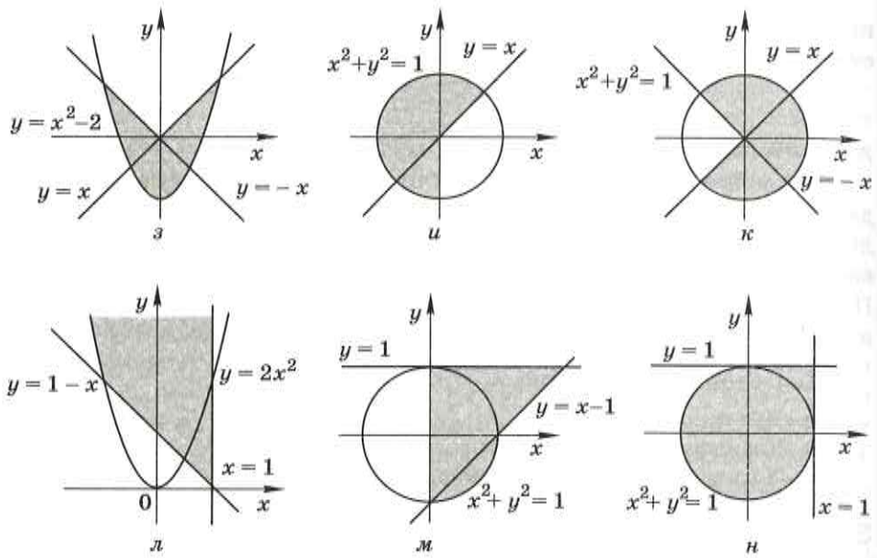


3. Поставьте в начале каждого предложения одно из слов: «все» или «не все».

а) «... окуни — рыбы».
б) «... рыбы умеют плавать».
в) «... реки впадают в моря».
г) «... моря солёные».
д) «... числа чётные».
е) «... ломаные состоят из отрезков».
ж) «... прямоугольники — квадраты».
з) «... кошки — млекопитающие».

4. Запишите с помощью кванторов следующие утверждения.

а) «Существует x, такой что $x > y$».
б) «Не существует x, такой что $x > y$».
в) «Для любого x имеем $x^2 > 1$».
г) «Любая река впадает в Каспийское море».
д) «Существует река, которая впадает в Каспийское море».
е) «Для любой реки существует море, в которое она впадает».
ж) «Для любого моря существует река, которая в него впадает».
з) «Существует река, которая впадает во все моря».
и) «Существует море, в которое впадают все реки».

5. Запишите с помощью кванторов следующие утверждения:

а) «Некоторые школьники ходят в театр».
б) «Все кошки серые».
в) «Встречаются злые собаки».
г) «Все люди разные».
д) «Люди ошибаются».
е) «Никто не обращает на него внимания».
ж) «Ни одна фирма не обанкротилась».
з) «Все лебеди — белые или чёрные».

6. Запишите отрицание для следующих утверждений.

а) $\exists x(x^2 = 5)$;
б) $\exists x(x + y = 5)$;
в) $\forall y(x + y = 5)$;
г) $\forall y \exists x(x + y = 5)$;
д) $\exists x$ («x работает в вузе»);
е) $\forall x$ («x — студент»);
ж) $\exists x$ («x — учитель y»);
з) $\exists x \forall y$ («x — учитель y»).

§ 24
Логические элементы компьютера

Простейшие элементы

В компьютерах все вычисления выполняются с помощью **логических элементов** — электронных схем, выполняющих логические операции. Обозначения простейших элементов приводятся на рис. 3.23 (ГОСТ 2.743-91). Заметьте, что небольшой кружок на выходе (или на входе) обозначает операцию «НЕ» (отрицание, инверсию).

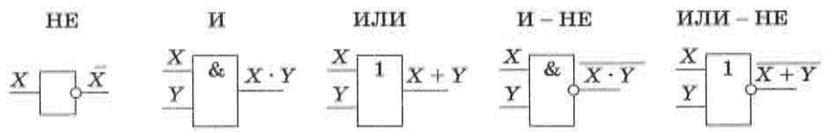


Рис. 3.23

Может показаться, что для реализации сложных логических функций нужно много разных видов логических элементов. Однако, как мы видели в § 22, любую логическую функцию можно представить с помощью операций «НЕ», «И» и «ИЛИ» (такой набор элементов называется **полным**). Именно эта классическая «тройка» используется в книгах по логике, а также во всех языках программирования. Тем не менее инженеры часто предпочитают строить **логические схемы** на основе элементов «И–НЕ». Как показано в § 19, эта функция (штрих Шеффера) позволяет реализовать операции «НЕ», «И» и «ИЛИ», а значит и любую другую операцию.

Если нужно составить схему по известному логическому выражению, её начинают строить с конца. Находят операцию, которая будет выполняться последней, и ставят на выходе соответствующий логический элемент. Затем повторяют то же самое для сигналов, поступающих на вход этого элемента. В конце концов должны остаться только исходные сигналы — переменные в логическом выражении.

Составим схему, соответствующую выражению

$$X = \overline{A} \cdot B + A \cdot \overline{B} \cdot \overline{C}.$$

Последняя операция — это логическое сложение, поэтому на выходе схемы будет стоять элемент «ИЛИ» (рис. 3.24).

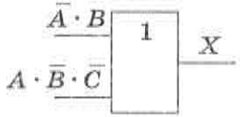


Рис. 3.24

Для того чтобы получить на первом входе $\overline{A} \cdot B$, нужно умножить $\overline{A}$ на B, поэтому добавляем элемент «И» (рис. 3.25).

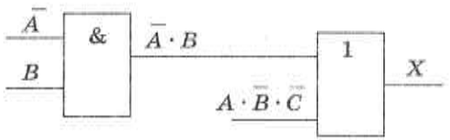


Рис. 3.25

Чтобы получить $\overline{A}$, ставим элемент «НЕ» (рис. 3.26).

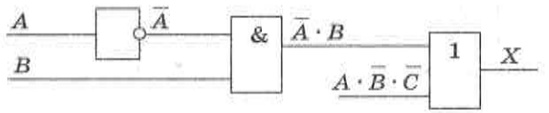


Рис. 3.26

Аналогично разбираем вторую ветку, которая поступает на второй вход элемента «ИЛИ» (рис. 3.27).

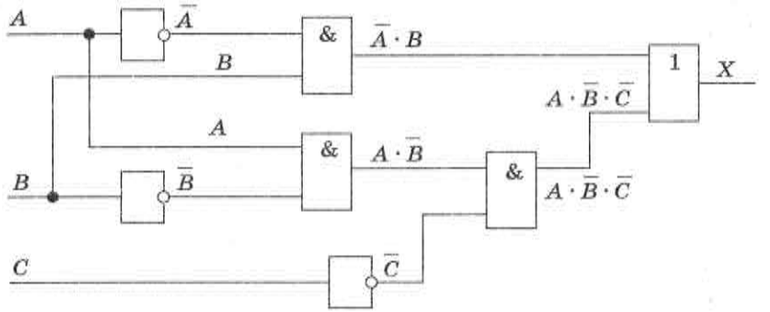


Рис. 3.27

Схема составлена, её входами являются исходные сигналы A, B и C, а выходом — X.

Триггер

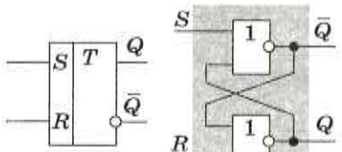
Слово **«триггер»** происходит от английского слова «trigger» — «защёлка» или спусковой крючок[1]. Так называют электронную схему, которая может находиться только в двух состояниях (их можно обозначить как 0 и 1) и способна почти мгновенно переходить из одного состояния в другое. Триггер изобрели независимо друг от друга М. А. Бонч-Бруевич и англичане У. Иккльз и Ф. Джордан в 1918 г.

[1] В английском языке триггер называется flip-flop.

В современных компьютерах на основе триггеров строится быстродействующая оперативная память. Один триггер способен хранить один бит данных. Соответственно, для того, чтобы запомнить 1 байт информации, требуется 8 триггеров, а для хранения 1 килобайта данных — $8 \cdot 1024 = 8192$ триггера.

Триггеры бывают разных типов. Самый распространённый — это **RS-триггер**. Он имеет два входа, которые обозначаются как S (англ. *set* — установить) и R (англ. *reset* — сброс), и два выхода — Q и $\bar{Q}$, причём выходной сигнал $\bar{Q}$ является логическим отрицанием сигнала Q (если $Q = 1$, то $\bar{Q} = 0$, и наоборот).

RS-триггер можно построить на двух элементах «И–НЕ» или на двух элементах «ИЛИ–НЕ». На рисунке 3.28 показано условное обозначение RS-триггера, внутреннее устройство триггера на элементах «ИЛИ–НЕ» и его таблица истинности.



S	R	Q	$\bar{Q}$	
0	0	Q	$\bar{Q}$	Хранение бита
0	1	0	1	Сброс в 0
1	0	1	0	Установка в 1
1	1	0	0	Запрещено

Рис. 3.28

Триггер использует так называемые **обратные связи** — сигналы с выходов схем «ИЛИ–НЕ» поступают на вход соседней схемы. Именно это позволяет хранить информацию.

Построим таблицу истинности триггера. Начнем с варианта, когда $S = 0$ и $R = 1$. Элемент «ИЛИ–НЕ» в нижней части схемы можно заменить на последовательное соединение элементов «ИЛИ» и «НЕ». Независимо от второго входа, на выходе «ИЛИ» будет 1, а на выходе «НЕ» — ноль. Это значит, что $Q = 0$ (рис. 3.29).

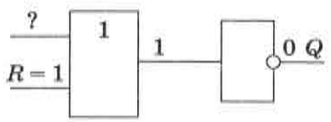


Рис. 3.29

Тогда на входе другого элемента «ИЛИ–НЕ» будут два нуля, а на выходе $\bar{Q}$ — единица (рис. 3.30).

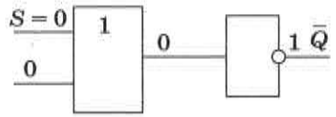


Рис. 3.30

Поскольку основным выходом считается Q, мы записали в триггер значение 0. Схема симметрична, поэтому легко догадаться, что при $S = 1$ и $R = 0$ мы запишем в триггер 1 ($Q = 1$).

Теперь рассмотрим случай, когда $S = 0$ и $R = 0$. На входе первого элемента «ИЛИ» будет сигнал $Q + 0 = Q$, поэтому на выходе $\bar{Q}$ останется его предыдущее значение (рис. 3.31).

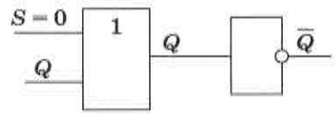


Рис. 3.31

Аналогично легко показать, что на выходе Q тоже остается его предыдущее значение. Это **режим хранения бита**.

Для случая $S = 1$ и $R = 1$ мы увидим, что оба выхода становятся равны нулю — в этом нет смысла, поэтому такой вариант запрещён.

Для хранения многоразрядных данных триггеры объединяются в единый блок, который называется **регистром**. Регистры (размером от 8 до 64 битов) используются во всех процессорах для временного хранения промежуточных результатов.

Над регистром, как над единым целым, можно производить ряд стандартных операций: сбрасывать (обнулять), заносить в него код и т. д. Часто регистры способны не просто хранить информацию, но и обрабатывать её. Например, существуют регистры-счётчики, которые подсчитывают количество импульсов, поступающих на вход.

Сумматор

Как следует из названия, сумматор предназначен для сложения (суммирования) двоичных чисел. Сначала рассмотрим более простой

элемент, который называют **полусумматором**. Он выполняет сложение двух битов с учетом того, что в результате может получиться двухразрядное число (с переносом в следующий разряд).

Обозначим через A и B входы полусумматора, а через P и S — выходы (перенос в следующий разряд и бит, остающийся в текущем разряде). Таблица истинности этого устройства показана на рис. 3.32.

A	*B*	*P*	*S*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Рис. 3.32

Легко увидеть, что столбец P — это результат применения операции «И» ко входам A и B, а столбец S — результат операции «исключающее ИЛИ»:

$$P = A \cdot B, \qquad S = A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}.$$

Формулу для S можно также записать в таком виде:

$$S = \overline{A} \cdot B + A \cdot \overline{B} = (A + B) \cdot (\overline{A} + \overline{B}) = (A + B) \cdot \overline{(A \cdot B)} = (A + B) \cdot \overline{P},$$

что позволяет построить полусумматор, используя всего четыре простейших элемента (рис. 3.33).

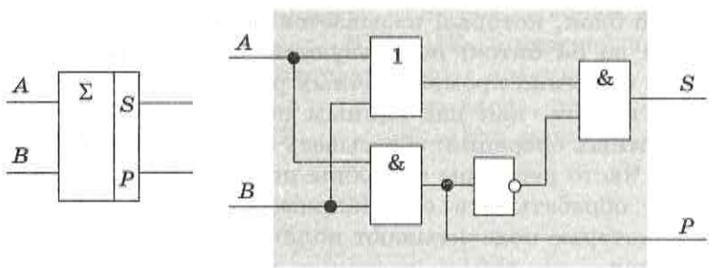


Рис. 3.33

Слева показано условное обозначение полусумматора, греческая буква Σ здесь (и в математике) обозначает сумму.

Полный **одноразрядный сумматор** учитывает также и третий бит C — перенос из предыдущего разряда. Сумматор имеет три входа и два выхода. Таблица истинности и обозначение сумматора показаны на рис. 3.34, 3.35.

A	*B*	*C*	*P*	*S*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Рис. 3.34

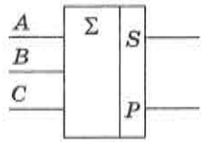


Рис. 3.35

Логические функции для выходов сумматора вы можете найти самостоятельно.

Сумматор можно построить с помощью двух полусумматоров и одного элемента «ИЛИ» (рис. 3.36).

Сначала складываются биты B и C, а затем к результату добавляется бит A. Перенос на выходе сумматора появляется тогда, когда любое из двух промежуточных сложений даёт перенос.

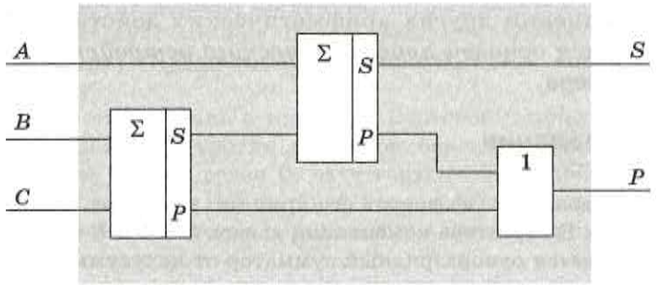


Рис. 3.36

Для сложения многоразрядных чисел сумматоры объединяют в цепочку. При этом выход P одного сумматора (перенос в следу-

ющий разряд) соединяется с входом C следующего. На рисунке 3.37 показано, как складываются два трёхразрядных числа: $X = 110_2$ и $Y = 011_2$. Сумма $Z = 1001_2$ состоит их четырёх битов, поэтому на выходе последнего сумматора бит переноса будет равен 1.

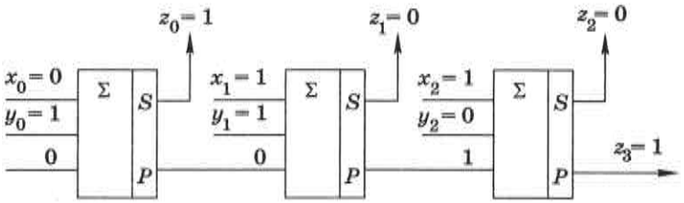


Рис. 3.37

Сложение начинается с самого младшего (нулевого) разряда. На вход первого сумматора подаются младшие биты исходных чисел, x_0 и y_0, а на третий вход — ноль (нет переноса из предыдущего разряда). Выход S первого сумматора — это младший бит результата, z_0, а его выход P (перенос) передаётся на вход второго сумматора и т. д. Выход P последнего из сумматоров представляет собой дополнительный разряд суммы, т. е. z_3.

Сумматор с последовательным переносом, показанный на рис. 3.37, работает слишком медленно. Поэтому в реальных процессорах применяют схемы с ускоренным переносом, которые выполняют сложение намного быстрее, но используют дополнительные логические элементы.

Сумматор играет важную роль не только при сложении чисел, но при выполнении других арифметических действий. Фактически он является *основой арифметического устройства* современного компьютера.

Вопросы и задания

1. Что такое триггер? Объясните его принцип действия.
2. Почему для RS-триггера комбинация входов $S = 1$ и $R = 1$ запрещена?
3. Чем отличается одноразрядный сумматор от полусумматора?
4. Как можно построить сумматор с помощью двух полусумматоров?
5. Постройте логические выражения для выходов сумматора и нарисуйте соответствующие им схемы.
6. Объясните, как работает многоразрядный сумматор.
7. Что такое перенос? Как он используется в многоразрядном сумматоре?

Подготовьте сообщение

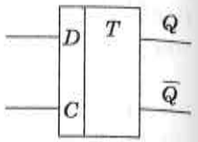
а) «Обозначения логических элементов в России и за рубежом»
б) «Типы триггеров»
в) «Что такое регистр?»
г) «Что такое мультиплексор?»
д) «Что такое демультиплексор?»
е) «Шифратор и дешифратор»

Задачи

1. Используя логические элементы, постройте схемы, соответствующие логическим выражениям:
   а) $X_1 = \overline{A} \cdot C + B \cdot \overline{C}$;
   б) $X_2 = A \cdot B + \overline{B} \cdot \overline{C}$;
   в) $X_3 = \overline{A} \cdot \overline{B} + B \cdot \overline{C}$.
2. Соревнования по поднятию тяжестей судит бригада из трёх человек, один из них старший. Лампочка «Вес взят» должна зажигаться, если проголосовали, по крайней мере, два судьи, причём один из них — старший. Предложите логическую схему, которая решала бы эту задачу.
3. В двухэтажном коттедже есть два выключателя, которые управляют освещением лестницы, один из них — на первом этаже, а второй — на втором. Каждый выключатель имеет два состояния, при нажатии на кнопку состояние изменяется. В исходный момент оба выключателя выключены. Когда человек заходит в неосвещенное здание, он нажимает кнопку выключателя на первом этаже, при этом должна загореться лампочка, освещающая всю лестницу. Поднявшись на второй этаж, он нажимает на кнопку второго выключателя, и лампочка должна погаснуть. Когда следом идет другой человек, он действует так же (хотя оба выключателя находятся в другом положении). Предложите логическую схему, которая решала бы эту задачу.
4. В самолёте есть три бака с горючим. Бортовой компьютер получает сигналы от датчиков уровня в каждом баке: если горючего в баке достаточно, то сигнал равен 0, если горючее кончилось — 1. Когда горючее заканчивается, по крайней мере, в двух баках, должна загореться лампочка «Тревога». Предложите логическую схему, которая решала бы эту задачу.
5. В парламенте некоторой страны выбирают спикера из трёх кандидатов. Каждый парламентарий должен нажать одну и только одну из трёх кнопок. Если он проголосовал правильно (нажал ровно одну кнопку), на пульте должна загореться зелёная лампочка. Предложите логическую схему, которая решала бы эту задачу.

6. Постройте RS-триггер на элементах «И–НЕ» и составьте его таблицу истинности.

*7. Постройте таблицу истинности и логическую схему D-триггера, который запоминает сигнал, поступающий на вход D (англ. *data* — данные), при подаче логической единицы на вход C (англ. *clock* — синхронизация). Для этого можно, например, немного изменить входную часть RS-триггера.



§ 25 Логические задачи

Метод рассуждений

Задача 1. Среди трёх приятелей (их зовут Сеня, Вася и Миша) один всегда говорит правду, второй говорит правду через раз, а третий всё время обманывает. Как-то раз они впервые прогуляли урок информатики. Директор школы вызвал их в свой кабинет для разговора. Сеня сказал: «Я всегда прогуливаю информатику. Не верьте тому, что скажет Вася». Вася сказал: «Я раньше не прогуливал этот предмет». Миша сказал: «Всё, что говорит Сеня, — правда». Директору стало всё понятно. Кто из них правдив, кто лгун, а кто говорит правду через раз?

Для решения используем **метод рассуждений**. Во-первых, есть точная информация, которая не подвергается сомнению: все трое прогуляли урок информатики в первый раз.

Запишем высказывания мальчиков:

Сеня:	1. Я всегда прогуливаю информатику.
	2. Вася сейчас скажет неправду.
Вася:	1. Я раньше не прогуливал информатику.
Миша:	1. Сеня говорит правду.

Известно, что один из них говорит правду всегда, второй через раз, а третий всё время лжёт. Отметим, что если у нас есть только одно высказывание «полулжеца», оно может быть как истинным, так и ложным.

Сопоставив первое высказывание Сени и высказывание Васи с точной информацией, сразу определяем, что тут Сеня соврал, а Вася сказал правду. Это значит, что второе высказывание Сени тоже неверно, поэтому Сеня всегда лжёт.

Тогда один из оставшихся, Вася или Миша, правдив, а второй говорит правду через раз. Мишино высказывание неверно, поскольку мы уже определили, что Сеня всегда лжёт; это значит, что Миша не всегда говорит правду, он — «полулжец». Тогда получается, что Вася правдив.

Табличный метод

Задача 2. Перед началом турнира по шахматам болельщики высказали следующие предположения по поводу результатов:

А) Максим победит, Борис будет вторым;
Б) Борис займёт третье место, а Коля — первое;
В) Максим будет последним, а Дима — первым.

Когда соревнования закончились, оказалось, что каждый из болельщиков был прав только в одном из своих прогнозов. Как распределились призовые места, если каждое место занял ровно один участник?

Запишем высказывания трёх болельщиков в форме таблицы (заголовок строки обозначает место в турнирной таблице):

	А	Б	В
I	Максим?	Коля?	Дима?
II	Борис?		
III		Борис?	
IV			Максим?

Начнём «раскручивать» эту таблицу с той строчки, где больше всего информации, в данном случае — с первой. Предположим, что Максим действительно занял первое место, как и сказал болельщик «А». В этом случае «В» ошибся, поставив на первое место Диму. Тогда получается, что второй прогноз болельщика «В» верен, и Максим — последний.

Так как мы предполагали, что Максим занял первое место, получается противоречие. Следовательно, первый прогноз «А» не

сбылся. Но тогда должен быть верен его второй прогноз, и Борис занял второе место. При этом он не мог занять ещё и третье место, поэтому первый прогноз болельщика «Б» неверный, а верен его второй прогноз: Коля — первый.

В этом случае Дима не может быть первым, поэтому верен первый прогноз «В»: Максим — последний. Диме осталось единственное свободное третье место.

	А	Б	В
I		Коля	
II	Борис		
III			
IV			Максим

В результате места распределились так: I — Коля, II — Борис, III — Дима и IV — Максим.

Задача 3. На одной улице стоят в ряд четыре дома, в каждом из них живёт по одному человеку. Их зовут Василий, Семён, Геннадий и Иван. Известно, что все они имеют разные профессии: скрипач, столяр, охотник и врач. Известно, что:

(1) Столяр живёт правее охотника.
(2) Врач живёт левее охотника.
(3) Скрипач живёт с краю.
(4) Скрипач живёт рядом с врачом.
(5) Семён не скрипач и не живёт рядом со скрипачом.
(6) Иван живёт рядом с охотником.
(7) Василий живёт правее врача.
(8) Василий живёт через дом от Ивана.

Определите, кто где живёт.

Из условий (1) и (2) следует, что охотник живет не с краю, потому что справа от него живет столяр, а слева — врач. Скрипач по условию (3) живет с краю, он может жить как слева, так и справа от остальных:

скрипач?	врач	охотник	столяр	скрипач?

Согласно условию (4), скрипач живет рядом с врачом, поэтому он занимает крайний дом слева:

1	2	3	4
скрипач	врач	охотник	столяр

Профессии жильцов определили, остаётся разобраться с именами. Из условия (5) «Семён не скрипач и не живет рядом со скрипачом» следует, что Семён — охотник или столяр:

1	2	3	4
скрипач	врач	охотник	столяр
		Семён?	Семён?

Из условия (6) «Иван живет рядом с охотником» следует, что он — врач или столяр:

1	2	3	4
скрипач	врач	охотник	столяр
		Семён?	Семён?
	Иван?		Иван?

Из условия (7) «Василий живет правее врача» определяем, что Василий — охотник или столяр:

1	2	3	4
скрипач	врач	охотник	столяр
		Семён?	Семён?
	Иван?		Иван?
		Василий?	Василий?

Согласно условию (8), «Василий живет через дом от Ивана», поэтому Иван — врач, а Василий — столяр:

1	2	3	4
скрипач	врач	охотник	столяр
	Иван	Семён?	Василий

Тогда сразу получается, что Семён — охотник, а Геннадий должен занять оставшееся свободное место, он — скрипач:

1	2	3	4
скрипач	врач	охотник	столяр
Геннадий	Иван	Семён	Василий

Задача 4. Шесть приятелей, Саша, Петя, Витя, Дима, Миша и Кирилл, встретившись через 10 лет после окончания школы, выяснили, что двое из них живут в Москве, двое — в Санкт-Петербурге, а двое — в Перми. Известно, что:

(1) Витя ездит в гости к родственникам в Москву и Санкт-Петербург.
(2) Петя старше Саши.
(3) Дима и Миша летом были в Перми в командировке.
(4) Кирилл и Саша закончили университет в Санкт-Петербурге и уехали в другие города.
(5) Самый молодой из них живет в Москве.
(6) Кирилл редко приезжает в Москву.
(7) Витя и Дима часто бывают в Санкт-Петербурге по работе.

Определите, кто где живёт.

Составим таблицу, где каждая строка соответствует городу, а столбец — человеку:

	Саша	Петя	Витя	Дима	Миша	Кирилл
Москва						
Санкт-Петербург						
Пермь						

Единица в таблице будет обозначать, что человек живёт в данном городе, а ноль — что не живёт. По условию в каждом городе живут ровно 2 человека, каждый живет только в одном городе. Поэтому в каждой строке должно быть две единицы, а в каждом столбце — одна.

Из условия (1) следует, что Витя живёт в Перми:

	Саша	Петя	Витя	Дима	Миша	Кирилл
Москва			0			
Санкт-Петербург			0			
Пермь			1			

Из (2) и (5) находим, что Петя живёт не в Москве. Кроме того, как следует из (6), Кирилл — тоже не москвич.

	Саша	Петя	Витя	Дима	Миша	Кирилл
Москва		0	0			0
Санкт-Петербург			0			
Пермь			1			

Согласно (3), Дима и Миша живут не в Перми:

	Саша	Петя	Витя	Дима	Миша	Кирилл
Москва		0	0			0
Санкт-Петербург			0			
Пермь			1	0	0	

Из условия (4) делаем вывод, что Кирилл и Саша живут не в Санкт-Петербурге, отсюда сразу следует, что Кирилл живёт в Перми. Двух пермяков мы уже определили, поэтому Саша и Петя живут не в Перми:

	Саша	Петя	Витя	Дима	Миша	Кирилл
Москва		0	0			0
Санкт-Петербург	0		0			0
Пермь	0	0	1	0	0	1

Далее определяем из таблицы, что Саша — москвич, а Петя живёт в Санкт-Петербурге:

	Саша	Петя	Витя	Дима	Миша	Кирилл
Москва	1	0	0			0
Санкт-Петербург	0	1	0			0
Пермь	0	0	1	0	0	1

По условию (7), Витя и Дима — не петербуржцы, поэтому в Петербурге живёт Миша. Тогда Дима живёт в Москве:

	Саша	Петя	Витя	Дима	Миша	Кирилл
Москва	1	0	0	1	0	0
Санкт-Петербург	0	1	0	0	1	0
Пермь	0	0	1	0	0	1

Таким образом, Саша и Дима живут в Москве, Петя и Миша — в Санкт-Петербурге, а Витя и Кирилл — в Перми.

Использование алгебры логики

Когда в условии задачи встречаются сложные логические высказывания, удобно использовать методы алгебры логики. Покажем этот подход на примерах.

Задача 5. Следующие два высказывания истинны:

1. Неверно, что если корабль А вышел в море, то корабль С — нет.
2. В море вышел корабль В или корабль С, но не оба вместе.

Определить, какие корабли вышли в море.

Введём три высказывания: A — «Корабль A вышел в море»; B — «Корабль В вышел в море»; C — «Корабль С вышел в море». Вспомним, что связка «если..., то» в логических выражениях заменяется импликацией, поэтому фразу «Если корабль A вышел

в море, то корабль C — нет» можно записать как $A \to \overline{C} = 1$. Но в условии сказано, что это утверждение неверно, поэтому:

$$A \to \overline{C} = 0, \text{ или } \overline{A \to \overline{C}} = 1.$$

Второе условие — это операция «исключающее ИЛИ», т. е. $B \oplus C = 1$. Оба условия истинны одновременно, т. е. их логическое произведение (И) тоже истинно:

$$\overline{(A \to \overline{C})} \cdot (B \oplus C) = 1.$$

Нам нужно решить это уравнение и найти неизвестные A, B и C. Для этого выразим импликацию и операцию «исключающее ИЛИ» через базовый набор логических операций (НЕ, И, ИЛИ), а затем раскроем инверсию сложного выражения с помощью закона де Моргана:

$$\overline{(A \to \overline{C})} \cdot (B \oplus C) = \overline{(\overline{A} + \overline{C})} \cdot (B \cdot \overline{C} + \overline{B} \cdot C) = A \cdot C \cdot (B \cdot \overline{C} + \overline{B} \cdot C) = 1.$$

В последнем выражении раскроем скобки и учтём, что $C \cdot \overline{C} = 0$ и $C \cdot C = C$. Получим:

$$A \cdot \overline{B} \cdot C = 1.$$

Это уравнение имеет единственное решение: $A = 1$, $B = 0$ и $C = 1$. Это значит, что в море вышли корабли A и C.

Выше был показан общий подход к решению подобных задач, однако эту конкретную задачу можно решить намного проще. Как вы знаете, импликация $A \to B$ ложна только при $A = 1$ и $B = 0$. Поэтому из условия $A \to \overline{C} = 0$ сразу следует, что $A = C = 1$. Теперь остаётся только применить второе условие $B \oplus C = 1$, которое при $C = 1$ даёт $B = 0$, и получаем тот же ответ, что и раньше.

Задача 6. На вопрос «Кто из твоих учеников изучал логику?» учитель ответил: «Если логику изучал Андрей, то изучал и Борис. Однако неверно, что если изучал Семён, то изучал и Борис». Кто же изучал логику?

Обозначим переменными высказывания: A — «Логику изучал Андрей»; B — «логику изучал Борис» и C — «Логику изучал Семён». Оба высказывания учителя можно записать в виде импликаций:

«Если логику изучал Андрей, то изучал и Борис». $A \to B = 1$
«Неверно, что если изучал Семён, то изучал и Борис». $C \to B = 0$

Дальше есть два варианта решения. Во-первых, можно поступить так же, как и в предыдущей задаче: применить операцию

«НЕ» ко второму высказыванию и составить уравнение с помощью логического произведения:

$$(A \to B) \cdot \overline{(C \to B)} = 1.$$

Теперь представляем импликацию через базовые операции и применяем закон де Моргана

$$(\overline{A} + B) \cdot \overline{(\overline{C} + B)} = (\overline{A} + B) \cdot C \cdot \overline{B} = \overline{A} \cdot C \cdot \overline{B} = 1.$$

Это уравнение имеет единственное решение: $A = 0$, $B = 0$ и $C = 1$. Значит, логику изучал только Семён.

Можно поступить иначе, вспомнив, что импликация ложна только в том случае, когда первое высказывание истинно, а второе ложно. Поэтому из условия $C \to B = 0$ сразу следует, что $B = 0$ и $C = 1$. Тогда первое условие, $A \to B = A \to 0 = 1$ сразу даёт $A = 0$.

Подготовьте сообщение

а) «Задача Эйнштейна»
б) «Задачи о лжецах»
в) «Задачи о шляпах»
г) «Задачи о двух городах»

Задачи

1. Три школьника — Миша, Коля и Сергей — остававшиеся в классе на перемене, были вызваны к директору по поводу разбитого в это время окна в кабинете. На вопрос директора о том, кто это сделал, мальчики ответили следующее:

   Миша: «Я не разбивал окно, и Коля тоже.»
   Коля: «Миша не разбивал окно, это Сергей разбил футбольным мячом!»
   Сергей: «Я не делал этого, стекло разбил Миша».

   Выяснилось, что один из ребят сказал правду, второй в одной части заявления солгал, а другое его высказывание истинно, а третий оба раза солгал. Кто разбил стекло в классе?
2. В финал соревнований по настольному теннису вышли Наташа, Маша, Люда и Рита. Болельщики высказали свои предположения о распределении мест в дальнейших состязаниях. Один считает, что первой будет Наташа, а Маша — второй. Другой болельщик на второе место прочит Люду, а Рита, по его мнению, займёт четвёртое место. Третий считает, что Рита займёт третье место, а Наташа будет второй. Когда соревнования закончились, оказалось, что каждый из болельщиков был прав только в одном из своих прогнозов. Как распределились места?

3. На одной улице стоят в ряд четыре дома, в каждом из них живёт по одному человеку. Их зовут Алексей, Егор, Виктор и Михаил. Известно, что все они имеют разные профессии: рыбак, пчеловод, фермер и ветеринар. Известно, что:
   (1) Фермер живёт правее пчеловода.
   (2) Рыбак живёт правее фермера.
   (3) Ветеринар живёт рядом с рыбаком.
   (4) Рыбак живёт через дом от пчеловода.
   (5) Алексей живёт правее фермера.
   (6) Виктор не пчеловод.
   (7) Егор живёт рядом с рыбаком.
   (8) Виктор живёт правее Алексея.

   Определите, кто где живёт.
4. Дочерей Василия Лоханкина зовут Даша, Анфиса и Лариса. У них разные профессии, и они живут в разных городах: одна в Ростове, вторая — в Париже и третья — в Москве. Известно, что:
   (1) Даша живёт не в Париже, а Лариса — не в Ростове.
   (2) Парижанка не актриса.
   (3) В Ростове живёт певица.
   (4) Лариса не балерина.

   Определите, где живёт каждая из дочерей и чем занимается.
5. В состав экспедиции входят Михаил, Сергей и Виктор. На обсуждении распределения обязанностей с руководителем проекта были высказаны предположения, что командиром будет назначен Михаил, Сергей не будет механиком, а Виктор будет утвержден радистом, но командиром точно не будет. Позже выяснилось, что только одно из этих четырёх утверждений оказалось верным. Как распределились должности?
6. В ходе заседания суда выяснилось, что:
   (1) Если Аськин не виновен или Баськин виновен, то виновен Сенькин.
   (2) Если Аськин не виновен, то Сенькин не виновен.

   Что можно сказать о виновности Аськина, Баськина и Сенькина?
7. Аськин, Баськин и Васькин стали свидетелями ограбления банка. Во время расследования Аськин сказал, что взломщики приехали на синей «Тойоте». Баськин считает, что это был красный «BMW», а Васькин утверждает, что это был «Форд-Фокус», но не синий. Выяснилось, что каждый из них назвал неправильно либо марку, либо цвет машины. На каком автомобиле приехали преступники?

Практические работы к главе 3

Работа № 7 «Тренажёр "Логика"»
Работа № 8 «Исследование запросов для поисковых систем»

ЭОР к главе 3 на сайте ФЦИОР (http://fcior.edu.ru)

- Высказывание. Простые и сложные высказывания
- Основные логические операции
- Теория множеств
- Логические законы и правила преобразования логических выражений
- Построение отрицания к простым высказываниям, записанным на русском языке
- Построение отрицания к сложным высказываниям, записанным на русском языке
- Решение логических задач
- Сумматор двоичных чисел

Самое важное в главе 3

- Логическое выражение может принимать два значения: «истина» и «ложь». Если обозначить эти значения символами 0 и 1, то получится что с помощью логических операций можно описать правила обработки двоичных кодов.
- Все вычисления в компьютерах выполняются с помощью логических операций с двоичными кодами данных.
- Для определения логической операции используют таблицы истинности. Таблица истинности однозначно определяет некоторую логическую функцию — правило преобразования входных данных в результат.
- Используя только логические операции «И», «ИЛИ» и «НЕ», можно записать любую логическую функцию. Также любая логическая функция может быть записана только с помощью операции «И-НЕ» или «ИЛИ-НЕ».
- Любой логической функции соответствует множество логических выражений. Используя законы алгебры логики, логические выражения можно преобразовывать и упрощать.
- Предикат — это утверждение, содержащее переменные. Если значения всех переменных заданы, предикат превращается в логическое высказывание.
- Триггер — это логическая схема, которая способна запоминать один бит данных.
- Сумматор — это логическая схема для сложения двоичных чисел.

Глава 4
Компьютерная арифметика

§ 26
Особенности представления чисел в компьютере

На уроках математики вы никогда не обсуждали, как хранятся числа. Математика — это теоретическая наука, для которой совершенно неважно, записаны они на маленьком или большом листе бумаги, зафиксированы с помощью счётных палочек, счётов, или внутри полупроводниковой схемы. Поэтому число в математике может состоять из любого количества цифр, которое требуется в решаемой задаче.

В то же время инженеры, разрабатывающие компьютер, должны спроектировать реальное устройство из вполне определённого количества деталей. Поэтому число разрядов, отведённых для хранения каждого числа, ограничено, и точность вычислений тоже ограничена. Из-за этого при компьютерных расчётах могут возникать достаточно серьёзные проблемы. Например, сумма двух положительных чисел может получиться отрицательной, а выражение $A + B$ может совпадать с A при ненулевом B. В этой главе мы рассмотрим важные особенности компьютерной арифметики, которые нужно учитывать при обработке данных. В первую очередь, они связаны с тем, как размещаются целые и вещественные числа в памяти компьютера.

Предельные значения чисел

Как вы уже поняли, числа, хранящиеся в компьютере, не могут быть сколь угодно большими и имеют некоторые предельные значения. Представим себе некоторое вычислительное устройство, которое работает с четырехразрядными неотрицательными целыми десятичными числами (рис. 4.1). Для вывода чисел используется четырёхразрядный индикатор, на котором можно отобразить числа от 0 (все разряды числа минимальны) до 9999 (все разряды максимальны) — рис. 4.2.

Рис. 4.1

Диапазон от  до 

Рис. 4.2

Вывести на такой индикатор число 10000 невозможно: не хватает технического устройства для пятого разряда. Такая «аварийная» ситуация называется переполнением разрядной сетки или просто переполнением (англ. *overflow* — переполнение «сверху»).

Переполнение разрядной сетки — это ситуация, когда число, которое требуется сохранить, не умещается в имеющемся количестве разрядов вычислительного устройства.

В нашем примере переполнение возникает при значениях, больших $9999 = 10^4 - 1$, где 4 — это количество разрядов. В общем случае, если в системе счисления с основанием B для записи числа используется K разрядов, максимальное допустимое число C_{max} вычисляется по аналогичной формуле[1]

$$C_{max} = B^K - 1.$$

Именно эта формула для $B = 2$ неоднократно применялась в главе 2.

Подчеркнём, что переполнение никак не связано с системой счисления: оно вызвано *ограниченным количеством разрядов* устройства и не зависит от количества возможных значений в каждом из этих разрядов.

Рассмотрим теперь, что получится, если наше устройство будет работать не только с целыми, но и с дробными числами. Пусть, например, один из четырёх разрядов относится к целой части числа, а остальные три — к дробной (рис. 4.3). Конечно, эффект переполнения сохранится и здесь: максимально допустимое число равно 9,999. Кроме того, дробная

Рис. 4.3

[1] Докажите эту формулу самостоятельно, например, подсчитав количество всех возможных комбинаций значений цифр в K разрядах.

часть числа тоже ограничена, поэтому любое число, имеющее более трёх цифр после запятой, не может быть представлено точно: младшие цифры придется отбрасывать (или округлять).

Не все вещественные числа могут быть представлены в компьютере точно.

При ограниченном числе разрядов дробной части существует некоторое минимальное ненулевое значение C_{min}, которое можно записать на данном индикаторе (в нашем примере это 0,001, рис. 4.4). В общем случае, если число записано в системе счисления с основанием B и для хранения дробной части числа используется F разрядов, имеем

Рис. 4.4

$$C_{min} = B^{-F}.$$

Любое значение, меньшее чем C_{min}, неотличимо от нуля. Такой эффект принято называть *антипереполнением* (англ. *underflow* — переполнение «снизу»).

Кроме того, два дробных числа, отличающиеся менее чем на C_{min}, для компьютера неразличимы. Например, 1,3212 и 1,3214 на нашем индикаторе выглядят совершенно одинаково (рис. 4.5).

Рис. 4.5

Дополнительная погрешность появляется при переводе дробных чисел из десятичной системы счисления в двоичную. При этом даже некоторые «круглые» числа (например, 0,2) в памяти компьютера представлены неточно, потому что в двоичной системе они записываются как бесконечные дроби и их приходится округлять до заданного числа разрядов.

Так как вещественные числа хранятся в памяти приближённо, сравнивать их (особенно если они являются результатами сложных расчётов) необходимо с большой осторожностью. Пусть при вычислениях на компьютере получили $X = 10^{-6}$ и $Y = 10^6$. Дробное значение X будет неточным, и произведение $X \cdot Y$ может незначительно отличаться от 1. Поэтому при сравнении вещественных чисел в компьютере условие «равно» использовать не рекомендуется. В таких случаях числа считаются равными, если

их разность достаточно мала по модулю. В данном примере нужно проверять условие $|1 - X \cdot Y| < \varepsilon$, где ε — малая величина, которая задает нужную точность вычислений. К счастью, для большинства практических задач достаточно взять ε порядка $10^{-2} \ldots 10^{-4}$, а ошибка компьютерных расчётов обычно значительно меньше[1] (не более 10^{-7}).

Введение разряда для знака числа не меняет сделанных выше выводов, только вместо нулевого минимального значения появляется отрицательное, которое зависит от разрядности (оно равно −9999 в первом из обсуждаемых примеров).

Различие между вещественными и целыми числами

Существуют величины, которые по своей природе могут принимать только целые значения, например, счётчики повторений каких-то действий, количество людей или предметов, координаты пикселей на экране и т. п. Кроме того, как показано в главе 2, кодирование нечисловых видов данных (текста, изображений, звука) сводится именно к целым числам.

Чтобы сразу исключить все возможные проблемы, связанные с неточностью представления в памяти вещественных чисел, целочисленные данные кодируются в компьютерах особым образом.

Целые и вещественные числа в компьютере хранятся и обрабатываются по-разному.

Операции с целыми числами, как правило, выполняются значительно быстрее, чем с вещественными. Не случайно в ядре современных процессоров реализованы только целочисленные арифметические действия, а для вещественной арифметики используется специализированный встроенный блок — *математический сопроцессор*.

Кроме того, использование целых типов данных позволяет экономить компьютерную память. Например, целые числа в диапазоне от 0 до 255 в языке Паскаль можно хранить в переменных типа *byte*, которые занимают всего один байт в памяти. В то же

[1] Тем не менее встречаются ситуации, когда вычислительные трудности все же возникают: классический пример — разность близких по значению десятичных дробей, отличающихся в последних значащих цифрах.

время самое «короткое» вещественное число (типа *single*) требует четырёх байтов памяти.

Наконец, только для целых чисел определены операции деления нацело и нахождения остатка[1]. В некоторых задачах они удобнее, чем простое деление с получением дробного (к тому же не совсем точного) результата: например, без них не обойтись при вычислении суммы цифр какого-то числа.

Таким образом, для всех величин, которые не могут иметь дробных значений, нужно использовать целочисленные типы данных.

Дискретность представления чисел

Из § 7 вы знаете, что существует непрерывное и дискретное представление информации. Их принципиальное различие состоит в том, что дискретная величина может принимать конечное количество различных значений в заданном диапазоне, а непрерывная имеет бесконечно много возможных значений. Для нашего обсуждения важно, что:

- целые числа дискретны;
- вещественные (действительные, дробные) числа непрерывны;
- современный компьютер работает только с дискретными данными.

Таким образом, для хранения вещественных чисел в памяти компьютера нужно выполнить *дискретизацию* — записать непрерывную величину в дискретной форме. При этом может происходить искажение данных, поэтому большинство трудностей в компьютерной арифметике (антипереполнение, приближённость представления дробной части и др.) связано именно с кодированием дробных чисел.

Программное повышение точности вычислений

Современные модели процессоров Intel «умеют» обрабатывать 8-, 16-, 32- и 64-разрядные двоичные целые числа, а также (в математическом сопроцессоре) 32-, 64- и 80-разрядные вещественные числа. Для большинства практических задач такой разрядности вполне достаточно. Если для каких-либо особо точных расчётов требуется повысить разрядность вычислений, это можно сделать программно. Например, можно считать, что четыре

[1] В языке Python они определены и для вещественных чисел.

последовательно хранящихся целых 64-разрядных числа — это единое «длинное» число, и написать программу обработки таких «удлинённых» чисел. Очень удобно хранить числа в виде последовательности десятичных цифр[1], правда, программы, выполняющие обработку таких чисел, получаются сложными и медленными.

Использование этих и других программных методов позволяет увеличить разрядность обрабатываемых чисел по сравнению с аппаратной разрядностью компьютера. Однако ограничение разрядности (и связанный с ним эффект переполнения) все равно остаётся: в программу заложено конкретное число разрядов, да и объём памяти компьютера конечен.

Вопросы и задания

1. Чем отличается компьютерная арифметика от «обычной»? Почему?
2. Почему диапазон чисел в компьютере ограничен? Связано ли это с двоичностью компьютерной арифметики?
3. Что такое переполнение разрядной сетки?
4. Какие проблемы появляются при ограниченном числе разрядов в дробной части?
5. Что называется антипереполнением? Что, по-вашему, опаснее для вычислений — переполнение или антипереполнение?

*6. Может ли антипереполнение сделать невозможными дальнейшие вычисления?

7. Сколько битов информации несёт знаковый разряд?
8. Приведите примеры величин, которые по своему смыслу могут иметь только целые значения.
9. Какая математическая операция между двумя целыми числами может дать в результате нецелое число?
10. Чем различается деление для целых и вещественных чисел?
11. Какие преимущества даёт разделение в компьютере целых и вещественных (дробных) чисел?
12. Вспомните определение дискретных и непрерывных величин. Какие множества чисел в математике дискретны, а какие — нет? Ответ обоснуйте.
13. Объясните, почему ограниченность разрядов дробной части приводит к нарушению свойства непрерывности.

[1] Такие задачи часто даются на школьных олимпиадах по информатике; для них даже придумано специальное название: «длинная» арифметика.

14. Можно ли организовать вычисления с разрядностью, превышающей аппаратную разрядность компьютера? Попробуйте предложить способы решения этой задачи.

Подготовьте сообщение

а) «Проблемы вычислений с вещественными данными»
б) «Длинная арифметика»

Задачи

1. Вычислите максимальное целое значение для 8-разрядного двоичного числа и 3-разрядного десятичного (считать, что значения не бывают отрицательными). Какое из них оказалось больше?
2. Вычислите максимальное целое положительное значение для 16- и 32-разрядных двоичных чисел.
3. Пользуясь калькулятором, вычислите границу антипереполнения для чисел с 16 двоичными разрядами в дробной части. Напишите два близких дробных числа, которые для полученного значения окажутся неразличимыми.

*4. Вычислите минимально возможное отрицательное значение для 16-разрядных двоичных чисел (учесть, что один из двоичных разрядов является знаковым).

*5. Придумайте простую вычислительную задачу, в которой для хранения результата не хватает 16 двоичных разрядов.

§ 27 Хранение в памяти целых чисел

Целые числа без знака

Беззнаковые (англ. *unsigned*) типы данных, т. е. величины, не имеющие отрицательных значений, широко используются в вычислительной технике. Дело в том, что в задачах, решаемых на компьютерах, есть много таких значений: всевозможные счётчики (количество повторений циклов, число параметров в списке или символов в тексте), количество людей или предметов и др.

Чтобы закодировать целое число без знака, достаточно перевести его в двоичную систему счисления (см. § 11) и дополнить

слева нулями до нужной разрядности. Например, число 28, записывается в 8-разрядную ячейку памяти так:

0001 1100

Это же число в 16-разрядном представлении будет иметь слева ещё 8 нулей. Восьмиразрядные коды некоторых характерных чисел приведены в табл. 4.1.

Таблица 4.1

X_{10}	0	1	...	127	128	129	...	255
X_{16}	00	01	...	7F	80	81	...	FF
X_2	0000 0000	0000 0001	...	0111 1111	1000 0000	1000 0001	...	1111 1111

Минимальное значение для беззнаковых целых чисел всегда равно 0 (все разряды нулевые), а максимальное число $X_{max} = 2^K - 1$ состоит из всех единиц и определяется разрядностью (количеством битов) K (табл. 4.2).

Таблица 4.2

K, битов	8	16	32	64
$X_{max} = 2^K - 1$	255	65 535	4 294 967 295	18 446 744 073 709 551 615

Возникает вопрос: что будет, если увеличить максимальное число в K-битной ячейке на единицу? Рассмотрим случай $K = 8$ и попытаемся прибавить единицу к числу $255_{10} = 1111\ 1111_2$. Добавляя дополнительный бит слева, получим:

$$\begin{array}{rc|l} & 0 & 1111\ 1111 \\ + & 0 & 0000\ 0001 \\ \hline & 1 & 0000\ 0000 \end{array}$$

Отбросив несуществующий дополнительный разряд[1], получаем 255 + 1 = 0. Как ни странно, именно это произойдёт в реальном компьютере. Говорят, что при K разрядах арифметика выполняется **по модулю 2^K**, т. е. при $K = 8$ имеем[2]:

$$(255 + 1) \bmod 256 = 256 \bmod 256 = 0.$$

[1] На самом деле, для того, чтобы обнаружить факт переполнения, этот разряд сохраняется в специальном управляющем бите процессора, который называется битом переноса.

[2] Здесь запись $a \bmod b$ обозначает остаток от деления a на b.

Вместе с тем, вычитая единицу из минимального значения 0, к которому добавлен старший единичный разряд за пределами 8-битной ячейки, получим $1111\ 1111_2 = 255_{10}$ (проверьте это самостоятельно).

Можно заметить, что при многократном увеличении числа на единицу мы доходим до максимального значения и скачком возвращаемся к минимальному. При вычитании единицы получается обратная картина — дойдя до минимума (нуля), мы сразу перескакиваем на максимум (255). Поэтому для изображения допустимого диапазона чисел лучше подходит не отрезок числовой оси (как в математике), а окружность (рис. 4.6).

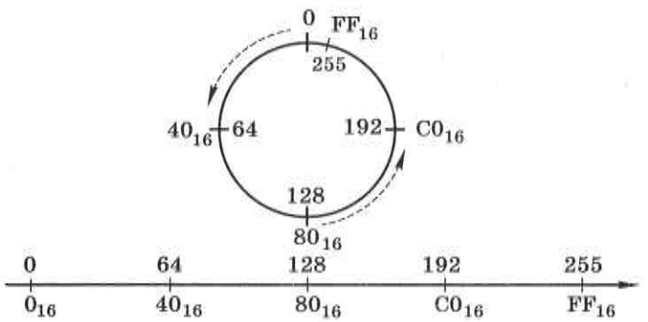


Рис. 4.6

Факт переполнения всегда фиксируется процессором, но выполнение программы не прерывается. Программе (точнее, программисту) предоставляется возможность как-то реагировать на переполнение или «не заметить» его.

Целые числа со знаком

Теперь рассмотрим числа со знаком (англ. *signed*). Для того, чтобы различать положительные и отрицательные числа, в двоичном коде выделяется один бит для хранения знака числа — **знаковый разряд**. По традиции для этого используют самый старший бит, причём нулевое значение в нём соответствует знаку «плюс», а единичное — знаку «минус». Ноль формально является положительным числом, так как все его разряды, включая знаковый, нулевые.

Поскольку один бит выделяется для хранения информации о знаке, ровно половина из всех 2^K чисел будут отрицательными. Учитывая, что одно значение — нулевое, положительных чисел будет на единицу меньше, т. е. допустимый диапазон значений оказывается несимметричным.

Положительные числа записываются в знаковой форме так же, как и в беззнаковой, но для значения остаётся на один разряд меньше. А как поступить с отрицательными числами? Первое, что приходит в голову, это кодировать отрицательные значения точно так же, как и положительные, только записывать в старший бит единицу. Такой способ кодирования называется **прямым кодом**. Несмотря на свою простоту и наглядность, он не применяется в компьютерах для представления целых чисел[1]. Это неудобно, потому что действия над числами, записанными в прямом коде, выполняются по-разному для разных сочетаний знаков чисел. Поэтому в современных компьютерах отрицательные числа кодируются с помощью другого метода, который менее нагляден, но позволяет выполнять арифметические действия с положительными и отрицательными числами по одному и тому же алгоритму.

Как же представить целые числа, чтобы арифметика выглядела максимально просто? Попробуем, например, вычислить код, соответствующий числу –1. Для этого просто вычтем из нуля единицу:

$$\begin{array}{rcc} & 1 & 0000\ 0000 \\ - & 0 & 0000\ 0001 \\ \hline & 0 & 1111\ 1111 \end{array}$$

Чтобы вычитание «состоялось», придется занять из несуществующего старшего бита единицу, что не очень естественно, но зато быстро приводит к правильному результату[2]. Заметим, что фактически мы вычитали не из 0, а из 256. В общем случае вычисление происходит по формуле $2^K - X$, где для данного примера $K = 8$, а $X = 1$.

[1] Тем не менее прямой код используется в представлении вещественных чисел.

[2] Для проверки можно прибавить к полученному коду единицу, в результате должен получиться ноль.

Однако предложенный способ перевода не слишком хорош, поскольку мы использовали дополнительный «несуществующий» разряд. Вместо этого можно использовать равносильный алгоритм:

$$256 - X = (255 - X) + 1 = \text{not } X + 1.$$

Здесь «not» обозначает логическую операцию «НЕ» (инверсию), применяемую к каждому биту числа отдельно (все нули заменяются на единицы и наоборот).

Итак, для получения дополнительного кода целого числа ($-X$) нужно:

Алгоритм А1

1) Выполнить инверсию каждого разряда двоичного представления числа X (такой код называется обратным).
2) К полученному результату прибавить единицу.

В результате получается **дополнительный код** — он *дополняет* число X до 2^K (если сложить его с числом X, мы получим 2^K).

Алгоритм А1 приводится в большинстве учебников, но его можно немного изменить так, чтобы облегчить человеку «ручные» вычисления:

Алгоритм А2

1) Вычислить число $X - 1$ и перевести его в двоичную систему.
2) Выполнить инверсию каждого разряда результата.

Оба алгоритма дают одинаковые результаты, но алгоритм А2 для человека существенно проще, потому что ему легче вычесть единицу в «родной» десятичной системе, чем прибавлять её в двоичной (при использовании алгоритма А1).

Наконец, оба пункта алгоритма А1 можно объединить, получив еще один вариант:

Алгоритм А3

Выполнить инверсию всех старших битов числа, кроме последней (младшей) единицы и тех нулей, которые стоят после неё.

Например, определим дополнительный код числа «–16», которое хранится в 8-разрядной ячейке. Здесь $X = 16$. Используя алгоритмы А1 и А2, получаем:

	A1
X	$0001\ 0000_2$
not	$1110\ 1111_2$
+1	$1111\ 0000_2$

	A2
$X-1$	15
$(X-1)_2$	$0000\ 1111_2$
not	$1111\ 0000_2$

Применение алгоритма А3 к числу $16 = 00010000_2$ сводится к замене первых трёх нулей единицами: $1111\ 0000_2$.

Для проверки можно сложить полученный результат с исходным числом и убедиться, что сумма обратится в ноль (перенос из старшего разряда не учитываем).

Повторное применение любого из алгоритмов А1–А3 всегда приводит к восстановлению первоначального числа (убедитесь в этом самостоятельно). Это свойство также удобно использовать для проверки.

В таблице 4.3 показаны шестнадцатеричные и двоичные коды некоторых характерных 8-разрядных чисел.

Таблица 4.3

X_{10}	–128	–127	...	–1	0	1	...	127
X_{16}	80	81	...	FF	00	01	...	7F
X_2	1000 0000	1000 0001	...	1111 1111	0000 0000	0000 0001	...	0111 1111

Обратите внимание на скачок при переходе от –1 к 0 и на два граничных значения: 127 и «–128». «Кольцо» для чисел со знаком выглядит так, как показано на рис. 4.7.

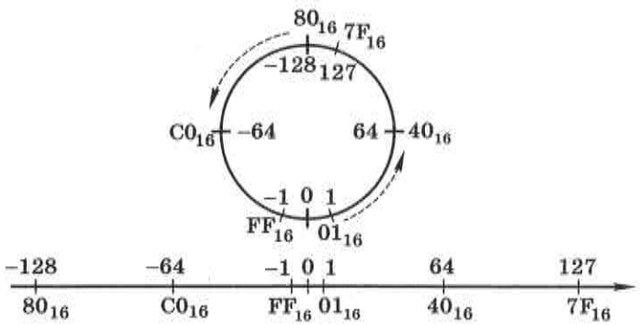


Рис. 4.7

Чтобы сравнить коды целых чисел без знака и со знаком, объединим обе таблицы (4.1 и 4.3) — получим табл. 4.4.

Таблица 4.4

Код	0	1	2	...	7F	80	81	...	FE	FF
Без знака	0	1	2	...	127	128	129	...	254	255
Со знаком	0	1	2	...	127	–128	–127	...	–2	–1

Общее количество значений со знаком и без знака одинаково, но их диапазоны сдвинуты друг относительно друга на числовой оси (рис. 4.8).

Рис. 4.8

В наших рассуждениях использовались 8-разрядные числа, но все выводы справедливы для чисел любой разрядности. От числа разрядов K зависят только граничные значения X_{max} и X_{min}, приведённые в табл. 4.5.

Таблица 4.5

K	8	16	32	64
X_{max}	127	32 767	2 147 483 647	9 223 372 036 854 775 807
X_{min}	–128	–32 768	–2 147 483 648	–9 223 372 036 854 775 808

Хотя дополнительный код гораздо менее нагляден, чем прямой, он значительно упрощает выполнение арифметических операций в компьютере. Например, вместо вычитания используется сложение с дополнительным кодом вычитаемого, поэтому не нужно проектировать специальное устройство для вычитания чисел.

Вопросы и задания

1. Чем отличается представление в компьютере целых чисел со знаком и без знака?
2. Приведите примеры величин, которые всегда имеют целые неотрицательные значения.
3. Как представлены в компьютере целые числа без знака?
4. Как изменится диапазон представления чисел, если увеличить количество разрядов на 1? На 2? На n?
5. Какое максимальное целое беззнаковое число можно записать с помощью K двоичных разрядов? Что произойдёт, если прибавить единицу к этому максимальному значению?
6. Как действует процессор при переполнении?
7. Почему максимальное положительное и минимальное отрицательное значения у целых двоичных чисел со знаком имеют разные абсолютные значения?
8. Верно ли, что положительные числа кодируются одинаково в знаковом и беззнаковом форматах?
9. Сформулируйте различные алгоритмы получения дополнительного кода для отрицательного числа.
10. Докажите, что алгоритмы А1, А2 и А3 всегда дают один и тот же результат.
11. Какое минимальное отрицательное значение можно записать с помощью K двоичных разрядов?
12. Может ли быть переполнение при сложении двух отрицательных чисел? Какой знак будет у результата?
13. Что получится, если правила перевода в дополнительный код применить к отрицательному числу?
14. Как можно проверить правильность перевода в дополнительный код?
15. В чём главное преимущество дополнительного кода при кодировании отрицательных чисел?
16. Почему компьютер может обойтись без вычитания?

Подготовьте сообщение

а) «Способы кодирования отрицательных целых чисел»

б) «Целочисленные типы данных в языках программирования»

Задачи

1. Цвет пикселя изображения кодируется как целое беззнаковое число. Найдите максимальное количество цветов при двух- и трёхбайтовом кодировании.
2. Используя арифметику 8-разрядных чисел без знака, выполните действия: 250 + 10 и 8 – 10. Объясните полученные результаты.
3. Выполните сложение десятичных чисел 65530 + 9 в 16-битной арифметике без знака.
4. Выполните сложение десятичных чисел 32760 + 9 в 16-битной арифметике со знаком.
5. Переведите в дополнительный код отрицательные числа –1, –10, –100 и запишите их с помощью 8 двоичных разрядов.
6. Постройте прямой код для отрицательных чисел –1, –10, –100, записанных с помощью 8 двоичных разрядов.
7. Рассматриваются 8-разрядные числа со знаком. Какие из приведённых чисел, записанных в шестнадцатеричной системе счисления, отрицательные: 1, 8, F, 10, 18, 20, 30, 3F, 40, 70, 7F, 80, 90, A1, CC, F0, FF? Как это можно быстро определять?
8. Отвечая на вопрос учителя о том, как вычислить максимальное положительное и минимальное отрицательное значения у целых K-разрядных двоичных чисел со знаком, ученик ответил кратко: 2^{K-1}. В чём он ошибся, а в чём нет? Вычислите правильные значения для $K = 12$.
9. Каков будет результат операции 127 + 3 в 8-разрядной арифметике со знаком? Объясните полученный результат.
10. Факториалом называется произведение последовательных целых чисел, например 3! (читается «3 факториал») $= 1 \cdot 2 \cdot 3 = 6$. Вычисления выполняются в 16-разрядной целочисленной арифметике со знаком. Для какого максимального значения n удастся вычислить $n!$ и что получится при вычислении $(n+1)!$?

§ 28 Операции с целыми числами

Сложение и вычитание

Сложение и вычитание требуются не только для расчётов по формулам, но и для организации вычислений. Например, для того, чтобы повторить какое-то действие R раз, используют переменную-счётчик, к которой после каждого выполнения этого действия прибавляют единицу, а затем результат сравнивают с R.

Вместо этого можно сразу записать в счётчик значение R и после каждого повторения вычитать из него единицу, пока не получится ноль[1].

Благодаря тому, что отрицательные числа кодируются в дополнительном коде, при сложении можно не обращать внимания на знаки слагаемых, т. е. *со знаковым разрядом обращаются точно так же, как и со всеми остальными.*

Например, сложим числа 5_{10} ($0000\ 0101_2$) и -9_{10} ($1111\ 0111_2$), используя 8-разрядную двоичную арифметику. Применим сложение столбиком, не задумываясь о знаках чисел:

$$\begin{array}{r} 0000\ 0101 \\ +\ 1111\ 0111 \\ \hline 1111\ 1100 \end{array}$$

Для расшифровки получившегося отрицательного числа применим к нему схему получения дополнительного кода: $1111\ 1100 \rightarrow 0000\ 0100_2 = 4_{10}$. Таким образом, результат равен -4_{10}, что совпадает с правилами «обычной» арифметики.

При сложении двух чисел с одинаковыми знаками может случиться переполнение — сумма будет содержать слишком большое количество разрядов. Покажем, как это выглядит для положительных и отрицательных чисел.

Сложим десятичные числа 96 и 33. Их сумма 129 выходит за 8-битную сетку. Для того чтобы обнаружить переполнение, добавим к обоим слагаемым ещё один старший бит, совпадающий со знаковым (рис. 4.9).

$$\begin{array}{r} 0\ |\ 0\ \ 1\ 1\ 0\ 0\ 0\ 0\ 0 \\ +\ 0\ |\ 0\ \ 0\ 1\ 0\ 0\ 0\ 0\ 1 \\ \hline \boxed{0}\ \boxed{1}\ \ 0\ 0\ 0\ 0\ 0\ 0\ 1 \\ S'\ S \end{array}$$

Рис. 4.9

Знаковый разряд S результата равен 1, т. е. сумма получилась отрицательной, хотя оба слагаемых положительны! Процессор определяет переполнение, сравнивая биты S и S': если они различны, то произошло переполнение и результат неверный (см. рис. 4.9).

[1] Второй вариант более эффективен, потому что процессор автоматически сравнивает результат очередного действия с нулём.

То же самое получается, если сложить два достаточно больших по модулю отрицательных числа, например –96 и –33. Добавим к кодам обоих чисел один старший разряд, равный знаковому разряду (рис. 4.10).

$$\begin{array}{r} 1\ |\ 1\ \ 0\ 1\ 0\ 0\ 0\ 0 \\ +\ 1\ |\ 1\ \ 1\ 0\ 1\ 1\ 1\ 1 \\ \hline \boxed{1}\ \boxed{0}\ 1\ 1\ 1\ 1\ 1\ 1 \\ S'\ S \end{array}$$

Рис. 4.10

Получается, что в результате бит $S = 0$, хотя ответ должен быть отрицательным. Биты S' и S не совпадают, это говорит о том, что произошло переполнение. Несложно проверить (сделайте это самостоятельно), что если переполнения нет, значения битов S и S' всегда одинаковы независимо от знаков слагаемых.

Сложение многоразрядных двоичных чисел выполняет специальное устройство — **сумматор** (см. главу 3). Как мы уже говорили, вычитание сводится к сложению с дополнительным кодом вычитаемого, поэтому отдельного блока вычитания в процессоре нет.

Умножение и деление

Умножение и деление выполнять труднее, чем сложение и вычитание. Вспомните, например, что в математике умножение часто заменяют многократным сложением, а деление — многократным вычитанием.

К двоичным числам можно применять обычную схему умножения «столбиком». Перемножим, например, числа 9_{10} ($0000\ 1001_2$) и 5_{10} ($0000\ 0101_2$):

$$\begin{array}{r} 00001001 \\ \times\ 00000101 \\ \hline 00001001 \\ +\ 00000000 \\ 00001001 \\ \hline 0000101101 \end{array}$$

Легко проверить, что это число равно 45_{10}.

В сравнении с десятичной системой, здесь есть серьёзное упрощение: первый сомножитель умножается на единицу (в этом случае результат равен ему самому) или на ноль (результат — 0). Поэтому компьютерное умножение целых чисел состоит из следующих элементарных действий:

1) вычисление очередного произведения в зависимости от младшего бита второго сомножителя: оно равно нулю (если этот бит нулевой) или первому сомножителю (если бит равен единице);
2) сложение содержимого сумматора с очередным произведением;
3) сдвиг содержимого первого сомножителя влево на 1 разряд;
4) сдвиг второго сомножителя вправо на 1 разряд (при этом следующий бит попадёт в младший разряд).

Таким образом, удаётся построить схему умножения без использования таблицы умножения. Заметим, что умножение — это довольно трудоёмкая операция, и для её ускорения конструкторы используют различные «хитрые» приёмы. Поэтому в реальных компьютерах всё может выглядеть значительно сложнее, чем в учебном примере.

Умножение, как и сложение, выполняется одинаково для положительных и отрицательных чисел (в дополнительном коде). Если в нашем примере вместо числа 9 подставить –9, то получится:

$$\begin{array}{r} \times\ 11110111 \\ \underline{\ 00000101} \\ 11110111 \\ +\ 00000000 \\ \underline{\ 11110111} \\ 10011010011 \end{array}$$

Оставив только 8 младших битов, можно убедиться (применяя алгоритмы А1–А3), что результат — это дополнительный код числа –45.

Теория деления нацело намного сложнее, чем приёмы умножения, поэтому мы её обсуждать не будем.

Сравнение

В отличие от арифметических действий, операция сравнения *по-разному* выполняется для чисел со знаком и без него. Еще раз внимательно посмотрим на таблицы кодов 8-битных чисел, приведённые в § 27 (табл. 4.4). Если сравниваемые коды не превышают $7F_{16}$, то оба числа положительны и сравнение однозначно. Если это не так, то сравнение чисел с учётом и без учёта знака дает разные результаты. Например, для беззнаковых чисел 81_{16} (129_{10}) больше, чем $7F_{16}$ (127_{10}). Для чисел со знаком, наоборот, отрицательное значение 81_{16} (-127_{10}) будет меньше, чем $7F_{16}$ (127_{10}). Поэтому современные процессоры имеют разные команды для сравнения чисел со знаком и без знака. Чтобы не путаться, при сравнении с учётом знака обычно используют термины «больше»/«меньше», а при сравнении без учёта знака — «выше»/«ниже».

Поразрядные логические операции

В главе 3, изучая основы математической логики, мы увидели, что обработка истинности и ложности высказываний может быть представлена как набор операций с двоичными кодами. Оказывается, что логические операции, введённые первоначально для обработки логических данных, можно формально применить к битам двоичного числа, и такой подход широко используется в современных компьютерах.

Рассмотрим электронное устройство для управления гирляндой лампочек. Состояние каждой лампочки будет задаваться отдельным битом в некотором управляющем регистре: если бит равен нулю, лампочка выключена, если единице — включена. Для получения различных световых эффектов (типа «бегущих огней») требуется зажигать или гасить отдельные лампочки, менять их состояние на противоположное и т. д. (рис. 4.11). Точно так же биты регистров используются для управления внешними устройствами.

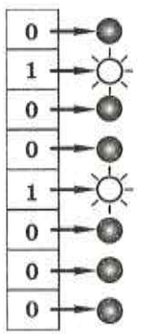


Рис. 4.11

Будем применять логические операции к каждому биту числа, как обычно, считая, что 1 соответствует значению «истина», а 0 — «ложь». Эти операции часто называют **поразрядными** или **битовыми**,

поскольку действия совершаются над каждым разрядом в отдельности, независимо друг от друга[1].

Введём несколько терминов, которые используются в литературе по вычислительной технике. **Сброс** — это запись в бит нулевого значения, а **установка** — запись единицы. Таким образом, если бит в результате какой-то операции становится равным нулю, то говорят, что он сбрасывается. Аналогично, когда в него записана единица, говорят, что бит установлен.

Маска — это константа (постоянная), которая определяет область применения логической операции к битам многоразрядного числа. С помощью маски можно скрывать (защищать) или открывать для выполнения операции отдельные биты[2].

Основные логические операции в современных процессорах — это «НЕ» (not), «И» (and), «ИЛИ» (or) и «исключающее ИЛИ» (xor).

Логическое «НЕ» (инвертирование, инверсия, not) — это замена всех битов числа на обратные значения: 0 на 1, а 1 — на 0. Эта операция используется, например, для получения дополнительного кода отрицательных чисел (см. алгоритм А1 в § 27). «НЕ» — это *унарная операция*, т. е. она действует на все биты *одного* числа. Маска здесь не используется.

D	M	D and M
0	0	0
1	0	0
0	1	0
1	1	1

Рис. 4.12

Логическое «И» (and). Обозначим через D содержимое некоторого бита данных, а через M — значение соответствующего ему бита маски. Операция «И» между ними задаётся таблицей, показанной на рис. 4.12.

Из таблицы видно, что при выполнении логического «И» нулевой бит в маске всегда сбрасывает (делает равным нулю) соответствующий бит результата, а единичный бит позволяет сохранить значение D (как бы пропускает его, открывая «окошко»).

[1] Для сравнения, сложение (как и другие арифметические действия) не является поразрядной операцией, поскольку возможен перенос из младшего разряда в старший.

[2] Использование маски аналогично выделению области рисунка в графическом редакторе — для выделенных пикселей маска равна 1, для остальных — нулю (или наоборот).

С помощью логической операции «И» можно *сбросить* отдельные биты числа (те, для которых маска нулевая), не меняя значения остальных битов (для которых в маске стоят единицы).

Например, операция X and 1 сбросит у любого числа X все биты, кроме самого младшего. С помощью этого приёма легко узнать, является ли число чётным: остаток от деления на 2 равен последнему биту!

Логическое «ИЛИ». Вспомнив таблицу истинности логической операции «ИЛИ» (or), можно обнаружить, что ноль в маске сохраняет бит (D or 0 = D), а единица устанавливает соответствующий бит результата (D or 1 = 1) (рис. 4.13).

D	M	D or M
0	0	0
1	0	1
0	1	1
1	1	1

Рис. 4.13

С помощью логической операции «ИЛИ» можно *установить* отдельные биты числа (те, для которых маска единичная), не меняя значения остальных битов (для которых в маске стоят нули).

Например, операция X or 80_{16} установит старший бит восьмиразрядного числа X, формально сделав тем самым число отрицательным.

Таким образом, используя операции «И» и «ИЛИ», можно сбрасывать и устанавливать любые биты числа, т. е. *строить любой нужный двоичный код*. Где это может пригодиться? Рассмотрим примеры решения конкретных задач.

Пример 1. На клавиатуре набраны 3 цифры, образующие значение целого числа без знака. Определить, какое число было введено.

При нажатии клавиши на клавиатуре в компьютер поступает код нажатой клавиши. Выпишем десятичные и шестнадцатеричные коды всех символов, обозначающих цифры:

Символ	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
X_{10}	48	49	50	51	52	53	54	55	56	57
X_{16}	30	31	32	33	34	35	36	37	38	39

Будем пользоваться шестнадцатеричными кодами: как видно из таблицы, их связь с цифрами числа гораздо нагляднее. Чтобы получить числовое значение цифры из кода символа X, достаточно сбросить его старшие четыре бита, не изменяя значений четырёх младших битов. Для этого нужно использовать операцию X and $0F_{16}$.

Пусть S_1 — код первого введенного символа, S_2 — второго, S_3 — третьего, а N обозначает искомое число. Тогда алгоритм перевода кодов символов в число выглядит так:

1. $N = 0$.
2. $W = S_1$ and $0F_{16}$ (выделяем первую цифру).
3. $N = 10 \cdot N + W$ (добавляем её к числу).
4. $W = S_2$ and $0F_{16}$ (выделяем вторую цифру).
5. $N = 10 \cdot N + W$ (добавляем её к числу).
6. $W = S_3$ and $0F_{16}$ (выделяем третью цифру).
7. $N = 10 \cdot N + W$ (добавляем её к числу).

Пусть, например, набраны символы '1', '2' и '3'. Тогда по таблице находим, что $S_1 = 31_{16}$, $S_2 = 32_{16}$ и $S_3 = 33_{16}$. Значение W на втором шаге вычисляется так:

$$\begin{array}{rll} & 0011 & 0001 \\ \text{and} & 0000 & 1111 \\ \hline & 0000 & 0001 \end{array}$$

Так как $W = 1$, на третьем шаге получаем $N = 1$. Следующая пара шагов — четвёртый и пятый — дают результаты $W = 2$ и $N = 12$ соответственно. Наконец, результат завершающих шагов: $W = 3$ и $N = 123$.

Такая процедура используется в каждом компьютере: именно так коды цифровых символов, набранные на клавиатуре, преобразуются в числа, с которыми компьютер выполняет арифметические действия. Заметим, что фактически здесь использована схема Горнера для представления целого числа (см. § 10).

Пример 2. Создадим структуру данных S, которая отражает, есть или нет в некотором числе каждая из цифр от 0 до 9. В математике такая структура называется **множеством**. Для хранения S будем использовать 16-разрядное целое число (рис. 4.14).

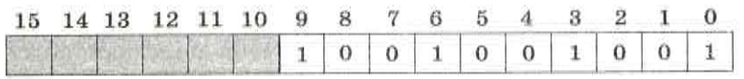


Рис. 4.14

Договоримся, что младший бит числа имеет номер 0, и хранит информацию о том, есть ли во множестве цифра 0 (если этот бит равен 0, такой цифры нет, если равен 1, то есть). Аналогично первый бит (второй по счёту справа) показывает, есть ли во множестве цифра 1, и т. д. Старшие биты 10–15 при этом не используются. Например, во множестве, изображенном на рис. 4.14, есть только цифры 0, 3, 6 и 9.

Для записи элементов во множество и проверки их наличия удобно использовать логические операции. Рассмотрим для примера бит 5. Маска, которая потребуется для обращения к нему, — это единица в пятом разряде и нули во всех остальных, т. е. $M = 0020_{16}$. С её помощью можно добавить элемент к множеству с помощью операции «ИЛИ»: $S = S$ or M. А узнать, есть ли во множестве интересующая нас цифра, можно, выделив соответствующий бит с помощью логического «И» ($P = S$ and M) и проверив результат на равенство нулю.

Исключающее ИЛИ. Как видно из таблицы истинности, операция «исключающее ИЛИ» (xor) не изменяет биты, когда маска нулевая, и меняет биты на противоположные при единичной маске (рис. 4.15).

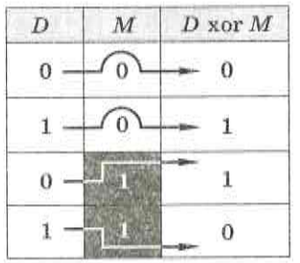


Рис. 4.15

Например, команда $X = X$ xor FF_{16} выполняет инверсию всех битов 8-разрядного целого числа X.

С помощью логической операции «исключающее ИЛИ» можно выполнить *инверсию* отдельных битов числа (тех, для которых маска единичная), не меняя значения остальных битов (для которых в маске стоят нули).

Пример 3. Пусть X — это результат выполнения некоторого вычислительного теста, а Y — то, что ожидалось получить («правильное» значение). Нужно определить, в каких разрядах различаются эти числа (для инженера это очень полезная подсказка, где искать неисправность).

Предположим, что $X = 7$, $Y = 3$. В результате операции X xor Y устанавливаются (в единицу) только те разряды, которые в этих числах не совпали, а остальные сбрасываются[1]. В данном случае находим, что числа различаются только одним битом:

$$\begin{array}{rr} & 0000\ \ 0111 \\ \text{xor} & 0000\ \ 0011 \\ \hline & 0000\ \ 0100 \end{array}$$

Пример 4. Используя логическую операцию «исключающее ИЛИ», можно шифровать любые данные. Покажем это на примере простого текста '2*2=4'.

Выберем любую маску, например $17_{16} = 0001\ 0111_2$. Эта маска представляет собой *ключ шифра* — зная ключ, можно расшифровать сообщение. Возьмём первый символ — цифру '2', которая имеет код $50_{10} = 0011\ 0010_2$, и применим операцию «исключающее ИЛИ» с выбранной маской:

$$\begin{array}{rr} & 0011\ \ 0010 \\ \text{xor} & 0001\ \ 0111 \\ \hline & 0010\ \ 0101 \end{array}$$

Полученное значение $0010\ 0101_2 = 37_{10}$ — это код символа '%'. Для расшифровки применим к этому коду «исключающее ИЛИ» с той же маской:

$$\begin{array}{rr} & 0010\ \ 0101 \\ \text{xor} & 0001\ \ 0111 \\ \hline & 0011\ \ 0010 \end{array}$$

[1] Профессиональные программисты часто используют операцию xor для обнуления переменной: команда `R := R xor R` в языке Паскаль запишет в переменную R ноль, независимо от её начального значения.

В результате получили число 50 — код исходной цифры '2'.

Повторное применение операции «исключающее ИЛИ» с той же маской *восстанавливает исходное значение*, т. е. эта логическая операция *обратима*.

Если применить такую процедуру шифрования ко всем символам текста '2*2=4', то получится зашифрованный текст '%=%*#'.

Обратимость операции «исключающее ИЛИ» часто используется в компьютерной графике для врéменного наложения одного изображения на другое. Это может потребоваться, например, для выделения области с помощью инвертирования её цвета.

Сдвиги

Об операции сдвига вспоминают гораздо реже, чем она того заслуживает. Перечитайте ещё раз алгоритм умножения, описанный выше, и вы убедитесь, что он весь построен на сдвигах. Сдвиги незаменимы тогда, когда требуется проделать ту или иную обработку *каждого* бита, входящего в число. Наконец, сдвиги двоичного числа позволяют быстро умножить или разделить число на степени двойки: 2, 4, 8 и т. д. Поэтому программисты очень ценят и широко применяют всевозможные разновидности сдвигов.

Идея операции сдвига довольно проста: все биты кода одновременно сдвигаются в соседние разряды[1] влево или вправо (рис. 4.16).

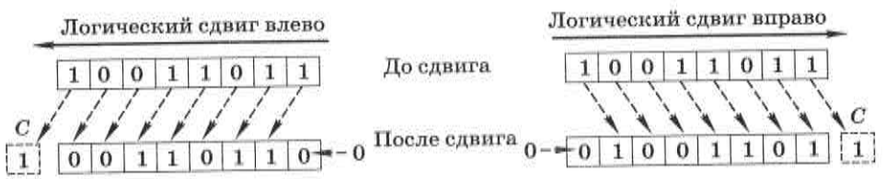


Рис. 4.16

[1] Аппаратно сдвиг реализуется необычайно просто и изящно: регистр, содержащий число, сбрасывается в ноль, при этом из тех разрядов, где исчезла единица, электрический импульс проходит в соседние и устанавливает их в единицу. При этом важно, что все разряды обрабатываются *одновременно*.

Отдельно надо поговорить о двух крайних битах, у которых «нет соседей». Для определённости обсудим сдвиг влево. Для самого младшего бита (на рис. 4.16 он крайний справа) данные взять неоткуда, поэтому в него просто заносится ноль. Самый старший (крайний слева) бит должен потеряться, так как его некуда сохранить. Чтобы данные не пропали, содержимое этого разряда копируется в специальную ячейку процессора — **бит переноса**[1] *C* (от англ. *carry* — перенос).

Рассмотренный тип сдвига обычно называется **логическим сдвигом**. Его можно использовать для быстрого умножения и деления. Рассмотрим, например, 8-разрядный двоичный код 0000 1100, который представляет число 12_{10}. Выполнив логический сдвиг влево, получим 0001 1000, т. е. число 24_{10}, которое вдвое больше! Это не случайность: вспомните, что происходит, если к десятичному числу справа приписать дополнительный ноль, например, 34 → 340.

При сдвиге вправо любое четное число уменьшается ровно в 2 раза. В случае нечётного значения происходит деление нацело, при котором остаток отбрасывается. Например, из 0001 0001 = 17_{10} при сдвиге вправо получается 0000 1000 = 8_{10}.

Логический сдвиг влево на 1 разряд увеличивает целое положительное число вдвое, а сдвиг вправо — делит на 2 нацело.

Пример. Для умножения числа, находящегося в ячейке *Z*, на 10 можно использовать такой алгоритм:

1. Сдвиг влево *Z* (в ячейке *Z* получаем $2Z_0$, где Z_0 — исходное число).
2. $X = Z$ (сохраним $2Z_0$).
3. Сдвиг на 2 бита влево *X* (вычислили $8Z_0$).
4. $X = X + Z$ ($X = 8Z_0 + 2Z_0 = 10Z_0$).

Для некоторых компьютеров такая последовательность выполняется быстрее, чем стандартная операция умножения.

Посмотрим теперь, что получится при сдвиге отрицательных чисел в дополнительном коде. Сдвинем влево код 1111 1000

[1] При программировании на языках высокого уровня бит переноса недоступен.

(8-разрядное представление числа –8): получится 1111 0000. Легко проверить, что это дополнительный код числа –16, т. е. значение удвоилось! Но со сдвигом вправо ничего не получается: из 1111 1000 получаем 0111 1100 — это код положительного числа! Дело в том, что при сдвиге вправо отрицательных чисел, в отличие от положительных, старший разряд надо заполнять не нулём, а единицей! Чтобы исправить положение, вводится ещё одна разновидность сдвига — **арифметический сдвиг**. Его единственное отличие от логического состоит в том, что старший (знаковый) бит не меняется, т. е. знак числа остаётся прежним (рис. 4.17).

Рис. 4.17

Если применить арифметический сдвиг к коду 1111 1000, получается 1111 1100 — дополнительный код числа –4, т. е. произошло деление на 2. В качестве упражнения проверьте, как ведёт себя отрицательное нечётное число при арифметическом сдвиге вправо.

Арифметический сдвиг влево не требуется, поскольку он ничем не отличается от обычного логического сдвига.

То, что в результате логических сдвигов содержимое крайних разрядов теряется, не всегда удобно. Поэтому в компьютере предусмотрен **циклический сдвиг**, при котором бит из одного крайнего разряда переносится в другой («по циклу», рис. 4.18).

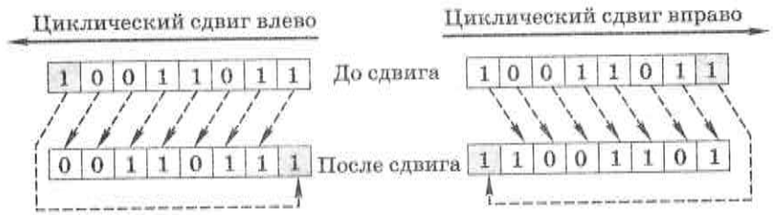


Рис. 4.18

Циклический сдвиг позволяет «просмотреть» все биты и вернуться к исходному значению. Если сделать последовательно 8 циклических сдвигов 8-битного числа, каждый его бит на каком-то шаге окажется на месте младшего разряда, где его можно выделить с помощью логической операции «И» с маской 1. Так можно «просматривать» не только младший, но и любой другой разряд (например, для выделения старшего разряда нужно использовать маску 80_{16}).

Вопросы и задания

1. Покажите на примере, как складываются два положительных целых числа, записанные в 8-разрядные ячейки. Что изменится, если числа будут отрицательными?
2. Что такое дополнительный код? Сформулируйте правила получения дополнительного кода числа.
3. При каких комбинациях знаков слагаемых в результате сложения может возникнуть переполнение?
4. Какое устройство выполняет в компьютере сложение? Вспомните, что вы знаете об этом устройстве.
5. Почему не нужно разрабатывать специальное устройство для вычитания целых чисел?
6. Перемножьте столбиком два положительных целых числа в двоичной системе счисления. Изменится ли алгоритм выполнения операции, если у одного из сомножителей поменять знак?
7. Почему коды чисел со знаком и без знака нужно сравнивать по-разному?
8. Что такое поразрядные операции? Приведите примеры.
9. Почему арифметические операции нельзя отнести к поразрядным?
10. Что такое маска?
11. Как, используя маску, сбросить определённый бит (записать в него 0)?
12. Напишите значение маски для того, чтобы сбросить в 16-разрядном числе 2 младших бита, не изменяя все остальные. Какую логическую операцию нужно для этого использовать?
13. Как, используя маску, установить определённый бит?
14. Напишите значение маски для того, чтобы установить в 16-разрядном числе 2 старших бита, не изменяя все остальные. Какую логическую операцию нужно для этого использовать?
15. Как, используя логические операции, определить, делится ли число на 4? На 8?
16. В каких практических задачах можно применять установку или сброс битов двоичного кода?
17. Каковы возможности операции «исключающее ИЛИ»?

*18. Попробуйте придумать алгоритм шифрования кода с помощью операции «исключающее ИЛИ». Постарайтесь предложить простой алгоритм изменения маски, а не просто использовать константу.
19. Прочитайте ещё раз материал, связанный с переполнением при сложении. Какой логической операцией можно определить, совпадают или нет биты S' и S?
20. Какую роль играет операция «НЕ» при получении отрицательных чисел?
21. Как выполнить инверсию всех битов, не используя логическую операцию «НЕ»?
22. Что такое сдвиг? Какие вы знаете виды сдвига?
23. Как обрабатываются самый старший и самый младший биты при различных типах сдвига?
24. Покажите на примерах, что сдвиг влево двоичного кода удваивает число, а сдвиг вправо — уменьшает вдвое.
25. Почему логический сдвиг не годится для уменьшения в два раза отрицательных чисел? Как работает арифметический сдвиг?
26. Почему не требуется арифметический сдвиг влево?
*27. Выведите правило вычисления результата арифметического сдвига отрицательного нечётного числа на один разряд вправо. Проверьте, применимо ли это правило к положительным нечётным числам. Как упрощается формула для чётных исходных значений?
28. Где могут применяться сдвиги?

Подготовьте сообщение

а) «Битовые логические операции»
б) «Шифрование с помощью операции ‘исключающее ИЛИ’»
в) «Применение сдвигов»

Задачи

1. Переведите в 8-разрядный двоичный код десятичные числа 31 и 19 и сложите их. Для проверки переведите полученную сумму в десятичную систему счисления.
2. Повторите вычисления предыдущей задачи, заменив первое слагаемое на –31. Подумайте, что изменится, если код сделать 16-разрядным?
3. Выберите произвольные значения двух целых чисел A и B и запишите их в виде 8-разрядных двоичных кодов. Проверьте путем непосредственных вычислений справедливость тождества $A - B = A + (-B)$.

4. Сложение ведётся в 8-разрядной арифметике со знаком. Какое максимальное число можно прибавить к двоичной константе 100000_2, чтобы не возникло переполнения? Как изменится результат, если число будет беззнаковым?
5. Переведите в двоичный код десятичные числа 12 и 7 и перемножьте их. Для проверки переведите результат в десятичную систему счисления.
6. Повторите вычисления предыдущей задачи, заменив первый сомножитель на –12. Считайте, что числа представлены в 8-разрядном коде.
7. Братья Петя и Коля часто спорят по поводу решения задач по информатике. Главная причина состоит в том, что Петя всегда решает задачу, как показал учитель, а Коля вечно придумывает что-то свое, причем не всегда удачно. Сегодня, например, они осваивали двоичную арифметику, умножая 1000_2 на 11011_2. Петя добросовестно умножал столбиком, а Коля взял второй сомножитель и, приписав к нему три нуля, получил такой же ответ. После объяснений Петя был вынужден признать правоту брата. Как объяснил свое решение Коля?
8. Какое из двух беззнаковых чисел больше: 0111 0111 или 1000 1000? Изменится ли ваш ответ, если вам скажут, что исходные коды — это 8-разрядные числа со знаком? Переведите оба значения для случаев чисел со знаком и без него в десятичную систему счисления.
9. Код строчной латинской буквы 'a' равен 61_{16}, а заглавной 'A' — 41_{16}. Используя логическую операцию «И», преобразуйте код строчной буквы в код заглавной. Проверьте, работает ли предложенный вами метод для других букв.
10. Используя логическую операцию «ИЛИ», преобразуйте код заглавной буквы 'A' в код строчной 'a'. Проверьте, работает ли предложенный вами метод для других букв.
11. Петя и Коля решают домашнюю задачу: известны коды двух введённых цифр C_1 и C_2. Найти сумму значений этих цифр. Петя, как обычно, глядя на решение задач в классе, пишет:
    1) $N_1 = C_1$ and $0F_{16}$;
    2) $N_2 = C_2$ and $0F_{16}$;
    3) $S = N_1 + N_2$.

    Коля предлагает более короткое решение:
    1) $S = C_1 + C_2$;
    2) $S = S$ and $0F_{16}$.

    Петя, ссылаясь на образцы задач в учебнике, критикует такой подход. Но Коля показывает на двух примерах ('2' и '3'; '5' и '7'), что его алгоритм даёт правильные результаты. Что скажет учитель по поводу Колиного решения?

12. Выполните битовую операцию X and 3 для следующих десятичных значений X: 4, 5, 8, 15, 16. Для каких из них получился нулевой ответ? Что общего у этих чисел?
13. Разработайте аналогичные способы определения делимости на 2, 8 и 16.

*14. Попробуйте разработать алгоритм, который позволяет поменять местами значения двух ячеек памяти, используя только операцию «исключающее ИЛИ».

15. Цвет точки в формате RGB хранится как 4-байтовое целое число, которое в шестнадцатеричном виде выглядит так: 00 RR GG BB (т. е. старший байтов не используется, а в каждом из последующих байтов хранится одна из трёх цветовых компонент[1]). Напишите последовательность операций, позволяющих выделить из 32-битного числа каждую из трёх цветовых компонент. Какая из них потребует большего числа операций?
16. Петя и Коля решают задачу: цвет точки в формате RGB хранится как 4-байтовое целое число N, которое в шестнадцатеричном виде выглядит так: 00 RR GG BB. Написать последовательность операций, позволяющих выделить из 32-битного числа красную компоненту. Петино решение:
    1) $N = N$ and $FF0000_{16}$;
    2) логический сдвиг N вправо на 16 разрядов.

    Колино решение содержит только вторую из этих операций. Чьё решение правильное?
17. Используя только сдвиги, сбросьте 4 старших разряда 8-битного значения. Как с помощью сдвигов сбросить 4 младших разряда?
18. Каков результат логического сдвига влево на 4 разряда целого числа $FEDC_{16}$ в 16-битном регистре? Сравните его с результатом циклического сдвига.
19. Два целых числа записаны в 16-битные регистры: 1234_{16} и $FEDC_{16}$. К каждому из них применяются логический, циклический и арифметический сдвиги вправо на 4 разряда (каждый раз сдвигается первоначальное значение, а не результат предыдущего сдвига!) Напишите и объясните результаты для каждой операции.

[1] Несмотря на то, что старший байт кажется лишним, этот способ хранения не лишен смысла. Дело в том, что процессор не приспособлен к обработке 3-байтовых данных, тогда как с 4-байтовыми работает очень быстро. Описанный формат, в частности, применяется при хранении таблиц цветов в графическом формате BMP. В других форматах (например, в PNG) старший байт используется для хранения степени прозрачности пикселя (альфа-канала).

20. Запишите число –18 в 8-разрядном двоичном коде. Что получится, если применить к нему логический сдвиг вправо? Арифметический сдвиг вправо? Сравните полученные результаты и объясните их.
21. Переведите число –1 в дополнительный двоичный код и дважды примените к нему арифметический сдвиг вправо. Какой будет результат?
22. Выполните приведённый в тексте параграфа алгоритм умножения на 10 для $Z = 1100_2$. Для проверки переведите результат в десятичную систему счисления.

§ 29
Хранение в памяти вещественных чисел

В начале главы мы отмечали принципиальное различие между вещественными и целыми числами: целые числа дискретны, а вещественные, напротив, непрерывны, а значит, не могут быть полностью корректно перенесены в дискретную по своей природе вычислительную машину. Как же всё-таки кодируются в компьютерах вещественные числа?

В первых ЭВМ использовалось кодирование **с фиксированной запятой**. Это значит, что положение запятой, отделяющей целую часть от дробной, было жёстко закреплено в разрядной сетке конкретной ЭВМ — раз и навсегда для всех чисел и для всех технических устройств этой машины. Все вычислительные алгоритмы были заранее «настроены» на это фиксированное размещение. Но в задачах, которые решаются на компьютерах, встречаются самые разнообразные по величине числа, от размера атома до астрономических расстояний. Чтобы согласовать их с таким жёстким представлением, программист, подготавливая задачу к решению на ЭВМ, выполнял большую предварительную работу по *масштабированию* данных: маленькие числа умножались на определённые коэффициенты, а большие, напротив, делились. Масштабы подбирались так, чтобы результаты всех операций, включая промежуточные, не выходили за пределы разрядной сетки и в то же время обеспечивалась максимально возможная точность (все разряды данных по возможности находились в пределах сетки). Эта работа требовала много времени и часто являлась источником ошибок.

Тем не менее работа с фиксированным размещением запятой не только показала недостатки метода, но и наметила путь их устранения. В самом деле, если наиболее сложным и трудоёмким местом является масштабирование данных, надо его автоматизировать. Иными словами, надо научить машину самостоятельно размещать запятую так, чтобы числа при счёте не выходили за разрядную сетку и по возможности сохранялись с максимальной точностью. Конечно, для этого нужно разрешить компьютеру «перемещать» запятую, а значит, дополнительно как-то сохранять в двоичном коде числа информацию о её текущем положении. В этом и заключается главная идея представления чисел **с плавающей запятой**[1].

Удобное представление вещественных чисел не пришлось специально придумывать. В математике уже существовал подходящий способ записи, основанный на том, что любое число A в системе счисления с основанием B можно записать в виде

$$A = \pm Z \cdot B^P,$$

где Z называют **значащей частью**, а показатель степени P — **порядком** числа (рис. 4.19). Для десятичной системы это выглядит привычно, например, заряд электрона равен $-1{,}6 \cdot 10^{-19}$ кулона, а скорость света в вакууме составляет $3 \cdot 10^8$ м/с.

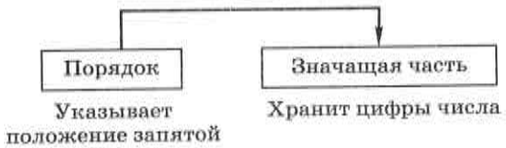


Рис. 4.19

Однако представление числа с плавающей запятой не единственно. Например, число 23,4 можно записать следующими способами:

$$2340 \cdot 10^{-2} = 234 \cdot 10^{-1} = 23{,}4 \cdot 10^0 = 2{,}34 \cdot 10^1 = \\ = 0{,}234 \cdot 10^2 = 0{,}0234 \cdot 10^3 = \ldots$$

На первый взгляд, выбор очень широкий, однако большинство вариантов обладают серьезными недостатками. В частности,

[1] В англоязычных странах используется термин *floating point — плавающая точка*, поскольку в этих странах традиционно целая часть отделяется от дробной не запятой, как у нас, а точкой.

все представления, в которых значащая часть содержит нули непосредственно после запятой (0,0234, 0,00234 и т. п.) или перед ней (2340, 23400 и т. п.), не подходят, поскольку, сохраняя эти незначащие нули, мы напрасно увеличиваем разрядность чисел. Согласно математической теории, для обеспечения максимальной точности при сохранении цифр числа в фиксированном количестве разрядов надо выбирать такой метод, при котором значащие цифры числа следует поместить как можно ближе к запятой. С этой точки зрения оптимальным будет вариант, когда целая часть равна нулю, а первая ненулевая цифра находится сразу после запятой (в нашем примере 0,234). При этом вместо двух частей (целой и дробной) остаётся только дробная, что фактически делает ненужной «разделительную» запятую.

Но взгляните на рис. 4.20, *а*, изображающий такое число на индикаторе: первый разряд всегда равен нулю, что делает его практически бесполезным. Поэтому с точки зрения экономии разрядов лучше взять другой вариант, в котором значащая часть равна 2,34 (рис. 4.20, *б*). Именно такой выбор закреплён в стандарте IEEE 754[1], на котором основана арифметика вещественных чисел в современных компьютерах.

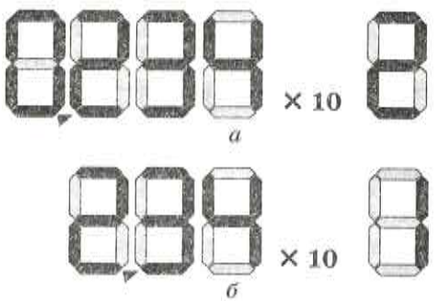


Рис. 4.20

Итак, существуют два приблизительно равноценных способа представления чисел с плавающей запятой:

- оптимальный с теоретической точки зрения, в котором целая часть нулевая, а первая цифра дробной части ненулевая ($0{,}234 \cdot 10^2$);

[1] Последняя версия стандарта называется IEEE 754-2008.

- более удобный с практической точки зрения, в котором целая часть состоит из единственной ненулевой цифры ($2{,}34 \cdot 10^1$).

К сожалению, эта «двойственность» порождает некоторую путаницу. В теоретической литературе, как правило, используется первый способ[1]. Все описания конкретных компьютерных систем, напротив, базируются на втором. Причём в обоих случаях обычно используется один и тот же русский термин — **мантисса**. Зато в англоязычной компьютерной литературе приняты два разных термина: в первом случае значащая часть называется *mantissa* (слово «мантисса» для математиков однозначно связано с дробной частью числа), а во втором — *significand* (значащая часть).

Далее мы будем использовать второй вариант, поскольку именно он даёт возможность решать задачи, связанные с практическим кодированием вещественных чисел. В связи с этим мы будем применять термин «значащая часть», а не «мантисса».

В компьютере используется такое представление вещественных чисел с плавающей запятой, при котором значащая часть Z удовлетворяет условию $1 \le Z < B$, где B — основание системы счисления. Такое представление называется **нормализованным**.

Нормализованное представление числа единственно — в нашем примере это $2{,}34 \cdot 10^1$. Любое число может быть легко нормализовано. Единственное, но важное исключение из правила составляет нуль — для него невозможно получить $Z \ge 1$. Ради такого важного случая было введено дополнительное соглашение: число 0, в котором все биты нулевые, в качестве исключения считается нормализованным.

Всё сказанное выше можно применить и к двоичной системе:

$$A = \pm Z \cdot 2^P, \text{ причём } 1 \le Z < 2.$$

Например: $-7_{10} = -111 \cdot 2^0 = -1{,}11 \cdot 2^{10}$ (не забывайте, что значащая часть и порядок записаны в двоичной системе!); отсюда $Z = 1{,}11$ и $P = 10$. Двоичная значащая часть *всегда* (исключая, разумеется, ноль!) *начинается с единицы*, так как $1 \le Z < 2$. Поэтому

[1] К этой группе относится большинство отечественных книг по основам вычислительной техники.

во многих компьютерах (в том числе в компьютерах на базе процессоров Intel), эта так называемая **скрытая единица** не хранится в ОЗУ, что позволяет сэкономить еще один дополнительный разряд значащей части[1].

Идея «скрытой единицы» раньше действительно давала заметное увеличение точности представления чисел, поскольку количество разрядов в устройствах ЭВМ того времени было невелико и поэтому усложнение метода кодирования было оправдано. Сейчас, когда процессоры работают с 64-битными данными, это скорее дань традиции, чем практически полезная мера[2].

Таким образом, при кодировании вещественного числа с плавающей запятой фактически *хранятся две величины: его значащая часть* (significand) *и порядок*. От разрядности значащей части зависит точность вычислений, а от разрядности порядка — диапазон представления чисел. В таблице 4.6 приведены характеристики стандартных вещественных типов данных, используемых в математическом сопроцессоре Intel.

Таблица 4.6

Тип	Диапазон	Число десятичных значащих цифр	Размер (байтов)
single	$1{,}4 \cdot 10^{-45} - 3{,}4 \cdot 10^{38}$	7–8	4
double	$4{,}9 \cdot 10^{-324} - 1{,}8 \cdot 10^{308}$	15–16	8
extended	$3{,}6 \cdot 10^{-4951} - 1{,}2 \cdot 10^{4932}$	19–20	10

Рассмотрим, как «распланированы» 4 байта, отводимые под простейший тип *single*. Тип *double* устроен совершенно аналогично, а тип *extended*, который является основным форматом для

[1] В результате то, что осталось после «скрытия» единичной целой части, можно вполне обоснованно называть мантиссой.

[2] Оценим величину добавки для математического сопроцессора Intel. Значащая часть чисел двойной точности вместо «скрытой единицы» приобретает дополнительный 53-й (!) бит, что прибавляет к значению числа поправку $2^{-53} \approx 1{,}1 \cdot 10^{-16}$, влияющую на 16-й десятичный знак; согласно IEEE 754-2008, в 128-битовых числах эта поправка будет и того меньше: $2^{-113} \approx 9{,}6 \cdot 10^{-35}$.

вычислений в математическом сопроцессоре, отличается только тем, что в нём единица в целой части не «скрывается».

В числах типа *single* 23 младших бита (с номерами от 0 до 22) хранят значащую часть числа, следующие 8 битов (с 23 по 30) — порядок, а старший (31-й) бит отведен под знак числа (рис. 4.21).

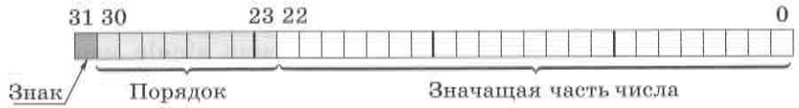


Рис. 4.21

Правила двоичного кодирования вещественных чисел во многом отличаются от правил кодирования целых чисел. Для того чтобы в них разобраться, рассмотрим конкретный пример — закодируем число –17,25 в формате *single*. Прежде всего, переведём модуль числа в двоичную систему, отдельно целую и дробную части (см. § 11):

$$17{,}25 = 10001{,}01_2.$$

Для нормализации нужно передвинуть запятую на $4_{10} = 100_2$ разряда влево:

$$10001{,}01 \cdot 2^0 = 1{,}000101 \cdot 2^{100}.$$

Построим значащую часть, «скрыв» единицу в целой части:

$$M = Z - 1 = 0{,}0001010...0.$$

Так как число отрицательное, знаковый разряд нужно установить в 1, т. е. $S = 1$.

В отличие от целых чисел, значащая часть вещественных чисел хранится в прямом коде.

Таким образом, значащие части положительного и равного по модулю отрицательного числа одинаковы, а отличаются они только старшим (знаковым) битом.

Теперь остается закодировать двоичный порядок 100. Порядок — это целое число со знаком, для него используется **кодиро-**

вание со смещением: чтобы вообще избавится от знака порядка, к нему добавляют некоторое положительное смещение d:

$$P_d = P + d.$$

Величина смещения подбирается так, чтобы число P_d было всегда положительным. В этом случае оказывается легче сконструировать математический сопроцессор для обработки вещественных чисел.

Для кодирования порядка в числах типа single используют смещение $d = 127_{10} = 7F_{16}$. Таким образом, для нашего примера

$$P_d = 100 + 111\ 1111 = \mathbf{1000\ 0011}.$$

Собирая теперь S, P_d и M в единое 32-разрядное число, получаем:

$$1\ \mathbf{10000011}\ 00010100000000000000000.$$

или более компактно в шестнадцатеричной системе:

$$C1\ 8A\ 00\ 00.$$

Этот код и будет записан в память[1].

Не все двоичные комбинации для вещественных чисел соответствуют «правильным» числам: некоторые из них кодируют бесконечные значения, а некоторые — нечисловые данные (*англ.* NaN = not a number, «не число»). Они отличаются от остальных чисел тем, что имеют максимально возможный порядок[2] (например, для типа single это смещённый порядок $P_d = 255$, а для типа double — 2047). Подобные «неправильные» данные возникают только в результате ошибок в вычислениях.

Таким образом, мы увидели, что целые и вещественные числа хранятся в памяти компьютера совершенно по-разному. Поэтому не удивительно, что свойства значений, скажем, 3 и 3,0, в компьютерной арифметике совершенно различные.

[1] На самом деле в IBM-совместимых персональных компьютерах байты будут сохранены в памяти в обратном порядке: 00 00 8A C1.

[2] Это вполне логично, поскольку для вещественных чисел переполнение (получение «бесконечного» значения) наступает именно при больших порядках.

Вопросы и задания

1. Чем вызваны трудности, возникающие при представлении вещественных чисел в компьютере? Как они связаны с непрерывностью вещественных чисел в математике?
2. Объясните, как хранятся вещественные числа с фиксированной запятой. Почему этот метод не используется в современных компьютерах?
3. Что такое плавающая запятая? Из каких частей состоит число при кодировании с плавающей запятой?
4. Приведите примеры физических величин, которые обычно записывают в форме с плавающей запятой.
5. Почему метод представления чисел с плавающей запятой неоднозначен? Как изменится порядок, если запятую сместить на один разряд влево (вправо)?
6. Что такое нормализованная форма записи числа?
7. Как требования нормализации связаны с точностью представления вещественных чисел?
8. Единственно ли нормализованное представление числа? Все ли числа имеют нормализованное представление?
9. Почему старший бит значащей части нормализованного двоичного числа всегда равен единице? Как этот факт используется на практике?
10. Какие числа сохраняются в памяти с нулевой значащей частью?
11. На что влияет разрядность значащей части и разрядность порядка?
12. Почему задание разрядности для целых чисел однозначно определяет их свойства, а для вещественных — нет?
13. Что вы знаете о типах single, double и extended?
14. Как хранится порядок во всех рассмотренных форматах вещественных чисел? Почему не хранится знак порядка?
15. Сравните методы хранения отрицательных целых и вещественных чисел.
16. Как по двоичному представлению вещественного числа определить, положительное оно или отрицательное? Подходит ли этот метод для целых чисел?
17. В каком из вещественных форматов не используется «скрытая единица» и почему?
18. Какие логические операции и с какой маской надо применить, чтобы в переменной типа single:
    а) выделить значащую часть, сбросив порядок и знаковый бит;
    б) восстановить в полученной знаковой части «скрытую единицу»?
19. Как можно выделить смещённый порядок из числа типа single? Как получить истинное значение порядка?

20. С помощью какой маски можно выделить знаковый бит числа, хранящегося в формате single?
21. Что такое NaN?
22. Чем различаются представление в памяти целого числа и равного ему вещественного с нулевой дробной частью (например, 12 и 12,0)?

Подготовьте сообщение

а) «Типы данных для хранения вещественных чисел»
б) «Стандарт IEEE-754»

Задачи

1. Рассмотрим вымышленный 32-разрядный компьютер, в котором вещественные числа кодируются с фиксированной запятой, причём к целой части относится один байт, а к дробной — три. Рассчитайте для такой машины максимальное и минимальное допустимые числа и сравните с аналогичными значениями для типа *single*; объясните разницу. Вычислите также «порог» антипереполнения, т. е. минимальное число, отличное от нуля.
2. Запишите в нормализованном виде следующие десятичные вещественные числа: $43 \cdot 10^{21}$; 1040; 1,5; 0,32; 0,0005; $0{,}34 \cdot 10^{-12}$.
3. Запишите в нормализованном виде следующие двоичные вещественные числа (значащая часть и порядок даны в двоичной системе счисления): $11 \cdot 2^{10100}$; 10110; 1,1; 0,101; 0,0001; $11{,}001 \cdot 2^{-1000}$. Обратите внимание на значение первого бита значащей части.
4. Сравните диапазон чисел, который представляется в 32-битной форме *single*, с диапазоном целых 32-разрядных чисел со знаком.
5. Рассчитайте величину порядка со смещением для чисел типа *single* с двоичными порядками -11_2, 0 и 11_2.
6. Определите, как хранятся в памяти в формате single следующие вещественные десятичные числа: 1; 100; 0,1. Ответ запишите в шестнадцатеричной системе счисления.
7. Используя результаты предыдущей задачи, получите соответствующие коды для вещественных чисел –1, –100 и –0,1.
8. Определите, какому десятичному значению соответствуют коды (тип *single*): $3FC0\,0000_{16}$, $BFC0\,0000_{16}$, $3F44\,0000_{16}$.

*9. Как с помощью битовых логических операций и сдвигов преобразовать некоторое небольшое положительное вещественное число с нулевой дробной частью, например $9_{10} = 1{,}001_2 \cdot 2^{11}$, в форму беззнакового целого?

§ 30
Операции с вещественными числами

Сложение и вычитание

Рассмотрим принципы вещественной компьютерной арифметики на простых примерах. Сложим $7{,}25_{10} = 111{,}01$ и $1{,}75_{10} = 1{,}11$ (здесь и далее будем явно указывать систему счисления только для десятичных чисел). Представим эти числа в нормализованном виде: $111{,}01 \cdot 2^0 = 1{,}1101 \cdot 2^{10}$ и $1{,}11 \cdot 2^0$ (ещё раз подчеркнём, что значащие части и порядки чисел указываются в двоичной системе!). Не будем сейчас использовать «скрытую» единицу: это нужно только при сохранении чисел в памяти, а при изучении арифметических действий удобнее иметь «развёрнутые» значения.

Числа, записанные в форме с плавающей запятой, нельзя непосредственно сложить. Дело в том, что когда числа имеют разные порядки, их значащие части оказываются сдвинутыми друг относительно друга. Поэтому первое, что делает процессор перед сложением вещественных чисел, — **выравнивает** их **порядки** до большего. Число, имеющее меньший порядок p_2 (и значащую часть z_2), «подгоняется» к числу с большим порядком p_1 следующим образом:

1. Если $p_2 = p_1$, то порядки выровнены и преобразования закончены.
2. $p_2 = p_2 + 1$.
3. Сдвинуть значащую часть z_2 на один разряд вправо.
4. Перейти к шагу 1.

Для нашего примера разность порядков равна $10 - 0 = 10 = 2_{10}$, так что для выравнивания порядков значащую часть придётся сдвинуть дважды (порядок при этом увеличится на 2): $1{,}11 \cdot 2^0 = 0{,}0111 \cdot 2^{10}$. Подчеркнём, что ради проведения сложения нормализацию пришлось временно нарушить.

Теперь числа имеют одинаковый порядок и их **значащие части можно складывать:**

$$\begin{array}{r} 1{,}1101 \\ +\ 0{,}0111 \\ \hline 10{,}0100 \end{array}$$

Полный результат (с учётом порядка) равен $10{,}01 \cdot 2^{10}$ (убедитесь, что получившееся число равно 9_{10}). Но значащая часть результата больше 2, поэтому для записи числа в память его необходимо **нормализовать**: $10{,}01 \cdot 2^{10} = 1{,}001 \cdot 2^{11}$.

В этом примере мы нигде не учитывали ограниченность разрядной сетки и для простоты специально взяли короткие числа. Как же обстоит дело в реальных вычислениях? При выравнивании порядков происходит сдвиг значащей части меньшего из чисел вправо, при этом ее младшие (правые) разряды могут выйти за пределы разрядной сетки и будут отброшены. При сложении чисел с большой разностью порядков в результате таких сдвигов меньшее число может стать равно нулю. Например, представьте себе, что при 24-битной значащей части у слагаемых A и B разность порядков составляет, например, 26_{10}. В этом случае при выравнивании порядков произойдёт 26 сдвигов значащей части вправо, так что абсолютно все(!) её разряды исчезнут. В результате сложения окажется, что $A + B = A$, хотя $B \neq 0$ — это очередной (но далеко не единственный) пример погрешности компьютерных вычислений.

Умножение и деление

Числа, представленные в форме с плавающей запятой, «хорошо приспособлены» для выполнения умножения и деления. При **перемножении** достаточно (в полном соответствии с правилами математики) **перемножить** их **значащие части**, а **порядки сложить**. При **делении** значащие части **делятся**, а **порядки вычитаются**. Конечно, результат может оказаться ненормализованным, но это легко устраняется стандартной процедурой.

Рассмотрим, как выполняется умножение чисел $1{,}25_{10} = 1{,}01_2$ и $4{,}0_{10} = 100{,}0_2$. В нормализованном виде они запишутся как $1{,}01 \cdot 2^0$ и $100 \cdot 2^0 = 1{,}0 \cdot 2^{10}$. В этом примере значащие части можно перемножить устно: $1{,}01 \cdot 1{,}0 = 1{,}01$. Теперь сложим порядки: $0 + 10 = 10$. Таким образом, результат равен $1{,}01 \cdot 2^{10}$. Он уже удовлетворяет требованиям нормализации, поэтому никаких дополнительных действий не требуется. Легко показать, что $1{,}01 \cdot 2^{10} = 101 \cdot 2^0 = 5_{10}$.

Знакомство с вещественной арифметикой убедительно показывает важную роль битовых операций, изученных в § 28. Для нормализации постоянно используются сдвиги; для выделения значащей части или порядка из общего кода числа обязательно потребуется логическая операция «И», а для получения единого кода

из порядка и значащей части можно использовать логическое «ИЛИ». В обеих последних задачах также необходимы сдвиги.

Вопросы и задания

1. Почему перед сложением или вычитанием вещественных чисел требуется выравнивать порядки?
2. Какое число — большее или меньшее — подвергается сдвигу при выравнивании порядков? Почему?
3. Верно ли, что при выравнивании порядков значащая часть всегда сдвигается вправо?
4. Как вычислить количество сдвигов, которое потребуется произвести для выравнивания порядков?
5. Может ли получиться так, что при выполнении операции сложения значащие части придется не складывать, а вычитать?
6. Как изменится двоичный код вещественного числа, если это число умножить на 2? Сравните с тем, как меняется код целого числа при удвоении.
7. Почему в компьютерной арифметике возможны случаи, когда $A + B = A$ при $B \neq 0$? При каких условиях может так получиться?

*8. Может ли при вычитании быть переполнение? Антипереполнение?

9. Сформулируйте правила умножения и деления вещественных чисел.

*10. Верно ли, что когда для размещения результата умножения в значащей части не хватает разрядов — это переполнение? Когда возникает переполнение?

11. Может ли в результате арифметической операции нарушиться нормализация? Как в таких случаях нужно поступать? Приведите пример.

Задачи

1. Суммируются два двоичных числа: $1{,}0 \cdot 2^{100}$ и $1{,}11 \cdot 2^{11}$. Какое из них будет сдвигаться при выравнивании порядков? На сколько разрядов?
2. Выполните сложение двух десятичных чисел 2,5 и 0,125, предварительно преобразовав их к двоичной форме и выделив значащую часть и порядок. Результат представьте в формат *single* и запишите в шестнадцатеричном коде.
3. Выполните вычитание десятичных чисел 0,125 – 2,5, предварительно преобразовав их к двоичной форме и выделив значащую часть и порядок. Вычтите из большего числа (2,5) меньшее (0,125), а знак добавьте в конце. Результат представьте в формате *single* и запишите в шестнадцатеричном коде.

*4. Выполните сложение двух десятичных чисел 0,1 и 0,2, предварительно преобразовав их к двоичной форме и выделив значащую часть и порядок. Предположите, что на значащую часть выделяется 8 разрядов, «скрытую единицу» не используйте.

5. Докажите, что если двоичное число $1{,}01 \cdot 2^{10}$ сложить с самим собой, то его значащая часть не изменится, а порядок возрастёт на 1.

*6. Заданы 5 вещественных чисел, причем одно из них $A = 1{,}0 \cdot 2^{111}$, а все остальные равны между собой: $B = C = D = E = 1{,}0 \cdot 2^{-10}$ (значения значащих частей и порядков записаны в двоичной системе счисления). Убедитесь, что в случае, когда значащая часть представлена 8 битами («скрытая единица» не используется), результат сложения всех чисел оказывается зависящим от порядка сложения, в частности $A+B+C+D+E \neq E+D+C+B+A$. Объясните результат.

7. Перемножьте два десятичных числа 0,75 и 1,25, предварительно преобразовав их к двоичной форме и выделив значащую часть и порядок. Для проверки переведите результат в десятичную систему счисления.

8. Двоичное число $1{,}1 \cdot 2^{1000}$ записано в формате, в котором под порядок вещественного числа отводятся 4 бита. Произойдет ли переполнение, если число возвести в квадрат?

9. Выполните деление десятичных чисел 0,75 и 0,25, предварительно преобразовав их к двоичной форме и выделив значащую часть и порядок. Для проверки переведите результат в десятичную систему счисления.

10. Двоичное число $1{,}0 \cdot 2^{-1000}$ записано в формате, в котором под порядок вещественного числа отводятся 4 бита. Что произойдет, если его разделить на $1{,}0 \cdot 2^{1001}$?

Практические работы к главе 4

Работа № 9 «Целые числа в памяти»
Работа № 10 «Арифметические операции»
Работа № 11 «Логические операции и сдвиги»

ЭОР к главе 4 на сайте ФЦИОР (http://fcior.edu.ru)

- Дополнительный код числа. Алгоритм получения дополнительного кода отрицательного числа
- Число и его компьютерный код
- Числа с фиксированной и плавающей запятой

Самое важное в главе 4

- Для хранения чисел в памяти компьютера используется конечное количество разрядов. Из-за этого диапазон чисел, которые можно хранить и обрабатывать в компьютере, ограничен, а результаты вычислений могут быть неточными.
- Для хранения целого числа может быть использовано 8, 16, 32 или 64 бита памяти. Каждый дополнительный бит расширяет диапазон допустимых чисел в 2 раза.
- Отрицательные целые числа хранятся в дополнительном двоичном коде, который позволяет выполнять вычисления с положительными и отрицательными числами по одному и тому же алгоритму.
- Вещественные числа хранятся в памяти компьютера в формате с плавающей запятой: отдельно значащая часть и порядок. Из-за ограниченности числа разрядов вещественное число, как правило, не удается точно представить в памяти.
- При вычислениях с вещественными числами накапливаются ошибки. Для повышения точности расчётов нужно стараться, если возможно, использовать только операции с целыми числами.

Глава 5
Как устроен компьютер

Компьютер — это универсальный программируемый автомат для обработки данных.

Из этого определения можно сделать вывод, что компьютер состоит из двух важнейших составляющих: **аппаратной части** и **программного обеспечения** (ПО). В технической литературе их часто называют английскими терминами **hardware** и **software**[1].

Поскольку одно и то же оборудование может быть перенастроено на выполнение новых задач простой заменой ПО, такие универсальные компьютеры можно выпускать большими партиями, и это делает их производство проще и дешевле. За счёт этого во многих областях они заменили специализированные устройства.

Исторически существовали два принципиально разных типа вычислительных машин — *аналоговые* и *цифровые*. Они различались по способу представления обрабатываемых данных: в аналоговой или цифровой форме. Цифровая техника быстро доказала свои преимущества:

- высокую точность вычислений;
- универсальность и быстроту перехода от одной задачи к другой;
- способность хранить большие объёмы данных.

В результате почти все современные компьютеры работают только с дискретной (цифровой) информацией. Поэтому в этой главе рассматривается только цифровая вычислительная техника.

[1] Слово «hardware» означает металлические изделия, а применительно к компьютеру — его детали (платы, монитор и прочее «железо»). Термин «software» исключительно компьютерный, он возник из противопоставления слов «soft» (мягкий, гибкий, податливый) и «hard» (твёрдый, жёсткий, негнущийся). Это значит, что software гибко «подстраивает» hardware для решения разнообразных задач.

§ 31
История развития вычислительной техники

История появления и развития вычислительной техники — это обширная тема, которой посвящено множество книг. С точки зрения курса информатики, исторические сведения интересны, прежде всего, тем, что позволяют отследить основные направления развития компьютерной техники и попытаться предвидеть её ближайшее будущее.

В отечественной технической литературе приблизительно до 80-х годов прошлого века везде использовался термин **«электронная вычислительная машина» (ЭВМ)**. Позднее это словосочетание стало постепенно вытесняться новым более коротким названием **«компьютер»**. Все разновидности современной вычислительной техники сейчас называются только компьютерами, но, тем не менее, старые модели по традиции именуются ЭВМ.

Блез Паскаль (1623–1662)

Вехи истории

Первая **механическая машина**, с помощью которой можно было производить вычисления, была изготовлена известным французским ученым *Блезом Паскалем* в 1645 г. Чтобы отдать дань уважения этому изобретению, один из языков программирования впоследствии был назван именем *Паскаль*.

Идея о реализации вычислений в автоматическом режиме (без участия человека) впервые была предложена и детально развита английским учёным *Чарльзом Бэббиджем*. Он спроектировал и описал **Аналитическую машину**, состав и принципы действия которой фактически повторились в будущих ЭВМ.

Чарльз Бэббидж (1791–1871)

Ада Лавлейс
(1815–1852)

Бэббидж посвятил всю свою жизнь работе над машиной, но построить ее из механических деталей не удалось: уровень техники XIX века не позволял изготовить столь сложный и точный механизм.

Разработкой принципов программирования Аналитической машины Бэббиджа занималась *Ада Лавлейс* (дочь английского поэта Д. Г. Байрона). Её идеи оказали большое влияние на развитие программирования. Например, ей принадлежат термины «цикл» и «рабочая ячейка». В честь первого в мире программиста один из языков программирования получил имя *Ада*.

Первой ЭВМ, продемонстрировавшей на практике возможность автоматических расчётов по программе, считается **ЭНИАК** (сокращение от английского словосочетания *Electronic Numeric Integrator and Computer*) — рис. 5.1. Эта ЭВМ была построена в 1944 г. в США под руководством *Джона Моучли*; главным инженером проекта был *Преспер Эккерт*. ЭНИАК содержал 18 000 электронных ламп и, занимая зал 9 × 15 м^2, потреблял около 150 кВт электроэнергии; выполнял более 350 умножений и 5000 сложений за секунду. Данные вводились в машину с помощью перфокарт, а программа обработки набиралась с помощью штекеров на специальных панелях.

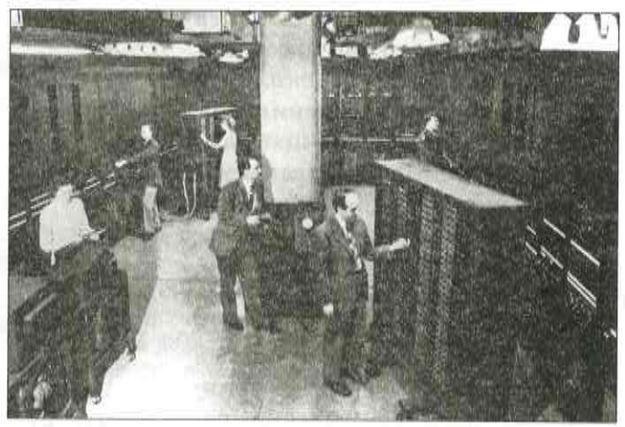


Рис. 5.1. ЭВМ «ЭНИАК» (www.fi.edu)

Опыт построения первой ЭВМ был проанализирован *А. Берксом*, *Г. Голдстайном* и *Дж. фон Нейманом*. В 1946 г. они опубликовали работу «Предварительное рассмотрение логической конструкции электронного вычислительного устройства», ставшую классической. Сформулированные в ней принципы построения вычислительных машин используются и сейчас, несмотря на то, что со времени публикации прошло более полувека. В компьютерной литературе эти принципы часто называют **фон-неймановской архитектурой ЭВМ**, хотя Джон фон Нейман не был её единоличным автором.

Дж. фон Нейман
(1903–1957)

Дальнейший прогресс вычислительной техники во многом определялся развитием её **элементной базы**. Важной вехой на этом пути стало создание в 1947 г. **транзистора** (*У. Шокли*, *Д. Бардин* и *У. Браттейн*). Транзистор — это полупроводниковый прибор для управления электрическими сигналами (рис. 5.2). На основе транзисторов могут собираться цифровые электронные схемы, такие как логические элементы и триггеры.

Рис. 5.2. Транзисторы

В 1958 г. *Дж. Килби* разработал первую **интегральную микросхему** — кристалл, в котором размещается не один транзистор, а целая схема на нескольких транзисторах — например, один или несколько триггеров. Все её вспомогательные радиодетали (резисторы, конденсаторы и другие) также изготавливаются с помощью полупроводниковых технологий.

Первый **микропроцессор** Intel 4004 был разработан под руководством инженера *М. Хоффа* и выпущен в 1971 г. Он был че-

С. Джобс и С. Возняк с компьютером Apple-I (cedmagic.com)

тырёхбитным, поскольку предназначался для микрокалькуляторов и должен был производить вычисления над десятичными числами (см. § 14, где рассказано о двоично-десятичном кодировании). Интересно, что цель его создания была чисто технологическая: заменить 12 специализированных микросхем одной универсальной, но программируемой. Достоинства микропроцессора как главного узла для компьютеров были оценены позднее.

В ходе совершенствования элементной базы вычислительные машины становились все более мощными и компактными. В результате появились персональные компьютеры (ПК), которыми мы сейчас пользуемся. В 1976 г. два молодых приятеля *С. Джобс* и *С. Возняк* в гараже родителей Джобса собрали ПК **Apple**, положивший начало известному ныне семейству компьютеров. А в 1981 г. был продемонстрирован первый компьютер другого семейства — **IBM PC** (IBM Personal Computer), потомки которого в нашей стране особенно широко распространены.

Поколения ЭВМ (совершенствование элементной базы)

Неудачная попытка Бэббиджа построить механическую Аналитическую машину показала, насколько важную роль играет элементная база. Именно поэтому дальнейшую историю вычислительной техники принято делить на периоды в соответствии с теми элементами, из которых изготавливались машины.

Первое поколение ЭВМ относят к периоду примерно 1945–1955 гг. Эти машины были построены на базе **электронных ламп**[1]. Открыл его уже описанный ранее **ЭНИАК**. В нашей стране машинами первого поколения были **МЭСМ** (Малая электронная счётная машина, 1951 г., рис. 5.3), **БЭСМ** (Большая, или Быстродействующая, электронная счётная машина, 1952 г.), **Стрела**

[1] В середине XX века было разработано несколько счётных машин на электромагнитных реле, но их из-за малого количества не принято включать в классификацию поколений.

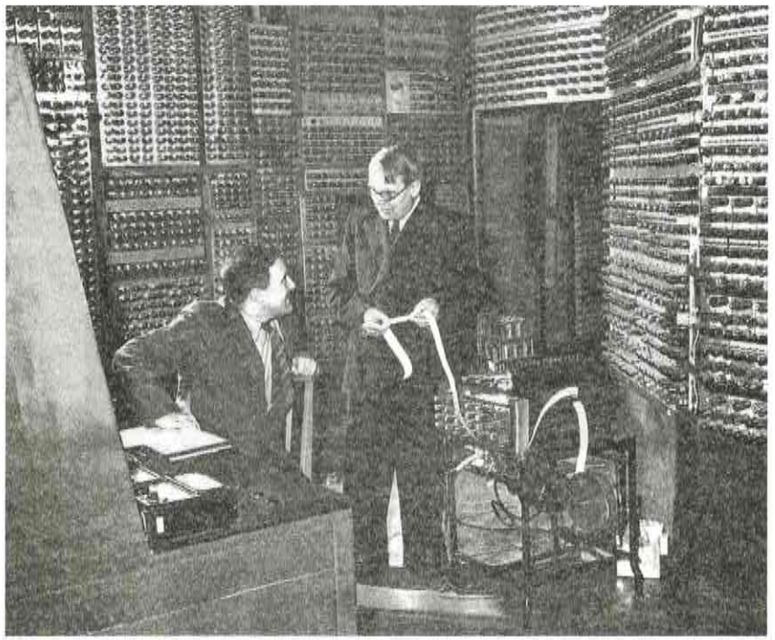


Рис. 5.3. ЭВМ первого поколения МЭСМ (фото из Единой коллекции цифровых образовательных ресурсов)

(1953 г.), **Урал** (1954 г.), **М-20** (1959 г.). Все эти машины были огромными, неудобными и дорогими.

Второе поколение ЭВМ (примерно 1955–1965 гг.) появилось, когда на смену лампам в электронных схемах пришли **транзисторы**. Первый экспериментальный компьютер на транзисторах **ТХ-0** был создан в 1955 г. в Массачусетском технологическом институте (США). ЭВМ на транзисторах были значительно меньше и имели существенно более высокое быстродействие; они потребляли гораздо меньше энергии, были надёжнее и не требовали таких громоздких систем отвода тепла, как ламповые машины. Многие машины второго поколения уже помещались в обычной комнате среднего размера, например, ЭВМ серии **Наири** (1964 г.) или **МИР** (Машина инженерных расчётов, 1965 г.). Наиболее производительными ЭВМ этого поколения стали **Стретч** (США, 1960 г.), **Атлас** (Великобритания, 1961 г.), **CDC 6600** (США, 1964 г.) и **БЭСМ-6** (СССР, 1967 г., рис. 5.4).

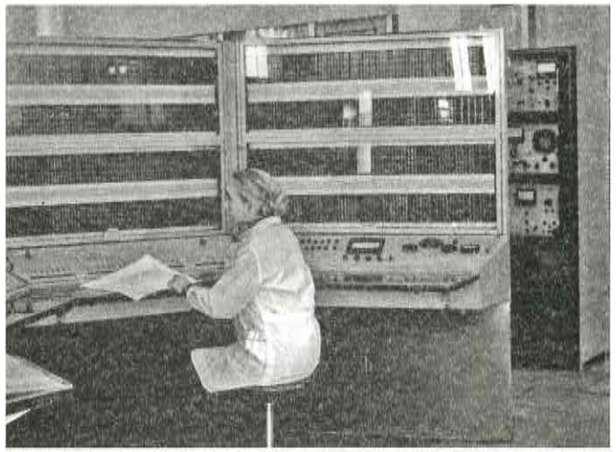


Рис. 5.4. ЭВМ второго поколения БЭСМ-6
(фото из Единой коллекции цифровых образовательных ресурсов)

Третье поколение ЭВМ (примерно 1965–1975 гг.) связано с появлением **интегральных микросхем**. Размеры элементов, из которых строились вычислительные машины, существенно уменьшились (рис. 5.5). Казалось бы, размеры самих этих ЭВМ снова должны были существенно уменьшиться, но этого не произошло. Дело в том, что ЭВМ третьего поколения были предназначены для коллективной (многопользовательской) работы. Это было время крупных вычислительных центров, предоставлявших услуги огромному числу пользователей из многих организаций. Поэтому главное внимание уделялось не уменьшению размеров и стоимости машин, а повышению их вычислительной мощности и эффективности обработки больших объёмов данных.

Отличительная черта третьего поколения — выпуск семейств вычислительных машин, которые были совместимы между собой как аппаратно (все устройства сконструированы по одинаковым стандартам), так и программно (имели одинаковую систему команд). Впервые идею общей **архитектуры**, обеспечивающей выполнение написанных ранее программ на любой новой модели, предложила фирма *IBM*, которая разработала семейства больших ЭВМ **IBM/360** и **IBM/370**. В этот период в СССР было принято решение перейти к копированию зарубежной техники ради обеспечения совместимости. В результате в странах Восточной Европы

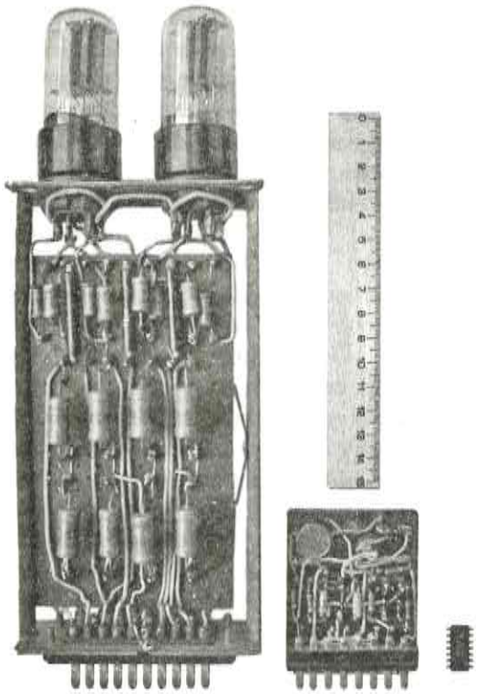


Рис. 5.5. Уменьшение размеров ячеек памяти ЭВМ от первого до третьего поколений (в каждой из них — по два триггера)

были выпущены «аналоги» упомянутых выше семейств под общим названием **ЕС ЭВМ** (Единая система ЭВМ). Одновременно появились мини-ЭВМ семейства **СМ ЭВМ** (Семейство малых ЭВМ), аналогичное известному зарубежному семейству **PDP** фирмы **DEC**.

Четвёртое поколение ЭВМ берёт своё начало примерно с 1975 г. Прогресс в электронике дал возможность существенно увеличить плотности «упаковки» элементов на кристалле, и в одной микросхеме теперь удавалось собрать целый узел, например, микропроцессор. Микросхемы такого уровня стали называть **БИС** (большие интегральные схемы, от 1000 до 10 000 элементов на кристалле), а позднее — **СБИС** (сверхбольшие ИС, более 10 000 элементов). Именно они стали основой четвёртого поколения ЭВМ, которое существует вплоть до настоящего времени.

Увеличение плотности схемы позволило, в первую очередь, повысить быстродействие компьютеров[1]. Кроме того, возросла и надёжность, поскольку значительная часть электрических соединений выполнена внутри кристалла. Однако при высокой плотности монтажа увеличивается теплоотдача от миниатюрных деталей, поэтому требуются специальные меры по отводу тепла (например, установка вентиляторов охлаждения).

Б. Гейтс и П. Аллен

Первый восьмиразрядный процессор **Intel 8080**, предназначенный специально для компьютеров, был выпущен в 1974 г. На его базе был разработан микрокомпьютер **Альтаир**, имевший большой коммерческий успех. Он вошел в историю еще и потому, что в 1975 г. молодой студент *Билл Гейтс* со своим другом *Полом Алленом* реализовали на нём язык программирования **BASIC**. Чуть позднее они создали известную компанию *Microsoft*.

Кроме **персональных компьютеров**, к четвёртому поколению относятся **серверы** — мощные вычислительные машины, которые используются для управления компьютерными сетями. Они предоставляют свои ресурсы (например, принтеры, файлы или программы) в коллективное пользование. Серверы могут эффективно обслуживать большое количество пользователей одновременно. Например, два сервера *Hewlett-Packard T600* (по 12 процессоров в каждом), установленные в системе резервирования билетов *Amadeus*, способны практически без задержек обслуживать примерно 60 миллионов запросов в сутки (система имеет около 180 тысяч терминалов в более чем ста странах мира).

Важное направление в компьютерах четвёртого поколения — **параллельная** (одновременная) **обработка данных**. Если решаемую задачу удается разбить на независимые друг от друга действия, то их необязательно делать друг за другом, а можно для экономии времени выполнять одновременно. Правда, для этого

[1] При быстродействии 10^9 элементарных операций в секунду (типичное по порядку величины значение для современного компьютера) за время каждой из них электрический сигнал со скоростью $3 \cdot 10^8$ м/с успевает пройти путь всего 30 см.

требуется несколько процессоров, но современный уровень техники это позволяет. Более того, в последнее время были сконструированы **многоядерные процессоры**, т. е. фактически несколько процессоров в одном кристалле.

Мощные многопроцессорные компьютеры, в которых выполняется параллельная обработка данных, называют **суперкомпьютерами**. Это уникальные устройства, поэтому они изготавливаются штучно.

В литературе часто упоминаются суперкомпьютеры серии **CRAY**, разработанные под руководством *Сеймура Крэя*. Первая модель этой серии, *CRAY-1*, была построена в США в 1976 г. и имела огромный коммерческий успех.

Все развитые страны ведут жёсткую конкуренцию в области суперкомпьютеров, поскольку обладание такой техникой позволяет решать стратегически важные вычислительные задачи:

- исследование геофизики Земли, прогнозирование изменений климата на планете;
- создание математических моделей молекул (полимеров, кристаллов и т. п.), синтез новых материалов и лекарств;
- расчёт процессов горения и взрыва, а также моделирование других физических задач (обтекание летательных аппаратов, прочность кузовов автомобилей);
- моделирование и прогнозирование ситуации в экономике;
- расчёты процессов нефте- и газодобычи, а также сейсморазведки недр;
- проектирование новых электронных устройств.

Приведём несколько примеров применения суперкомпьютеров. Исследователи фирмы IBM на протяжении десятилетий изучают деятельность мозга и пытаются моделировать её. В 2009 г. появилось сообщение о том, что полученная в рамках проекта модель мозга по своим возможностям превзошла уровень кошки: моделируется 1 миллиард нейронов и 10 триллионов связей между ними! Модель работает на базе суперкомпьютера *Blue Gene/P*, имеющего 147 456 процессоров и 144 Тбайт оперативной памяти.

По данным компании *Ford Motor*, благодаря детальному моделированию на суперкомпьютере, количество разрушаемых в крэш-тестах[1] автомобилей удается сократить на треть.

[1] Крэш-тесты — это тесты, исследующие поведение машин при сильном ударе о бетонное препятствие.

Применение суперкомпьютеров для расчёта состава лекарств позволяет уменьшить срок их разработки с нескольких лет до полугода.

Мощные компьютеры используются при создании компьютерных спецэффектов в кино. Например, для фильма «Властелин колец» фирме *WETA Digital* потребовалось столько суперкомпьютеров, что Новая Зеландия вышла на первое место в мире по их количеству на душу населения.

В 2009 г. в МГУ введён в строй самый мощный **российский суперкомпьютер «Ломоносов»** (рис. 5.6) производительностью около 400 Тфлопс[1]. В его состав входят 8892 многоядерных процессора (общее число ядер — 35 776). На момент запуска «Ломоносов» занимал в мировом рейтинге суперкомпьютеров Top500 двенадцатое место.

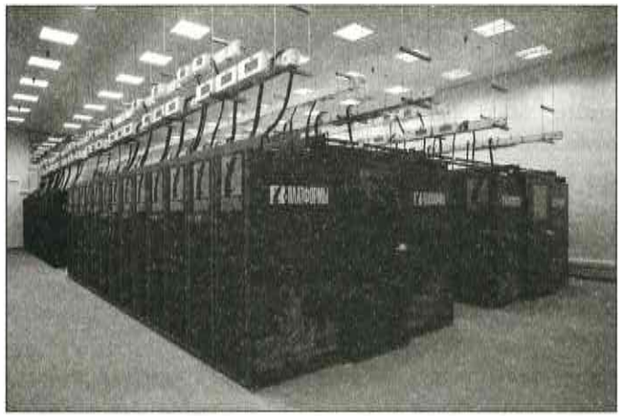


Рис. 5.6. Суперкомпьютер Ломоносов[2]

Много шума наделал *японский проект по созданию компьютеров пятого поколения* (1982–1992 гг.). Было заявлено, что в основе компьютеров пятого поколения будут уже не вычисления,

[1] Флопс (англ. *FLOPS — floating point operations per second*) — единица измерения количества операций с вещественными числами за 1 секунду; приставка «тера» добавляется по тем же правилам, что и при измерении информации; очевидно, что операции над вещественными числами намного сложнее и выполняются гораздо дольше, чем над целыми.

[2] Фотография предоставлена компанией «Т-Платформы» (www.t-platforms.ru).

а логические заключения, т. е. произойдёт переход от обработки данных к обработке знаний. Машину обещали научить воспринимать речевые команды человека, читать рукописный текст, анализировать графические изображения и делать многие другие нетривиальные для компьютера вещи. Планы проекта были грандиозны. Но, несмотря на щедрое финансирование и передовые позиции японских технологий, успехи оказались весьма скромными. На основе программного моделирования на компьютерах четвёртого поколения удалось реализовать лишь отдельные детали проекта, причём реальная машина, работающая на базе логических выводов, так и не вышла за стены лабораторий.

Таким образом, создание принципиально новых компьютеров пятого поколения закончилось неудачей. Все компьютеры, используемые в настоящее время, по-прежнему построены на базе идей четвёртого поколения. Классификация поколений «замерла» в ожидании новых революционных идей. Такие идеи особенно необходимы ещё и потому, что электронная техника уже подошла к пределу быстродействия, который определяется законами физики: для увеличения скорости передачи данных требуется уменьшать размеры электронных деталей, но плотность упаковки транзисторов в полупроводниковом кристалле и так уже практически достигла максимально возможной. Поэтому *идёт поиск неэлектронных средств хранения и обработки данных*.

В первую очередь, ученые попытались использовать в качестве носителя информации свет — так появились **оптические процессоры**. В них можно применять параллельную обработку данных, например, одновременно выполнять какую-то операцию со всеми пикселями изображения. В 2003 г. был выпущен оптический процессор *Enlight256*, который имеет оптическое ядро, а входные и выходные данные представлены в электронном виде. Быстродействие этого процессора — 8 триллионов операций в секунду. Он состоит из 256 лазеров, набора линз и фотоприёмников. Оптические процессоры используются в военной технике и при обработке видеоданных в реальном времени.

Большие надежды связаны с **разработкой квантовых компьютеров**, в которых применяются идеи квантовой физики, описывающей законы микромира и поведение отдельных элементарных частиц. Данные для обработки в квантовом компьютере записываются в систему *кубитов* — квантовых битов. Затем с помощью специальных операций состояние этой системы изменяют по определённому алгоритму. Конечное состояние системы куби-

тов — это и есть ответ в задаче. Особые свойства кубитов[1] позволяют организовать параллельную обработку данных, так же, как и в многопроцессорных системах. Поэтому многие задачи, для решения которых сейчас не хватает вычислительных ресурсов (например, раскрытие шифров), будут достаточно быстро решены, как только квантовый компьютер будет построен.

В некоторых лабораториях ведется **разработка биологических компьютеров** (биокомпьютеров), которые работают как живой организм. Ячейки памяти биокомпьютеров — это молекулы сложных органических соединений, например, молекулы ДНК, в которых хранится наследственная информация. Сам процесс вычислений — это химическая реакция, результат — состав и строение получившей молекулы.

Проводятся также исследования и в области **нанотехнологий**, с помощью которых планируют построить транзистор размером с молекулу.

Развитие возможностей от поколения к поколению

С каждым поколением вычислительных машин развивались их **аппаратные возможности**. ЭВМ становились более мощными и универсальными. Расширялось количество обрабатываемых **типов данных**:

первое поколение — только числовые данные;
второе поколение — числа и символы;
третье поколение — числа, текстовые и графические данные;
четвёртое поколение — добавились аудио- и видеоданные.

Появившись как устройство для облегчения вычислений, компьютер сейчас всё более активно обрабатывает разнообразную нечисловую информацию. Чтобы подчеркнуть широкие возможности современных компьютеров, введён специальный термин «мультимедиа».

Мультимедиа (от латинских слов *multum* — много и *medium* — средства) — одновременное использование различных форм представления информации (графика, текст, видео, фотографии, анимация, звук и т. д.) и их объединение в одном объекте.

[1] В отличие от привычного нам бита, кубит устроен так, что способен вместить в себя гораздо больше информации.

Термин «мультимедиа» также используется в словосочетаниях «мультимедиа компьютер», «мультимедиа носитель», «программные и аппаратные средства мультимедиа». Пример мультимедиа объекта — компьютерная презентация.

Как правило, при использовании технологий мультимедиа человек может влиять на показ материалов: перейти вперёд или вернуться назад, изменить настройки, выбрать один из предложенных вариантов и т. п. Подобное взаимодействия человека и компьютера называют **интерактивностью** (взаимной активностью).

Другое направление в развитии аппаратной части — это увеличение разнообразия и одновременно рост сложности **внешних устройств**, присоединяемых к ЭВМ:

первое поколение — штекеры и переключатели, индикаторные лампочки, устройства ввода с перфокарт;

второе поколение — перфоленты, магнитные ленты и барабаны, печатающие устройства;

третье поколение — магнитные диски, текстовые и графические мониторы, графопостроители;

четвёртое поколение — огромное разнообразие внешних устройств, в том числе:

- устройства для хранения данных на оптических дисках;
- мышь, джойстик, шлемы виртуальной реальности и др.;
- возможность подключения бытовой электроники (фотоаппаратов, музыкальных плееров, сотовых телефонов и др.) с помощью кабелей и беспроводных соединений.

Каждое новое поколение компьютеров расширяет возможности **программного обеспечения (ПО)**. Использовать современный компьютер без ПО практически невозможно. Стоимость ПО нередко намного превышает стоимость аппаратной части (в первых поколениях было наоборот!).

Первое поколение. Программы разрабатывали хорошо подготовленные специалисты на машинном языке, сама программа представляла собой последовательность чисел (машинных кодов). Стандартного программного обеспечения практически не было.

Второе поколение. Появились первые языки программирования. Некоторые из них были разработаны для конкретных машин, но значительно удобнее оказались машинно-независимые языки, такие как **Фортран** (1957) и **Алгол** (1960). Написать программу на таком языке было значительно проще: с этим уже

вполне мог справиться рядовой научный работник, причем не обязательно с математическим образованием. В конце второго поколения появились **специальные программы**, управляющие последовательным прохождением заданий через ЭВМ в автоматическом режиме (они назывались **мониторами**). Их дальнейшее развитие привело к появлению операционных систем.

Третье поколение. Созданы **операционные системы** (ОС), которые обеспечивали работу компьютеров в многопользовательском режиме и управляли большим количеством сложных внешних устройств (в первую очередь, магнитными дисками). Для «общения» с ОС разработаны специальные языки управления заданиями. Широкое распространение получили созданные ранее языки программирования, например, **Фортран** для математических вычислений и **Кобол** для экономических расчётов. Начали появляться пакеты прикладных программ для решения задач в конкретных областях.

Четвёртое поколение. Для управления компьютером пользователь теперь использует не язык программирования, а различные меню и кнопки. Например, необходимую команду можно выбрать из меню (перечня доступных в данной ситуации возможностей) на естественном языке. Стало реально освоить компьютер после очень короткой подготовки. Программное обеспечение для ПК становится необычайно разнообразным — написано столько программ, что их трудно даже просто систематизировать. Казалось бы, такое разнообразие ПО должно сделать программирование ненужным, но нередко проще написать собственную небольшую программу решения конкретной задачи, чем тратить время на поиск и освоение возможностей готовых универсальных пакетов.

Таким образом, при переходе от поколения к поколению возрастает вычислительная мощность компьютеров. Значительная часть новых возможностей направляется на повышение удобства работы пользователя. В человеко-машинном общении отчетливо прослеживается движение от машинного языка к языкам, естественным для человека. В результате расширяется область применения и круг пользователей компьютерной техники.

Вопросы и задания

1. Что такое компьютер?
2. Охарактеризуйте программную и аппаратную части компьютера.

3. Почему универсальный компьютер с изменяемой программой удобнее, чем специализированная техника? Ответ обоснуйте.
4. Что такое цифровая и аналоговая техника?
5. Почему цифровая техника вытеснила аналоговую?
6. Перечислите основные вехи в истории развития вычислительной техники.
7. Какова заслуга Чарльза Бэббиджа?
8. В честь кого названы языки программирования Ада и Паскаль? Какое отношение эти люди имеют к вычислительной технике?
9. Что такое транзистор и микросхема? Из чего они изготавливаются?
10. С какой целью разрабатывались первые микропроцессоры?
*11. Почему микропроцессор Intel 4004 был специально спроектирован для работы только с четырехбитными данными? Указание: вспомните, как можно хранить отдельные десятичные цифры числа.
12. По какому принципу ЭВМ делятся на поколения?
13. Почему время существования того или иного поколения всегда указывается приблизительно?
14. Перечислите все поколения ЭВМ и назовите элементную базу каждого из них.
15. Что даёт уменьшение базовых элементов вычислительной техники?
16. Почему электронные схемы требуют охлаждения? Все ли элементы нуждаются в дополнительном охлаждении?
17. Какие поколения вычислительных машин построены на базе полупроводниковых технологий? Чем отличается друг от друга их элементная база?
18. Объясните, почему большинство ЭВМ третьего поколения имели крупные габариты, несмотря на очередное уменьшение размеров элементной базы.
19. Когда появились первые семейства ЭВМ? Какая фирма предложила идею? В чем преимущества выпуска совместимых моделей?
20. Компьютеры какого поколения сейчас стоят на полках магазинов?
21. Какие разновидности компьютеров входят в четвёртое поколение?
22. Как вы понимаете термин «персональный компьютер»?
23. Какие семейства персональных компьютеров вы знаете? Какое из них появилось раньше?
24. Перечислите бытовые приборы, в которых применяются микропроцессоры.
25. Что такое суперкомпьютеры? Зачем они используются?
26. Найдите в Интернете рейтинг суперкомпьютеров Top500. Какие страны занимают в нём лидирующее положение? Есть ли там российские компьютеры?
*27. Зачем в суперкомпьютерах так много процессоров? Подумайте, любая ли задача может быть решена быстрее, если её считать параллельно на множестве процессоров? (В качестве помощи можно вос-

пользоваться аналогией с распределением частей одного большого задания между учениками класса.)

28. Назовите примеры вычислительных машин каждого из четырёх поколений. Найдите в Интернете дополнительный материал об этих машинах.
29. Что вы можете сказать о судьбе пятого поколения компьютеров?
*30. Почему, по-вашему, уже довольно давно не происходило смены поколений?
31. Данные каких типов обрабатывались на ЭВМ каждого из поколений?
32. Как изменялся набор внешних устройств при переходе от одного поколения к другому?
33. Опишите, как происходило развитие программного обеспечения.
34. Что вы можете сказать по поводу роли программного обеспечения: уменьшается она или увеличивается по сравнению с предыдущими поколениями?
35. Предположим, что появился процессор с каким-то принципиально новым свойством. Как быстро этим свойством смогут воспользоваться потребители? Какова роль программного обеспечения в этом?
36. Быстродействие вычислительной техники постоянно растёт. Как же тогда объяснить, что пользователи жалуются на «медлительные» компьютеры и все время стараются купить новые, еще более производительные?
*37. Влияет ли развитие программных средств на развитие аппаратной части?
38. Что представляли собой программы для первых машин? Почему для их записи было удобно использовать не двоичную систему счисления, а восьмеричную или шестнадцатеричную?
39. Зачем были созданы языки программирования? Когда они появились?
40. Попробуйте назвать положительные и отрицательные последствия огромного разнообразия существующих программ.
41. Почему развитие ПО расширяет количество пользователей компьютера?
42. Когда появились операционные системы и с чем это связано?
*43. Насколько сейчас, по-вашему, актуально умение программировать? Попробуйте найти аргументы «за» и «против» (учитывайте разные цели работы на компьютере у людей).

Подготовьте сообщение

а) «Что такое микропроцессор?»
б) «Физические пределы быстродействия компьютеров»
в) «Много программ — это хорошо или плохо?»
г) «Зачем нужно программировать?»

д) «Первые ЭВМ»
е) «Поколения ЭВМ»
ж) «Программное обеспечение и поколения ЭВМ»
з) «Разработка компьютеров будущего»
и) «Квантовые компьютеры»
к) «Суперкомпьютеры»

§ 32
Принципы устройства компьютера

В предыдущем параграфе вы увидели, что вычислительная техника в своем развитии прошла целый ряд характерных этапов. Несмотря на это некоторые фундаментальные (базовые, основные) принципы устройства ЭВМ почти не изменились. Поэтому логично начать знакомство с устройством компьютера именно с них.

Классические принципы построения ЭВМ были предложены в работе *А. Беркса, Г. Голдстайна и Дж. фон Неймана «Предварительное рассмотрение логической конструкции электронного вычислительного устройства»*. Обычно выделяют[1] следующие наиболее важные идеи этой работы:

- состав основных компонентов вычислительной машины;
- принцип двоичного кодирования;
- принцип адресности памяти;
- принцип иерархической организации памяти;
- принцип хранимой программы;
- принцип программного управления.

Рассмотрим их подробнее.

Общие принципы

Основные компоненты машины. В самом первом разделе с таким названием фон Нейман с соавторами определили и обосновали состав ЭВМ:

«Так как законченное устройство будет универсальной вычислительной машиной, оно должно содержать несколько основных органов, таких как орган арифметики, памяти, управле-

[1] Эта техническая статья не содержит отдельного пронумерованного перечня принципов, поэтому в учебной литературе встречаются непринципиальные отличия в их формулировке и описании.

ния и связи с оператором. Мы хотим, чтобы машина была полностью автоматической, т. е. после начала вычислений работа машины не зависела от оператора».

Таким образом, ЭВМ должна состоять из нескольких блоков, каждый из которых выполняет вполне определённую функцию. Эти блоки есть и в сегодняшних компьютерах:

- **арифметико-логическое устройство** (АЛУ), в котором выполняется обработка данных;
- **устройство управления** (УУ), обеспечивающее выполнение программы и организующее согласованное взаимодействие всех узлов машины; сейчас АЛУ и УУ изготавливают в виде единой интегральной схемы — микропроцессора;
- **память** — устройство для хранения программ и данных; память обычно делится на **внутреннюю** (для временного хранения данных во время обработки) и **внешнюю** (для длительного хранения между сеансами обработки);
- **устройства ввода**, преобразующие входные данные в форму, доступную компьютеру;
- **устройства вывода**, преобразующие результаты работы ЭВМ в форму, удобную для восприятия человеком.

В классическом варианте все эти устройства взаимодействовали через процессор (рис. 5.7).

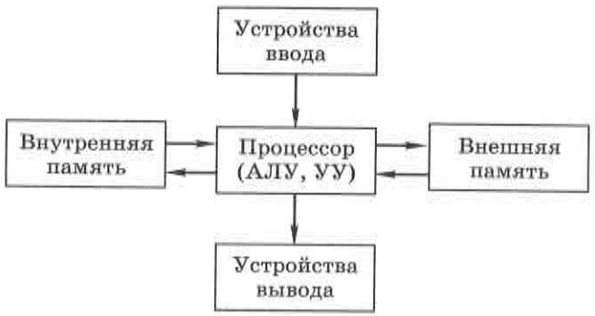


Рис. 5.7

Принцип двоичного кодирования. Устройства для хранения двоичной информации и методы её обработки наиболее просты и дешевы. Поскольку в ЭВМ используется двоичная система счисления, необходимо переводить данные из десятичной формы

в двоичную (при вводе) и наоборот (при выводе результатов). Однако такой перевод легко автоматизируется, и многие пользователи даже не знают об этих внутренних преобразованиях.

В первых машинах использовались только числовые данные. В дальнейшем ЭВМ стали обрабатывать и другие виды информации (текст, графика, звук, видео), но это не привело к отмене принципа двоичного кодирования. Даже цифровые сигнальные процессоры[1], предназначенные для обработки цифровых сигналов в реальном времени, используют двоичное представление данных.

В истории известен пример успешной реализации *троичной ЭВМ «Сетунь»* (1959 г., руководитель проекта *Н. П. Брусенцов*), но он так и остался оригинальным эпизодом и не оказал влияния на эволюцию вычислительной техники. В первую очередь, это связано с серьёзными проблемами, которые возникают при изготовлении элементов троичного компьютера на основе полупроводниковых технологий. Эти проблемы так и не были решены, тогда как наладить массовое производство аналогичных устройств для двоичных компьютеров оказалось значительно проще.

Принципы организации памяти

Принцип адресности памяти. Оперативная память машины состоит из отдельных **битов**. Для записи или считывания группы соседних битов объединяются в **ячейки памяти**, каждая из которых имеет свой адрес (номер). Нумерацию ячеек принято начинать с нуля.

Адрес ячейки памяти — это её номер.

Ячейка содержит минимально возможный считываемый из памяти объём данных: невозможно прочитать меньшее количество битов, а тем более отдельный бит.

Использование чисел для нахождения в памяти требуемых ячеек выглядит абсолютно естественно: в компьютерах любая информация кодируется числами, так что адреса ячеек — не исключение из этого фундаментального правила. Если номера соседних

[1] В англоязычной литературе их называют **DSP** — *Digital Signal Processor*.

ячеек отличаются на единицу, удобно организовывать их последовательную обработку.

Разрядность ячеек памяти (количество битов в ячейке) в разных поколениях была различной. Первоначально ЭВМ были построены исключительно для математических расчётов. Числа желательно было представлять как можно точнее, поэтому ячейки ОЗУ в первых машинах были длинными. Кроме чисел, машина должна была хранить в памяти еще и команды программы; как правило, в то время размер числовой ячейки совпадал с размером команды, что существенно упрощало устройство памяти.

Примерно на стыке второго и третьего поколений ЭВМ стали использовать для обработки символьной информации, что привело к серьёзному неудобству: в существующую числовую ячейку памяти помещалось 4–5 символов. Инженеры выбрали наиболее простое решение проблемы — уменьшить размер ячейки так, чтобы можно было обращаться к каждому символу отдельно. **Байтовая память**, основой которой стала **восьмибитная ячейка**, прекрасно зарекомендовала себя и используется в компьютерной технике до настоящего времени.

В результате перехода к «коротким» ячейкам памяти числа стали занимать несколько ячеек (байтов), каждая из которых имеет собственный адрес. На рисунке 5.8, *а* показана организация ячеек памяти первых ЭВМ, а на рис. 5.8, *б* — современная (байтовая) структура памяти.

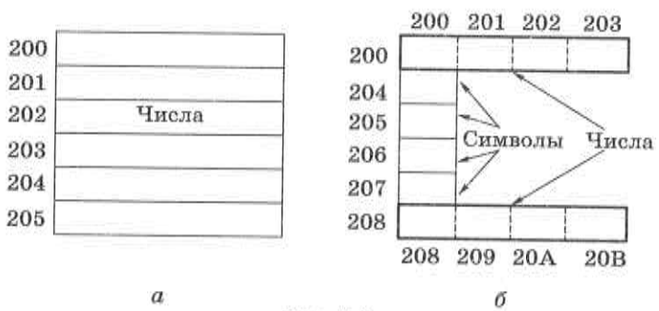


Рис. 5.8

На рисунке 5.8, *а* числа занимают по одной ячейке, причём номера этих ячеек отличаются на единицу. Справа показаны два 32-битных числа, которые хранятся в байтах 200–203 и 208–20B (адреса указаны в шестнадцатеричной системе). По принятому правилу за адрес числа принимается наименьший из адресов, так

что в данном случае адреса чисел — 200 и 208. Кроме того, на рис. 5.8, *б* между числами (в байтах с 204 по 207) размещены четыре однобайтных символа. Заметим, что современные компьютеры могут извлекать из памяти до восьми соседних байтовых ячеек за одно обращение к памяти.

Очень важно, что информация может считываться из ячеек и записываться в них в произвольном порядке, поэтому организованную таким образом память принято называть **памятью с произвольным доступом** (англ. **RAM** — *random access memory*). Чтобы лучше понять смысл этого термина, сравните такую память с магнитной лентой, данные с которой можно получить только путем последовательного чтения.

Часто термин RAM отождествляют с русским термином **ОЗУ — оперативное запоминающее устройство**. Это не совсем точно. Дело в том, что кроме ОЗУ существует еще одна разновидность памяти с произвольным доступом — **постоянное запоминающее устройство (ПЗУ**, англ. **ROM** — *Read Only Memory* — память только для чтения). Главное отличие ПЗУ от ОЗУ заключается в том, что при решении задач пользователя содержимое ПЗУ не может быть изменено. ПЗУ гораздо меньше ОЗУ по объёму, но это очень важная часть компьютера, поскольку в нём хранится доступное в любой момент программное обеспечение. Благодаря этому ПО компьютер сохраняет работоспособность даже тогда, когда в ОЗУ нет никакой программы.

Таким образом, ОЗУ и ПЗУ — это два вида памяти с произвольным доступом, обращение к данным в которых построено на основе принципа адресности.

Принцип иерархической организации памяти. К памяти компьютера предъявляются два противоречивых требования: её объём должен быть как можно больше, а скорость работы — как можно выше. Ни одно реальное устройство не может удовлетворить им одновременно. Любое существенное увеличение объёма памяти неизбежно приводит к уменьшению скорости её работы. Действительно, если память большая, то обязательно усложняется поиск в ней требуемых данных[1], а это сразу замедляет чтение

[1] Например, память большого объёма требует многоразрядного адреса, что, в свою очередь, приводит к очень большому количеству линий связи. В итоге приходится как-то изменять способ адресации, например, передавать адрес по частям.

из памяти. Кроме того, чем быстрее работает память, тем она дороже, и, следовательно, меньше памяти можно установить за приемлемую для потребителей стоимость.

Чтобы преодолеть противоречие между объёмом памяти и её быстродействием, используют несколько различных видов памяти, связанных друг с другом. Когда в 1946 г. впервые формулировался этот принцип, в состав ЭВМ предполагалось включить всего два вида памяти: оперативную память и память на магнитной проволоке (предшественник устройств хранения данных на магнитной ленте). Дальнейшее развитие вычислительной техники подтвердило необходимость построения иерархической памяти: в современном компьютере уровней иерархии гораздо больше.

Выполнение программы

Принцип хранимой программы. Первые ЭВМ программировались путём установки перемычек на специальных панелях (рис. 5.9), так что процесс подготовки к решению задачи мог растянуться на несколько дней. Такое положение дел никого не устраивало, и в фон-неймановской архитектуре было предложено представлять команды в виде двоичного кода. Код программы, записанный заранее[1] на перфокарты или магнитную ленту, можно было ввести в машину достаточно быстро.

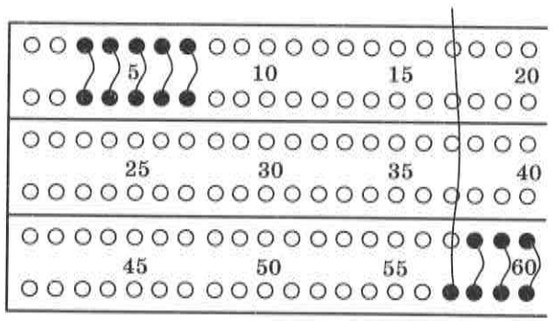


Рис. 5.9. Фрагмент коммутационной панели устройства IBM-557; требуемая операция получается соединением (коммутацией) соответствующих отверстий

[1] До появления персональных компьютеров для этого использовались специальные *устройства подготовки данных*. Такая схема ускоряла процесс ввода и исключала простои ЭВМ, связанные с длительным набором программ.

Поскольку команды программы и данные по форме представления стали одинаковыми, их можно хранить *в единой памяти*[1] вместе с данными. Не существует принципиальной разницы между двоичными кодами машинной команды, числа, символа и т. д. Это утверждение иногда называют **принципом однородности памяти**. Из него следует, что команды одной программы могут быть получены как результат работы другой программы. Именно так текст программы на языке высокого уровня переводится (транслируется) в машинные коды конкретной машины.

Код программы может сохраняться во внешней памяти (например, на дисках) и затем загружаться в оперативную память для повторных вычислений. Благодаря простоте замены программ, ЭВМ стали **универсальными устройствами**, способными решать самые разнообразные задачи в произвольном порядке и даже одновременно.

Принцип программного управления. Любая обработка данных в вычислительной машине происходит по программе. Принцип программного управления определяет наиболее общий механизм автоматического выполнения программы.

Важным элементом устройства управления в машине фон-неймановской архитектуры является специальный регистр — **счётчик адреса команд**[2]. В нём в любой момент хранится адрес команды программы, которая будет выполнена следующей.

Используя значение из счётчика, процессор считывает из памяти очередную команду программы, расшифровывает её и выполняет. Затем те же действия повторяются для следующей команды и т. д. Процессор выполняет команды по следующему алгоритму (его часто называют **основным алгоритмом работы процессора**):

1) из ячейки памяти, адрес которой записан в счётчике адреса команд, выбирается очередная команда программы; на время выполнения она сохраняется в специальном **регистре команд**;

[1] Известна также так называемая **гарвардская архитектура**, в которой программы и данные хранятся в разных областях памяти. Несмотря на повышение надёжности, она не получила широкого распространения.

[2] В различных процессорах этот регистр может называться по-разному: например, в семействе Intel он обозначается *IP* — Instruction Pointer (указатель на инструкцию).

2) значение счётчика адреса команд увеличивается так, чтобы он указывал на следующую команду;
3) выбранная команда выполняется (например, при сложении двух чисел оба слагаемых считываются в АЛУ, складываются и результат операции сохраняется в регистре или ячейке памяти);
4) далее весь цикл повторяется сначала.

Таким образом, автоматически выполняя одну команду программы за другой, компьютер может исполнить любой линейный алгоритм. Для того чтобы в программе можно было использовать ветвления и циклы, необходимо нарушить естественную последовательность выполнения команд. Для этого существуют специальные **команды перехода**, которые на этапе 3 заносят в счётчик адреса новое значение — адрес перехода. Чаще всего в программах используется **условный переход**, т. е. переход происходит только при выполнении определенного условия.

Легко понять, что для запуска основного алгоритма работы процессора в счётчик адреса команд должно быть предварительно занесено начальное значение — адрес первой выполняемой команды. В первых ЭВМ оператор вводил этот адрес вручную. В современных компьютерах при включении питания в счётчик аппаратно заносится некоторое значение, которое указывает на начало программы, хранящейся в ПЗУ. Эта программа тестирует устройства компьютера и приводит их в рабочее состояние, а затем загружает в ОЗУ **начальный загрузчик операционной системы** (как правило, с диска). Ему и передаётся дальнейшее управление, а стартовая программа из ПЗУ завершает свою работу. Начиная с этого момента, поведение компьютера уже определяется установленным на нём программным обеспечением (см. главу 6).

Чтобы ускорить выполнение программы, основной алгоритм работы процессора был значительно усовершенствован. Идея была заимствована из **конвейерного** производства, где несколько рабочих одновременно выполняют различные операции (каждый над своим экземпляром изделия). Аналогично в современных микропроцессорах для каждого этапа выполнения команды создан отдельный аппаратный блок. Выполнив свою операцию, он передает результаты следующему блоку, а сам начинает выполнять очередную команду.

Проще всего понять этот механизм на примере первого этапа — выборки команды из ОЗУ. Специализированный блок

выборки извлекает из памяти последовательно расположенные команды, не дожидаясь окончания их обработки. Прочитанные команды размещаются в специальной рабочей памяти внутри микропроцессора. В итоге, когда первая из выбранных команд будет завершена, за следующей не придется обращаться к ОЗУ, так как она уже находится во внутренней памяти микропроцессора. Учитывая, что обращение к ОЗУ занимает значительно большее времени, чем пересылка данных внутри процессора, такая *опережающая выборка* значительно ускоряет выполнение программы.

На практике применение конвейерного метода не так просто. Например, следующую команду часто не удается выполнить, поскольку она использует результат предыдущей, или сразу нескольким командам потребуется одновременно обратиться к ОЗУ. Тем не менее этот метод широко применяется в микропроцессорах. В некоторых моделях используются **параллельные конвейеры**, так что во многих случаях к моменту завершения выполнения одной команды уже готов результат следующей.

Что называют архитектурой

Описанные фон Нейманом и его соавторами классические принципы построения вычислительных устройств применялись во всех поколениях ЭВМ. В дополнение к ним в каждом конкретном семействе (PDP, ЕС ЭВМ, Apple, IBM PC и др.) формулируются свои собственные принципы устройства, благодаря которым обеспечивается аппаратная и программная совместимость моделей. Для пользователей это означает, что все существующие программы будут работать и на новых моделях того же семейства компьютеров. В литературе общие принципы построения конкретного семейства компьютеров называют **архитектурой**. К архитектуре обычно относят:

- принципы построения системы команд и их кодирования;
- форматы данных и особенности их машинного представления;
- алгоритм выполнения команд программы;
- способы доступа к памяти и внешним устройствам;
- возможности изменения конфигурации оборудования.

Стоит обратить внимание на то, что архитектура описывает именно общее устройство вычислительной машины, а не особенности изготовления конкретного компьютера (набор микросхем, тип жёсткого диска, ёмкость памяти, тактовая частота). Напри-

мер, наличие видеокарты как устройства для организации вывода информации на монитор входит в круг вопросов архитектуры. А вот является ли видеокарта частью основной платы компьютера или устанавливается на неё в виде отдельной платы, с точки зрения архитектуры значения не имеет. Иначе могло бы получиться, что для интегрированной в плату видеокарты потребовалась бы отдельная версия графического редактора!

Вопросы и задания

1. Найдите материалы, подтверждающие, что Джон фон Нейман не был единоличным автором «фон-неймановской» архитектуры ЭВМ.
2. Перечислите принципы фон-неймановской архитектуры и кратко объясните каждый из них.
3. Назовите основные компоненты вычислительного устройства. Каково их назначение? Согласны ли вы с тем, что полученный набор узлов логичен и обоснован?
4. В чём состоит принцип двоичного кодирования?
5. Вспомните, как кодируются в компьютере числа, тексты, графика. Соблюдается ли принцип двоичного кодирования?
6. По какому алгоритму вводимые в компьютер десятичные числа можно перевести во внутреннее двоичное представление? Как перевести обратно результаты расчёта?
7. Что такое ячейка памяти? Что такое адрес ячейки?
8. Что вы знаете о разрядности ячеек ОЗУ разных поколений?
9. Почему появилась байтовая память?
10. Можно ли заменить в ячейке памяти содержимое одного бита, не затрагивая значений соседних? Почему?
11. Приведите примеры различных типов данных и назовите их разрядность. Сколько байтов памяти потребуется для хранения данных каждого из этих типов?
12. Что такое иерархическая организация памяти?
13. Почему большая по объёму память обычно работает медленнее, чем маленькая?
14. В чём состоит принцип хранимой программы?
15. Где может храниться программа?
16. Можно ли к нечисловым данным (символам, графическим и звуковым данным) применять арифметические операции?
17. Как вы понимаете фразу «любая обработка данных в вычислительной машине происходит по программе»? Чем компьютер в этом отношении отличается от простого калькулятора?
18. Сформулируйте основной алгоритм выполнения команды в компьютере.

19. Что такое счётчик адреса команд и какова его роль в основном алгоритме?
20. Опишите, что происходит в момент включения компьютера с точки зрения принципа программного управления.
21. Можно ли нарушить последовательность выполнения команд в программе? Для чего это может потребоваться?
22. Всегда ли в новом компьютере есть какая-либо программа?
23. Что такое конвейер и как он работает при выполнении программы?
24. Почему команды перехода нарушают работу конвейера?
25. Какие из принципов, предложенных в работе «Предварительное рассмотрение логической конструкции электронного вычислительного устройства», продолжают применяться в современных компьютерах безо всяких изменений, а какие сохранились, но в несколько изменённом виде? Объясните, почему потребовались эти изменения.
26. Что такое архитектура? Какие детали устройства компьютера к ней не относятся?
27. В чем преимущества единой архитектуры семейств ЭВМ для пользователей и для производителей?
28. Какие семейства вычислительных машин вы знаете?

Подготовьте сообщение

а) «Джон фон Нейман и его вклад в науку»
б) «Троичная ЭВМ "Сетунь"»
в) «Гарвардская архитектура»
г) «Архитектуры современных компьютеров»

Задачи

1. Используя условные команды «считать байт из памяти», «записать байт в память», а также битовые логические операции «И», «ИЛИ» и «НЕ», составьте алгоритм который в байте, хранящемся в памяти:
   а) установит младший бит данных в единицу, не затрагивая содержимого остальных двоичных разрядов;
   б) сбросит его в ноль.
2. Описанная в «Предварительном рассмотрении...» конструкция ЭВМ имела ячейки памяти, состоящие из 40 двоичных разрядов. Сколько современных байтовых ячеек потребуется для хранения числа такой же разрядности?
3. В языке Паскаль есть тип данных word, значением которого являются целые положительные 16-разрядные числа. Сколько байтов занимает такое число в памяти? Какое максимальное значение может иметь этот тип чисел?

4. В микропроцессорах семейства Intel для увеличения на единицу одного из регистров процессора используется команда, имеющая код 41_{16}. Пользуясь таблицей символов, определите, какая буква соответствует этому же самому коду. Если рассматривать этот код как целое число, чему оно будет равно в десятичной системе счисления?

*5. Изучите содержимое текстового файла, используя программу, которая способна отображать данные в виде шестнадцатеричных кодов. Попробуйте найти коды известных вам символов. Сделайте то же самое с графическим файлом: удастся ли вам найти там те же самые коды («символы»)?

*6. Найдите таблицу кодов команд процессора Intel. Сравните эти коды с кодами, которые вы нашли в предыдущем задании. Удалось ли найти совпадения?

§ 33 Магистрально-модульная организация компьютера

Что понимается под устройством компьютера?

Компьютер — это пример очень сложной техники. При изучении таких систем возможно несколько разных подходов. Например, можно изучать:

- устройство конкретного экземпляра компьютера: набор микросхем, тип основной платы, конструкцию и разновидности модулей памяти и т. п.;
- семейство компьютеров, например, IBM-совместимых персональных компьютеров;
- различные конструкции компьютеров (ноутбуки, планшетные компьютеры, смартфоны);
- функциональное устройство компьютера, т. е. его основные узлы и способы взаимодействия между ними.

Каждый из этих подходов полезен при решении определённых задач. Так для настройки конкретного компьютера необходимо точно знать марки и параметры его устройств. Определить эти

данные можно с помощью специального программного обеспечения. К сожалению, любые знания в этой области очень быстро устаревают, поскольку аппаратура постоянно меняется.

Если изучать особенности одного семейства компьютеров, мы получим «однобокое» представление об устройстве компьютерной техники, так как каждое семейство имеет свои особенности.

Современные компьютеры очень разнообразны и поэтому имеют самую различную конструкцию и внешний вид. **Настольный ПК** состоит из системного блока и подключённых к нему внешних устройств. Такая конструкция удобна для пользователя, поскольку все устройства можно разместить на столе так, как ему хочется.

В **переносных компьютерах** весь минимально необходимый набор устройств собран в одном корпусе. Сейчас такие компьютеры называют **ноутбуками** (англ. *notebook* — тетрадь, блокнот) или ультрабуками. По своим вычислительным возможностям они практически не уступают настольным ПК.

Стремительно растёт популярность **планшетных компьютеров**, в которых для ввода данных нажимают пластиковой палочкой (она называется **стилус**) или пальцем на **сенсорный** (реагирующий на прикосновение) экран.

Вместе с тем мощные серверы и суперкомпьютеры по-прежнему собираются в виде крупных «шкафов», напоминающих ЭВМ предыдущих поколений. Наконец, нельзя не упомянуть и о бытовой электронике, которая все больше и больше приближается к традиционным компьютерам.

Разнообразие типов современных компьютеров говорит о том, что конструкция — это не самое главное. В то же время, как показано в § 32, их функциональное устройство практически не изменяется. Поэтому далее мы подробно рассмотрим основные узлы компьютера (процессор, память и устройства ввода и вывода) и взаимодействие между ними.

Взаимодействие устройств

Процессор должен обмениваться данными с внутренней памятью и устройствами ввода и вывода. Выделить отдельные каналы для связи процессора с каждым из многочисленных устройств нереально. Вместо этого сделана общая линия связи, доступ к которой имеют все устройства, использующие её по очереди. Такой информационный канал называется шиной.

Шина (или **магистраль**) — это группа линий связи для обмена данными между несколькими устройствами компьютера.

Традиционно шина делится на три части (рис. 5.10) — это:

- **шина данных**, по которой передаются данные;
- **шина адреса**, определяющая, куда именно передаётся информация;
- **шина управления**, которая организует процесс обмена (несёт сигналы чтение/запись, обращение к внутренней/внешней памяти, данные готовы/не готовы и т. п.).

Рассмотрим процесс записи данных из процессора в память. На шину данных процессор выставляет данные для записи, на шину адреса — нужный адрес памяти, а на шину управления — сигналы для записи информации в память. Далее он вынужден ожидать, пока данные будут «взяты» с шины. В это время все остальные устройства постоянно «слушают» шину (проверяют её состояние). В нашем примере по сигналам на шине память обнаруживает, что для неё имеются данные. Она сохраняет их по заданному адресу и должна по шине управления сообщить процессору, что операция завершена. На практике, учитывая высокую надёжность работы памяти, сигнал подтверждения часто не ис-

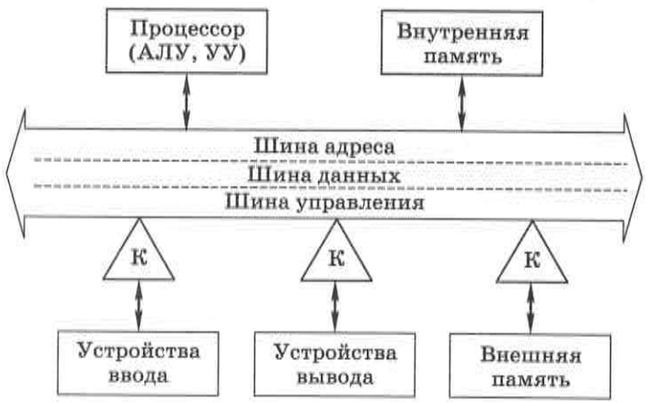


Рис. 5.10

пользуется: процессор просто выжидает определённое время и продолжает выполнение программы. Из этого примера понятно, что для успешного обмена данными по шине необходимы чёткие правила (их принято называть **протоколом шины**), которые должны соблюдать все устройства.

По сравнению с первыми ЭВМ, взаимодействие процессора с внешними устройствами организуется теперь по-другому. В классической архитектуре процессор контролировал все процессы ввода/вывода. Получалось так, что быстродействующий процессор тратил много времени на ожидание при работе с значительно более медленными внешними устройствами. Поэтому появились специальные электронные схемы, которые руководят обменом данными между процессором и внешними устройствами. В третьем поколении такие устройства назывались **каналами ввода/вывода**, а в четвёртом — **контро́ллерами**[1] (на рис. 5.10 они обозначены буквой К).

Контро́ллер — это электронная схема для управления внешним устройством и простейшей предварительной обработки данных.

Современный контроллер — это специальный микропроцессор, предназначенный для обслуживания одного или нескольких однотипных устройства ввода/вывода (УВВ) или внешней памяти. Нагрузка на центральный процессор при этом существенно снижается, и это увеличивает эффективность работы всей системы в целом. Контроллер, собранный в виде отдельной микросхемы, называют **микроконтроллером**.

В качестве примера рассмотрим контроллер современного жёсткого диска. Его основная задача — по принятым от процессора координатам найти на диске требуемые данные, прочитать их и передать в ОЗУ. Но контроллер способен выполнять и другие, порой весьма нетривиальные функции. Так, он сохраняет в служебной области диска информацию обо всех имеющихся на магнитной поверхности некачественно изготовленных секторах (а их

[1] Это название происходит от английского слова *control* — «управление»; не следует путать с русским словом «контролёр».

при современной высокой плотности записи избежать не удаётся!) и способен «на ходу» подменять их резервными, что создаёт видимость диска, который полностью свободен от дефектов.

Как видно из приведённой на рис. 5.10 схемы, теперь данные могут передаваться между внешними устройствами и ОЗУ напрямую, минуя процессор. Кроме того, наличие шины существенно упрощает подсоединение к ней новых устройств. Архитектуру, которую можно легко расширять за счёт подключения к шине новых устройств, часто называют **магистрально-модульной архитектурой**.

Если спецификация на шину (детальное описание всех её логических и физических параметров) является открытой (опубликована), то производители могут разрабатывать к такой шине любые дополнительные устройства. Такой подход называют **принципом открытой архитектуры**. При этом в компьютере предусмотрены стандартные разъёмы для подключения новых устройств, удовлетворяющих стандарту. Поэтому пользователь может собрать такой компьютер, который ему нужен. Необходимо только помнить, что при подключении любого нового устройства нужно установить специальную программу — **драйвер**, которая управляет обменом данными между этим устройством и процессором.

В современных компьютерах для повышения эффективности работы используется несколько шин, например, одна — между процессором и памятью, другая связывает процессор с видеосистемой и т. д.

Обмен данными с внешними устройствами

Существуют три режима обмена данными между центральным процессором (ЦП) и внешними устройствами:

- программно управляемый ввод/вывод;
- обмен с устройствами по прерываниям;
- прямой доступ к памяти (ПДП).

При **программно управляемом обмене** все действия по вводу или выводу предусмотрены в теле программы. Процессор полностью руководит ходом обмена, включая ожидание готовности периферийного устройства и прочие временные задержки, связанные с процессами ввода/вывода. *Достоинства* этого метода — простота и отсутствие дополнительного оборудования, *недостаток* — большие потери времени из-за ожидания быстро работающим процессором более медленных устройств ввода/вывода.

При **обмене по прерываниям** устройства ввода/вывода в случае необходимости сами «требуют внимания» процессора. Например, клавиатура оповещает процессор каждый раз, когда была нажата или отпущена клавиша; всё остальное время процессор выполняет программу, «не отвлекаясь» на клавиатуру. Когда прерывание произошло, ЦП «откладывает» на некоторое время выполнение основной программы и переходит на служебную программу обработки прерывания. Завершив его обработку, ЦП снова возвращается к тому месту программы, где она оказалась прервана. При этом основная программа даже «не заметит» возникшей задержки. Этот режим обмена более сложен, но зато значительно эффективнее — процессор не тратит время на ожидание.

Представим себе, что в кабинете начальника идет совещание, и в этот момент по телефону поступает важная информация, требующая немедленного принятия решения. Секретарша, не дожидаясь конца совещания, сообщает начальнику о звонке. Тот, прервав свое выступление, снимает трубку, выясняет суть дела и сообщает свое решение. Затем он продолжает совещание, как ни в чем не бывало. Здесь роль ЦП играет начальник, а телефонный звонок — это запрос (требование) на прерывание. «Секретарша» в компьютере тоже предусмотрена — это контроллер прерываний, анализирующий и сортирующий все поступающие прерывания с учётом их важности (*приоритета*).

Механизм прерываний используется не только в аппаратной части, но и в программах, которые основаны на обработке событий (нажатий на клавиши, команд управления от мыши и т. п.). Такая технология лежит в основе современных операционных систем и применяется в системах разработки программ Microsoft Visual Studio, Delphi, Lazarus и им подобных.

В обоих описанных выше вариантах управление обменом выполнял центральный процессор. Именно он извлекал из памяти выводимые данные (или записывал туда вводимые), подсчитывал их количество и полностью контролировал работу шины. Если передаваемые данные не требуют сложной обработки, ЦП напрасно расходует время на проведение обмена. Чтобы освободить процессор от этой работы и увеличить скорость передачи крупных блоков данных от устройства ввода в память и обратно, применяется **прямой доступ к памяти** (**ПДП**, англ. **DMA** — *Direct Memory Access*).