных, определяемое n -й комбинацией битов адреса. Символ «А» в обозначении входов/выходов (как и на **Рис. 2.12**) указывает на эту взаимосвязь. Для включения выходных буферов с тремя состояниями на входе \overline{OE} должен быть НИЗКИЙ уровень.

Адресованное слово записывается в память при $R/\overline{W} = 0$. Байт данных, который должен быть записан в n -ю ячейку, подается на входы $I/O_7...I/O_0$. Такая двунаправленная передача данных является отличительной особенностью компьютерных шин.

В обоих случаях микросхема ОЗУ должна быть выбрана подачей лог. 0 на вывод $\overline{CS1}$ и лог. 1 — на вывод $CS2$. В зависимости от версии микросхемы интервал между подачей сигналов выборки и началом обращений к ней составляет от 100 до 150 нс. Если напряжение питания не пропадает, время хранения данных не ограничено. По этой причине микросхема 6264 называется статическим ОЗУ (SRAM). Вместо того чтобы использовать для хранения одного бита пару транзисторов, данные можно хранить в виде заряда емкости затвор-исток одного по-

¹⁾ Отечественный аналог — микросхема К555ИР26. — *Примеч. пер.*

²⁾ Строго говоря, ПЗУ тоже следует называть памятью с произвольным доступом, однако по традиции этот термин используется только для ОЗУ.

³⁾ Отечественный аналог — микросхема К537РУ17. — *Примеч. пер.*

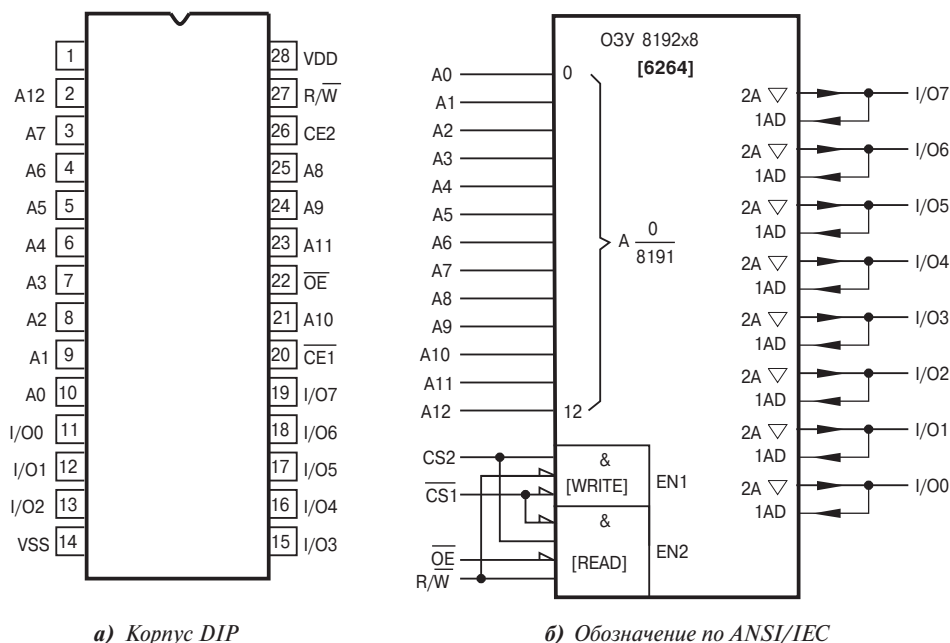


Рис. 2.26. Микросхема ОЗУ 6264 (8196 × 8 бит)

левого транзистора. Время рассасывания подобного заряда составляет несколько миллисекунд, поэтому заряд необходимо периодически обновлять. Такая динамическая память (DRAM) дешевле в изготовлении, и микросхемы данного типа имеют большую емкость. Обычно память подобного типа используется там, где требуется очень большой объем памяти, например в персональных компьютерах. В этом случае стоимость схемы регенерации компенсируется дешевизной микросхем памяти.

Оба типа памяти являются энергозависимыми, т.е. они не сохраняют свое содержимое после выключения питания. Однако некоторые микросхемы статического ОЗУ позволяют хранить данные при напряжении, которое ниже, чем рабочее, потребляя при этом очень маленький ток. В таких случаях для сохранения содержимого в течение нескольких месяцев можно использовать батарею.

ОБРАБОТКА ХРАНИМОЙ ПРОГРАММЫ

В предыдущей главе мы с вами разработали простейший процессор, состоящий из арифметико-логического устройства (АЛУ) и регистра с параллельным вводом/выводом данных. Собственно АЛУ выступает в роли «числодробилки», а рабочий регистр используется для хранения операндов, а также результатов всех операций. В нашем примере, описанном на стр. 33, мы складывали вместе два числа, накапливая результат в рабочем регистре. Если задавать код режима работы АЛУ перед каждым шагом, то мы в принципе можем заставить наше вычислительное устройство выполнить любую задачу, которая может быть описана последовательностью арифметических и логических операций. Эта совокупность кодов команд (например, «сложить», «вычесть», «логическое И», ...) может храниться во внешней памяти. Там же могут находиться различные операнды, передаваемые в АЛУ, а также результаты выполнения команд. Таким образом, эти коды включают в себя как собственно *программу* программируемого устройства, так и различные операнды, или *данные*. **Извлекая** (fetch) эти *команды* по очереди, мы можем **выполнять** заданную *программу*. Такая структура вместе с соответствующими каналами передачи данных, дешифраторами и логическими схемами обычно называется цифровым *компьютером* или цифровой вычислительной машиной.

Как мы с вами вскоре убедимся, в основе архитектуры микроконтроллера лежит архитектура компьютера. С учетом этого обстоятельства в данной главе рассматривается архитектура и рабочий ритм некоего обобщенного компьютера. Несмотря на то что этот компьютер является чисто гипотетическим устройством, при его «разработке» принимались во внимание именно те микроконтроллеры, которые рассматриваются в данной книге.

Прочитав эту главу, вы:

- Познакомитесь с фон-неймановской архитектурой и узнаете ее недостатки.
- Познакомитесь с гарвардской архитектурой, с ее параллельно работающими блоками выборки и дешифрации, а также отдельными адресными пространствами.
- Поймете, какая взаимосвязь существует между цифровым компьютером, микропроцессором и микроконтроллером.
- Познакомитесь со структурой памяти программ, а также ее взаимодействием со счетчиком команд и конвейером.

- Узнаете формат типичных команд.
- Познакомитесь с назначением и структурой памяти данных.

С исторической точки зрения электронные цифровые вычислительные машины в том виде, в котором мы их сегодня знаем, являются косвенным результатом Второй мировой войны. В то время были созданы различные опытные образцы компьютеров, причем некоторые из них действительно работали¹⁾. Как правило, эти вычислительные машины представляли собой специализированные устройства, предназначенные для выполнения какой-либо конкретной задачи при различных входных данных. Алгоритм функционирования некоторых из таких машин можно было менять, но при этом требовалась их частичная переделка.

Поскольку принципиальный вопрос возможности создания таких вычислительных систем был уже решен, основным достижением группы инженеров, работавших с Джоном фон Нейманом²⁾, было осознание того факта, что программа может храниться в памяти вместе с данными. Основным преимуществом такого подхода является его гибкость, так как для изменения программы достаточно просто загрузить новый код в соответствующую область памяти. По существу, фон-неймановская³⁾ архитектура, показанная на **Рис. 3.1**, состоит из *центрального процессора* (ЦПУ), памяти и общей шины (называемой также магистралью), по которой в обоих направлениях пересылаются данные. На практике ЦПУ также должен взаимодействовать и с окружающим миром. При этом данные к/от соответствующих интерфейсных портов передаются по **одной** общей шине данных.

Огромным преимуществом фон-неймановской архитектуры является ее простота, поэтому данная концепция легла в основу большинства компьютеров общего назначения. Однако использование общей шины означает, что в любой момент времени может выполняться только одна операция. Соответственно, пересылка данных между ЦПУ и *памятью данных* не может осуществляться

¹⁾ В качестве примера можно привести английский компьютер под названием «Колосс», который на протяжении нескольких лет использовался для расшифровки секретных кодов немецкой армии. Более подробную историческую и техническую информацию об этих первых компьютерах вы сможете узнать, посетив Web-сайт, посвященный оригинальному изданию данной книги.

²⁾ Джон фон Нейман — венгерский математик, принимавший участие в Манхэттенском проекте (американская программа по созданию ядерной бомбы), осуществлявшемся во время Второй мировой войны. После войны был принят на должность консультанта в проекте по созданию машины EDVAC, проводившемся в Муровской школе электрических разработок (*The Moore School of Electrical Engineering*) Пенсильванского университета (*The University of Pennsylvania*). В этом компьютере предполагалось реализовать концепцию хранения программ и данных в общей памяти. Эти идеи фон Нейман опубликовал в 1946 году, однако EDVAC был запущен только в 1951 году. По иронии судьбы англичане смогли реализовать эти идеи гораздо раньше — компьютер «Марк 1», созданный в Манчестерском университете (*The University of Manchester*), выполнил свою первую программу уже в июне 1948 года! Совсем немного отстали от своих коллег разработчики из Кембриджского университета — их компьютер EDSAC был запущен в мае 1949 года, почти за два года до создания EDVAC.

³⁾ Справедливости ради стоит упомянуть и об одной из первых отечественных ЭВМ «М-1», эксплуатация которой была начата весной 1952 года. В этой машине тоже была реализована концепция программы, хранимой в оперативной памяти, хотя отчет Принстонского университета, в котором были сформулированы архитектурные принципы Дж. фон Неймана, в то время и не был известен разработчикам «М-1». — *Примеч. пер.*

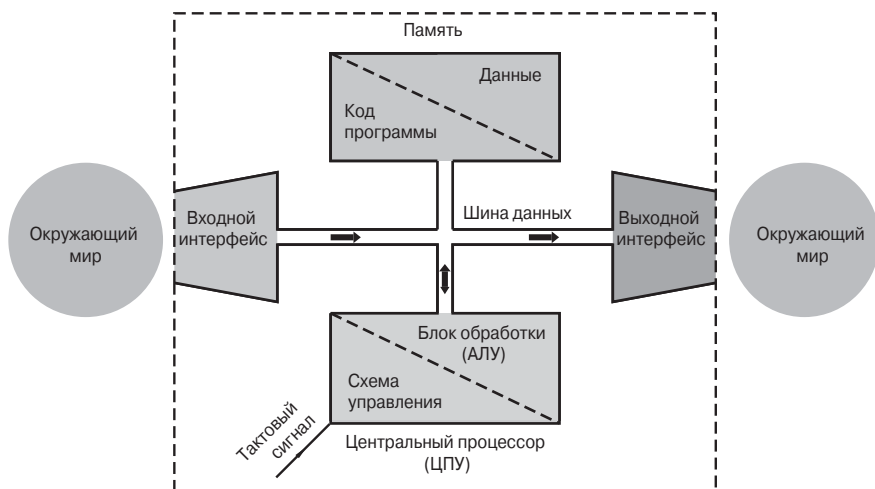


Рис. 3.1. Элементарная фон-неймановская вычислительная машина (шина адреса не показана)

одновременно с выборкой команды из *памяти программ*. Эту особенность иногда называют фон-неймановским узким местом.

В первое послевоенное десятилетие в Гарвардском университете было создано несколько компьютеров семейства «Марк», от «Марк 1» до «Марк 4», в которых память программ была полностью отделена от памяти данных (в первых машинах «Марк 1» и «Марк 2» программа считывалась с бумажной перфоленты). Такая концепция была более эффективной, чем фон-неймановская (или, как ее иногда называют, *принстонская*¹⁾ архитектура, поскольку код программы мог считываться из памяти программ одновременно с обменом между ЦПУ и памятью данных или с операциями ввода/вывода. Однако такие машины были намного сложнее и дороже в изготовлении. А с учетом уровня технического развития 50-х годов, да еще и после проигрыша в конкурсе на создание компьютера для контроля сети континентальных радиолокационных станций, устроенного Министерством обороны США, они и вовсе не получили широкого распространения. Однако с развитием сложных интегральных схем эта *гарвардская архитектура* снова оказалась в центре внимания.

На Рис. 3.2 показаны две физически разделенные шины, используемые для передачи информации между ЦПУ и этими неперекрывающимися областями памяти. Каждая память имеет собственную шину адреса, поэтому адрес ячейки памяти программ никоим образом не связан с адресом ячейки памяти данных. В таком случае говорят, что обе области памяти находятся в **различных адресных пространствах**. Память данных иногда называют файловой памятью, в этом случае *n*-я ячейка обозначается как **файл *n***.

¹⁾ Причем это название более верно с исторической точки зрения, поскольку авторство принадлежит не фон Нейману, а группе разработчиков компьютера ENIAC, у которых фон Нейман проходил стажировку. — *Примеч. пер.*

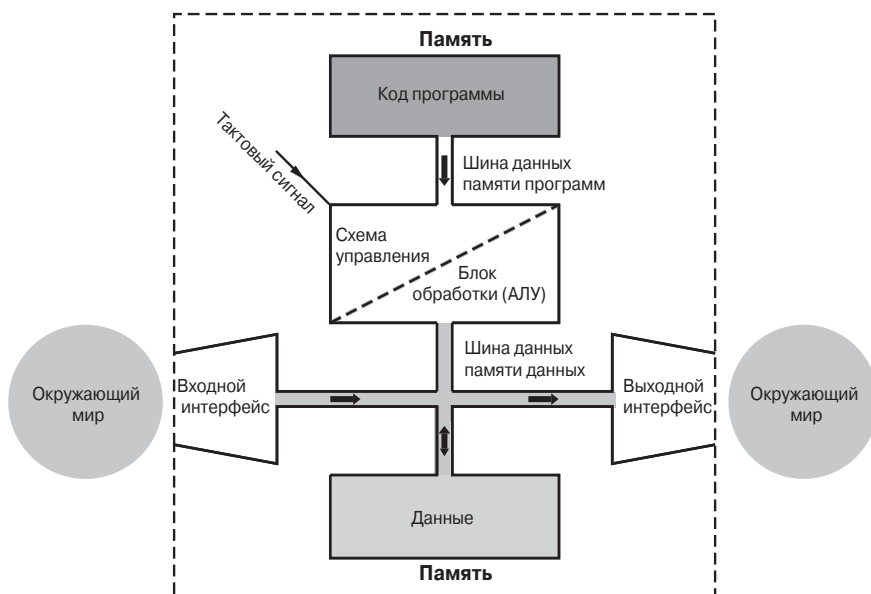


Рис. 3.2. Элементарная гарвардская вычислительная машина (шины адреса не показаны)

А теперь давайте познакомимся поближе с различными элементами компьютерной архитектуры.

Центральный процессор

Центральный процессор состоит из связки АЛУ/рабочий регистр и соответствующей управляющей логики. По сигналам схемы управления команды программы выбираются из памяти, дешифруются и исполняются. Данные, которые получаются или используются во время выполнения программы, также располагаются в памяти. Этот цикл «выборка — исполнение» образует рабочий ритм вычислительной машины и повторяется непрерывно в течение всего времени, когда система находится в активном состоянии.

Память

Во всех вычислительных устройствах память используется для хранения как кода программы, так и данных. Память с произвольным доступом характеризуется **содержимым**, хранящимся в группе ячеек, и расположением (**адресом**) каждой ячейки. В случае фон-неймановской архитектуры и программа, и данные располагаются в одной области памяти, тогда как при использовании гарвардской архитектуры эти объекты располагаются в совершенно разных областях. То есть адреса одной области памяти никоим образом не связаны с адресами другой области. В обоих случаях данные, хранящиеся в памяти, передаются в ЦПУ по шине дан-

ных. При этом ЦПУ выставляет на шину адреса код адреса той ячейки, к которой он собирается обратиться. В системах с гарвардской архитектурой каждая область памяти имеет собственные шины адреса и данных (**Рис. 3.4**). В запоминающих устройствах с произвольным доступом длительность операции чтения или записи любой из ячеек не зависит от положения этой ячейки в адресном пространстве.

В большинстве компьютеров используется долговременная память больших объемов, обычно на магнитных или оптических дисках, в которой время доступа к ячейке зависит от ее физического расположения в памяти. Помимо этого недостатка, присущего памяти с последовательным доступом, такие устройства, как правило, слишком медленны для использования их в качестве основной памяти. Поэтому они используются для резервного хранения больших объемов данных (например, ответов на экзаменационные билеты) или программ, которые перед выполнением необходимо подгружать в основную память.

Память программ

В памяти программ хранится двоичный код, составляющий *программу*, или *программное обеспечение* (software). Это слово созвучно термину hardware (аппаратные средства) и отражает тот факт, что данный код не связан с каким-либо физическим изменением схемы устройства. В идеале память, в которой находится программа, должна быть такой же быстрой, как и ЦПУ, поэтому для данных целей обычно используется полупроводниковая память, созданная при помощи технологий, подобных рассмотренным в предыдущей главе¹⁾.

Память данных

В памяти данных хранятся данные, используемые во время работы программы. И опять же быстродействие этой памяти обычно сравнимо с быстродействием ЦПУ. Также в адресном пространстве памяти данных могут располагаться специальные регистры, например порты ввода/вывода.

Интерфейсные порты

Независимо от своего назначения компьютер должен иметь возможность взаимодействовать с окружающим миром. Хотя обычно вспоминаются такие устройства, как клавиатура и монитор, можно считывать и изменять состояние практически любого физического устройства. Так, данные об объеме топлива, впрыскиваемого в цилиндр двигателя, в совокупности со значением скорости вращения вала могут использоваться для управления моментом зажигания искры в камере сгорания бензинового двигателя.

¹⁾ Однако это не всегда верно — самые первые быстродействующие устройства памяти программ были построены на ферритовых кольцах, которые могли намагничиваться в одном из двух направлений. Память на магнитных сердечниках использовалась с 50-х годов вплоть до начала 70-х, но и до сих пор в литературе можно встретить термин *core* (сердечник), применяемый для обозначения памяти программ.

Шина данных

Все элементы фон-неймановского компьютера соединяются между собой одной **общей** магистралью передачи данных, или шиной (понятие шины было введено во 2-й главе, см. **Рис. 2.4** на стр. 34). Вся информация передается по этим общим линиям в обоих направлениях, при этом ЦПУ играет роль главного контроллера. В компьютере с гарвардской архитектурой память программ имеет отдельную шину данных, что позволяет осуществлять выборку команд одновременно с действиями на шине данных памяти программ. Другие шины используются для передачи адресов различным областям памяти, а также управляющей информации и информации о состоянии (см. **Рис. 3.4**).

Микроконтроллеры, которым посвящена эта книга, имеют гарвардскую архитектуру, поэтому дальше мы будем рассматривать только ее. Взяв за основу ЦПУ, показанный на **Рис. 2.20** (стр. 49), и добавив к нему память программ, память данных, а также схемы управления и дешифрации, мы получим примитивный компьютер с гарвардской архитектурой (**Рис. 3.3**). Серым цветом на рисунке выделены элементы исходной схемы с **Рис. 2.21**, приведенного на стр. 50.

Благодаря подключению шины данных АЛУ к памяти данных, мы получаем возможность считывать из памяти первый операнд, а также при необходимости помещать в нее результат операции. Адрес этого операнда является частью кода команды, считанного из памяти программ и дешифрованного устройством управления. Это же устройство управления формирует сигналы выбора режима АЛУ, который зависит от текущей команды. Результат, получаемый на выходе АЛУ, может быть загружен либо в рабочий регистр (устройство управления формирует импульс на линии W), либо обратно в ту же ячейку памяти, откуда был считан операнд (устройство управления формирует импульс на линии F). Информация об адресате результата операции также содержится в коде команды.

Команды (в виде кодовых слов) обычно располагаются в памяти программ последовательно. Для поочередной адресации каждой команды используется двоичный суммирующий счетчик (см. **Рис. 2.24** на стр. 53). Если, предположим, при сбросе компьютера *счетчик команд* (Program Counter — PC) обнуляется, то первая команда будет расположена по адресу h'000' памяти программ, вторая — по адресу h'001' и т.д. (см. **Рис. 3.4**). Устройство управления просто инкрементирует счетчик после выборки каждой команды. Непосредственно загружая новый адрес в счетчик команд, можно осуществить переход к другому участку кода.

Последовательность операций «выборка команды/ее дешифровка/исполнение», т.е. так называемый *цикл выборки — исполнения* команды, является фундаментальным понятием, необходимым для понимания работы компьютера. Чтобы проиллюстрировать этот рабочий ритм, рассмотрим простую программу, которая считывает переменную NUM_1, прибавляет к ней число 4 и записывает результат в

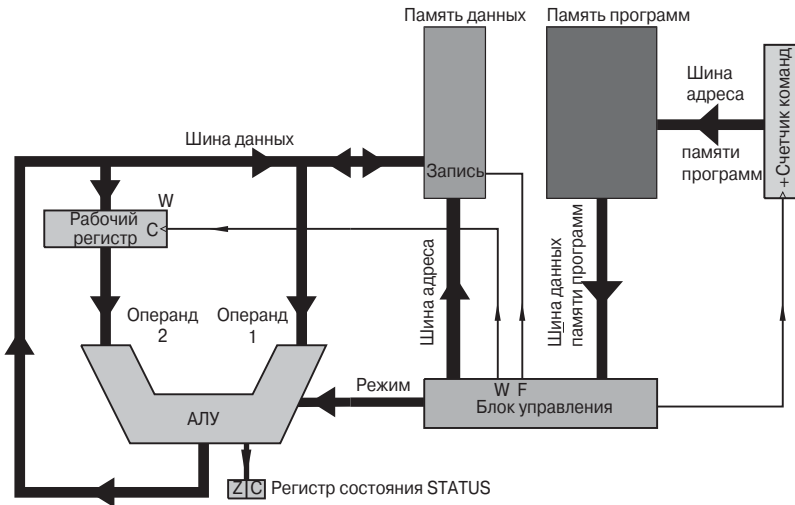


Рис. 3.3. Структура элементарного гарвардского компьютера на системном уровне

переменную `NUM_2`. На языке высокого уровня Си эту операцию можно записать следующим образом¹⁾:

```
NUM_2 = NUM_1 + 4;
```

Несколько более подробно структура нашего компьютера, который я назвал BASIC (аббревиатура от Basic All-purpose Stored Instruction Computer — базовый универсальный компьютер с хранимой программой), изображена на Рис. 3.4. На этом рисунке показаны ЦПУ и обе области памяти со своими шинами данных и соответствующими шинами адреса.

Центральный процессор можно условно разделить на две части. Узлы, расположенные в левой части рисунка, осуществляют **выборку** кодов команд и поочередно передают их в дешифратор команд ID. Узлы, расположенные в правой части, **исполняют** каждую из команд в соответствии с сигналами, формируемыми этим дешифратором.

Сначала разберемся с процессом выборки команд.

Счетчик команд

Команды обычно располагаются в памяти программ последовательно, а счетчик команд PC является обычным счетным регистром, определяющим местонахождение текущей команды. Этот суммирующий счетчик иногда называют (может быть, даже более правильно) указателем команд.

¹⁾ На языке Паскаль или Модуля-2 это же выражение будет иметь вид `NUM_2 := NUM_1 + 4`.

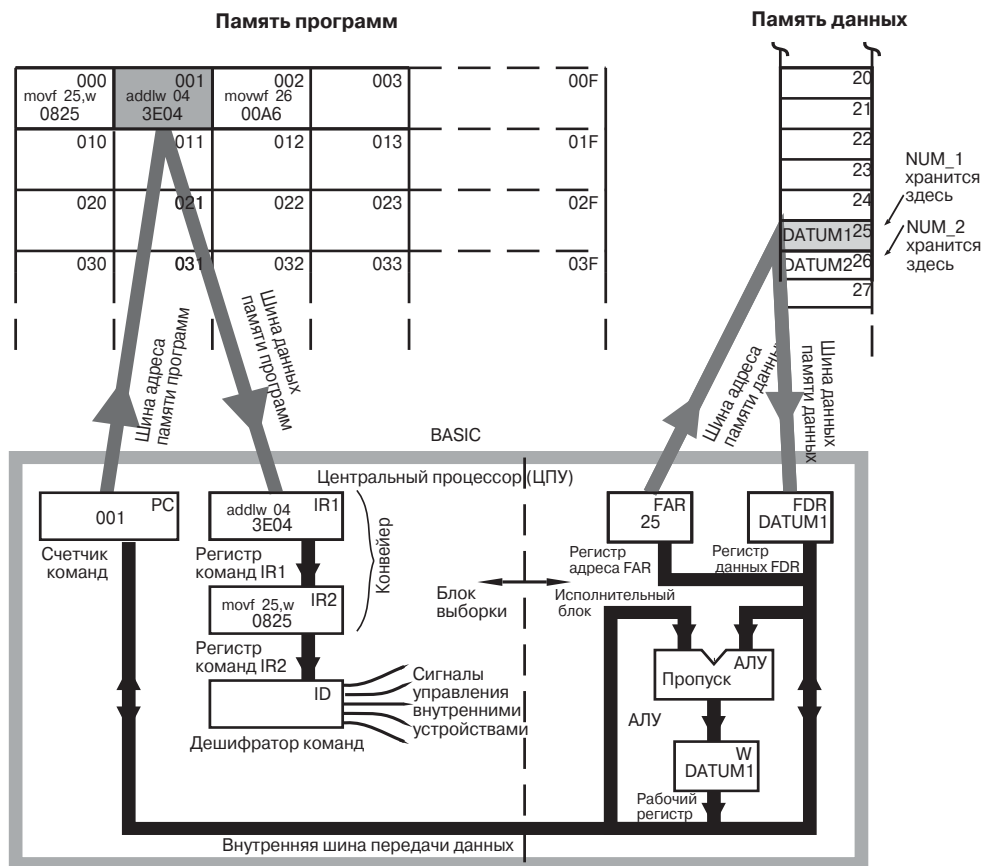


Рис. 3.4. Состояние ЦПУ в момент выполнения первой команды при одновременной выборке второй команды.
Все адреса/данные представлены в шестнадцатеричной системе

Поскольку счетчик команд подключен через внутреннюю шину данных к исполнительному блоку, мы можем использовать АЛУ для управления этим регистром и изменения predetermined последовательности исполнения команд. Таким образом, можно реализовать различные команды перехода к другим частям программы, а также команды пропуска.

Конвейер

В двух регистрах команд содержатся коды команд, считанные из памяти программ. В начало конвейера (в первый регистр команд, IR1) загружается код n -й команды и хранится там для обработки в следующем цикле. Это позволяет испол-

нять команду $n - 1$, находящуюся в конце конвейера (во втором регистре команд, IR2) одновременно с выборкой n -й команды и загрузкой ее в конвейер. Работа конвейера показана на **Рис. 3.7**.

Дешифратор команд

Дешифратор команд ID является «мозгами» ЦПУ — он дешифрует код команды, находящийся в регистре IR2, и формирует в определенной последовательности сигналы для исполнительного блока, необходимые последнему для определения местоположения операнда (если таковой имеется) в памяти данных и для переключения АЛУ в заданный режим. На **Рис. 3.4** показано выполнение команды `movf h'25',w` (копирование содержимого регистра данных с адресом h'25' в рабочий регистр).

Исполнительный блок осуществляет обращения к памяти данных и конфигурирование АЛУ. Работой исполнительного блока управляет дешифратор команд, функционирование которого, в свою очередь, зависит от значения кода команды $n - 1$, находящегося в регистре команд IR2.

Исполнительный блок обрабатывает все числа группами по восемь битов, данные во всех регистрах и в памяти данных также хранятся побайтно. Поэтому о таком компьютере обычно говорят как о 8-битном процессоре.

Регистр адреса

Когда ЦПУ собирается обратиться к ячейке (регистру) памяти данных, он помещает адрес этой ячейки в регистр адреса FAR. При этом производится непосредственная адресация памяти данных по ее шине адреса. Как показано на **Рис. 3.4**, из памяти данных считывается регистр с адресом h'25', и его содержимое зашелкивается во внутреннем регистре данных FDR процессора.

Регистр данных

Этот «двунаправленный» регистр выполняет две функции:

- Хранит содержимое адресованного регистра данных, если ЦПУ осуществляет *цикл чтения*. Именно это происходит при выполнении 1-й команды (`movf h'25',w`), которая пересылает (копирует) содержимое регистра с адресом h'25' в рабочий регистр.
- Хранит данные, которые ЦПУ собирается записать в адресованный регистр данных. Такой *цикл записи*, в частности, имеет место при выполнении команды `movwf h'26'`, которая пересылает (копирует) содержимое рабочего регистра в регистр с адресом h'26'.

Арифметико-логическое устройство

АЛУ выполняет арифметические и логические операции, определяемые значением кода режима (см. **Рис. 2.10** на стр. 39), который извлекается из кода команды дешифратором команд.

Регистр состояния

Этот регистр содержит флаги нуля **Z** и переноса **C**, которые устанавливаются соответственно, если результат операции равен нулю и если в результате сложения возник перенос.

Рабочий регистр

В рабочем регистре АЛУ (**W**) обычно находится один из операндов команды — либо источник, либо адресат. Например, команда `addwf h'20',w` складывает содержимое рабочего регистра с содержимым регистра `h'20'` и помещает сумму обратно в рабочий регистр **W**. В некоторых компьютерах этот регистр называется также аккумулятором.

Помимо ЦПУ, в нашем компьютере имеется две области памяти, предназначенные для хранения кода программы и данных.

Память программ

Каждая позиция (или ячейка) памяти программ может содержать одну команду, которая кодируется 14-битным словом. Из **Рис. 3.4** видно, что каждая из этих ячеек имеет свой адрес, выставяемый счетчиком команд на шину адреса памяти программ. На этом рисунке содержимое **PC** равно `h'001'` (или `b'00000000000001'`), что приводит к выдаче содержимого ячейки `h'001'` на шину данных памяти программ и, следовательно, загрузке его в начало конвейера. В рассматриваемом примере считанное значение равно `h'3E04'` (или `b'11111000000100'`), что является машинным кодом команды `addlw 04`. В конечном счете дешифратор команд интерпретирует этот код как операцию сложения константы «4» с рабочим регистром.

Память данных

Каждая ячейка (или регистр) памяти данных содержит один байт (восемь битов) данных. Адрес регистра формируется исполнительным блоком в регистре адреса **FAR** и выставяется на шину адреса памяти данных. Содержимое адресованного регистра либо считывается в регистр данных **FDR**, либо перезаписывается находящимся в нем значением.

Шины адреса и данных памяти данных совершенно независимы от одноименных шин памяти программ, что позволяет одновременно обращаться к обеим областям памяти. Таким образом, адреса памяти программ и памяти данных имеют различный смысл, т.е. адрес $h'25'$ в памяти программ совсем не то же самое, что адрес $h'25'$ в памяти данных, который соответствует регистру $h'25'$.

Теперь, когда у нашего ЦПУ появилась память программ и память данных, рассмотрим более подробно саму программу. Наша иллюстративная программа состоит всего из трех команд и, как уже упоминалось, предназначена для копирования увеличенного на 4 значения однобайтной переменной, расположенной по адресу NUM_1, в ячейку с адресом NUM_2. Из Рис. 3.4 видно, что переменная NUM_1 представляет собой псевдоним для обозначения содержимого регистра данных $h'25'$. Аналогично, имя NUM_2 является символическим выражением для указания содержимого регистра данных $h'26'$.

А теперь рассмотрим используемые в программе команды.

movf

Эта команда (MOVe File) копирует содержимое заданного регистра данных в рабочий регистр (как правило) или же обратно в тот же регистр данных (см. стр. 141). Таким образом, команда `movf NUM_1, w` загружает байт, расположенный по адресу $h'25'$ памяти данных, в рабочий регистр. Если содержимое указанного регистра равно нулю, то при выполнении команды устанавливается флаг **Z**, в противном случае этот флаг будет сброшен.

addlw

Эта команда (ADD Literal to Working register) прибавляет однобайтную константу к содержимому рабочего регистра. Таким образом, команда `addlw 04` прибавляет число 4 к содержимому рабочего регистра и записывает результат обратно в этот же регистр. Если возникает переполнение, то устанавливается флаг **C**, а если результат равен нулю — флаг **Z**.

movwf

Эта команда (MOVe Working register to File) копирует содержимое рабочего регистра в заданный регистр данных. Таким образом, команда `movwf NUM_2` сохраняет содержимое рабочего регистра по адресу $h'26'$ памяти данных. На состояние флагов данная команда не влияет.

Описывая команды, мы использовали мнемонические обозначения, такие как `addlw`. Разумеется, реальные логические схемы, декодирующие эти команды, работают исключительно с двоичными кодами. Мнемонические обозначения просто играют роль своеобразных «памяток» для программиста. Хотя крайне маловероятно, что кто-либо станет писать программы в *машинных кодах*, двоичный формат всех команд имеет логическую основу, и его знание будет полезно для понимания недостатков и ограничений набора команд и реальных аппаратных средств, которые мы будем обсуждать в следующих двух главах.

Пока мы рассмотрим две категории команд.

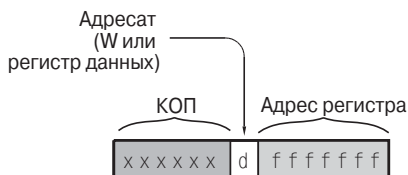
Прямая адресация регистра данных

КОП	d	ffffff
-----	---	--------

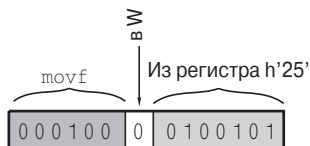
Данный способ адресации используют команды, в которых указывается адрес регистра данных, являющийся их операндом. Например, в команде `movf h'25',w` операндом является регистр `h'25'`.

Из **Рис. 3.5** видно, что 14-битный код команды состоит из трех частей:

- Шесть старших битов (13...8) называются *кодом операции* или, сокращенно, КОП. Каждая команда имеет уникальный КОП, и именно по его значению схема дешифратора определяет тип обрабатываемой команды.
- Седьмой бит кода команды, обозначенный символом «d», определяет адресата результата операции. Например, команда `addwf h'30',w` означает «сложить содержимое рабочего регистра с регистром `h'30'` и поместить результат обратно в рабочий регистр», тогда как команда `addwf h'30',f` означает «сложить содержимое рабочего регистра с регистром данных `h'30'` и поместить результат обратно в регистр `h'30'`». В первом случае адресатом операции является рабочий регистр W и бит d равен 0, а во втором случае адресатом является регистр данных и бит d равен 1. Мы еще вернемся к этой команде в пятой главе. В символической записи команды символы «w» соответствуют сброшенному биту адресата d, а символы «f» соответствуют установленному биту d.
- Младшие семь битов (6...0) определяют адрес регистра данных. Так, в нашем примере используется регистр `h'25'`, поэтому в указанном поле содержится значение `b'0100101'`. Так как размер поля адреса равен семи битам, то посредством прямой адресации можно адресовать только один банк памяти, вмещающий в себя $2^7 = 128$ регистров, т.е. с регистра `h'00'` по регистр `h'7F'` (см. **Рис. 4.7** на стр. 97).



а) Формат кода команды



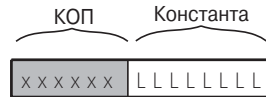
б) Код команды для `movf h'25',w`

Рис. 3.5. Формат кода команд, использующих прямую адресацию

Операции с константами

КОП	LLLLLLLL
-----	----------

Команды, работающие с константами, кодируются немного иначе, как показано на **Рис. 3.6**. В старших шести битах по-прежнему содержится код операции, а в младших восьми битах находится собственно значение константы. Результат выполнения таких команд всегда помещается в рабочий регистр, поэтому не требуется ни бит адресата, ни значение адреса в памяти данных.



а) Формат кода команды



б) Код команды для `addlw 04`

Рис. 3.6. Формат слова команд операций с константами

Код операции команды из нашего примера (`addlw 04`) равен `b'111110`, а константа равна `b'00000100`. Значение константы должно лежать в диапазоне `b'00000000'...b'11111111'` (`h'00'...h'FF'` или, в десятичной системе, `0...255`), что вполне логично, поскольку рабочий регистр, как и все внутренние регистры исполнительного блока, является восьмибитным¹⁾.

Не только команды могут иметь мнемонические обозначения. Как мы видели, символические имена можно присваивать и ячейкам памяти данных. Так, на **Рис. 3.4** идентификатор `NUM_1` используется для указания содержимого регистра данных `h'25'`, а `NUM_2` — регистра данных `h'26'`. Таким образом, нашу программу можно символически записать следующим образом:

```
NUM_2 = NUM_1 + 4;
```

Возвращаясь к собственно компьютеру, мы видим, что, начиная с адреса `h'000'`, наша программа имеет вид

```
00100000100101
11111000000100
00000010100110
```

Если только вы не киборг, то чтение такой программы — весьма сомнительное удовольствие²⁾.

¹⁾ Одна из наиболее распространенных ошибок заключается в упущении из виду этого 8-битного ограничения и в использовании в программе команд, подобных `addlw h'500'`. Результат этой операции аналогичен попытке заполнить литровую бутылку содержимым 4-литрового ведра!

²⁾ Мне ли этого не знать! Я писал программы подобным образом в середине 70-х.

Если мы воспользуемся шестнадцатеричной системой¹⁾, то будет уже удобнее:

```
0825
3E04
00A6
```

но ненамного. К тому же ЦПУ все равно понимает только двоичные числа, поэтому нам в любом случае понадобится программа-транслятор для перевода шестнадцатеричных значений в двоичные, запускаемая, скажем, на персональном компьютере.

Раз уж мы все равно собираемся использовать компьютер для перевода нашей программы, называемой также *исходным кодом*, в двоичный машинный код, называемый *объектным кодом*, то имеет смысл не мелочиться и записать ее полностью в символическом виде. При этом различные команды будут представлены их мнемоническими обозначениями, а адреса переменных — своими именами. В результате наша программа примет вид:

```
movf    NUM_1,w      ; Копируем содержимое NUM_1 в W
addlw   4             ; Прибавляем к нему число 4
movwf   NUM_2        ; Копируем NUM_1 + 4 в NUM_2
```

Текст после символов «;» называется *комментариями*, которые используются для облегчения понимания программы.

Код, записанный таким образом, представляет собой программу на языке *ассемблера*. Синтаксису этого языка, а также процессу преобразования написанных на нем программ в исполняемый двоичный код полностью посвящена восьмая глава книги.

При написании программ с использованием языка ассемблера необходимо помнить, что каждая инструкция программы один в один соответствует исходной машинной команде и ее двоичному коду. В главе 9 мы увидим, что в языках высокого уровня это соотношение нарушается.

В основе работы любого вычислительного устройства лежит периодическое выполнение цикла *выборка — исполнение*. При этом каждая команда поочередно выбирается из памяти программ, интерпретируется и исполняется. Поскольку память, к которой обращается программа в процессе выполнения, является памятью данных, а каждое устройство памяти имеет собственные шины, операции выборки и исполнения могут осуществляться **параллельно**. Таким образом, во время выборки n -й команды исполняется команда $n - 1$. На **Рис. 3.4** показано, что коды обеих команд, как следующей, так и текущей, хранятся в двух внутренних регистрах команд — IR1 и IR2 соответственно. Команды, считанные из памяти программ, загружаются в начало этого конвейера и «выталкиваются» в дешифратор команд с конца конвейера. На **Рис. 3.7** изображен развернутый во времени процесс выполнения нашей команды, разбитый на машинные циклы. Во время каждого цикла, за исключением самого первого, выборка новой команды и исполнение предыдущей осуществляются одновременно.

¹⁾ Не забывайте, что мы используем шестнадцатеричную нотацию исключительно для удобства. Если вы возьмете электронный микроскоп и посмотрите внутрь этих ячеек, то сможете «увидеть» только двоичную структуру.

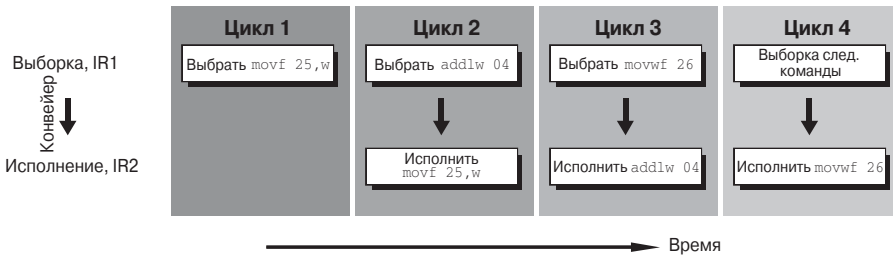


Рис. 3.7. Параллельные потоки выборки и исполнения команд

А теперь, чтобы лучше понять работу конвейера, давайте пройдемся по нашей программе. Предположим, что компьютер (т.е. счетчик команд) был сброшен и только что завершилась операция выборки первого цикла.

Выборка (Рис. 3.4) Цикл 2

- Инкрементируется счетчик команд, чтобы указать на 2-ю команду.
- Одновременно код 1-й команды перемещается по конвейеру (из регистра IR1 в регистр IR2).
- Содержимое счетчика команд (h'001') выставляется на шину адреса памяти программ.
- После этого на шине данных памяти программ появляется код 2-й команды, который загружается в регистр IR1.

Исполнение (Рис. 3.4) Цикл 2

- Адрес операнда h'25' (т.е. NUM_1) заносится в регистр адреса FAR и выставляется на шину адреса памяти данных.
- Искомое значение, находящееся по адресу NUM_1, выставляется на шину данных памяти данных, после чего загружается в регистр FDR.
- АЛУ переключается в режим пропуска, в котором аргумент с входа АЛУ без изменений копируется в рабочий регистр.

Выборка Цикл 3

- Инкрементируется счетчик команд, чтобы указать на 3-ю команду.
- Одновременно код 2-й команды перемещается по конвейеру (из регистра IR1 в регистр IR2).
- Содержимое счетчика команд (h'002') выставляется на шину адреса памяти программ.
- После этого на шине данных памяти программ появляется код 3-й команды, который загружается в регистр IR1.

Исполнение Цикл 3

- АЛУ переключается в режим сложения, в котором константа, содержащаяся в коде 2-й команды, прибавляется к содержимому рабочего регистра.

- Результат операции $\text{NUM_1} + 4$ с выхода АЛУ помещается обратно в рабочий регистр.

Выборка Цикл 4

- Инкрементируется счетчик команд, чтобы указать на 4-ю команду.
- Одновременно код 3-й команды перемещается по конвейеру (из регистра IR1 в регистр IR2).
- Содержимое счетчика команд (h'003') выставляется на шину адреса памяти программ.
- После этого на шине данных памяти программ появляется код 4-й команды, который загружается в регистр IR1.

Исполнение Цикл 4

- Адрес операнда h'26' (т.е. NUM_2) заносится в регистр адреса FAR и выставляется на шину адреса памяти данных.
 - АЛУ переключается в режим пропуска, в котором содержимое рабочего регистра без изменений копируется в регистр FDR и выставляется на шину данных памяти данных.
 - Содержимое регистра FDR заносится в память данных по адресу, выставленному на шину адреса, т.е. в регистр NUM_2.
-

Обратите внимание на автоматическое изменение счетчика команд на этапе выборки каждого цикла. Такой последовательный характер изменения его содержимого будет сохраняться до тех пор, пока не встретится команда, напрямую модифицирующая этот счетчик, например команда `goto h'200'`. При выполнении этой команды адрес h'200' помещается в счетчик команд, нарушая обычный процесс инкрементирования, в результате чего ЦПУ перейдет к команде, расположенной по адресу h'200'. Далее изменение счетчика команд снова примет линейный характер.

Хотя наша программа делает, прямо скажем, немного, на выполнение каждой команды затрачивается всего около 1 мкс. Миллион простейших операций в секунду — это сила! По существу, все компьютеры, какими бы «умными» они ни казались, просто выполняют с очень большой скоростью множество относительно простых операций. Поэтому основной задачей программиста является принятие решения о том, в какой последовательности необходимо расположить команды и структуры данных для выполнения соответствующей задачи.

До настоящего момента мы рассматривали одни компьютероподобные структуры. Для логического завершения главы нам осталось только провести связь между предметом обсуждения и собственно микроконтроллерами.

Так что же это такое — микроконтроллер? Кратко говоря, *микроконтроллер* — это микропроцессор, который объединен с памятью и различными устройствами ввода/вывода в одной интегральной микросхеме. То есть по сути дела это микро-

процессор со встроенными вспомогательными узлами. Так что нам придется совершить небольшое путешествие во времени к моменту рождения микропроцессора. Вся эта история началась в 1968 году, когда Роберт Нойс (один из изобретателей интегральных микросхем), Гордон Мур¹⁾ и Эндрю Грув уволились из компании Fairchild Corporation и основали свою собственную компанию, назвав ее Intel²⁾. В течение трех лет новая компания освоила выпуск полупроводниковой памяти всех основных типов, используемых в настоящее время, — динамического и статического ОЗУ, а также микросхем EEPROM.

Одним из неосновных направлений деятельности компании была разработка интегральных микросхем большой степени интеграции (БИС) по спецификациям заказчика. В 1970 году к Intel обратились представители японской корпорации Busicom с предложением изготовить подходящий набор микросхем (так называемый чипсет) для линейки калькуляторов. В то время рынок калькуляторов развивался очень динамично, поэтому любые микросхемы пришлось бы менять каждые несколько лет. Естественно, при этом снижалась рентабельность производства БИС и увеличивалась их стоимость. И вот инженеру Тэду Хоффу³⁾ пришла в голову революционная идея: а почему бы не создать простой ЦПУ на кристалле? Его можно было бы запрограммировать на реализацию функций калькулятора, а для расширения функциональности устройства по мере появления новых требований достаточно будет просто усовершенствовать его программное обеспечение. Все это значительно увеличивало срок жизни такой микросхемы и ее рентабельность. Кроме того, не следует забывать, что основной сферой деятельности компании Intel было производство микросхем памяти, а компьютероподобные архитектуры требуют ее очень много! Это была поистине блестящая мысль. Разумеется, в конце 1969 года японские заказчики одобрили предложение компании Intel, поскольку оно было простым и гораздо более гибким, нежели традиционные решения.

Весной 1970 года в Intel появился Федерико Фаггин⁴⁾, а уже к концу года были изготовлены рабочие образцы первого в мире чипсета. Эксклюзивными правами на его приобретение обладала компания Busicom. Однако к середине 1971 года у нее возникли серьезные финансовые затруднения, и компания Intel, возместив затраты на разработку этого чипсета в размере 65 000 долл., получила права на его продажу всем желающим. В то время рыночные перспективы этого изделия были достаточно туманны, но Intel все же решила рискнуть и опубликовала в ноябрьском номере журнала «Electronic News» за 1971 год рекламу своего «микропрограмируемого компьютера на кристалле», получившего обозначение 4004 (тер-

¹⁾ Именно он высказал в 1965 году предположение, что число транзисторов на кристалле будет удваиваться каждые 18 месяцев (в то время типичная ИС состояла из 50 транзисторов). Эту зависимость, которая получила название первого закона Мура, он вывел путем экстраполяции роста емкости микросхем с 1959 года. Впоследствии эта зависимость пересматривалась каждые два года и все время с блеском подтверждалась.

²⁾ Считается, что это название произошло от слов INTELLigence (интеллект) или INTEgrated ELectronics (интегрированная электроника).

³⁾ Говорят, что эта идея возникла у него, когда он отдыхал на топлесс-пляже на Таити.

⁴⁾ Именно он позже основал компанию Zilog, которая стала известна благодаря своему микропроцессору Z80 — основному конкуренту процессора Intel 8085.

мин микропроцессор вошел в обиход только после 1972 года). Появление процессора 4004 вызвало бурный интерес, поскольку он позволял внедрить «интеллект» в электронную технику.

Микропроцессор 4004 имел фон-неймановскую архитектуру с 4-битной шиной данных и позволял напрямую адресовать до 512 байт памяти. Он работал на частоте 108 кГц и содержал 2300 транзисторов¹⁾. Годом позже был выпущен 8-битный микропроцессор 8008, работавший на частоте 200 кГц и позволявший адресовать 16 Кбайт памяти. Эта микросхема состояла уже из 3500 транзисторов. Если четырех битов было вполне достаточно для работы с BCD-числами, использующимися в калькуляторах, то 8-битная архитектура уже годилась для создания интеллектуальных терминалов (наподобие кассовых аппаратов), в которых требовалась поддержка разнообразных алфавитно-цифровых символов. В 1974 году на смену 8008 пришел микропроцессор 8080²⁾, а в 1976 году появился слегка модифицированный вариант последнего — 8085. Нужно сказать, что 8-битный микропроцессор 8085 до сих пор выпускается компанией Intel.

Идея микропроцессора оказалась настолько удачной, что множество других производителей электронных компонентов тоже поспешили застолбить место на этом рынке. Более того, многие бывшие разработчики открывали собственные компании, взять, к примеру, ту же Zilog. К 1976 году было выпущено или хотя бы анонсировано 54 различных моделей микропроцессоров. Так, родоначальником одного из семейств, имевших наибольший успех на рынке, был микропроцессор 6800, разработанный компанией Motorola³⁾. Этот микропроцессор имел ясную и гибкую фон-неймановскую архитектуру, мог работать на частоте 2 МГц и адресовать до 64 Кбайт памяти. В модели 6802 (1977) была даже встроенная память объемом 128 байт и внутренний тактовый генератор. В 1979 году была выпущена усовершенствованная модель 6809, ставшая последним представителем этого семейства 8-битных микропроцессоров. Основными ее конкурентами были такие микропроцессоры, как 8085 компании Intel, Z80 компании Zilog и 6502 компании MOS Technology.

Вообще говоря, изначально микропроцессоры не предназначались для использования в обычных компьютерах. Но в 1975 году небольшая компания — производитель калькуляторов MITS⁴⁾, оказавшись на грани банкротства, совершила рискованный ход и переориентировалась на изготовление и продажу компьютеров. Прimitивная вычислительная машина, разработанная инженером Эдом Робертсом (Ed Roberts), была построена на базе микропроцессора 8080 компании Intel. Взаимодействие с оператором осуществлялось посредством переключателей и лампочек, располагавшихся на передней панели, — никакой клавиатуры и монитора. В течение нескольких недель после начала рекламной акции компания получила

¹⁾ Сравните с 5.5 млн транзисторов процессора Pentium Pro (известного также как P6 или 80686).

²⁾ Разработанный Масатоши Шима (Masatoshi Shima), который позже перешел в компанию Zilog и занялся там разработкой микропроцессора Z80, совместимого с 8080.

³⁾ Компания Motorola была основана в 30-х годах как производитель автомобильных радиоприемников. Во время написания книги (2005 год) она занимала ведущую позицию на мировом рынке микроконтроллеров.

⁴⁾ Располагавшаяся в Нью-Мексико по соседству с массажным кабинетом.

около 650 предварительных заказов на этот компьютер, названный «Альтаир»¹⁾ (сумма каждого заказа составляла около 400 долл.). В результате вместо долга на сумму 400 000 долл. компания получила прибыль в размере 250 000 долл.

Этот первый *персональный компьютер* (ПК) породил целое поколение компьютерных фанатов. Так, однажды, в декабре 1975 года, никому доселе не известный 19-летний студент факультета вычислительной техники Гарвардского университета Билл Гейтс (Bill Gates) и зашедший к нему в гости приятель Пол Аллен (Paul Allen) увидели фотографию «Альтаира»²⁾ на обложке журнала «Popular Electronics» и решили заняться написанием программного обеспечения для этого ПК. Они связались по телефону с Эдом Робертсом и сообщили ему, что у них есть уже почти завершённый транслятор языка Бейсик для «Альтаира» (вообще-то это была, мягко говоря, неправда). Так на свет появилась корпорация Microsoft.

Примерно двумя месяцами позже несколько десятков человек основали в Сан-Франциско своеобразный компьютерный клуб, приобретая в складчину один «Альтаир» на всех. В числе членов этого клуба были Стив Джобс (Steve Jobs) и Стив Возняк (Steve Wozniak). В качестве демонстрации работы клуба они собрали собственный ПК, назвав его «Apple»³⁾. К 1978 году объём продаж компьютеров «Apple II» составил 700 000 долл.; в течение 1979 года этих компьютеров было продано на сумму 7 млн долл., в следующем году — на сумму 48 млн долл....

Компьютер «Apple II» был построен на базе недорогого микропроцессора 6502 производства компании MOS Technology. Разработчиком этого процессора (и одним из основателей компании) был Чак Педдл (Chuck Peddle), ранее служивший в компании Motorola и отвечавший там за разработку процессора 6800. Неудивительно, что микропроцессор 6502 до боли напоминал это предыдущее детище Чака. Компания Motorola даже подала в суд с требованием запретить продажу микропроцессора 6501, тем более что его цоколевка полностью совпадала с цоколевкой их процессора 6800. Вплоть до конца 70-х годов микропроцессор 6502 оставался одним из основных игроков на рынке ПК, будучи, помимо всего прочего, «сердцем» таких известных в то время компьютеров, как «BBC Micro»⁴⁾ и «Commodore PET»⁵⁾.

Основным фактором, реально повлиявшим на популярность компьютера «Apple II», было наличие у последнего пакета программ «VisiCalc» для работы с электронными таблицами. Когда бизнес-сообщество осознало, что ПК — это не просто игрушка и что с его помощью можно решать «реальные» задачи, объём продаж этих компьютеров резко подскочил. То же самое произошло и с компьютерами IBM PC. Этот ПК, представленный компанией IBM в 1981 году, был пост-

¹⁾ По названию планеты (Altair) из популярного телесериала «Star Trek».

²⁾ На фотографии была изображена всего лишь экспериментальная модель — к тому времени компьютеры еще не были готовы. Так что первые экземпляры «Альтаиров» существовали только в воображении заказчиков!

³⁾ От англ. слова «apple» — яблоко. Дело в том, что Джобс придерживался исключительно фруктовой диеты, кроме того, одно время он работал в садоводческом хозяйстве.

⁴⁾ Один из первых домашних компьютеров. Был спроектирован и разработан компанией Acorn Computers Ltd. для корпорации BBC. — *Примеч. пер.*

⁵⁾ Разработка компании Commodore Business Machines — предшественник одного из самых популярных домашних компьютеров «Commodore 64». — *Примеч. пер.*

роен на базе микропроцессора 8088, работавшего на частоте 4.77 МГц. В компьютере имелось ОЗУ объемом 128 Кбайт, два дисководов для дискет объемом 360 Кбайт и монохромный дисплей, работавший в текстовом режиме. В качестве операционной системы в нем использовалась ОС PC/MS-DOS версии 1.0 компании Microsoft. В комплекте с ней поставлялся пакет программ для обработки электронных таблиц «Lotus 1-2-3».

Уровень развития технологии изготовления кремниевых СБИС, достигнутый к концу 70-х, сделал возможным размещение на одном кристалле нескольких десятков и даже сотен тысяч транзисторов. И сразу же перед разработчиками микропроцессоров встал вопрос: каким образом использовать эту возможность? Наиболее очевидным и, соответственно, наиболее популярным направлением совершенствования микроконтроллеров было увеличение разрядности АЛУ и емкости шин/памяти. По этому пути, в частности, пошла компания Intel, выпустив в 1978 году микропроцессор 8086 (16-битный вариант микропроцессора 8085¹⁾), имевший в своем составе 29 000 транзисторов.

При его разработке особое внимание было уделено программной и аппаратной совместимости с его 8-битным предшественником. С коммерческой точки зрения это было очень мудрым решением, поскольку позволяло удержать многочисленных потребителей процессора 8085 от перехода к изделиям конкурентов. В то же время это решение было весьма неоднозначным с технической точки зрения. Так или иначе, но выпуск этого микропроцессора оказался несколько преждевременным, поэтому в оригинальных компьютерах IBM PC компании IBM использовалась модифицированная версия 8086 — процессор 8088, имевший урезанную 8-битную шину данных и 20-битную шину адреса²⁾.

В 1979 году компания Motorola представила свой вариант 16-битного микропроцессора, получившего название 68000, а также его модификацию 68008 с 8-битной шиной данных. Однако внутренняя организация всех этих микропроцессоров была 32-битной, за счет чего удалось обеспечить их совместимость с более поздними моделями, вплоть до выпущенного в 1995 году микропроцессора 68060, а также RISC-процессора ColdFire, появившегося в 1997 году. Микропроцессоры семейства 68000 представляли собой совершенно новую разработку и технически были более прогрессивными по сравнению со своими конкурентами семейства 80x86.

Компания Apple решила использовать процессор 68000 в своих новых ПК «Macintosh». Однако, несмотря на все преимущества этого процессора, объем продаж компьютеров Apple Mac составил менее 5% от объема продаж IBM PC. Гораздо больших успехов компания Motorola добилась на рынке микропроцессоров для встраиваемых систем — начиная от яйцеварок и заканчивая системами управления самолетов. Конечно же, микропроцессоры изначально были разработаны именно для этой области, поэтому количество микропроцессоров, продан-

¹⁾ На долю микропроцессоров с интеловской архитектурой 8086 приходится наибольшее число продаж.

²⁾ При этом объем адресуемой памяти составлял $2^{20} = 1$ Мбайт. Именно по этой причине для обеспечения обратной совместимости в MS-DOS было введено ограничение на 1 Мбайт основной памяти. В среде Microsoft Windows эта область памяти называется реальной памятью.

ных для использования во встраиваемых системах, более чем на порядок превысило количество, проданное для нужд компьютерного рынка.

В таких устройствах микропроцессор «спрятан» в недрах системы вместе с памятью и различными интерфейсными схемами ввода/вывода. То есть он выступает в роли центрального контроллера, управляя системой в соответствии с программой, зашитой в его памяти программ. Ежегодно для использования во встраиваемых системах продается свыше 3.5 млрд микропроцессоров и сопутствующих микросхем, что составляет более 95% всего рынка микропроцессоров.

Другое направление совершенствования микроконтроллеров, ставшее возможным в конце 70-х, заключалось в сохранении относительно простого ЦПУ и использовании оставшихся ресурсов кристалла для реализации встроенной памяти и интерфейсов ввода/вывода. Это дало возможность создавать простые встраиваемые системы управления на одной-единственной микросхеме, значительно уменьшая таким образом общее число микросхем, необходимое для реализации заданной функции. Для реализации подавляющего большинства задач управления большой вычислительной мощности не требуется, а вот уменьшение размера готовых устройств и, соответственно, их стоимости крайне желательно. В качестве простого примера можно привести смарт-карту с интегрированным процессором. Такие микропроцессорные устройства получили название микроконтроллеров¹⁾. Например, в каждом доме, незаметно для нас, обитает несколько сот микроконтроллеров. Они есть повсюду — в бытовой технике, аудио- и видеоаппаратуре, персональных компьютерах, телекоммуникационных устройствах, смарт-картах и, в том числе, в автомобилях.

Если микропроцессор с точки зрения архитектуры (см. **Рис. 3.1** и **Рис. 3.2**) представляет собой только блок центрального процессора, то микроконтроллер уже является законченной самодостаточной компьютероподобной системой. Рассмотрим в качестве примера электронную часть системы контроля автомобильного одометра, которая отображает общий пробег автомобиля с момента изготовления, а также дальность последней поездки (так называемый путевой одометр). Основным входным сигналом системы является сигнал от автомобильного тахометра, который формирует импульсы при каждом обороте маховика двигателя. Подсчитав суммарное количество этих импульсов, можно определить количество оборотов двигателя, а по интервалу между импульсами можно вычислить скорость движения автомобиля. Разумеется, реальный путь, проходимый автомобилем, зависит от передаточного числа коробки передач, поэтому нам необходимо знать о том, какая из пяти передач была включена водителем в каждый момент времени. Эта информация поступает из коробки передач по линиям G5, ..., G1 (обычно обозначаемым как G[5:1]). Включенной передаче соответствует напряжение ВЫСОКОГО уровня на соответствующей линии (передача заднего хода не учитывается). Два дополнительных входа предназначены для задания единицы измерения отображаемых значений (мили или километры) и для обнуления путевого одометра.

¹⁾ Изначально существовал еще один термин, *микрокомпьютер*, однако он вскоре исчез из употребления, поскольку точно так же назывались персональные компьютеры того времени.

Собственно дисплей одометра представляет собой семиразрядный 7-сегментный индикатор (см. **Рис. 6.8** на стр. 183), который может отображать значения до **999999.9**. Поскольку общее число сегментов довольно велико (целых 49), то для управления индикатором используется сдвиговый регистр (см. **Рис. 2.22** на стр. 51), данные в который передаются по одной линии (**Рис. 3.8**). По второй линии передаются тактовые импульсы — для полного обновления содержимого дисплея необходимо 49 импульсов¹⁾.

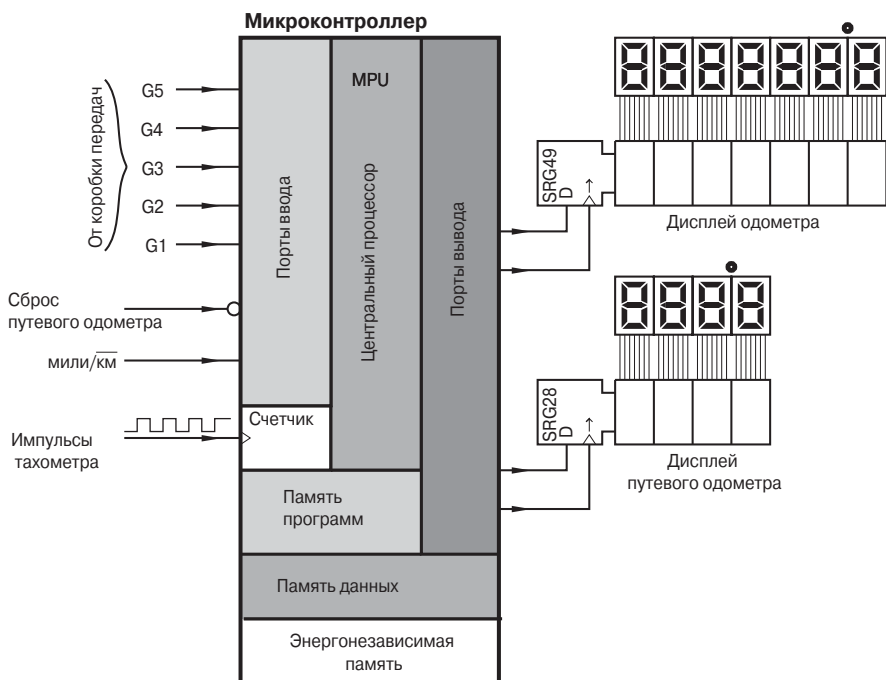


Рис. 3.8. Пример микроконтроллерной системы

Дисплей путевого одометра является 4-разрядным и позволяет отображать значения до **999.9**. Сдвиговый регистр этого дисплея тоже управляется по двум линиям, только в данном случае для вывода нового 4-разрядного значения необходимо 28 тактовых импульсов.

Для реализации этой системы нам потребуются следующие ресурсы (так называемый *бюджет ресурсов*):

- Вход, срабатывающий по фронту и подключенный к счетчику/таймеру для подсчета числа оборотов вала двигателя (на этот вход поступают импульсы от тахометра).
- Семь цифровых входов для ввода текущего передаточного отношения, задания единицы отображения (мили/км) и для сброса путевого одометра.

¹⁾ Во многих индикаторах уже имеется встроенный сдвиговый регистр.

- Четыре цифровых выхода для тактирования двух сдвиговых регистров и передачи информации о сегментах.
- Микропроцессор для выполнения вычислений, считывания входных сигналов и формирования выходных.
- Память программ, обычно ПЗУ какого-либо типа.
- Память данных для хранения рабочих переменных программы, обычно статическое ОЗУ.
- Энергонезависимая память для долговременного хранения информации, такой как суммарный путь, пройденный автомобилем, и расстояние, пройденное с момента последнего сброса путевого одометра.

Все эти функции могут быть реализованы в одной-единственной интегральной микросхеме, называемой в данном случае микроконтроллером, т.е. микропроцессором, интегрированным на одном кристалле со вспомогательными схемами и выполняющим работу целого микрокомпьютера. Разумеется, перечисленные ресурсы имеют отношение только к нашему примеру. Хотя основные узлы (микропроцессор и память) являются общими для широкого круга приложений, интерфейс ввода/вывода необходимо подбирать под каждую конкретную задачу. Причем эти интерфейсные модули могут быть самыми разными, например:

- Модули приема/передачи данных по последовательным каналам с использованием разнообразных синхронных и асинхронных протоколов.
- Модули счетчиков/таймеров для подсчета числа внешних событий и формирования цифровых сигналов с точными временными параметрами.
- Модули аналого-цифрового преобразователя для считывания и оцифровки входных аналоговых сигналов.
- Модули цифро-аналогового преобразователя для формирования выходных аналоговых сигналов.
- Модули специализированного интерфейса для управления многоразрядными жидкокристаллическими индикаторами (ЖКИ).

Такое использование дополнительных ресурсов кристалла привело к появлению в конце 70-х годов первых микроконтроллеров. К примеру, микроконтроллер Motorola 6801 (35 000 транзисторов), разработанный специально для использования в автомобилях, был построен на базе существующего микропроцессора 6800. Этот микроконтроллер имел ПЗУ программ объемом 2048 байт, ОЗУ данных объемом 128 байт, 29 линий ввода/вывода и 16-битный таймер. После того как микроконтроллеры доказали свою жизнеспособность, все ведущие производители микропроцессоров выпустили на рынок различные семейства микроконтроллеров. Каждое из этих семейств базировалось на определенном ядре, при этом различные представители одного и того же семейства отличались набором периферийных устройств. Например, в семействе 68HC11 компании Motorola (дальнейшее развитие микроконтроллера 6801) было использовано слегка модернизированное ядро 6800. Семейства 68HC12 и 68HC16 имели уже 16-битные ядра, которые, однако, обеспечивали совместимость с предыдущим 8-битным семейством 68HC11. Вскоре выяснилось, что во многих встраиваемых приложениях вовсе не требуются все вычислительные возможности древнего ядра 6800, поэтому было выпущено новое

семейство 68HC05¹⁾, представители которого имели значительно урезанное ядро и, соответственно, меньшую стоимость. Как это ни удивительно, но 4-битные микроконтроллеры, такие как TMS1000 компании Texas Instruments, лидировали по объему продаж среди всех остальных разновидностей процессоров вплоть до начала 90-х (и до сих пор продолжают пользоваться устойчивым спросом). Похоже, что и 8-битным микроконтроллерам, ставшим в последнее время наиболее популярными, в обозримом будущем уготована та же судьба. Кстати говоря, процессор 14500 компании Motorola вообще был однобитным!

В основе всех этих микропроцессоров и микроконтроллеров лежала фон-неймановская архитектура, используемая в универсальных ЭВМ. Альтернативная гарвардская архитектура впервые возродилась в микропроцессоре 8X300 компании Signetics, который в середине 70-х был приспособлен компанией General Instruments для работы в качестве *периферийного интерфейсного контроллера* (Peripheral Interface Controller — PIC). Компания собиралась использовать этот контроллер как программируемый порт ввода/вывода для своего 16-битного микропроцессора CP1600. После того как в 1988 году компания General Instruments продала свое подразделение интегральных микросхем молодой компании, названной Arizona Microchip Technology, это устройство вновь появилось на свет, но уже в виде самостоятельного микроконтроллера. Именно данному семейству микроконтроллеров и посвящена оставшаяся часть книги.

Примеры

Пример 3.1

Контроллер теплицы должен контролировать аналоговый сигнал от датчика влажности почвы и, если его величина ниже некоторого порогового значения, открывать водяной клапан на пять секунд с последующей 5-секундной паузой. В резервуаре с водой имеется поплавковый датчик, который замыкает контакты при снижении уровня воды ниже порогового значения. В этом случае должен включаться звуковой сигнализатор для индикации тревоги.

Можете ли вы придумать систему на базе микроконтроллера, реализующую указанные функции?

Решение

В решении, приведенном на **Рис. 3.9**, используется автомобильный одометр с **Рис. 3.8**. Единственным новым периферийным устройством является аналоговый порт, используемый для считывания и оцифровки аналогового сигнала от датчика влажности почвы. В основе работы этого датчика лежит зависимость сопротивления почвы от ее влажности. Электроды датчика, включенные последовательно с постоянным резистором, образуют делитель напряжения, выходное напряжение

¹⁾ Микроконтроллеры семейства 68HC05 прочно обосновались в нише процессоров для смарт-карт, где производительность является не самым важным параметром.

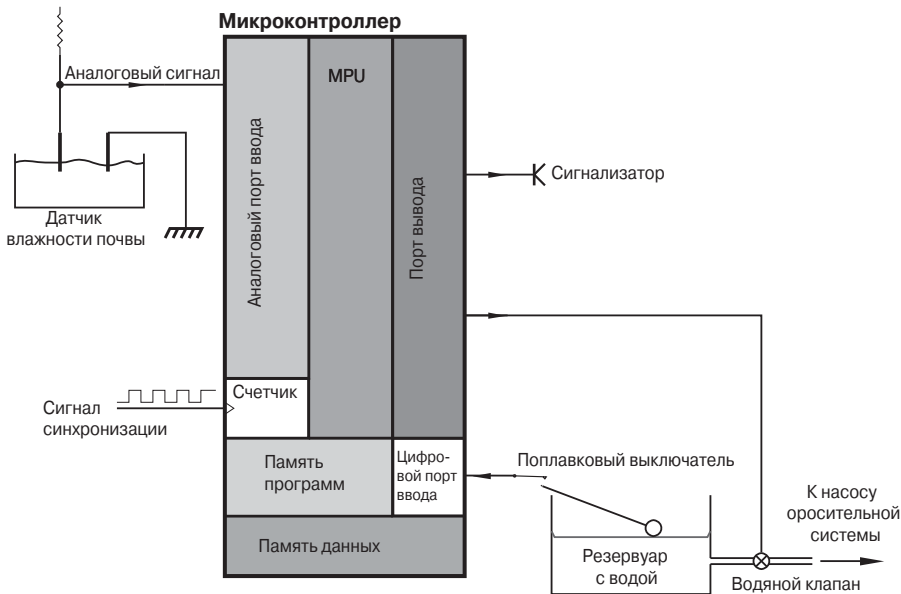


Рис. 3.9. Климатический контроллер теплицы

которого будет меняться в зависимости от влажности почвы. Микроконтроллер может преобразовать это аналоговое напряжение в соответствующий цифровой код, который затем будет сравниваться в программе с предустановленным значением. Также порт ввода может представлять собой обыкновенный аналоговый компаратор, формирующий на выходе лог. 1 или лог. 0, если входное напряжение превышает определенное значение, которое может задаваться программно.

Глядя на **Рис. 3.9**, мы можем оценить требуемые ресурсы:

- Вход для внешнего генератора, подключенный к счетчику/таймеру. Это необходимо для того, чтобы микроконтроллер мог отсчитывать временные интервалы. На практике такие таймеры очень часто работают от внутреннего тактового сигнала микроконтроллера.
- Один аналоговый вход для измерения уровня аналогового сигнала от датчика влажности.
- Один цифровой вход для контроля уровня воды в резервуаре.
- Один цифровой выход для открытия и закрытия водяного клапана.
- Один цифровой выход для управления звуковым сигнализатором.
- Микропроцессор для вычислений, считывания входных и формирования выходных сигналов.
- Память программ, обычно ПЗУ какого-либо типа.
- Память данных для хранения рабочих переменных программы, обычно статическое ОЗУ.

Если учесть, что для выполнения указанных задач требуется не так уж и много времени, можно задействовать дополнительные входы микроконтроллера для

контроля других параметров, таких как температура и освещенность. В результате мы сможем осуществлять более комплексное управление климатической обстановкой в теплице.

Пример 3.2

Наиболее сложной проблемой, с которой приходится сталкиваться программисту, часто является собственно постановка решаемой задачи. Для этого необходимо логическое мышление, которым обладает человек и которое отсутствует у машины. Именно способность принимать решения и является отличительной чертой хорошего программиста. Эта способность складывается из опыта, капельки таланта, а также хорошего понимания решаемой задачи.

Чтобы проиллюстрировать процесс принятия решения, продумаем последовательность элементарных действий, которые должен будет выполнить робот с микроконтроллерным управлением для перехода через регулируемый пешеходный переход на улице с оживленным движением.

Решение

1. Подойти к переходу и остановиться.
2. Посмотреть на светофор.
3. Принять решение — не горит ли в нашем направлении зеленый сигнал?
4. ЕСЛИ сигнал красный, ТО перейти к шагу 2, ИНАЧЕ продолжить.
5. Посмотреть налево.
6. Едут ли машины?
7. ЕСЛИ да, ТО перейти к шагу 5, ИНАЧЕ продолжить.
8. Посмотреть направо.
9. Едут ли машины (вообще-то все машины уже должны были остановиться, но кто знает!)?
10. ЕСЛИ да, ТО перейти к шагу 5, ИНАЧЕ продолжить.
11. Перейти через дорогу — задача решена!

На **Рис. 3.10** описанный алгоритм представлен в графическом виде. В этой *блок-схеме* прямоугольники используются для обозначения действий, ромбы — для обозначения условий, а прямоугольники со скругленными углами — для обозначения точек входа и выхода. Линии со стрелками указывают последовательность выполнения действий и дополнительно помечаются в точках принятия решений. В принципе в данном конкретном случае графическое представление алгоритма не имеет больших преимуществ по сравнению с текстовым. Однако в более сложных задачах, со множеством условий и вариантов выполнения, графическое представление может оказаться гораздо удобнее для документирования поведения системы. А когда система становится очень сложной, то и простой перечень задач, и блок-схема становятся одинаково бесполезными. В этом случае

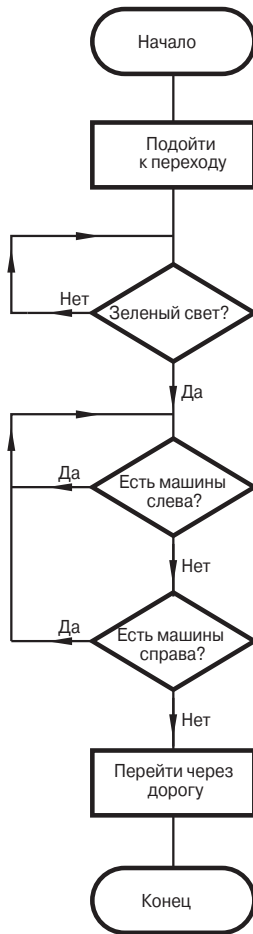


Рис. 3.10. Блок-схема алгоритма перехода через дорогу

описание системы необходимо строить по иерархическому принципу, начиная с самых общих вопросов и постепенно продвигаясь к более конкретным задачам.

На первый взгляд этот пример может показаться довольно глупым и надуманным, но именно эти операции вам придется совершать каждый раз при переходе загруженной улицы по регулируемому пешеходному переходу. И именно этот алгоритм вы должны заложить в робота, чтобы он смог сделать то же самое. Такая последовательность элементарных шагов, или инструкций, называется *программой*. Со стороны все эти действия, предпринимаемые роботом для перехода через улицу, могут показаться проявлением интеллекта. Но это не интеллект — интеллектом обладают люди. Это программист, запрограммировавший микроконтроллер робота, вложил в него необходимые знания.

Разумеется, робот не будет иметь ни малейшего понятия о том, что ему делать после перехода на другую сторону, если только мы не сообщим ему об этом. Что же касается человека, то он уже, образно говоря, «запрограммирован» — у него есть опыт!

Заметьте, что этапы пронумерованы в том порядке, в котором они должны выполняться. Счетчик команд, в данном случае читатель, начинает выполнение с 1-й команды (состояние сброса) и заканчивает выполнением 11-й команды. В микроконтроллере после выполнения действий, предписываемых текущим этапом, счетчик команд автоматически инкрементируется, указывая на следующий этап, если только текущая команда не была командой **пропуска** или **перехода**. При выполнении команды пропуска счетчик команд «перепрыгивает» через следующую команду, обычно при определенном условии или результате. А при выполнении команды перехода счетчик команд просто переходит к заданному этапу. Если бы таких команд не было, в программе нельзя было бы реализовать ветвления и циклы. Под циклом в данном случае понимается многократное повторение одних и тех же действий, например периодическая проверка наличия зеленого сигнала светофора до тех пор, пока он не включится.

Вопросы для самопроверки

- 3.1. Можете ли вы предложить вариант инкрементирования и декрементирования содержимого рабочего регистра, показанного на **Рис. 3.4**, с использованием трех команд, рассмотренных в этой главе?
- 3.2. Разработайте программу, которая позволит роботу с микроконтроллерным управлением из Примера 3.2 наполнить стакан водой из крана.
- 3.3. Компьютер BASIC, структура которого изображена на **Рис. 3.4**, может одновременно осуществлять выборку одной команды и исполнять другую команду. Объясните, за счет чего он может выполнять эти операции параллельно.
- 3.4. Составьте перечень задач, в соответствии с которым робот сможет пройти к ближайшему банкомату, снять со счета заданную сумму наличных, запросить баланс и вернуться на исходную позицию. Не забудьте про обработку запроса на печать баланса, а также продумайте действия робота при отсутствии на счете достаточной суммы денег!
- 3.5. Для подключения коробки передач к микроконтроллерной системе, показанной на **Рис. 3.8**, требуется пять выводов микросхемы. Многие микроконтроллеры выпускаются в корпусах с малым числом выводов (см., например, **Рис. 10.2** на стр. 304). Подумайте, как можно уменьшить требуемое число выводов, а также будет ли ваше решение экономически оправданным? **Подсказка:** взгляните на **Рис. 2.6** (стр. 36).
- 3.6. Подумайте, каким образом можно уменьшить на единицу количество выходов микроконтроллера, требуемых для управления дисплеями обоих одометров, и насколько это будет удобно?