ОГЛАВЛЕНИЕ

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 сайт[1)]

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 книги[1)] набиралась автором на различных ПК, работающих под управлением Microsoft® Windows™ с использованием среды $LaTeX2\varepsilon$ (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 гремлинов, 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

[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. Исследования[1)], 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-битных[1)] 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

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

ЧАСТЬ I

ОСНОВЫ

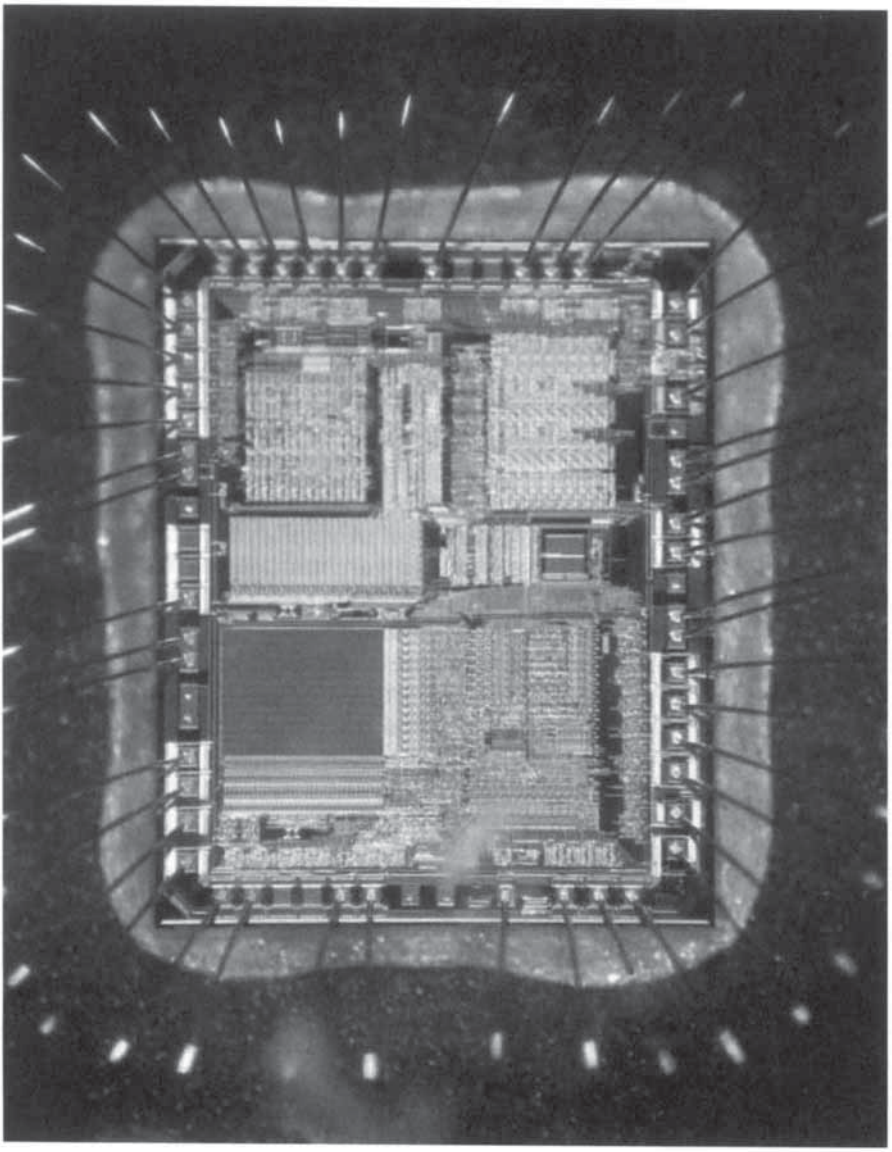
Эта книга посвящена микроконтроллерам. Микроконтроллеры представляют собой цифровые устройства, построенные по образу и подобию ЭВМ с хранимой программой и объединенные вместе со вспомогательными узлами, памятью различного типа и блоками сопряжения в микросхемах сверхвысокой степени интеграции. Хотя, 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 книг[1)] 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

ГЛАВА 1

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 «A» соответствует число 65, символу «B» — 66,..., «Z» — 90, «a» — 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 символы[2].

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 виде[3], 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 пальцев[4]. 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 микропроцессоров[2] 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 символы[3] 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:

10^3	10^2	10^1	10^0
1	9	9	8

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:

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

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, ..., 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 арифметика[1)] 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:

$$\begin{array}{ll} 0+0 & = 0 \\ \left.\begin{array}{l} 0+1 \\ 1+0 \end{array}\right\} & = 1 \\ 1+1 & = 10 \quad (0 \text{ и } 1 \text{ в переносе}) \end{array}$$

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		б) Двоичное число	
1 0 1 0 0 1		1 2 6 3 1 8 4 2 6 8 4 2 1	
96	1-е слагаемое	1100000	1-е слагаемое
+ 37	2-е слагаемое	+ 0100101	2-е слагаемое
~~11~~	Переносы	~~1 1~~	Переносы
133	Сумма	10000101	Сумма

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.

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

Например:

а) Десятичное число		б) Двоичное число	
1		6 3 1	
0 1		4 2 6 8 4 2 1	
96	Уменьшаемое	1100000	Уменьшаемое
– 37	Вычитаемое	– 0100101	Вычитаемое
–1–	*Заемы*	–1111111–	*Заемы*
59	Разность	0111011	Разность

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		б) Двоичное число	
37	Уменьшаемое	0100101	Уменьшаемое
– 96	Вычитаемое	– 1100000	Вычитаемое
–1–		–1–––––	
41	Разность (–59)	1000101	Разность (–0111011)

Обычно, если мы знаем, что уменьшаемое меньше вычитаемого, мы меняем операнды местами и добавляем знак минуса к результату, т.е. вычисляем выражение – (вычитаемое – 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 ($0 \to 1,\ 1 \to 0$) делает этот метод очень привлекательным. Обратимся к приведенному выше примеру:

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

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

$$\text{дополнительный код} \Leftrightarrow \text{прямой код}.$$

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'**1**1000101' будет соответствовать значению –59, а b'**0**0111011' — 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		б) Уменьшаемое больше вычитаемого	
01100000	(+96)	**0**0100101	(+37)
− **1**1011011	(–37)	− **1**0100000	(–96)
00111011	(+59)	**1**1000101	(–59)

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

```
  01000  (+8)
+ 01011  (+11)
  -----
  10011  (–13!!!)
```

б) Сумма двух отрицательных чисел получается положительной

```
  11000  (–8)
+ 10101  (–11)
  -----
  01101  (+13!!!)
```

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

```
  00011001  (+25)
+     0011  (+03)
  --------
    ????
```

```
  00011001  (+25)
+ 00000011  (+03)
  --------
  00011100  (+28)
```

б) Расширение отрицательного числа

```
  00011001  (+25)
+     1101  (–03)
  --------
    ????
```

```
  00011001  (+25)
+ 11111101  (–03)
  --------
  00010110  (+22)
```

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:

а) $+3 \times 8 = +24$

000000011 (3)
<<
000000110 (6)
<<
000001100 (12)
<<
000011000 (24)

б) $-3 \times 8 = -24$

111111101 (–3)
<<
111111010 (–6)
<<
111110100 (–12)
<<
111101000 (–24)

в) $+3 \times 10 = 30$

000000110 (3×2)
+ **0**00011000 (3×8)

000011110 ($3 \times 10 = 30$)

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:

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

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:

а) $+15/8 = 1.875$

01111.000 (+15)
>>
00111.100 (+7.5)
>>
00011.110 (+3.75)
>>
00001.111 (+1.875)

б) $-15/8 = -1.875$

10001.000 (–15)
>>
11000.100 (–7.5)
>>
11100.010 (–3.75)
>>
11110.001 (–1.875)

в) $15/10 = 1.5$

```
    0001.1
1010| 1111.0
   -1010
    0101
    -101.0
     000.0
```

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.

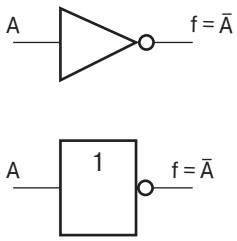
Деление на число, не являющееся степенью двойки, показано в примере (**в**). Эта операция осуществляется аналогично операции деления столбиком в десятичной системе. При ее выполнении по аналогии с умножением используется комбинирование операций сдвига и вычитания.

Арифметические действия — не единственные операции, которые можно осуществлять над двоичными числами. Английский математик Джордж Буль[1] (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 $f = \overline{A}$ 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: $\overline{\overline{f}} = f$[2].

A	f
0	1
1	0

а) *Таблица истинности*



б) *Альтернативные варианты изображения логического элемента*

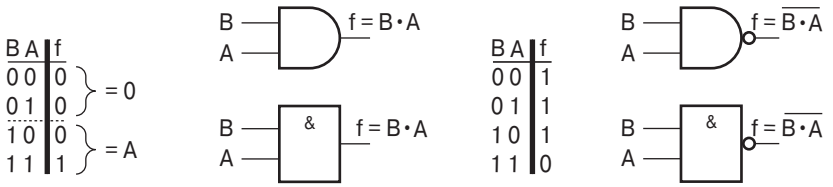
Рис. 1.1. Операция НЕ (NOT)

[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]. Кружок 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** имеется две входные переменные, и выражение для выходного значения записывается как $f = B \cdot A$, где символ «·» — булевый оператор И[2]. Количество входных переменных может быть любым, и в общем случае $f = A(0) \cdot A(1) \cdot A(2) \cdot \ldots \cdot A(n)$. 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.



а) *Таблица истинности* ***б)*** *Альтернативные варианты изображения логического элемента* ***в)*** *Операция И-НЕ (NAND)*

Рис. 1.2. Операция И (AND)

Если предположить, что вход B является управляющим входом, а вход A — 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]:

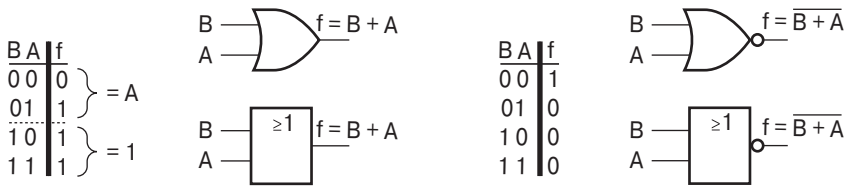
---

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

[2] Иногда для обозначения оператора И используется знак «/\». — *Примеч. пер.*

[3] В отечественной литературе оператор ИЛИ часто обозначается также знаком «\/». — *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.

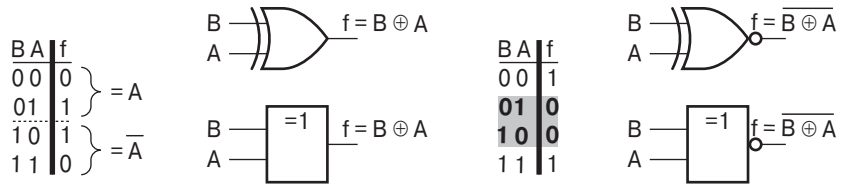


а) Таблица истинности *б) Альтернативные варианты изображения логического элемента* *в) Операция ИЛИ-НЕ (NOR)*

Рис. 1.3. Операция ИЛИ (OR)

Если предположить, что вход B является управляющим входом, а вход A — 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.



а) Таблица истинности *б) Альтернативные варианты изображения логического элемента* *в) Операция Исключающее ИЛИ-НЕ (XNOR)*

Рис. 1.4. Операция Исключающее ИЛИ (XOR)

Если предположить, что вход B — управляющий, а вход A — вход данных (или наоборот), тогда

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

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

Другим полезным применением функции Исключающее ИЛИ можно назвать использование ее в качестве логического дифференциатора. Из таблицы истинности (**Рис. 1.4, а**) видно, что выход элемента Исключающее ИЛИ истинен только тогда, когда состояния обоих входов различны. Аналогично, из таблицы истинности оператора Исключающее ИЛИ-НЕ (XNOR), показанной на **Рис. 1.4, в**, видно, что выход такого элемента истинен при одинаковых сигналах на обоих входах. Таким образом, вентиль Исключающее ИЛИ-НЕ можно рассматривать в качестве 1-битного компаратора. Равенство двух *n*-битных значений можно проверить, объединив по И набор вентилей Исключающее 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 Исключающее ИЛИ и Исключающее ИЛИ-НЕ рассмотрим задачу определения *переполнения* в знаковом бите (см. стр. 24). Эта ситуация возникает, если знаковые биты обоих операндов одинаковы $\overline{(S_B \oplus S_A)}$, а знаковый бит C 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 Исключающее ИЛИ можно использовать для определения четного количества истинных входов. При каскадном соединении $n + 1$ вентилей Исключающее 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.

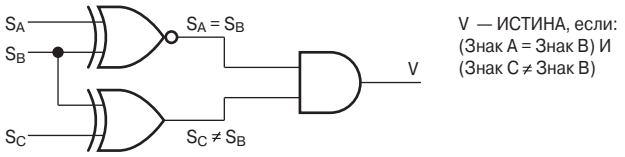


Рис. 1.5. Обнаружение переполнения в знаковом бите