Балакшин П.В. Соснин В.В.

Информатика

Методическое пособие

Санкт-Петербург, 2015 г

Оглавление

Глава 1

Единицы измерения объема данных

В области цифровой и вычислительной техники двоичная система счисления (основанная на степени двойки) получила широкое распространение. В следствие этого стали употреблять двоичные приставки:
$1kB = 2^{10}B$ - 1 килобайт равен 2^{10} байт
$1MB = 2^{20}B$ - 1 мегабайт равен 2^{20} байт
Однако, такая система противоречит СИ, которая использует десятичные приставки (основанные на степени десяти):
$1k = 10^3$, $1M = 10^6$
Поначалу это противоречие не было существенной проблемой. Число $2^{10} = 1024$ достаточно близко к тысяче и при объемах памяти, измерявшихся килобайтами, ошибка была всего в 2,4% . Но по мере развития технологий, разница между "двоичным"и "десятичным"гигабайтом была в 7% . Тогда IEEE, Институт инженеров электротехники и электроники (англ. Institute of Electrical and Electronics Engineers), утвердил стандарт IEEE 1541-2002.

Рекомендации стандарта IEEE 1541-2002

- Бит (bit) (символ 'b') - двоичный знак;
- Байт (byte) (символ 'B') - равен 8 битам ($1B = 8b$);
- Киби (kibi) (символ 'Ki') - $2^{10} = 1024$;
- Меби (mebi) (символ 'Mi') - $2^{20} = 1048576$;
- Гиби (gibi) (символ 'Gi') - $2^{30} = 1073741824$;

- Теби (tebi) (символ 'Ti') - $2^{40} = 1099511627776$;

Сегодня в кибибайтах, мебибайтах и т.д. измеряется память - оперативная память, жесткие диски, flash-накопители. Однако, скорость передачи данных измеряется в килобайтах и мегабайтах в секунду (например 512 kbps (kilobits per second) - 512 килобит в секунду). Операционные системы считают по-разному. *nix системы (unix, linux) используют "двоичные"приставки, в то время как Windows использует приставки СИ.

Глава 2

Системы счисления

В истории разные народы использовали системы счисления с разными основаниями. Каждый народ руководствовался своими доводами в пользу того или иного числа в основании. Например, африканские племена использовали 5-ричную систему счисления, потому что на руке 5 пальцев. Тибетцы и нигерийцы использовали 12-ричную, это количество фаланг на четырех пальцах. Привычная нам система счисления с основанием равным 10 появилась в Европе в 16, а в России в 17 веке.

2.1 Позиционная система счисления

Рассмотрим формулу записи числа в позиционной системе счисления:
$X_{(q)} = x_{n-1} \times q^{n-1} + x_{n-2} \times q^{n-2} + ... + x_1 \times q^1 + x_0 \times q^0 + x_{-1} \times q^{-1} + + x_{-2} \times q^{-2} + ... + x_{-m} \times q^{-m}$
Или

$$X_{(q)} = \sum_{i=-m}^{n-1} x_i \times q^i$$

Где:
$X_{(q)}$ - запись числа в системе счисления с основанием q
x_i - натуральные числа меньше q, то есть цифры
n - число разрядов целой части
m - число разрядов дробной части
q - показатель системы счисления

Само число $X_{(q)}$ имеет следующий вид:

$$X_{(q)} = x_{n-1}x_{n-2}...x_1x_0x_{-1}...x_{1-m}x_{-m}$$

Рассмотрим данную формулу на примере:
$123,45_{10} = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 + 4 \times 10^{-1} + 5 \times 10^{-2}$
Мы разложили число $123,45$ по этой формуле. В данном случае $q = 10$, $n = 3$, $m = 2$, $X_{(q)} = 123,45$, а $x_{3-1} = 1$ ($x_2 = 1$), $x_1 = 2$ и так далее.
В позиционной системе счисления важную роль имеет порядок цифр, то есть значение каждого числового знака (цифры) в записи числа зависит от его позиции (разряда).

2.2 Перевод чисел из одной системы счисления в другую

Существуют три способа перевода из одной системы счисления в другую:

1. Из десятичной системы счисления в систему счисления с основанием N
2. Из системы счисления с основанием N в десятичную систему счисления
3. Из системы счисления с основанием N в систему счисления с основанием N^k и обратно, при условии $k \in \mathbb{N}$

2.2.1 Перевод числа из десятичной системы счисления в систему счисления с основанием N

Чтобы перевести дробное число в систему счисления с основанием N необходимо разделить его на две части: целую и дробную, и каждую часть переводить отдельно.

Преобразования целой части числа

Для перевода целой части числа из десятичной системы счисления в другую необходимо:

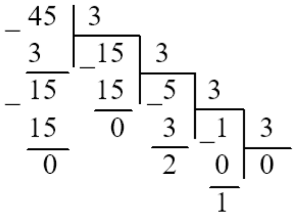
1. Разделить целую часть десятичного числа на основание новой системы счисления;
2. Записать остаток деления;

3. Разделить получившийся результат деления (п.1) на основание новой системы счисления (при необходимости);

4. Записать остаток деления;

Повторять, пока целая часть десятичного числа не будет равна 0. Получившиеся в ходе деления остатки и есть цифры искомого числа в новой системе счисления. Записать остатки в обратном порядке (начиная с последнего полученного). Стоит заметить, что в данном способе очень удобно применять деление столбиком.

Пример 1:
Задание: перевести число 45_{10} в троичную систему счисления.
Решение: последовательно разделим 45_{10} на 3, записывая остатки:



Полученные остатки: 0, 0, 2, 1. Записываем их в обратном порядке.
Ответ:$45_{10} = 1200_3$

Преобразования дробной части числа

Для перевода дробной части числа из десятичной системы счисления в другую необходимо:

1. Умножить дробную часть десятичного числа на основание новой системы счисления;

2. Отделить и записать целую часть;

3. Умножить дробную часть результата умножения (п.1) на основание новой системы счисления (при необходимости);

4. Отделить и записать целую часть;

Повторять, пока дробная часть десятичного числа не будет равна 0. Получившиеся в ходе умножения целые части и есть цифры искомого числа в новой системе счисления. Записать целые части в прямом порядке (начиная с первого полученного). Первая записанная целая часть (0)

идет в целую часть нового числа, а в дробную записываются полученные целые части, начиная со второй. Стоит заметить, что в данном способе очень удобно применять умножение столбиком.

Пример 2:
Задание: перевести число $0,625_{10}$ в четверичную систему счисления.
Решение: умножим дробную часть $0,625_{10}$ на 4, записывая целые части, пока не получим в дробной части 0:

$$\begin{array}{r} 0\,,\,625 \\ \times\quad\quad 4 \\ \hline 2\,,\,500 \\ \times\quad\quad 4 \\ \hline 2\,,\,000 \end{array}$$

Пояснение: сначала умножаем 0,625 на 4, получаем 2,5. 2 записываем в целые части, а далее используем дробную часть - 0,5. Умножаем 0,5 на 4, получаем 2, записываем в целые части. Так как дробная часть равна 0, то перевод окончен.
Полученные целые части: 0, 2, 2. Первая полученная целая часть (0) идет в целую часть нового числа. Остальные (2, 2) в дробную часть.
Ответ: $0,625_{10} = 0,22_4$

Пример 3:
Задание: перевести число $43,52_{10}$ в пятеричную систему счисления.
Решение: разделим $43,52_{10}$ на две части: целую (43_{10}) и дробную ($0,52_{10}$). Переведем целую и дробную части по отдельности:

$$\begin{array}{r|l} 43 & 5 \\ \hline 40 & 8 \quad 5 \\ \hline 3 & 5 \quad 1 \quad 5 \\ \hline & 1 \quad 0 \quad 0 \\ \hline & 1 \end{array} \qquad \begin{array}{r} 0\,,\,52 \\ \times\quad\quad 5 \\ \hline 2\,,\,60 \\ \times\quad\quad 5 \\ \hline 3\,,\,00 \end{array}$$

Полученные остатки (3, 1, 1) запишем в обратном порядке: $43_{10} = 113_5$
Полученные целые части (0, 2, 3) запишем в прямом порядке: $0,52_{10} = 0,23_5$
Объеденим полученные части
Ответ: $43,52_{10} = 113,23_5$

2.2.2 Перевод числа из системы счисления с основанием N в десятичную систему счисления

Формула перевода числа из системы счисления с основанием N в десятичную систему счисления это практически формула записи числа в позиционной системе счисления.

$$X_{(10)} = \sum_{i=-m}^{n-1} x_i \times q^i$$

Где:
$X_{(10)}$ - искомое число в десятичной системе счисления
x_i - натуральные числа меньше q, то есть цифры
n - число разрядов целой части
m - число разрядов дробной части
q - показатель системы счисления

Пример 1:
Задание: перевести число $1101,111_2$ в десятичную систему счисления.
Решение: $1101,111_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + + 1 \times 2^{-3} = 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 + 1 \times 0,5 + 1 \times 0,25 + 1 \times 0,125 = = 8 + 4 + 1 + 0,5 + 0,25 + 0,125 = 13,875_{10}$
Ответ: $1101,111_2 = 13,875_{10}$

2.2.3 Перевод числа из системы счисления с основанием N в систему счисления с основанием N^k и обратно, при условии $k \in \mathbb{N}$

Если основание системы счисления первого числа является степенью основания системы счисления второго числа ($N = N^k$), при условии $k \in \mathbb{N}$, то можно использовать следующий алгоритм.

Преобразование $N \to N^k$

1. Дополнить число (записанное в системе счисления N) незначащими нулями так, чтобы количество цифр было кратно k (если число дробное, то дополнить так, чтобы и в целой и в дробной частях количество цифр было кратно k).

2. Разбить это число на группы по k цифр, начиная от нуля (если число дробное, то целую часть разбивать, начиная от запятой в левую сторону, а дробную часть, начиная от запятой в правую сторону).

3. Заменить каждую такую группу эквивалентным числом, записанным в системе N^k.

Преобразование $N^k \to N$

1. Заменить каждую цифру числа, записанного в системе счисления N^k, эквивалентным набором из k цифр системы счисления N.

Рассмотрим данный метод на системах счисления с основанием $N = 2^k$. Для этого воспользуемся таблицей.

Десятичная	Двоичная	Восмеричная	Шестнадцатиричная
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Таблица 2.1: Основания вида 2^k

Пример 1:

Задание: пользуясь таблицей перевести число $1542,43_8$ в двоичную систему счисления

Решение: по таблице находим чему равны цифры исходного числа в двоичной системе. $1_8 = 001_2$, $5_8 = 101_2$ (незначащие нули убираем, так как необходимо, чтобы количество цифр в эквивалентном наборе было равно степени k из выражения $N = N^k$, где N^k - исходная система счисления. В данном случае $k = 3$) и так далее.

Заменяем каждую цифру числа эквивалентным набором.

Ответ: $1542,43_8 = 001101100010,100011_2$

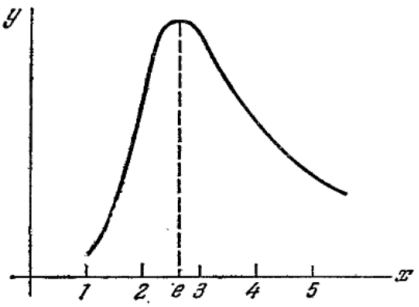
Пример 2:

Задание: пользуясь таблицей перевести число $11010,11_2$ в шестнадцатиричную систему счисления

Решение: первым делом, добавим незначащие нули так, чтобы количество цифр было кратно k (в данном случае $k = 4$). Так как число дробное, не забываем добавлять нули и в конце числа.
Получим $00011010,1100_2$.
Теперь необходимо разбить число на группы по k цифр (начинаем от запятой). Результат: 0001 1010 , 1100 $_2$.
Пользуясь таблицей, заменяем группы цифр эквивалентными числами, записанным в шестнадцатиричной системе счисления.
Ответ: $11010,11_2 = 1A,C_{16}$.

2.3 Оптимальная система счисления

Давайте представим, что Вы по несчастливой (или счастливой) случайности попали на необитаемый остров. Обеспокоенный количеством дней, которые Вам суждено провести на острове в ожидании спасателей, Вы решаете вести счет дней с помощью камней. Но вот незадача - камней на острове нашлось всего 60 штук (небогатый на камни остров оказался). И для того, чтобы вести учет дней как можно продуктивнее (учесть как можно больше дней) необходимо выбрать систему счисления, плотность записи числа которой максимальна при данных обстоятельствах.
Существует зависимость плотности записи информации от основания системы счисления. Если взять N камней, а за основание принять число X, то получится $^N/_X$ разрядов, которыми можно закодировать $X^{N/X}$ чисел. То есть с помощью 60 камней мы можем закодировать: 2^{30}, 3^{20}, 4^{15}, 5^{12}, 6^{10}, 10^6, 12^2, 15^4, 20^3, 30^2 или 60^1 чисел. Все зависит от того, какую систему счисления мы выберем. Возведя все числа в степени, мы увидим, что самое большое из них это $3^{20} = 3486784401$.
Удельная натуральнологарифмическая плотность записи числа зависит от основания системы счисления и выражается функцией $y = \frac{\ln x}{x}$. Эта функция имеет максимум при $x = e = 2,718281828....$



Таким образом, самая оптимальная система счисления имеет основание равное $e = 2,718281828...$, то есть нецелочисленное. Она обладает наибольшей плотностью записи информации.
Возвращаясь к нашему пребыванию на острове, мы не можем взять дробное основание для системы счисления. Поэтому мы берем самое близкое целое к e - это 3.

2.4 Округление чисел

Как происходит округление чисел в десятичной системе счисления? Каждый учил в школе правила округления: 1, 2, 3, 4 округляется в меньшую сторону, а 5, 6, 7, 8 и 9 - в большую. Рассмотрим два примера:

	Число	Округление
	5,9	6,0
	4,1	4,0
	5,0	5,0
	6,6	7,0
	2,4	2,0
Сумма	24,0	24,0

Пример 1

	Число	Округление
	5,5	6,0
	3,5	4,0
	2,5	3,0
	8,5	9,0
	1,5	2,0
Сумма	21,5	24,0

Пример 2

В примере 1 мы видим, что сумма до и после округления одинакова. Так случилось, потому что мы округляли то в большую сторону, то в меньшую, и в итоге количество округлений в большую сторону равно количеству округлений в меньшую. Округление нам не помешало получить красивую сумму.
А теперь посмотрим на специально подобранный пример 2. Разница сумм довольно большая. Так получилось, потому что мы округляли всегда только в большую сторону, хотя мы округляли по правилам.
Если проделать такой эксперимент с большим количеством чисел (тысяча, две тысячи или даже больше), то ошибка будет небольшая, но она будет.
Работая с числами, у которых показатель системы счисления четный, мы натыкаемся на следующую проблему: у нас нечетное количество чисел для округления. Разберем на примере десятичной системы счисления. В ней всего 10 цифр - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Числа $X.0$ мы не округляем. Остается 9 чисел: $X.1$, $X.2$, $X.3$, $X.4$ округляем в меньшую сторону (всего 4 числа), а $X.5$, $X.6$, $X.7$, $X.8$, $X.9$ - в большую (всего 5 чисел). $X.5$ - середина между $X+1$ и X, однако мы округляем в пользу $X+1$, то есть в большую. Таким образом, при работе с большим количеством чисел, мы всегда будем округлять в большую сторону чаще, чем в меньшую. Отсю-

да и ошибка в итоговой сумме.
Чтобы этого избежать, некоторые программы при автоматическом округлении большого количества чисел округляют $X.5$ по очереди то в большую сторону, то в меньшую.
В системах счисления с нечетным основанием такой ошибки нет. Возьмем, к примеру, пятиричную систему счисления. Она содержит цифры 0, 1, 2, 3, 4. Числа $X.0$ мы не округляем, остается 4 числа: $X.1$, $X.2$ округляем в меньшую сторону (всего 2 числа), а $X.3$, $X.4$ - в большую (всего 2 числа). Таким образом, количество чисел, округленных в меньшую сторону, равно количеству чисел, округленных в большую.

2.5 Нетрадиционные системы счисления

2.5.1 Факториальная система счисления

Любое натуральное число можно представить в виде

$$X = \sum_{k=1}^{n} d_k \times k! \qquad \text{,где } 0 \leqslant d_k \leqslant k$$

В основании факториальной системы счисления используется факториал. Запись числа в факториальной системе счисления будет иметь вид

$$X_{\text{Ф}} = d_n d_{n-1} ... d_1$$

Перевод из факториальной системы счисления в десятичную

Алгоритм перевода из факториальной системы счисления в десятичную очень похож на алгоритм перевода из системы счисления с основанием N в десятичную (раздел 2.2.1).

$$X_{10} = d_n \times n! + d_{n-1} \times (n-1)! + d_{n-2} \times (n-2)! + ... + d_2 \times 2! + d_1 \times 1!$$

Где:
X_{10} - искомое число в десятичной системе счисления
d_i - натуральные числа меньше или равные i
n - количество разрядов исходного числа
Пример 1:
Задание: перевести число $221_{\text{Ф}}$ в десятичную систему счисления
Решение: $221_{\text{Ф}} = 2 \times 3! + 2 \times 2! + 1 \times 1! = 2 \times 6 + 2 \times 2 + 1 \times 1 = 12 + 4 + 1 = 17_{10}$
Ответ: $221_{\text{Ф}} = 17_{10}$

Перевод из десятичной системы счисления в факториальную

Для перевода воспользуемся все той же формулой:

$$X_{10} = d_n \times n! + d_{n-1} \times (n-1)! + d_{n-2} \times (n-2)! + ... + d_2 \times 2! + d_1 \times 1!$$

Стоит обратить внимание, что $0 \leqslant d_1 \leqslant 1$; $0 \leqslant d_2 \leqslant 2$ и так далее.

1. Находим факториал $k!$, значение которого больше X_{10}, но ближе всего к нему. Тогда $n = k - 1$, где n - количество разрядов искомого числа в факториальной системе.

2. Записываем

   $$X_{10} = d_n \times n! + d_{n-1} \times (n-1)! + d_{n-2} \times (n-2)! + ... + d_2 \times 2! + d_1 \times 1!$$

   с уже полученным n.

3. Начиная с d_n с помощью ума и смекалки начинаем подбирать коэффициенты, помня, что $d_1 \in \{0, 1\}$, $d_2 \in \{0, 1, 2\}$ и т.д.

Пример 2:
Задание: перевести число 54_{10} в факториальную систему счисления
Решение: $54_{10} < 5!(5! = 120)$, значит количество разрядов равно $5-1 = 4$.
Запишем формулу для $n = 4$: $54_{10} = d_4 \times 4! + d_3 \times 3! + d_2 \times 2! + d_1 \times 1!$
Подберем коэффициенты: $54_{10} = 2 \times 4! + 1 \times 3! + 0 \times 2! + 0 \times 1!$
Ответ: $54_{10} = 2100_{\text{ф}}$

Применение факториальной системы счисления: декодирование и кодирование перестановок.

Пример 3:
Задание: Имеется $n = 5$ чисел (1, 2, 3, 4, 5), нужно найти все их перестановки. Известно, что существует $n! = 5! = 120$ таких перестановок. Найти перестановку, если известен ее номер $k = 52$.
Решение: Переведем k в факториальную систему: $52_{10} = 2 \times 4! + 0 \times 3! + 2 \times 2! + 0 \times 1! = 2020_{\text{ф}}$
Дополним результат до $n - 1$ разрядов (при необходимости), расставим символы по местам:
1. Справа от 5 есть 2 меньшие цифры $(- - 5 - -)$;
2. Справа от 4 есть 0 меньших цифр $(- - 5 \ -4)$;
3. Справа от 3 есть 2 меньшие цифры $(3 \ -5 \ -4)$;
4. Справа от 2 есть 0 меньших цифр $(3 \ -5 \ 2 \ 4)$;
Ответ: (3 1 5 2 4)

0	12345
1	?????
...	.....
52	?????
...	.....
119	54321

2.5.2 Система счисления Цекендорфа

Любое натуральное число можно представить в виде

$$X = \sum_{k=1}^{n} d_k \times F_k \qquad \text{,где } d_k \in \{0, 1\}, \text{а } F_k \text{ - числа Фибоначчи}$$

Каждое число Фибоначчи есть сумма двух предыдущих чисел:
$F_k = \{1, 1, 2, 3, 5, 8, 13, 21, ...\}$.
В записи чисел в системе счисления Цекендорфа первая единица из ряда чисел Фибоначчи **не используется**(т.к. первая единица это F_0).
Запись числа в системе счисления Цекендорфа будет иметь вид

$$X_{\text{Ц}} = d_n d_{n-1} ... d_1$$

В записи чисел в системе счисления Цекендорфа **не допускается использование двух единиц подряд**.

Перевод из системы счисления Цекендорфа в десятичную

Алгоритм перевода из системы счисления Цекендорфа в десятичную очень похож на алгоритм перевода из системы счисления с основанием N в десятичную (раздел 2.2.1).

$$X_{10} = d_n \times F_n + d_{n-1} \times F_{n-1} + d_{n-2} \times F_{n-2} + ... + d_2 \times F_2 + d_1 \times F_1$$

Где:
X_{10} - искомое число в десятичной системе счисления
d_i - число, принимающее значение 0 или 1
n - количество разрядов исходного числа

Пример 1:
Задание: перевести число $100101_{\text{Ц}}$ в десятичную систему счисления
Решение: $100101_{\text{Ц}} = 1{\times}13{+}0{\times}8{+}0{\times}5 = 1{\times}3{+}0{\times}2{+}1{\times}1 = 13{+}3{+}1 = 17_{10}$
Ответ: $100101_{\text{Ц}} = 17_{10}$

Перевод из десятичной системы счисления в систему счисления Цекендорфа

Для перевода воспользуемся все той же формулой:

$$X_{10} = d_n \times F_n + d_{n-1} \times F_{n-1} + d_{n-2} \times F_{n-2} + ... + d_2 \times F_2 + d_1 \times F_1$$

1. Находим число F_k в ряду чисел Фибоначчи, которое больше X_{10}, но ближе всего к нему. Тогда $n = k - 1$, где n - количество разрядов искомого числа в системе Цекендорфа.

2. Записываем

$$X_{10} = d_n \times F_n + d_{n-1} \times F_{n-1} + d_{n-2} \times F_{n-2} + ... + d_2 \times F_2 + d_1 \times F_1$$

с уже полученным n.

3. Начиная с d_n с помощью ума и смекалки начинаем подбирать коэффициенты, помня, что $d_1 \in \{0, 1\}$ и что две единицы не могут стоять рядом.

Пример 2:
Задание: перевести число 19_{10} в систему счисления Цекендорфа
Решение: $19_{10} < 21 (21 = F_7)$, значит количество разрядов равно $7-1=6$.
Запишем формулу для $n = 6$: $d_6 \times 13 + d_5 \times 8 + d_4 \times 5 + d_3 \times 3 + d_2 \times 2 + d_1 \times 1$
Подберем коэффициенты: $19_{10} = 1 \times 13 + 0 \times 8 + 1 \times 5 + 0 \times 3 + 0 \times 2 + 1 \times 1$
Ответ: $19_{10} = 101001_{\text{Ц}}$

Применение системы счисления Цекендорфа: кодирование данных с маркером завершения $'11'$, у некоторых народов в сельском хозяйстве - минимизация необходимого числа зерен.

2.5.3 Система счисления Бергмана

Любое действительное неотрицательное число можно представить в виде

$$X = \sum_{k=-\infty}^{\infty} d_k \times z^k \qquad \text{,где } d_k \in \{0, 1\}, \text{а } z = \frac{1+\sqrt{5}}{2}$$

Число z - число золотой пропорции.
Запись числа в системе счисления Бергмана будет иметь вид:

$$X_{\text{Б}} = d_n d_{n-1} ... d_2 d_1 d_0, d_{-1} d_{-2} ... d_{-m}$$

Чтобы исключить неоднозначность, используется запись с наибольшим количеством разрядов.

Перевод из системы счисления Бергмана в десятичную

Алгоритм перевода из системы счисления Бергмана в десятичную очень похож на алгоритм перевода из системы счисления с основанием N в десятичную (раздел 2.2.1).

$$X_{10} = d_n \times z^n + d_{n-1} \times z^{n-1} + ... + d_2 \times z^2 + d_1 \times z^1 + d_0 \times z^0 + d_{-1} \times z^{-1} + ... d_{-m} \times z^{-m}$$

Где:
X_{10} - искомое число в десятичной системе счисления
d_i - число, принимающее значение 0 или 1
n - количество разрядов целой части
m - количество разрядов дробной части

Пример 1:
Задание: перевести число $100,01_{\text{Б}}$ в десятичную систему счисления
Решение: $100,01_{\text{Б}} = 1\times(\frac{1+\sqrt{5}}{2})^2+0\times(\frac{1+\sqrt{5}}{2})^1+0\times(\frac{1+\sqrt{5}}{2})^0+0\times(\frac{1+\sqrt{5}}{2})^{-1}+$
$+1\times(\frac{1+\sqrt{5}}{2})^{-2} = (\frac{1+\sqrt{5}}{2})^2+(\frac{1+\sqrt{5}}{2})^{-2} = \frac{6+2\sqrt{5}}{4}+\frac{4}{6+2\sqrt{5}} = \frac{9+3\sqrt{5}}{3+\sqrt{5}} = \frac{3(3+\sqrt{5}}{3+\sqrt{5}} =$
3
Ответ: $100,01_{\text{Б}} = 3_{10}$
Перевод чисел из десятичной системы в систему счисления Бергмана происходит методом подбора
Применение системы счисления Бергмана: запись иррациональных чисел конечным числом цифр, контроль арифметических операций, коррекция ошибок, самосинхронизация кодовых последовательностей при передаче по каналу связи.

2.5.4 Нега-позиционная система счисления

Это системы счисления с отрицательным основанием. Числа в нега-позиционных системах счисления, которые содержат четное количество цифр - отрицательные.
Главная особенность таких систем в том, что не нужно никаких специальных знаков для обозначения отрицательных чисел.

Перевод чисел из нега-позиционных систем счисления полностью описан в разделе 2.2.1. Обратный перевод происходит методом подбора.

Примеры в нега-десятичной системе счисления:
$15_{-10} = 1\times(-10)^1+5\times(-10)^0 = -1\times10+5\times1 = -10+5 = -5_{10}$;
$532_{-10} = 5\times(-10)^2+8\times(-10)^1+2\times(-10)^0 = 5\times100-8\times10+2\times1 =$
$= 500-80+2 = 422_{10}$;

2.5.5 Симметричная система счисления

Это системы счисления с отрицательными числами. Центром симметрии является 0, поэтому основанием для таких систем счисления могут быть только нечетные числа.
Главная особенность таких систем, как и нега-позиционных, в том, что

не нужно никаких специальных знаков для обозначения отрицательных чисел.

Перевод чисел из симметричных систем счисления и полностью описан в разделе 2.2.1. Обратный перевод происходит методом подбора.

Примеры в симметричной пятеричной системе счисления: Если в обычной пятеричной системе используются цифры {0, 1, 2, 3, 4}, то в симметричной пятеричной системе используются {-2, -1, 0, 1, 2}.
$10\overline{2}\overline{1}2_{5\text{C}} = 1 \times 5^4 + 0 \times 5^3 + (-2) \times 5^2 + (-1) \times 5^1 + 2 \times 5^0 = 1 \times 625 + 0 \times 125 - 2 \times 25 - 1 \times 5 + 2 \times 1 = 625 - 50 - 5 + 2 = 572_{10}$;
$\overline{1}021\overline{2}_{5\text{C}} = (-1) \times 5^4 + 0 \times 5^3 + 2 \times 5^2 + 1 \times 5^1 + (-2) \times 5^0 = -1 \times 625 + 0 \times 125 + 2 \times 25 + 1 \times 5 - 2 \times 1 = -625 + 50 + 5 - 2 = -572_{10}$;

Стоит обратить внимание, что цифры с чертой сверху - отрицательные.

Глава 3

Арифметика в ограниченной разрядной сетке

3.1 Представление отрицательных чисел в ЭВМ

В электронных вычислительных машинах нет возможности обозначить знак "минус"перед числом. Существует несколько способов решения этой проблемы:

1. **Специальный знаковый бит** - определенный бит означает знак числа.
   Пример 1:
   $+5_{10} = 0101_2$, $-5_{10} = 1101_2$
   В данном случае, знаковый бит - старший.

2. **Фиксированное смещение** - все числа уменьшены на какое-то определенное число.
   Пример 2:
   $-5_{10} = 0000_2$, $-4_{10} = 0001_2$, ...,$+10_{10} = 1111_2$
   В данном случае, все числа уменьшены на 5.

3. **Нега-двоичная система счисления** - основание системы счисления равно -2.
   Пример 3:
   $-4_{10} = 1100_{-2}$, $+5_{10} = 0101_{-2}$

4. **Обратный (инверсный) код** - инвертируются все биты.

Пример 4:
$+5_{10} = 0101_2$, $-5_{10} = 1010_2$

5. **Дополнительный код** - инверсия всех бит плюс единица.
   Пример 5:
   $+5_{10} = 0101_2$, $-5_{10} = 1011_2$

Некоторые из этих способов были реализованы. В 50-х и 60-х годах широко использовался четвертый способ. И специалисты теории информации разбились на два лагеря: те, кто использовал четвертый способ, и те, кто использовал пятый. Долго не могли примириться и компьютеры существовали и в том и в другом виде. Это привело к тому, что в стандарте языка программирования Си не определено как именно представлять отрицательные числа. Если Вы не будете использовать стандартные конструкции языка (например, $a = b + c$), в которых язык Си сам считает значение из памяти и приведет к нужному отрицательному или положительному виду, а сразу обратитесь к внутреннему представлению памяти - к ячейке по адресу (возьмете значение напрямую из ячейки), то хранящиеся там биты будут различны. Все будет зависеть от того, как работает Ваш компьютер: по четвертому способу или по пятому. Так же при складывании некоторых чисел, интерпретированных в обратном коде, в ограниченной разрядной сетке, возникает ошибка и требуется корректировка результата. Пятый способ не требует корректировки. Поэтому было решено использовать для представления отрицательных чисел в ЭВМ дополнительный код.

Алгоритм перевода двоичного числа в дополнительный код и из дополнительного кода в прямой

1. Инвертировать все биты;

2. Прибавить единицу;

Отличить, в каком коде представлено число в разрядной сетке - в дополнительном или прямом, можно по старшему значащему биту. Если старший бит равен нулю - число положительное, единице - число отрицательное (например, в числе 1010_2 старший бит равен 1 - число отрицательное и представлено в дополнительном коде).
Важно! Нумерация бит в разрядной сетке начинается с нуля и идет справа налево.

Пример 6:
Задание: перевести число 1101011011_2 в дополнительный код.
Решение: инвертируем биты: $1101011011_2 \rightarrow 0010100100_2$;

Прибавляем единицу: $0010100100_2 + 1_2 = 0010100101_2$;
Ответ: 0010100101_2

Пример 7:
Задание: представить число -18_{10} в 8-разрядной сетке.
Решение: так как у нас число отрицательное, то сначала переведем модуль данного числа в двоичную систему счисления: $|-18_{10}| = 18_{10} = 10010_2$; Теперь представим его в дополнительном коде: $10010_2 \rightarrow 01101_2 + 1_2 = 01110_2$; Так как у нас 8-разрядная сетка, то дополним число незначащими нулями: $0000\ 1110_2$
Ответ: $-18_{10} = 0000\ 1110_2$

3.2 Флаги состояния процессора

Регистр флагов - регистр процессора, отражающий текущее состояние процессора.
После любой арифметической операции процессор автоматически без участия программиста заполняет регистр флагов состояния. Состояние процессора меняется после каждой арифметической операции.
Таблица 3.1 представлена для ознакомления. Для курса "Информатика"необходимо знать следующие флаги:

1. **CF (Carry Flag) - Флаг переноса**. Устанавливается (принимает значение 1) в случае, если происходит перенос за пределы разрядов или заем извне.

2. **PF (Parity Flag) - Флаг четности**. Устанавливается, если младший значащий байт результата содержит четное число единичных (ненулевых) бит. Изначально этот флаг был ориентирован на использование в коммуникационных программах: при передаче данных по линиям связи для контроля мог также передаваться бит четности.

3. **AF (Auxiliary Carry Flag) - Вспомогательный флаг переноса**. Устанавливается, если произошел заем или перенос между первым и вторым полубайтами (третьим и четвертым битами).

4. **ZF (Zero Flag) - Флаг нуля**. Устанавливается, если результат машинной операции по модулю 2 в степени k (где k - разрядность ячейки) равен нулю (другими словами, принимает значение 1, если результат выполнения операции равен нулю).

5. **SF (Sign Flag) - Флаг знака**. Устанавливается, если результат выполнения операции отрицателен (равен значению старшего значащего бита).

6. **OF (Overflow Flag) - Флаг переполнения**. Устанавливается, если в результате выполнения операции со знаковыми числами появляется одна из ошибок: при сложении положительных чисел получается отрицательный результат или при сложении отрицательных чисел получается положительный результат.

Пример 1:

Задание: сложить 14837_{10} и 5832_{10} в 16-разрядной сетке. Расставить флаги состояния процессора.

Решение: для начала переведем исходные числа в двоичную систему счисления. $14837_{10} = 0011\ 1001\ 1111\ 0101_2$, $5832_{10} = 0001\ 0110\ 1100\ 1000_2$.

$$\begin{array}{cll} & 0011\ 1001\ 1111\ 0101_2 & = 14837_{10} \\ + & 0001\ 0110\ 1100\ 1000_2 & = 5832_{10} \\ \hline & 0101\ 0000\ 1011\ 1101_2 & = 20669_{10} \end{array} \quad \boxed{\begin{array}{ll} \text{CF} = 0 & \text{ZF} = 0 \\ \text{PF} = 1 & \text{SF} = 0 \\ \text{AF} = 0 & \text{OF} = 0 \end{array}}$$

Так как $0101\ 0000\ 1011\ 1101_2 = 20669_{10}$, то результат операции сложения в 16-разрядной сетке корректен.

Пример 2:

Задание: сложить 21324_{10} и 13543_{10} в 16-разрядной сетке. Расставить флаги состояния процессора.

Решение: для начала переведем исходные числа в двоичную систему счисления. $21324_{10} = 0101\ 0011\ 0100\ 1100_2$, $13543_{10} = 0011\ 0100\ 1110\ 0111_2$.

$$\begin{array}{cll} & 0101\ 0011\ 0100\ 1100_2 & = 21324_{10} \\ + & 0011\ 0100\ 1110\ 0111_2 & = 13543_{10} \\ \hline & 1000\ 0101\ 1010\ 0100_2 & = -14940_{10} \end{array} \quad \boxed{\begin{array}{ll} \text{CF} = 0 & \text{ZF} = 0 \\ \text{PF} = 0 & \text{SF} = 0 \\ \text{AF} = 0 & \text{OF} = 1 \end{array}}$$

Так как $1000\ 0101\ 1010\ 0100_2 = -14940_{10} \neq 34867_{10} = 21324_{10} + 13543_{10}$, то результат операции сложения в 16-разрядной сетке некорректен.
OF = 1: при складывании положительных чисел получили отрицательное.

Пример 3:

Задание: аналогично примерам 1 и 2 - сложить -7453_{10} и 24732_{10}.

Решение: $24732_{10} = 0110\ 0000\ 1001\ 1100_2$, $-7453_{10} = 1110\ 0010\ 1110\ 0011_2$ (-7453_{10} представляем в дополнительном коде для 16-разрядной сетки).

$$\begin{array}{cll} & 0110\ 0000\ 1001\ 1100_2 & = 24732_{10} \\ + & 1110\ 0010\ 1110\ 0011_2 & = -7453_{10} \\ \hline 1 & 0100\ 0011\ 0111\ 1111_2 & = 17279_{10} \end{array} \quad \boxed{\begin{array}{ll} \text{CF} = 1 & \text{ZF} = 0 \\ \text{PF} = 0 & \text{SF} = 0 \\ \text{AF} = 0 & \text{OF} = 0 \end{array}}$$

Так как $0100\ 0011\ 0111\ 1111_2 = 17279_{10}$, то результат операции сложения в 16-разрядной сетке корректен. Несмотря на то, что произошел выход за пределы разрядности сетки.

Таблица 3.1: Регистр флагов Intel x86

Номер бита	Обозначение	Название	Описание
FLAGS			
0	CF	Carry Flag	Флаг переноса
1	-	-	Зарезервирован
2	PF	Parity Flag	Флаг четности
3	-	-	Зарезервирован
4	AF	Auxiliary Carry Flag	Вспомогательный флаг переноса
5	-	-	Зарезервирован
6	ZF	Zero Flag	Флаг нуля
7	SF	Sign Flag	Флаг знака
8	TF	Trap Flag	Флаг трассировки
9	IF	Interrupt Enable Flag	Флаг разрешения прерываний
10	DF	Direction Flag	Флаг направления
11	OF	Overflow Flag	Флаг переполнения
12 13	IOPL	I/O Privilege Level	Уровень приоритета ввода-вывода
14	NT	Nested Task	Флаг вложенности задач
15	-	-	Зарезервирован
EFLAGS			
16	RF	Resume Flag	Флаг возобновления
17	VM	Virtual-8086 Mode	Режим виртуального процессора 8086
18	AC	Alignment Check	Проверка выравнивания
19	VIF	Virtual Interrupt Flag	Виртуальный флаг разрешения прерывания
20	VIP	Virtual Interrupt Pending	Ожидающее виртуальное прерывание
21	ID	ID Flag	Проверка на доступность инструкции CPUID
22 ... 31	-	-	Зарезервированы
RFLAGS			
32 ... 63	-	-	Зарезервированы

Глава 4

Теория информации

4.1 Терминология теории информации

Понятие "информация"имеет различные трактовки в различных предметных областях. Например, в кибернетике "информация"может рассматриваться как сигналы направления, в биологии - мера разнообразия видов, в физике - мера хаоса и так далее. Но мы остановимся на понятиях, близких к информатике.

Информация - это некоторая упорядоченная последовательность сообщений, отражающих, передающих и увеличивающих наши знания.

Информация - это сведения об окружающем мире (объекте, процессе, явлении, событии), которые являются объектом преобразования (включая хранение, передачу и т.д.) и используются для выработки поведения, для принятия решения, для управления или для обучения.
Для информации необходимо понятие алфавита.

Алфавит - конечное множество различных знаков (букв), символов, для которых определена операция *конкатенации* (присоединения символа к символу или цепочке символов).

Знак (буква) - любой элемент алфавита (элемент x алфавита X, где $x \in X$).

Слово в алфавите (или над алфавитом) - конечная последовательность знаков (букв) алфавита.

Длина некоторого слова в алфавите (над алфавитом) - число составляющих его букв.

Словарь (словарный запас) - множество различных слов в алфавите (над алфавитом).

4.2 Признаки классификации информации

Существуют следующие признаки классификации информации:

- Отношение к источнику или приемнику (входная, выходная и внутренняя);
- Отношение к конечному результату (исходная, промежуточная и результирующая);
- Актуальность;
- Адекватность;
- Доступность (открытая, закрытая);
- Понятность;
- Полнота (достаточная, недостаточная, избыточная);
- Достоверность;
- Массовость;
- Изменчивость (постоянная, переменная, смешанная);
- Объективность;
- Точность;
- Стадия использования (первичная, вторичная);
- Ценность.

4.3 Измерение количества информации

Существует три большие группы методов получения информации.

Методы получения информации:

- *Эмпирические* - полученные опытным путем.

- *Теоретические* - опираются на рациональное познание (понятие, суждение, умозаключение) и логические процедуры вывода.
- *Эмпирико-теоретические* - смешанные.

Теперь нам известно понятие информации, но необходимо еще конкретно знать сколько этой информации. Поэтому есть два важных определения:

Количество информации - число, адекватно характеризующее разнообразие (неопределённость) в оцениваемой системе.

Мера информации - численная оценка количества информации, которая обычно задана неотрицательной аддитивной функцией (то есть, мера информации объединения событий (множеств) равна сумме мер каждого события).

4.3.1 Мера Хартли

Ральф Хартли
1888 – 1970

Пусть известны N состояний системы S (N опытов с различными, равновозможными, последовательными состояниями системы). Если каждое состояние системы закодировать двоичными кодами, то минимальная длина d полученного кода определяется из условия:

$$2^d \geq N \qquad \textit{или} \qquad d \geq \log_2 N$$

Значит, для однозначного описания системы требуется $\log_2 N$ бит. В общем случае количество информации в системе S равно:

$$H_s = \log_k N$$

Единицы измерения количества информации:

- Бит ($k = 2$)
- Трит ($k = 3$)
- Дит (харт) ($k = 10$)
- Нит (нат) ($k = e$)

Примеры использования меры Хартли

Пример 1:
Задание: мальчик загадывает число от 1 до 64. Какое количество вопросов типа "да-нет"понадобится, чтобы гарантированно угадать число?
Решение:
• Первый вопрос: "Загаданное число меньше 32?". Ответ: "Да".
• Второй вопрос: "Загаданное число меньше 16?". Ответ: "Нет".
...
• Шестой вопрос точно приведет к правильному ответу.
Значит, в соответствии с мерой Хартли в загадке мальчика содержится $\log_2 64 = 6$ бит информации ($N = 64$ так как возможно 64 вариантов загаданного числа).
Ответ: 6 бит.

Пример 2:
Задание: мальчик держит за спиной шахматного ферзя и собирается поставить его на произвольную клетку пустой доски. Какое количество информации содержится в его действии?
Решение: шахматная доска имеет размеры 8×8 клеток. Ферзь может быть как белым, так и черным, поэтому количество равновероятных состояний будет равно $8 \times 8 \times 2 = 128$. Получается, количество информации по мере Хартли равно $\log_2 128 = 7$ бит.
Ответ: 7 бит.

Мера Хартли подходит лишь для идеальных, абстрактных систем, так как в реальных системах состояния системы неодинаково осуществимы (неравновероятны).

4.3.2 Мера Шеннона

Клод Шеннон
1916 – 2001

Если состояния системы не равновероятны, используют меру Шеннона. Мера Шеннона оценивает информацию отвлеченно от ее смысла:

$$I = -\sum_{i=1}^{N} p_i \times \log_2 p_i$$

Где:
I - количество информации, выраженное в битах (в $\log_k p_i$ $k = 2$);
N - число состояний системы;
p_i - вероятность (относительная частота) перехода системы в i-е состояние (вероятность того, что система

находится в состоянии i)
Сумма всех p_i должна быть равна единице.

Примеры использования меры Шеннона

Пример 1:
Задание: девочка наугад вытаскивает из мешка мяч. Известно, что в мешке всего 8 мячей, из них: 4 красных, 2 синих, 1 зеленый и 1 белый. Какое количество информации содержится в этом событии?
Решение:
• Вероятность вытащить красный мяч равна ${}^4/_8 = 0,5$
• Вероятность вытащить синий мяч равна ${}^2/_8 = 0,25$
• Вероятность вытащить зеленый мяч равна ${}^1/_8 = 0,125$
• Вероятность вытащить белый мяч равна ${}^1/_8 = 0,125$
Значит количество информации, выраженное в битах равно: $I = -(0,5 \times \times \log_2 0,5 + 0,25 \times \log_2 0,25 + 0,125 \times \log_2 0,125 + 0,125 \times \log_2 0,125) = = -(-0,5 \times 1 - 0,25 \times 2 - 0,125 \times 3 - 0,125 \times 3) = -(-0,5 - 0,5 - 0,375 - - 0,375) = 1,75$ бит.
Ответ: 1,75 бит

Глава 5

Сжатие данных

Сжатие данных - это процесс, обеспечивающий уменьшение объема данных путем сокращения их избыточности.

К. Шеннон

Сжатие данных - это частный случай *кодирования* данных

Кодирование - процесс преобразования символов алфавита в символы алфавита Y.
Декодирование - процесс, обратный кодированию.
Символ - наименьшая единица данных, рассматриваемая как единое целое при кодировании/декодировании.
Алфавит - множество всех возможных символов.

Рассмотрим на примере. Возьмем обычную книгу и будем считать ее содержимое как за исходные данные. За символ мы можем взять как букву (и тогда алфавитом будет являться обычный алфавит русского языка), так и целое слово (тогда алфавит будет состоять из всех уникальных слов, встречающихся в этой книге). Символ - не обязательно один знак. Символом могут являться слова и даже целые предложения.

Кодовое слово - последовательность символов из алфавита Y, однозначно обозначающая конкретный символ алфавита .
Средняя длина кодового слова - это величина, которая вычисляется как взвешенная вероятностями сумма длин всех кодовых слов.
То есть:

$$L = \sum_{i=1}^{N} p_i \times l_i$$

Где:
N - количество кодовых слов в алфавите;
p_i - вероятность появления кодового слова в последовательности;
l_i - количество символов в кодовом слове (длина кодового слова).
Сумма всех p_i должна быть равна единице.

Если все кодовые слова имеют одинаковую длину, то код называется **равномерным** (фиксированной длины). Если встречаются слова разной длины, то - **неравномерным** (переменной длины)

Классический пример равномерного кода - таблица символов ASCII. Любой символ из этой таблицы будет закодирован одним байтом (один символ всегда кодируется двумя цифрами в шестнадцатеричной системе счисления).
Пример неравномерного кода - азбука Морзе.

Характеристики кодирования

$$\text{Коэффициент сжатия} = \frac{\text{Размер входного потока}}{\text{Размер выходного потока}}$$

$$\text{Отношение сжатия} = \frac{\text{Размер выходного потока}}{\text{Размер входного потока}}$$

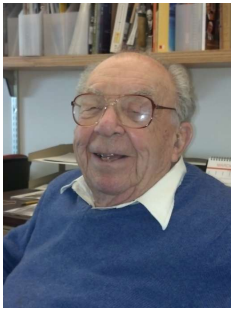
Существует два вида сжатия данных:

Сжатие без потерь (*полностью обратимое*) - сжатые данные после декодирования (распаковки) не отличаются от исходных.
Сжатие с потерями (*частично обратимое*) - сжатые данные после декодирования (распаковки) отличаются от исходных, так как при сжатии часть исходных данных была отброшена для увеличения коэффициента сжатия.

Префиксный код - это код, в котором никакое кодовое слово не является префиксом любого другого кодового слова. Эти коды имеют переменную длину.

Оптимальный префиксный код - это префиксный код, имеющий минимальную среднюю длину.

5.1 Алгоритм Шеннона-Фано



Роберт Фано
род.1917

Алгоритм Шеннона-Фано - один из первых алгоритмов сжатия. Его сформулировали два ученых - Клод Шеннон и Роберт Фано. Алгоритм основан на частоте повторения. Так, часто встречающийся символ кодируется кодом меньшей длины, а редко встречающийся - кодом большей длины. Коды, полученные при кодировании, префиксные, что позволяет декодировать любую последовательность.

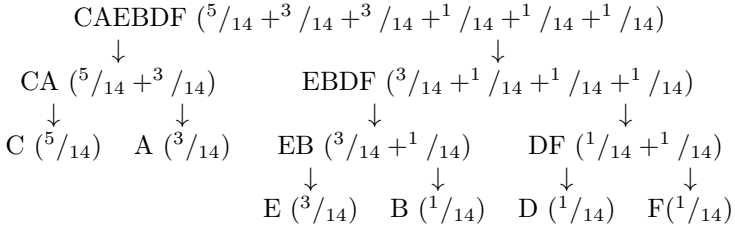
Алгоритм Шеннона-Фано для некоторых последовательностей может сформировать неоптимальные коды.

Алгоритм Шеннона-Фано

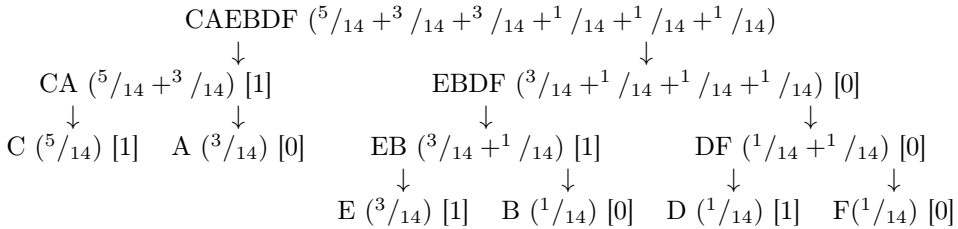
1. Символы входного (первичного) алфавита выписывают по убыванию вероятностей - это корень будущего дерева;
2. Строится дерево от корня к листьям. Находится середина, которая делит корень на два узла. Эти узлы (суммы вероятностей символов алфавита) примерно равны;
3. Полученные узлы - листья дерева. Левому узлу (с большей суммарной вероятностью) присваивается значение 1, а правому - 0;
4. Шаги 2-3 повторяются, пока в листьях дерева не останется один символ первичного алфавита.
5. Символ входного (первичного) алфавита кодируется последовательностью нулей и единиц в соответствии с распределением их от корня к листьям (узлам).

Пример 1:
Задание: составить код Шеннона-Фано для последовательности $AAABCCCCCDEEEF$. Найти среднюю длину кодового слова.
Решение: в последовательности AAABCCCCCDEEEF алфавит состоит из 4 символов: A, B, C, D. Выпишем символы первичного алфавита по убыванию вероятностей:
• Вероятность символа C - $^{5}/_{14}$;
• Вероятность символа A - $^{3}/_{14}$;
• Вероятность символа E - $^{3}/_{14}$;

• Вероятность символа B - $^1/_{14}$;
• Вероятность символа D - $^1/_{14}$;
• Вероятность символа F - $^1/_{14}$;
Полученная последовательность $CAEDBF$ является корнем будущего дерева.
Построим дерево от корня к листьям:



Присвоим левому символу (с большей вероятностью) значение 1, а правому - 0:

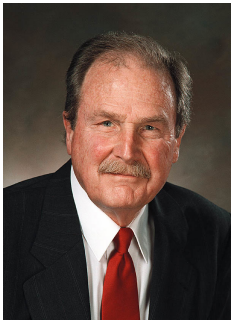


Получим следующую таблицу для кодировки:

Символ	Вероятность	Код
C	$^5/_{14}$	11
A	$^3/_{14}$	10
E	$^3/_{14}$	011
B	$^1/_{14}$	010
D	$^1/_{14}$	001
F	$^1/_{14}$	000

Исходная последовательность AAABCCCCCDEEEF кодируется следующей: 10.10.10.010.11.11.11.11.11.001.011.011.011.000 – 34 бита
Средняя длина кодового слова: $^5/_{14} \times 2 + ^3/_{14} \times 2 + ^3/_{14} \times 3 + ^1/_{14} \times 3 + + ^1/_{14} \times 3 + ^1/_{14} \times 3 = ^{34}/_{14} \approx 2,4$

5.2 Код Хаффмана



Дэвид Хаффман
1925 – 1999

Код (алгоритм) Хаффмана был разработан в 1952 году аспирантом Массачусетского технологического института Дэвидом Хаффманом при написании им курсовой работы.
Как и алгоритм Шеннона-Фано, основан на частоте повторения. Зная вероятности символов в сообщении, можно описать процедуру построения кодов переменной длины, состоящих из целого количества битов. Символам с большей вероятностью ставятся в соответствие более короткие коды. Коды Хаффмана обладают свойством префиксности, что позволяет однозначно их декодировать.
Однако, в отличие от кодов Шеннона-Фано, коды Хаффмана всегда являются оптимальными.
Сжатие данных по Хаффману применяется при сжатии фото- и видеоизображений (JPEG, стандарты сжатия MPEG), в архиваторах (PKZIP, LZH), в протоколах передачи данных MNP5 и MNP7.

Алгоритм Хаффмана для неоптимальных префиксных кодов

1. Символы входного алфавита образуют список свободных узлов. Каждый узел имеет вес, равный вероятности появления символа в сжимаемом тексте (исходной последовательности). Строится дерево от листьев (узлов) к корню:

2. Выбираются два свободных узла дерева с наименьшими весами;

3. Создается их родитель с весом, равным их суммарному весу;

4. Родитель добавляется в список свободных узлов, а двое его детей удаляются из этого списка.

5. Одной дуге, выходящей из родителя (узлу с большим весом), ставится в соответствие значение 1, а другой (узлу с меньшим весом) значение 0.

6. Повторяем шаги 2-4, выбирая в качестве одного из свободных узлов родителя, до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.

7. Символ входного (первичного) алфавита кодируется последовательностью нулей и единиц в соответствии с распределением их от корня дерева к узлам (листьям).

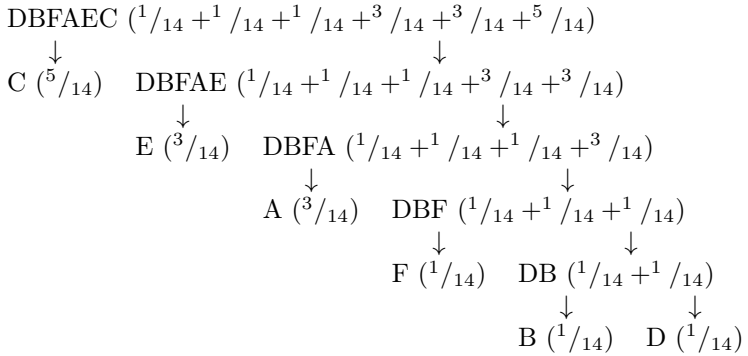
Пример 1:

Задание: составить код Хаффмана для неоптимальных префиксных кодов для последовательности AAABCCCCCDEEEF. Найти среднюю длину кодового слова.

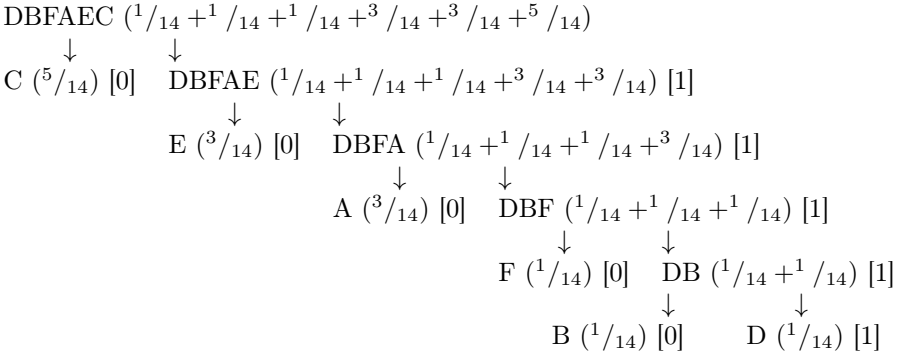
Решение: составим список свободных узлов для исходной последовательности:

- Символ A с вероятностью $^{3}/_{14}$;
- Символ B с вероятностью $^{1}/_{14}$;
- Символ C с вероятностью $^{5}/_{14}$;
- Символ D с вероятностью $^{1}/_{14}$;
- Символ E с вероятностью $^{3}/_{14}$;
- Символ F с вероятностью $^{1}/_{14}$;

Построим кодовое дерево:



Одной дуге, выходящей из родителя (узлу с большим весом), присвоим значение 1, а другой (узлу с меньшим весом) - значение 0.



Получим следующую таблицу для кодировки:

Символ	Код
A	110
B	11110
C	0
D	11111
E	10
F	1110

Исходная последовательность AAABCCCCCDEEEF кодируется следующей: 110.110.110.11110.0.0.0.0.0.11111.10.10.10.1110 - 34 бита.
Средняя длина кодового слова: ${}^{5}/_{14} \times 1 + {}^{3}/_{14} \times 2 + {}^{3}/_{14} \times 3 + {}^{1}/_{14} \times 4 + {}^{1}/_{14} \times 5 + {}^{1}/_{14} \times 5 = {}^{34}/_{14} \approx 2,4$

Алгоритм Хаффмана для оптимальных префиксных кодов

1. Символы входного (первичного) алфавита выписывают по убыванию вероятностей (весов) в таблицу.
2. Выбираются два свободных узла (элемента) с наименьшими весами (вероятностями);
3. Верхнему узлу (с большим весом) присваивается значение 1, а нижнему (с меньшим весом) - 0;
4. Создается их родитель с весом, равным их суммарному весу;
5. Родитель добавляется в список свободных узлов (таблицу), занимая соответствующее место в списке убывающих во величине весов (вероятностей), а двое его детей удаляются из этого списка.
6. Повторяем шаги 2-5, до тех пор, пока в списке свободных узлов (в таблице) не останется только два свободных узла (элемента).
7. Символ входного (первичного) алфавита кодируется последовательностью нулей и единиц в соответствии с распределением их от корня к узлам.

Пример 2:
Задание: составить код Хаффмана для неоптимальных префиксных кодов для последовательности AAABCCCCCDEEEF. Найти среднюю длину кодового слова.
Решение: составим список свободных узлов для исходной последовательности:

Символ	Вероятность, p
C	$^{5}/_{14}$
A	$^{3}/_{14}$
E	$^{3}/_{14}$
B	$^{1}/_{14}$
D	$^{1}/_{14}$
F	$^{1}/_{14}$

Построим таблицу:

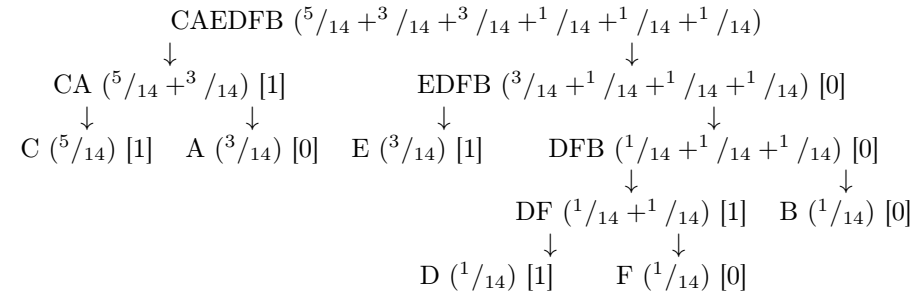
C	$^{5}/_{14}$	C	$^{5}/_{14}$	C	$^{5}/_{14}$	EDFB	$^{6}/_{14}$	CA	$^{8}/_{14}$
A	$^{3}/_{14}$	A	$^{3}/_{14}$	A	$^{3}/_{14}$	C	$^{5}/_{14}$	EDFB	$^{6}/_{14}$
E	$^{3}/_{14}$	E	$^{3}/_{14}$	E	$^{3}/_{14}$	A	$^{3}/_{14}$		
B	$^{1}/_{14}$	DF	$^{2}/_{14}$	DFB	$^{3}/_{14}$				
D	$^{1}/_{14}$	B	$^{1}/_{14}$						
F	$^{1}/_{14}$								

Верхнему узлу (с большим весом) присвоим значение 1, а нижнему (с меньшим весом) - 0:

C	$^{5}/_{14}$	C	$^{5}/_{14}$	C	$^{5}/_{14}$	EDFB	$^{6}/_{14}$
A	$^{3}/_{14}$	A	$^{3}/_{14}$	A	$^{3}/_{14}$	C [1]	$^{5}/_{14}$
E	$^{3}/_{14}$	E	$^{3}/_{14}$	E [1]	$^{3}/_{14}$	A [0]	$^{3}/_{14}$
B	$^{1}/_{14}$	DF [1]	$^{2}/_{14}$	DFB [0]	$^{3}/_{14}$		
D [1]	$^{1}/_{14}$	B [0]	$^{1}/_{14}$				
F [0]	$^{1}/_{14}$						

CA [1]	$^{8}/_{14}$
EDFB [0]	$^{6}/_{14}$

Построим кодовое дерево:



Получим следующую таблицу для кодировки:

Символ	Код
A	10
B	000
C	11
D	0011
E	01
F	0010

Исходная последовательность AAABCCCCCDEEEF кодируется следующей: 10.10.10.000.11.11.11.11.11.0011.01.01.01.0010 - 33 бита.
Средняя длина кодового слова: $^5/_{14} \times 2 + {}^3/_{14} \times 2 + {}^3/_{14} \times 2 + {}^1/_{14} \times 3 + {}^1/_{14} \times 4 + {}^1/_{14} \times 4 = {}^{31}/_{14} \approx 2,2$

5.3 Кодирование длин серий

Кодирование длин серий (кодирование повторов) - алгоритм сжатия данных, заменяющий повторяющиеся символы (серии) на один символ и число его повторов. Серией называется последовательность, состоящая из нескольких одинаковых символов. При кодировании (упаковке, сжатии) строка одинаковых символов, составляющих серию, заменяется строкой, содержащей сам повторяющийся символ и количество его повторов.
Кодирование длин серий активно применяется при сжатии графических изображений, таких как иконки или графические изображения.

Пример 1:
Задание: применить кодирование длин серий для последовательности *AAAAABAAAAABBBBBAAAAA*.
Решение: посчитаем количество повторяющихся символов:
1. 5 символов A;
2. 1 символ B;

3. 5 символов A;
4. 5 символов B;
5. 5 символов A;

Итого, найдено 5 серий. Заменим серии на число повторов и сам повторяющийся символ: $5A1B5A5B5A$. Получилась последовательность из 10 символов. Исходная последовательность состояла из 21 символа.
Коэффициент сжатия: $\frac{21}{10} = 2,1$.

Глава 6

Помехоустойчивое кодирование

При обработке данных (хранение данных в памяти, передача по каналам связи) существует вероятность битовых ошибок. Они могут возникать из-за альфа частиц от примесей в чипе микросхемы или от нейтронов из фонового космического излучения. Частота единичных битовых ошибок на 1 гигабайт данных составляет от 1 раза в час до 1 раза в тысячелетие. По данным исследования Google получилось, что 1 раз в сутки. Есть несколько способов обработки данных, полученных при передаче с ошибкой:

- *Использовать полученные данные без проверки на ошибки.* Для такого способа не нужно выполнять лишних действий. Например, при передаче текста или видеоизображения ошибка в одном блоке (слове, кадре) не исказит общего смысла.

- *Обнаружить ошибку, выполнить запрос повторной передачи поврежденного блока.* В случае, если пользователь обнаружил ошибку, потребуется передать не блок, а целое сообщение заново, и при передаче файлов большого размера (видеоизображений, фотоизображений и музыкальных файлов высокого качества) это очень проблематично. Если ошибку обнаружила операционная система, то она обнаружила ее еще при передаче данных, и заново передается только поврежденный блок.

- *Обнаружить ошибку и отбросить поврежденный блок.* Как и первый способ, он весьма оправдан при передаче текста или видеоизображений.

- *Обнаружить и исправить ошибку.* Не тратится время на повторную передачу данных, но полученные данные корректны. Однако, при таком способе необходимо применять особые методы кодирования и наряду с информационными битами передавать служебные, которые позволяют исправить ошибку.

Последний способ характеризует помехоустойчивый код. **Помехоустойчивые коды** - это коды, позволяющие обнаружить и (или) исправить ошибки в кодовых словах, которые возникают при передачи по каналам связи.

Классификация помехоустойчивых кодов

- **Блочные** - фиксированные блоки информации длиной k символов преобразуются в блоки длиной n символов (независимо друг от друга). Например, при передаче файла объемом в 1 гигабайт он делится на равные блоки по 1 килобайту и каждый килобайт снабжается служебной информацией, которая позволяет понять - корректен данный блок или нет. И при обнаружении некорректного блока передается только он.
  - **Неравномерные** - редко используемые символы кодируются большим количеством символов (имеют большую длину) (азбука Морзе)
  - **Равномерные** - длина блока (символа) постоянна (таблица ASCII).
    - **Неразделимые** - коды с постоянной плотностью единиц: информационные и проверочные разряды неразделимы (каждый блок данных на входе получает служебные биты, позволяющие обнаружить ошибки, которые далее не отделяются).
    - **Разделимые** - можно отделить (выделить) служебные биты от информационных.
      - **Систематические (линейные)** - циклические коды, биты четности/нечетности, код Хэмминга, код Рида-Соломона, код Боуза-Чоудхури-Хоквингема.
      - **Несистематические** - коды с контрольным суммированием.
- **Непрерывные** - передаваемая информационная последовательность не разделяется на блоки. Кодирование осуществляется целого потока.

– **Сверточные** - корректирующие ошибки коды; работают с непрерывным потоком данных, кодируя их при помощи регистров сдвига с линейной обратной связью.

Помехоустойчивый код характеризуется:

i - числом информационных разрядов;
r - числом проверочных разрядов;
n - общим числом разрядов ($n = k + r$);

Коэффициент избыточности: $КИ = \frac{r}{n}$;

6.1 Кодирование с помощью бита четности

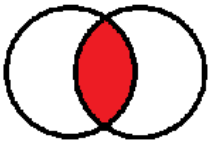
Контрольная сумма (check sum) - некоторое число, рассчитанное путем применения определенного алгоритма к набору данных и используемое для проверки целостности этого набора данных при их передаче или хранении.

Бит четности - частный случай контрольной суммы, представляющий из себя 1 контрольный бит, используемый для проверки четности количества единичных битов в двоичном числе.

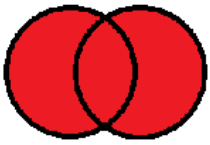
Метод расчета бита четности - суммирование по модулю 2 всех бит числа.

Сумма по модулю 2 - исключающее "ИЛИ"(для двух операндов), логическое или битовое сложение, разность двух/трех множеств.

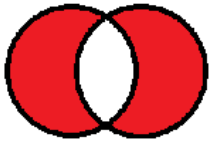
Обозначение: $A \ mod 2 \ B = A \oplus B$



$A \wedge B$



$A \vee B$



$A \oplus B$

$A \oplus B = (\neg(A \wedge B)) \wedge (A \vee B) = \neg((A \wedge B) \vee (\neg A \wedge \neg B))$

Таблица истинности для $A \oplus B$

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

A	B	C	A ⊕ B ⊕ C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

В случае двух переменных результат выполнения операции является истинным тогда и только тогда, когда лишь один из аргументов является истинным.
Для функции трех и более переменных результат выполнения операции будет истинным только тогда, когда количество аргументов, равных 1, - нечетное.

Пример обнаружения ошибок

Допустим, у нас есть один информационный бит $i = 1$. К нему идет один r_1 - бит четности, проверочный разряд №1.
$i = r_1$, $i \oplus r_1 = 0$.
При получении данных произошел сбой и данные некорректны. Нам известно, что сумма по модулю 2 информационного бита и бита четности равна 0.

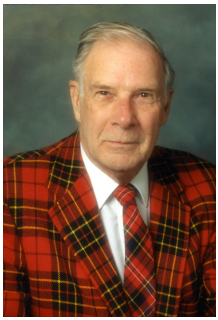
i исх	r_1 исх	i рез	r_1 рез	i рез ⊕ r_1 рез
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Соответственно, биты с пометкой "исх" - исходные, а "рез" - результирующие.
В первом случае видно, что несмотря на то, что оба бита пришли с ошибкой, сумма по модулю 2 равна нулю (верна), а значит компьютер не посчитает это как за ошибку.
Во втором и третьем случаях сумма по модулю 2 результирующих данных не верна, равна 1. Ошибка.
В последнем случае все верно. Биты корректны, сумма по модулю 2 верна. Ошибки нет.

Способ обнаружения ошибок с помощью бита четности применяется в RAID-хранилищах (RAID - (англ. redundant array of independent disks -

избыточный массив независимых дисков) технология виртуализации данных, которая объединяет несколько дисков в логический элемент для избыточности и повышения производительности).

6.2 Код Хэмминга



Ричард Уэсли Хэмминг
1915 – 1998

Код Хэмминга - блочный равномерный разделимый самокорректирующийся код. Исправляет одиночные битовые ошибки, возникшие при передаче или хранении данных.

На каждые i информационных бит используется r проверочных.

Значение каждого контрольного бита зависит от значений информационных бит: контрольный бит с номером N контролирует все последующие N бит через каждые N бит, начиная с позиции N.
Синдром последовательности S - набор контрольных сумм информационных и проверочных разрядов

Рассмотрим таблицу:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
2^k	r_1	r_2	i_1	r_3	i_2	i_3	i_4	r_4	i_5	i_6	i_7	i_8	i_9	i_{10}	S
1	X		X		X		X		X		X		X		s_1
2		X	X			X	X			X	X			X	s_2
4				X	X	X	X					X	X	X	s_3
8								X	X	X	X	X	X	X	s_4

Данная таблица представлена для кода из 14 бит, но ее можно расширить для нужного размера самостоятельно.
Сверху (в первой строчке) указан номер бита, а справа - синдром.
Знаком X обозначены те биты, которые контролирует проверочный бит, с номером, который указан в левом столбце (степень двойки). Чтобы узнать какими битами контролируется бит с номером N надо просто разложить N по степеням двойки.
Например, видно, что проверочный бит r_1 контролирует информационные биты i_1, i_2, i_4, i_5, i_7, i_9. А 11 бит (i_7) контролируется битами 1 (r_1), 2 (r_2) и 8 (r_4).
Чтобы узнать значение проверочного бита необходимо сложить по модулю 2 все информационные биты, которые он контролирует.

В данном случае:
$r_1 = i_1 \oplus i_2 \oplus i_4 \oplus i_5 \oplus i_7 \oplus i_9$;
$r_2 = i_1 \oplus i_3 \oplus i_4 \oplus i_6 \oplus i_7 \oplus i_1 0$;
и так далее.
Чтобы узнать значение синдрома, необходимо сложить по модулю 2 все биты, которые содержит синдром.
В данном случае:
$s_1 = r_1 \oplus i_1 \oplus i_2 \oplus i_4 \oplus i_5 \oplus i_7 \oplus i_9$;
$s_2 = r_2 \oplus i_1 \oplus i_3 \oplus i_4 \oplus i_6 \oplus i_7 \oplus i_{10}$;
и так далее.
Синдром последовательности S определяется составляющими синдромами s_1, s_2 и так далее. То есть для кода, где $r = 3$, синдром будет иметь следующий формат: $S(s_1; s_2; s_3)$.

Определение минимального количества контрольных разрядов:

$$2^k \geq r + i + 1$$

.
Классические коды Хэмминга с маркировкой (n, i): (7, 4); (15, 11); (31, 26).

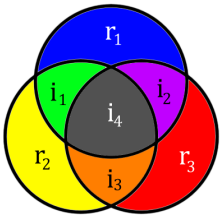
Разбрем код Хэмминга для $r = 3$.

Код Хэмминга для $r = 3$

В данном случае, у нас 7 бит, из них 4 информационных и 3 проверочных. То есть, код имеет маркировку (7, 4).
Составим таблицу кода (7, 4):

	1	2	3	4	5	6	7	
2^k	r_1	r_2	i_1	r_3	i_2	i_3	i_4	S
1	X		X		X		X	s_1
2		X	X			X	X	s_2
4				X	X	X	X	s_3

Рассмотрим конкретно контроль информационных бит проверочными на кругах Эйлера:



Видно, что r_1 контролирует i_1, i_2 и i_4; r_2 контролирует i_1, i_3 и i_4; а r_3 контролирует i_2, i_3 и i_4.
$r_1 = i_1 \oplus i_2 \oplus i_4$
$r_2 = i_1 \oplus i_3 \oplus i_4$
$r_3 = i_2 \oplus i_3 \oplus i_4$

$s_1 = r_1 \oplus i_1 \oplus i_2 \oplus i_4$
$s_2 = r_2 \oplus i_1 \oplus i_3 \oplus i_4$
$s_3 = r_3 \oplus i_2 \oplus i_3 \oplus i_4$

Рассмотрим таблицу значений синдрома $(s_1; s_2; s_3)$ и позицию ошибочного бита в сообщении:

Синдром $(s_1; s_2; s_3)$	Конфигурация ошибок	Ошибочный символ
000	НЕТ	НЕТ
001	0001000	r_3
010	0100000	r_2
011	0000010	i_3
100	1000000	r_1
101	0000100	i_2
110	0010000	i_1
111	0000001	i_4

Возьмем, к примеру, 2 строку: синдром $S(0, 0, 1)$, это значит что $s_1 = 0$, $s_2 = 0$, $s_3 = 1$. По построенной таблице кода (7,4) находится, в каком бите ошибка - какой бит содержится только в 3 синдроме. Проще говоря, в каком столбце X стоит только в 3 строчке (напротив s_3). По таблице видим, что такой бит - 4 (поэтому во втором столбце 0001000 - нули означают правильный бит, а единица - ошибочный. В данном случае ошибочный бит 4, поэтому он равен единице). Смотрим, какой именно бит занимает четвертое место - r_3.
Чтобы получить правильную последовательность, необходимо инвертировать ошибочный бит.

Пример 1:
Задание: получена последовательность 1100100. Вычислить ошибочный бит, записать правильную последовательность.
Решение: составим таблицу кода (7,4) с конкретной последовательностью.

	1	2	3	4	5	6	7	
	1	1	0	0	1	0	0	
2^k	r_1	r_2	i_1	r_3	i_2	i_3	i_4	S
1	X		X		X		X	s_1
2		X	X			X	X	s_2
4				X	X	X	X	s_3

Рассчитаем значение контрольных бит результата:

r_1 рез $= i_1 \oplus i_2 \oplus i_4 = 0 \oplus 1 \oplus 0 = 1$
r_2 рез $= i_1 \oplus i_3 \oplus i_4 = 0 \oplus 0 \oplus 0 = 0$
r_3 рез $= i_2 \oplus i_3 \oplus i_4 = 1 \oplus 0 \oplus 0 = 1$
Рассчитаем синдромы:
$s_1 = r_1 \oplus i_1 \oplus i_2 \oplus i_4 = r_1$ рез $\oplus\, r_1$ исх $= 1 \oplus 1 = 0$
$s_2 = r_2 \oplus i_1 \oplus i_3 \oplus i_4 = r_2$ рез $\oplus\, r_2$ исх $= 0 \oplus 1 = 1$
$s_3 = r_3 \oplus i_2 \oplus i_3 \oplus i_4 = r_3$ рез $\oplus\, r_3$ исх $= 1 \oplus 0 = 1$
Полученный синдром: $S(0, 1, 1)$.
Смотрим по таблице, какой бит содержится в s_2 и s_3 одновременно. 6, то есть i_3.
Инвертируем ошибочный бит и получаем правильную последовательность.
Ответ: 6 бит (i_3), правильная последовательность: 1100110.

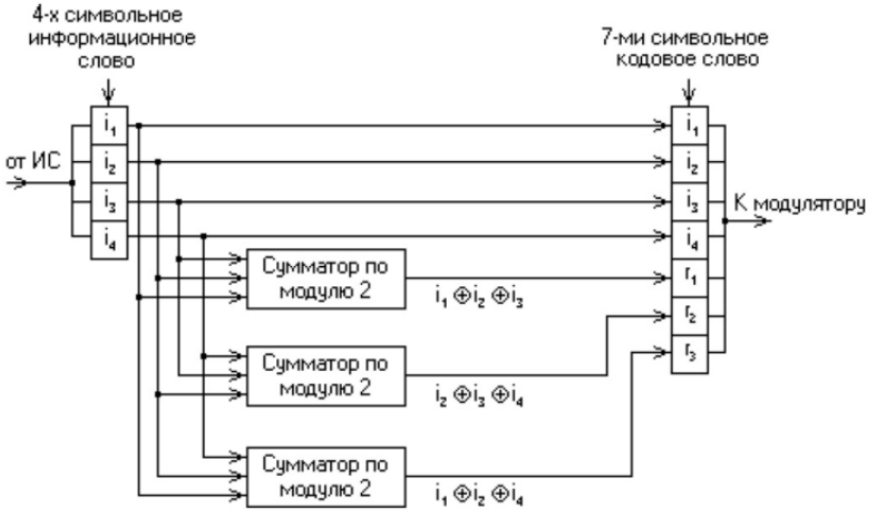


Рис. 6.1: Схема создания кода Хэмминга (7,4)

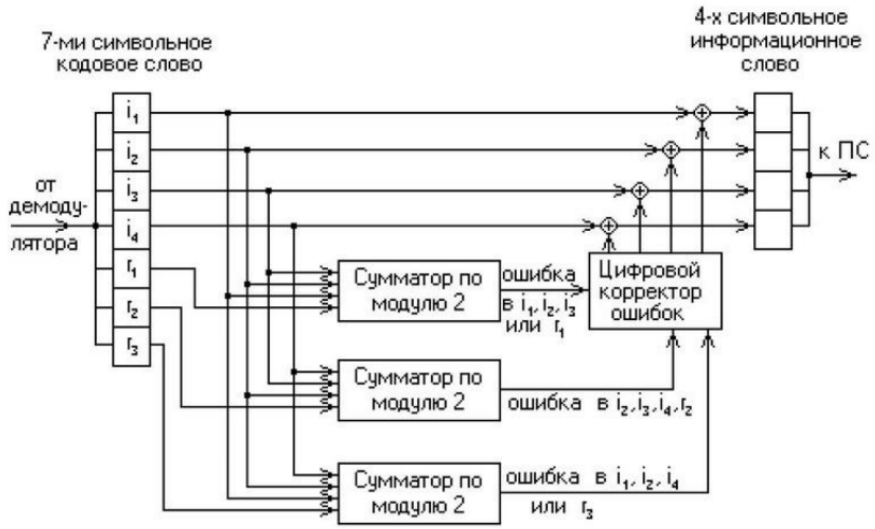
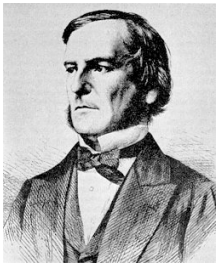


Рис. 6.2: Схема декодирования кода Хэмминга (7,4)

Глава 7

Алгебра логики

7.1 Определение



Джордж Буль
1815 – 1864

Алгебра двоичной логики - раздел математики, в котором изучаются логические операции над высказываниями. Чаще всего предполагается, что высказывания могут быть только истинными или ложными, то есть используется так называемая бинарная или двоичная логика. Один из основателей алгебры логики - Джордж Буль.

Логическая (булева) переменная – такая переменная, значения которой могут быть лишь "1"или "0".
В естественном языке: "булева переменная"= "высказывание".

Высказывание - утверждение, про которое можно однозначно сказать, истинно оно или ложно.
Обозначения:

- "истина" и "ложь"
- "true" и "false"
- "1" и "0"

Например, высказывание "Москва - столица России истина, а "трава синего цвета ложь.

Логическая (булева) функция $f(x, y, z, \dots)$ - в результате выполнения логических операций над логическими переменными $x, y, z, \dots$ получает значение 0 или 1.

Основные операции

$\overline{x}$ - отрицание или инверсия
$x \vee y$ - дизъюнкция или логическое сложение
$x \wedge y$ - конъюнкция или логическое умножение

7.2 Основные тождества

1. Аксиома двойного отрицания

$$\overline{\overline{x}} = x$$

2. Аксиома существования 1 и 0

$$0 = \overline{1} \qquad x \vee \overline{x} = 1$$

$$1 = \overline{0} \qquad x \wedge \overline{x} = 0$$

3. Закон идемпотенции

$$x \vee x = x \qquad x \wedge x = x$$

4. Закон коммутативности

$$x \vee y = y \vee x \qquad x \wedge y = y \wedge x$$

5. Закон поглощения

$$x \vee (x \wedge y) = x \qquad x \wedge (x \vee y) = x$$

6. Закон ассоциативности

$$x \vee (y \vee z) = (x \vee y) \vee z \qquad x \wedge (y \wedge z) = (x \wedge y) \wedge z$$

7. Закон дистрибутивности

$$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z) \qquad x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$$

8. Законы де Моргана

$$\overline{x \vee y} = \overline{x} \wedge \overline{y} \qquad \overline{x \wedge y} = \overline{x} \vee \overline{y}$$

9. Закон нейтральности

$$x \vee (y \wedge \overline{y}) = x \qquad x \wedge (y \vee \overline{y}) = x$$

7.3 Таблица истинности

Существует только 4 различные логические функции одной переменной $F(x)$. Любая другая функция (даже самая сложная вида $F(x) = \overline{x} \vee x \wedge \overline{x}$) будет иметь одну из следующих таблиц истинности:

x	$F_0(x)$	$F_1(x)$	$F_2(x)$	$F_3(x)$
0	0	1	0	1
1	0	0	1	1

Например $F(x) = \overline{x}$ будет иметь таблицу истинности $F_1(x)$, а $F(x) = 1$ - $F_3(x)$

Обычно логическую функцию $F(x)$ обозначают как FK, N, где K - количество операндов, а N - число в десятичной системе счисления, которое при переводе в двоичную является таблицей истинности функции. Например, $F(x) = \overline{x}$ обозначается как $F1, 1$, так как одна переменная и $2_{10} = 01_2$, что является таблицей истинности функции (смотреть снизу вверх).

Для k переменных будет существовать $N = 2^{2^k}$ функций.
Рассмотрим таблицу истинности:

	Значения k штук булевых операндов				Значение булевых функций F			
	x_1	x_2	...	x_k	$FK, 0$	$FK, 1$	...	FK, N
L штук (от 00..0 до 11..1 сверху вниз)	0	0	...	0	0	1	...	1
	0	0	...	1	0	0	...	1
	...	...	...	...	...	...	...	...
	1	1	...	1	0	0	...	1

от N штук (от 00..0 до 11..1 слева направо)

Так как всего k переменных, то для них будет $L = 2^k$ строк в таблице истинности, а количество булевых функций будет равно $N = 2^L$. Отсюда:

$$\begin{matrix} L = 2^k \\ N = 2^L \end{matrix} \quad \Rightarrow N = 2^{2^k}$$

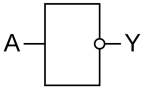
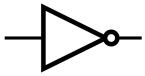
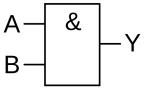
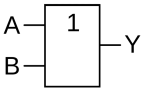
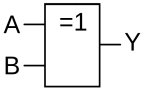
Так, для 1 переменной будет 4 булевых функции, для 2 переменных - 16 и так далее.

x	1	1	0	0	Обозначение	Название
y	1	0	1	0		
F	0	0	0	1	F2,1 $= x \downarrow y = xNORy =$ $= NOR(x,y) = x$НЕ-ИЛИ $y =$НЕ-ИЛИ(x,y)	Стрелка Пи́рса, НЕ-ИЛИ, антидизъюнкция
F	0	1	1	0	F2,6 $= x \oplus y =$ $= xXORy = XOR(x,y) =$ $= x >< y = x <> y =$ $= xNEy = NE(x,y)$	Сложение по модулю 2, исключающее "или" сумма Жегалкина, не равно
F	0	1	1	1	F2,7 = x | y = $= NAND(x,y) = xNAND$ $y = x$ НЕ-И $y =$ НЕ-И(x,y)	Штрих Шеффера, НЕ-И, 2И-НЕ, антиконъюнкция
F	1	0	0	0	F2,8 $= x \wedge y = x \cdot y =$ $= xy = x\&y = xANDy =$ $= AND(x,y) = x$И$y =$ $=$ И$(x,y) = min(x,y)$	Конъюнкция, 2И, минимум
F	1	0	0	1	F2,9 $= (x \equiv y) = x\ y =$ $= x \leftrightarrow y = xEQVy =$ $= EQV(x,y)$	Эквивалентность равенство
F	1	0	1	1	F2,11 $= x \rightarrow y = x \supset y =$ $= x \leq y = xLEy = LE(x,y)$	Импликация следование
F	1	1	1	0	F2,14 $= x \vee y = x + y =$ $=xORy = OR(x,y) =$ $= x$ ИЛИ $y =$ ИЛИ$(x,y) =$ $= max(x,y)$	Дизъюнкция, 2ИЛИ, максимум

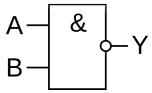
Таблица 7.1: Таблица значений и названий булевых функций от двух переменных

7.4 Обозначение на электрической схеме булевых функций

Булевы функции на электрической схеме обозначаются двумя способами, так как существует две организации, предложившие свои стандарты: IEC (англ. *International Electrotechnical Commission* - международная электротехническая комиссия) и ANSI (англ. *American national standards institute* - американский национальный институт стандартов). В основном встречаются обозначения IEC, но иногда можно увидеть и ANSI.

Название	IEC	ANSI
НЕ, $\overline{A}$ (Инвертор)		
И, $A \wedge B$		
ИЛИ, $A \vee B$		
Сумма по модулю 2, $A \oplus B$		

Это основные обозначения. Они могут совмещаться. Например, так будет выглядеть ИЛИ-НЕ:



Цена функции по Квайну – суммарное число входов логических элементов в составе схемы.
Минимизация функции – сокращение цены функции, с помощью преобразования её к более простому эквивалентному выражению. Наиболее простой вид получается при сведении функции к постоянной - 1 (истина) или 0 (ложь).

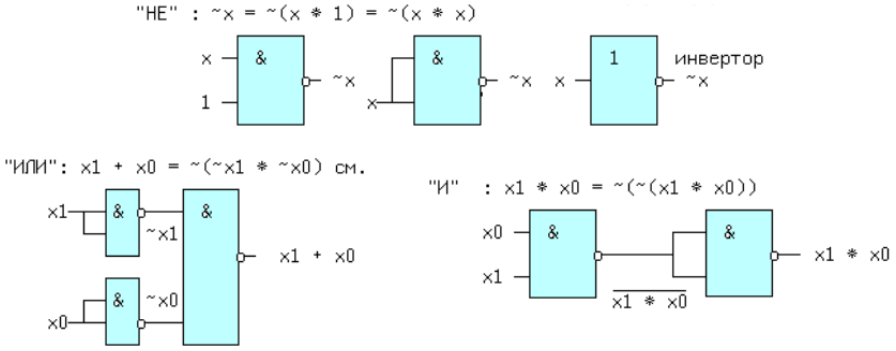
7.5 Логический базис

Логический базис - набор булевых функций, позволяющий реализовать любую другую булеву функцию.
Три наиболее востребованных логических базиса: И, ИЛИ, НЕ; ИЛИ-НЕ; И-НЕ.

7.6 Формы записи математических выражений

Форма записи по-другому называется нотацией.
"Арность операции"означает количество операндов, участвующих в операции.

Рис. 7.1: Пример реализации функций И, ИЛИ, НЕ в базисе И-НЕ



Например: $\sqrt{A}$ (унарная), $A \times B$ (бинарная).

Виды нотаций

1489 г. - инфиксная запись $A + B$

1920 г. - префиксная (польская) запись $+AB$

1957 г. - постфиксная (обратная польская) запись $AB+$

Стек - абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. *last in - first out*, "последним пришел - первым вышел"). Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно снять верхнюю.

7.6.1 Префиксная нотация

Пример:
Инфиксная нотация: $(A + B + C) - E^{D \times F \times G}$
Префиксная нотация: $- + +ABC\hat{}E \times \times DFG$

Рассмотрим, как же происходит запись в префиксной нотации:

1. Исходное выражение: $(A + B + C) - E^{D \times F \times G}$. Расставляем порядок выполнения операций, согласно математическим правилам.

2. Последним выполняется вычитание, а именно из $(A + B + C)$ вычитается $E^{D \times F \times G}$. **Ставим знак в начало.** Получается $-(A + B + C)(E^{D \times F \times G})$.

3. Рассмотрим $(A+B+C)$, расставим порядок действий. Сначала происходит сложение A и B, а потом $A+B$ и C. Начнем с последнего действия, получается: $+(A+B)C$. В $(A+B)$ происходит сложение A и B, получаем $(+AB)$. Совместим все вместе: $(+(+AB)C)$.

4. Рассмотрим $(E^{D\times F\times G})$, происходит возведение в степень. А именно E возводится в степень $D\times F\times G$. Получается $^\wedge E(D\times F\times G)$.

5. Рассмотрим $D\times F\times G$, расставляем порядок действий. Сначала происходит умножение D и F, а потом $D\times F$ и G. Начнем с последнего действия, получается: $\times(D\times F)G$. В $(D\times F)$ происходит умножение D и F, получаем $(\times DF)$. Совместим все вместе: $(\times(\times DF)G)$.

6. Совмещаем все вместе. Получается: $-(+(+AB)C)(^\wedge E(\times(\times DF)G)$. Убираем скобки. Получили $-++ABC^\wedge E\times\times DFG$.

Существует популярная Lisp-разновидность префиксной нотации (Lisp - семейство языков программирования). И в ней запись будет выглядеть так: $(-(+ABC)(^\wedge E(\times DFG)))$

Особенности префиксной нотации

- Не требуется скобок, если арность фиксирована.
- Запись выражения получается короче, чем инфиксная.
- Не требуется знать приоритет операций.
- Легко декодировать выражение с помощью стека.
- Малоприменима на практике (кроме Lisp).

7.6.2 Постфиксная нотация

Пример:
Инфиксная нотация: $(A+B+C)-E^{D\times F\times G}$
Постфиксная нотация: $CAB++EGDF\times\times^\wedge -$

Рассмотрим, как же происходит запись в постфиксной нотации:

1. Исходное выражение: $(A+B+C)-E^{D\times F\times G}$. Расставляем порядок выполнения операций, согласно математическим правилам.

2. Последним выполняется вычитание, а именно из $(A+B+C)$ вычитается $E^{D\times F\times G}$. **Ставим знак в конец.** Получается $(A+B+C)(E^{D\times F\times G})-$.

3. Рассмотрим $(A+B+C)$, расставим порядок действий. Сначала происходит сложение A и B, а потом $A+B$ и C. Начнем с последнего действия, получается: $(A+B)+$. В $(A+B)$ происходит сложение A и B, получаем $(AB+)$. Совместим все вместе: $(C(AB+)+)$.

4. Рассмотрим $(E^{D\times F\times G})$, происходит возведение в степень. А именно E возводится в степень $D\times F\times G$. Получается $E(D\times F\times G)^{\wedge}$.

5. Рассмотрим $D\times F\times G$, расставляем порядок действий. Сначала происходит умножение D и F, а потом $D\times F$ и G. Начнем с последнего действия, получается: $G(D\times F)\times$. В $(D\times F)$ происходит умножение D и F, получаем $(DF\times)$. Совместим все вместе: $(G(DF\times)\times)$.

6. Совмещаем все вместе. Получается: $(C(AB+)+)(E(G(DF\times)\times)^{\wedge})-$. Убираем скобки. Получили $CAB++EGDF\times\times^{\wedge}-$.

Особенности постфиксной нотации

- Не требуется скобок, если арность фиксирована.
- Запись выражения получается короче, чем инфиксная.
- Не требуется знать приоритет операций.
- Легко декодировать выражение с помощью стека.
- Успешно применяется в компиляторах, в небольшом количестве языков программирования и некоторых ЭВМ (калькуляторы "Электроника"и HP).

Алгоритм вычисления для постфиксной нотации

1. Обработка входного символа
   - Если на вход подан операнд, он помещается на вершину стека.
   - Если на вход подан знак операции, то соответствующая операция выполняется над требуемым количеством значений, извлеченных из стека, взятых в порядке добавления. Результат выполненной операции кладется на вершину стека.
2. Если входной набор символов обработан не полностью, перейти к шагу 1.
3. После полной обработки входного набора символов результат вычисления выражения лежит на вершине стека.

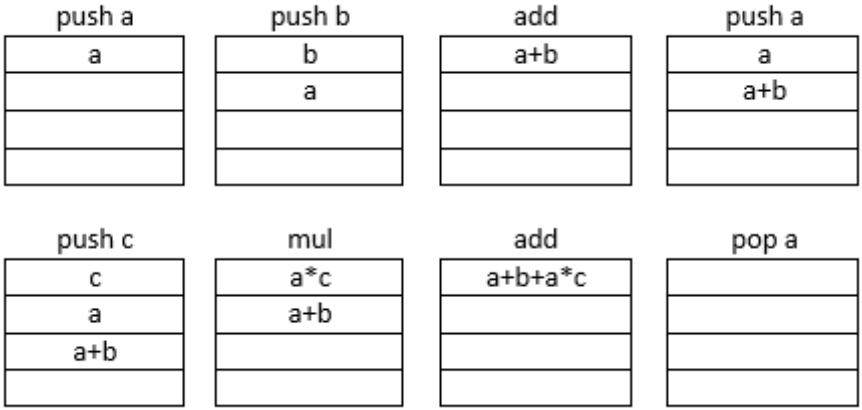


Рис. 7.2: Вычисления выражения $ab + ac \times +$ (в инфиксной нотации $a + + b + a \times c$)

Рассмотрим вычисление постфиксной нотации в стеке. Стек не имеет адресных операций. В нем есть две операции с переменными: *push* (положить) и *pop* (забрать). Если необходимо положить значение в стек, то он кладется с помощью команды *push* на вершину стека. При этом, все другие значения сдвигаются вниз. Арифметические операции (*add* - сложить, *mul* - умножить) выполняются с переменными (или переменной - зависит от арности операции), которые лежат на вершине стека.

Глава 8

Программное обеспечение

8.1 Офисное ПО

Компания Microsoft открыла формат doc (стандарт по которому он создается) только в 2000 г. До этого разработчикам приходилось методом обратного инженеринга вручную раскрывать этот стандарт. Это характеризует формат doc не в лучшую сторону - не все имели доступ (Microsoft Word - платное ПО) и формат проверен малым количеством людей. До сих пор существует много ошибок, которые исправлены только в docx.
Форматы odt и docx используют XML. Технология XML открыта - любой может посмотреть, найти ошибки, предложить исправление. Чтобы убедиться, что odt и docx используют XML, есть простой способ: создайте docx или odt документ, смените расширение на zip и распакуйте. Можно увидеть, что в основном все файлы имеют расширение xml.
Нормальная криптография появилась только в docx. В формате doc ничего нельзя было шифровать. Только сторонними приложениями.

Стоимость продуктов Microsoft Office

- Для дома и учебы (Word, Excel, PowerPoint, OneNote) $\approx$ 3000р.
- Для дома и бизнеса (Word, Excel, PowerPoint, OneNote, Outlook) $\approx$ 10000р.
- Профессиональный (Word, Excel, PowerPoint, OneNote, Outlook, Access, Publisher) $\approx$ 20000р.

LibreOffice, OpenOffice, Calligra Suite = 0р.

Уведомление
об утверждении стандарта

Обозначение документа		ГОСТ Р ИСО/МЭК 26300-2010	
Наименование документа (на русском языке)		Информационная технология. Формат Open Document для офисных приложений (OpenDocument) v1.0	
Наименование документа (на английском языке)		Information technology. Open Document Format for Office Applications (OpenDocument) v1.0	
Технический комитет РФ		22 - Информационные технологии	
Межгосударственный ТК			
Сведения о регистрации	800-ст	Дата регистрации	21.12.2010
Дата введения в действие документа		01.06.2011	
Введен (впервые, взамен, взамен в части)		Впервые	
Связь с другими НД		Идентичен ISO/IEC 26300:2006	

Рис. 8.1: ГОСТ Р ИСО/МЭК 26300-2010

В 2010 году был принят ГОСТ Р ИСО/МЭК 26300-2010, обязывающий госучреждения перейти на бесплатный формат документов (Open Document Format - ODF). Но это вовсе не означает, что будет использоваться LibreOffice и OpenOffice. В последних версиях Microsoft Office есть поддержка этого формата.

Факты о ODF

- Распоряжение Правительства Российской Федерации от 17 декабря 2010 г. №2299-р "О плане перехода федеральных органов исполнительной власти и федеральных бюджетных учреждений на использование свободного программного обеспечения (2011 - 2015 годы)"
- OpenDocument это единственный формат электронной документации, который реализован несколькими производителями ПО и утвержден ISO в качестве международного стандарта.
- Абсолютно любой производитель ПО может использовать формат OpenDocument при разработке своего собственного редактора электронных документов.
- Открытость формата OpenDocument позволяет пользователю быть свободным при выборе программного обеспечения, не зависеть от конкретного поставщика ПО и его маркетинговой политики.
- Исчезает угроза утери информации из-за изменения закрытых форматов

8.1.1 Сравнение возможностей OO и MS Office

Свойства	Open Office Calc	Microsoft Excel
Размерность	1 024 × 1 048 576	16 384 × 1 048 576
Кол-во цветов	104	16 777 216
Работа с датами	от 1 января 0001 г. до 31 декабря 9999г.	от 1 января 1900 г. до 31 декабря 9999г.
Поддержка графических форматов	met, pbm, pgm, ppm, psd, ras, sbm, sgg, svg, xpm, xbm	cdr, emz, mix, pcz, wmz, wpg, fpx, emz, drw
Прочее	Работа с MySQL. Макросы на разных языках (Python, JavaScript)	Продвинутые сводные таблицы и условное форматирование

Open Office Writer

- Частые обновления
- Независимые стили страниц в одном документе
- Автоматическое создание указателя формул
- Продвинутая навигация (по ссылкам, разделам, примечаниям, изображениям)
- Проверка орфографии любого количества языков в одном документе
- Вложенные, скрытые, защищенные паролем индивидуально оформленные разделы
- Перекрестные вычисления между разными таблицами в одном документе
- Несколько оглавлений в одном документе.
- Автоматизированное перемещение элементов оглавления и списков с подпунктами.

Microsoft Word

- Встроенные средства для продвинутой проверки грамматики русского языка
- Распознавание голоса и рукописного ввода

- Подробная справочная система с примерами
- Широкая распространенность

Если сравнивать *производительность* OpenOffice Writer и Microsoft Word, то Writer уступает приблизительно в два раза.
Если сравнивать *безопасность* OpenOffice и Microsoft Office, то Microsoft намного надежнее, чем OpenOffice (MS Office 2010: 17 крахов и 0 потенциальных уязвимостей, В OpenOffice 3.2.1: 163 краха и 18 потенциальных уязвимостей). Одна из причин - разработкой свободного ПО занимаются любители.

8.1.2 Концепция стилей и шаблонов

- 1-я ошибка - форматирование вручную без стилей.
- 2-я ошибка - создание оформления вместо создания структуры.
- При подготовке документа главное то, чем текст является. А как он выглядит - вторично.
- Забыть про "размер шрифта 14pt" "гарнитура Times New Roman" "расположение по центру"и так далее.
- Помнить только стили: "Заголовок" "Заголовок n-ого уровня" "основной текст и так далее.
- Создание нового документа начинается с продумывания структуры документа и создания системы стилей.
- Как будет выглядеть конечный документ, (шрифты, гарнитура, и так далее) решается, когда документ уже готов, путем изменения соответствующего стиля.

8.1.3 Панграммы

Панграмма (греч. "*все буквы*") или разнобуквица - текст, использующий все или почти все буквы алфавита. Используется для:

- Демонстрация шрифтов.
- Проверки передачи текста по линиям связи.
- Тестирование печатающих устройств.

Microsoft Съешь [же] ещё этих мягких французских булок, да выпей чаю.

KDE Широкая электрификация южных губерний даст мощный толчок подъёму сельского хозяйства.

Gnome В чащах юга жил бы цитрус? Да, но фальшивый экземпляр!

8.1.4 Автозаполнение

Lorem ipsum - название классического текста-"рыбы".
"Рыба" - слово из жаргона дизайнеров, обозначает условный, зачастую бессмысленный текст, вставляемый в макет страницы.

Lorem ipsum представляет собой искаженный отрывок из философского трактата Цицерона "О пределах добра и зла написанного в 45 году до нашей эры на латинском языке. Впервые этот текст был применен для набора шрифтовых образцов неизвестным печатником в XVI веке.

$$= rand(m, n)$$

Где:
m – количество абзацев;
n – количество предложений в каждом абзаце;
Так же

$$= lorem(m, n)$$

8.2 Вспомогательное ПО для программирования

- Автоматическое создание документации для программы (doxygen).
- Контроль версий (SVN, Git, Mercurial).
- Управления жизненным циклом найденных ошибок (bug tracking system).
- Автоматизированное тестирование кода и функциональности.

8.2.1 Автоматизированное создание документации

Самая известная система для автоматизации создания документации программного обеспечения на C/C++ - это *doxygen*. Используется в KDE, IBM, AbiWord, Adobe, DC++, Qt, ...
При работе с Doxygen размечается код (в комментариях появляется собственный синтаксис, в результате чего комментарии преобразовываются в документацию).

Такие программы значительно облегчают работу: не нужно отдельно создавать документацию. Более того, в комментариях все подробно описано и любой программист сможет разобраться в программе.

8.2.2 Системы управления (контроля) версиями

- **Клиент-серверные (централизованные)**: CVS, Subversion, Microsoft SourceSafe, Perforce, VSS
- **Распределенные:** Mercurial, git

Принцип работы: пометка версий, которые отдаются пользователю, выкладываются на сайт (release версий) и версий для разработчиков, в которую возможно внести изменения (которые пользователю не отдаются). Это необходимо для того, чтобы редактировать новые версии (вносить изменения, тестировать), и, при необходимости, была возможность откатиться на старую версию. Программа
Преимущества Git над SVN: удобная работа с большим количеством веток, хранение всей истории изменения файлов проекта. Система управления хранит все предыдущие версии.

8.2.3 Жизненный цикл обнаруженной ошибки

Тестировщик находит ошибки;

Менеджер проекта назначает того, кто исправит ошибку;

Программист исправляет или объясняет, почему нельзя исправить (дубль; нет смысла исправлять; нельзя воспроизвести);

Тестировщик проверяет, была ли исправлена ошибка.

8.2.4 Тестирование программного обеспечения

Самые известные СУБД ошибок: JIRA, Redmine, Bugzilla, TrackGear.

Описание ошибки

- кто сообщил об ошибке;
- дата и время обнаружения;
- серьезность ошибки;
- перечень шагов воспроизведения ошибки;

- текущий статус ошибки.

Автоматизированное тестирование программного обеспечения - часть процесса тестирования на этапе контроля качества в процессе разработки программного обеспечения.
Оно использует программные средства для выполнения тестов и проверки результатов выполнения, что помогает сократить время тестирования и упростить его процесс.

Наиболее известный инструментария для тестирования:

- JUnit — тестирование приложений для Java
- NUnit — порт JUnit под .NET
- xUnit — тестирование приложений для .NET
- TestNG — тестирование приложений для Java
- Selenium — тестирование приложений HTML
- WatiN — тестирование веб-приложений
- TOSCA Testsuite — тестирование приложений HTML, .NET, Java, SAP
- UniTESK — тестирование приложений на Java, Си.

8.3 Лицензии

Лицензии на программное обеспечение – это правовой инструмент, определяющий использование и распространение программного обеспечения, защищенного авторским правом.

Проприетарное ПО ⇒ закрытый исходный код. Может быть платным и бесплатным.

Свободное ПО ⇒ открытый исходный код. Может быть платным и бесплатным.

Коммерческое ПО ⇒ платное. Может иметь как закрытый, так и открытый исходный код.

Бесплатное ПО ⇒ бесплатное. Может иметь как закрытый, так и открытый исходный код.

Разновидности лицензий на свободное ПО

- **Пермиссивные лицензии (BSD)**: можно менять и закрывать.
- **Копилефт (GPL)**: можно менять, нельзя закрывать.

Ричард Мэттью Столлман род.1953

Основоположником движения за открытый код является Ричард Мэттью Столлман. Он и создал первую лицензию GNU (General Public License).
Всего около 70 лицензий на владение свободным ПО (одобренных на opensource.org). Самые популярные: Apache License, BSD license, GPL, LGPL, MIT license, MPL.

8.3.1 Базовые права, предоставляемые свободным ПО

Все они предоставляют 4 базовых права:

- Право на запуск программы в любых целях (только если она не нанесет вред своим действием или бездействием).
- Право на изучение исходного и бинарного кода программы
- Право на платное или бесплатное распространение программы.
- Право на развитие программы.

Если программист передает пользователю свою программу , но не прилагает лицензию, то действует "право свободного пользования":

- Можно установить программу на 1 компьютер.
- Можно запускать программу на 1 компьютере.
- Нельзя копировать программу на другие компьютеры.
- Нельзя модифицировать программу.
- Данная лицензия действует 5 лет (п.4 ст. 1235 ГК РФ).

8.3.2 Особенности различных свободных лицензий

GNU GPL

- Запрещено включать исходные тексты в закрытое ПО, запрещено менять тип лицензии (copyleft).
- Запрещено динамическое связывание GNU GPL-библиотек с не-GNUGPL библиотеками (dll).

GNU LGPL

- Допускается динамическое связывание с закрытыми библиотеками.
- Запрещено использование кода в другом ПО.

MPL (Mozilla public license)

- Можно использовать исходные тексты в закрытом ПО, но лишь частично и с гарантией доступа к изменениям.

BSD License

- Можно использовать исходные коды в закрытом ПО без ограничений.

8.3.3 Ответственность за пиратское ПО

Административная ответственность за пиратское ПО Статья 7.12 КоАП РФ: нарушение авторских прав при ущербе на сумму до 100 000 рублей:

- штраф до 2 000 (физическое лицо)
- штраф до 20 000 (должностное лицо)
- штраф до 40 000 (юридическое лицо)

Уголовная ответственность за пиратское ПО Статья 146.2 УК РФ: незаконное использование объектов авторского права (в т.ч. приобретение, хранение) при ущербе на сумму от 100 000 рублей:

- штраф до 200 000 р.
- исправительные работы вплоть до 2 лет
- – арест вплоть до 2 лет

Статья 146.3 УК РФ: Незаконное использование объектов авторского права (в т.ч. приобретение, хранение) при ущербе на сумму от 1 000 000 рублей:

- штраф до 500 000 р.
- арест вплоть до 6 лет

Уголовная ответственность за плагиат ПО Статья 146.1 УК РФ: присвоение авторства, если это причинило крупный ущерб автору:

- штраф до 200 000 р.
- исправительные работы вплоть до 1 года
- арест вплоть до 6 месяцев

Гражданская ответственность за нарушение лицензии ПО Статья 1301 ГК РФ: нарушение авторских, интеллектуальных и исключительных прав:

- штраф до 5 000 000 руб. в пользу обладателя ПО
  либо
- двукратное возмещение убытков обладателю ПО

8.4 Visual Basic for Applications

Visual Basic for Applications (VBA, Visual Basic для приложений) — немного упрощенная реализация языка программирования Visual Basic, встроенная в линейку продуктов Microsoft Office (включая версии для Mac OS), а также во многие другие программные пакеты, такие как AutoCAD, SolidWorks, CorelDRAW, WordPerfect и ESRI ArcGIS. VBA покрывает и расширяет функциональность ранее использовавшихся специализированных макро-языков, таких как WordBasic.

8.4.1 Имя переменной

- Начинается с буквы латинского алфавита.
- Не может содержать пробелы, точки символы операций (+, -, *, /, #, $, %, &, !, <, >, = и так далее).
- Не может превышать 254 символов в длину.
- Должно быть уникальным в своей области действия.
- Не может дублировать зарезервированные слова.
- Не различает регистр букв: MyNumber = mYnUmBeR.

8.4.2 Типы данных

Тип данных	Резервируемая память, байт	Минимальное значение	Максимальное значение
Byte	1	0	255
Boolean	2	False	True
Integer	2	-32768	32767
Long	4	-2147483648	2147483647
Date	8	1 января 100 г.	31 декабря 9999 г.
String	Длина строки	1	65400
Variant (число)	16		
Variant (символ)	22 байта + длина строки		2147483647 символов

Объявление переменных

1. **Неявное**:
   sum = 100
   В данном случае присваивается тип Variant. Это обобщенный тип, переменная нетипизирована.

2. **Явное**:
   Dim sum As Integer
   Преимущества:

   - Программа быстрее работает.
   - Программе требуется меньше памяти.
   - Легче обнаружить некоторые ошибки.
   - Не возникает проблем со сложными типами (например, как отличить дату от текста).

   Недостатки:

   - Приходится думать.
   - Требуется использовать больше переменных.

8.5 TEX

WYSIWYG (англ. *What You See Is What You Get* - "что видишь, то и получишь") - свойство прикладных программ или веб-интерфейсов, в которых содержание отображается в процессе редактирования и выглядит максимально близко похожим на конечную продукцию, которая может

быть печатным документом, веб-страницей или презентацией. В настоящее время для подобных программ также широко используется понятие "визуальный редактор". (пример: Microsoft Word)

WYSIWYM (англ. *What You See Is What You Mean* - "что видишь, есть то, что имеешь в виду") - парадигма редактирования документов, возникшая как альтернатива более распространенной парадигме WYSIWYG. В WYSIWYM редакторе пользователь задает только логическую структуру документа и собственно контент. Оформление документа, его итоговый внешний вид возложено на отдельное ПО, либо, во всяком случае, вынесено в отдельный блок. Таким образом достигается полная независимость содержания документа от его формы. (пример: TEX)

TEX - система компьютерной верстки, разработанная американским профессором информатики Дональдом Кнутом в целях создания компьютерной типографии. В нее входят средства для секционирования документов, для работы с перекрестными ссылками. Номер версии TEXприближается к π, номер редактора формул - к числу e.

8.6 Вебинары

Вебинар (онлайн-семинар) - разновидность веб-конференции, проведение онлайн-встреч или презентаций через Интернет. Во время веб-конференции каждый из участников находится у своего компьютера, а связь между ними поддерживается через Интернет посредством загружаемого приложения, установленного на компьютере каждого участника, или через веб-приложение.
1988 г. – появление первых IRC (англ. Internet Relay Chat).
Середина 1990-х – появление и распространение IM (Instant Messaging).
1998 г. – регистрация торгового знака "Webinar"Эриком Р. Корбом (Eric R. Korb).

Существуют следующие приложения для вебинаров

- GoToMeeting
  - Создана в 2004 году компанией Citrix Online.
  - Поддерживаемые ОС: Macintosh, Microsoft Windows
  - http://www.gotomeeting.com/
- StartMeeting
  - Создана в 2011 году как start-up.

  - Поддерживаемые ОС: Microsoft Windows
  - http://www.startmeeting.com/

- Team Viewer
  - Создана в 2005 году.
  - Поддерживаемые ОС: Windows, Mac OS, Linux, Android, Apple iOS, Windows Phone
  - http://www.teamviewer.com

Глава 9

Структура и принципы функционирования компьютера

Люди пытались сделать компьютер достаточно давно.
Первым (после антикитерского механизма) был **механизм для суммирования и умножения**. Изобрел его Вильгельм Шиккард (*1592 - 1635*) в 1623 году.
Машина содержала суммирующее и множительное устройства, а также механизм для записи промежуточных результатов. Первый блок - шестиразрядная суммирующая машина - представлял собой соединение зубчатых передач. На каждой оси имелись шестерня с десятью зубцами и вспомогательное однозубое колесо - палец. Палец служил для того, чтобы передавать единицу в следующий разряд (поворачивать шестеренку на десятую часть полного оборота, после того как шестеренка предыдущего разряда сделает такой оборот). При вычитании шестеренки следовало вращать в обратную сторону. Контроль хода вычислений можно было вести при помощи специальных окошек, где появлялись цифры. Для перемножения использовалось устройство, чью главную часть составляли шесть осей с "навернутыми"на них таблицами умножения.
Второй арифметической машиной был **механизм для суммирования и вычитания "Паскали́на"**. Изобрел его Блез Паскаль (*1623—1662*) в 1642 году.
Машина Паскаля представляла собой механическое устройство в виде ящичка с многочисленными связанными одна с другой шестеренками. Складываемые числа вводились в машину при помощи соответствующего поворота наборных колесиков. На каждое из этих колесиков, соответ-

ствовавших одному десятичному разряду числа, были нанесены деления от 0 до 9. При вводе числа, колесики прокручивались до соответствующей цифры. Совершив полный оборот, избыток над цифрой 9 колесико переносило на соседний разряд, сдвигая соседнее колесо на 1 позицию.
Следующим был **арифмометр Лейбница**, умеющий выполнять операции сложения, вычитания, деления и умножения. Изобрел ее Готфрид Вильгельм Лейбниц (*1646 - 1716*) в 1673 году.
Сложение чисел выполнялось при помощи связанных друг с другом колес, так же как на "Паскалине". Добавленная в конструкцию движущаяся часть и специальная рукоятка, позволявшая крутить ступенчатое колесо (в последующих вариантах машины - цилиндры), позволяли ускорить повторяющиеся операции сложения, при помощи которых выполнялось деление и перемножение чисел. Необходимое число повторных сложений выполнялось автоматически.
Прообразом современного компьютера стала **идея создания универсальной аналитической вычислительной машины**, которую выдвинул Чарльз Бэббидж (*1791 - 1871*) в 1823 году.
В 1822 году Бэббидж построил **малую разностную машину**. Ее работа была основана на методе конечных разностей. Малая машина была полностью механической и состояла из множества шестеренок и рычагов. В ней использовалась десятичная система счисления. Она оперировала 18-разрядными числами с точностью до восьмого знака после запятой и обеспечивала скорость вычислений 12 членов последовательности в 1 минуту. Малая разностная машина могла считать значения многочленов 7-й степени.
И в том же 1822 году Бэббидж задумался о создании **большой разностной машины** предназначенной для автоматизации вычислений путем аппроксимации функций многочленами и вычисления конечных разностей. Возможность приближенного представления в многочленах логарифмов и тригонометрических функций позволяло бы рассматривать эту машину как довольно универсальный вычислительный прибор. В 1823 году он приступил к проектированию, однако, в 1842 году государство отказалось финансировать проект и машина так и не была достроена. Но для развития вычислительной техники имело значение другое: идея создания аналитической вычислительной машины. В единую логическую схему Бэббидж увязал арифметическое устройство (названное им "мельницей"), регистры памяти, объединенные в единое целое ("склад"), и устройство ввода-вывода, реализованное с помощью перфокарт трех типов. Перфокарты операций переключали машину между режимами сложения, вычитания, деления и умножения. Перфокарты переменных управляли передачей данных из памяти в арифметическое устройство и обратно. Числовые перфокарты могли быть использованы как для ввода

данных в машину, так и для сохранения результатов вычислений, если памяти было недостаточно.
Следующим этапом в развитии вычислительной техники стал **электромеханический перфокарточный табулятор для переписи населения**, который изобрел Герман Холлерит (*1860 - 1929*) в 1880 году.
Табуляторы предназначены для автоматической обработки (суммирования и категоризации) числовой и буквенной информации, записанной на перфокартах, с выдачей результатов на бумажную ленту или специальные бланки. Умножение и деление выполнялись методом последовательного многократного сложения и вычитания. Работа табулятора производилась в соответствии с набираемой на коммутационной панели программой.
В 1911 году Алексей Николаевич Крылов (*1863 - 1945*) изобрел **аналоговый решатель дифференциальных уравнений**. Он интегрировал обыкновенные дифференциальные уравнения.
В 1919 году Николай Николаевич Павловский (*1884 - 1937*) сконструировал **аналоговую вычислительную машину (АВМ)**. Она была создана для реализации метода исследования природных явлений при помощи аналого-математического моделирования, который разработал Павловский в том же 1919 году.

9.1 ЭВМ Джона фон Неймана

Джон фон Нейман
1903 – 1957

В 1930-х годах началась разработка архитектуры ЭВМ для военно-морской артиллерии по заказу правительства США. В разработке участвовали Гарвардский университет и Принстонский университет (в том числе и Джон фон Нейман). На рисунке 9.1 приведена схема ЭВМ, предложенная фон Нейманом.

9.1.1 Узлы ЭВМ фон Неймана

- **Процессор** - исполнитель машинных инструкций (кода программ), главная часть аппаратного обеспечения ЭВМ. В состав процессора входят:

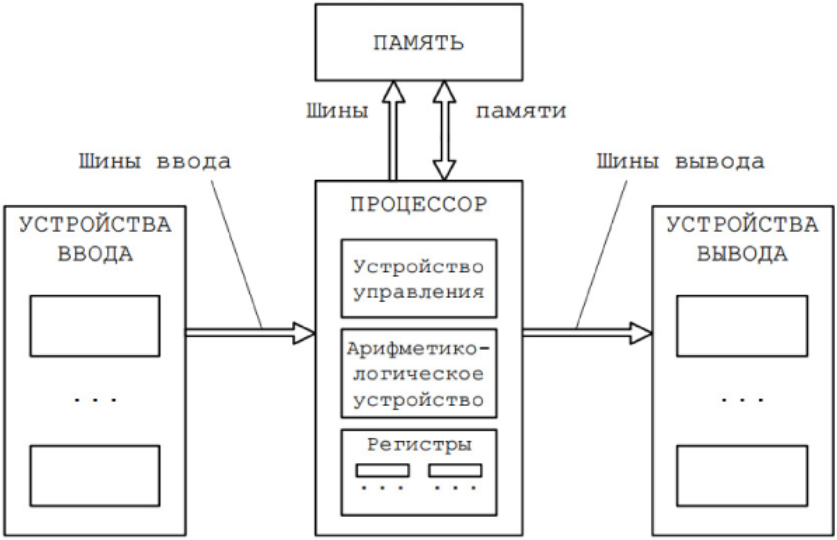


Рис. 9.1: Структурная схема ЭВМ фон Неймана

  - устройство управления выборкой команд из памяти и их выполнением;
  - арифметико-логическое устройство, производящее операции над данными;
  - регистры, осуществляющие временное хранение данных и состояний процессора;
  - схемы для управления и связи с подсистемами памяти и ввода-вывода.

- **Устройства ввода** обеспечивают считывание данных с носителей информации и ее представление в форме электрических сигналов, воспринимаемых другими устройствами ЭВМ (процессором или памятью) (мышь, клавиатура, сканер).

- **Устройства вывода** представляют результаты обработки данных в ЭВМ в форме, удобной для визуального восприятия человеком (монитор, принтер) или хранения (DVD-привод, стример).

- **Устройства ввода-вывода** используются как для хранения данных, так и для их считывания (жесткий диск, флешка).

9.1.2 Принципы работы архитектуры фон Неймана

Бёркс, Голдстайн и фон Нейман в 1946 г. в книге "Предварительное рассмотрение логического конструирования электронного вычислительного устройства"описали принципы:

- **Принцип двоичного кодирования** - вся информация, поступающая в ЭВМ, кодируется с помощью двоичных сигналов.
- **Принцип однородности памяти** - программы и данные хранятся в одной и той же памяти. Поэтому ЭВМ не различает, что хранится в данной ячейке памяти - число, текст или команда. Над командами можно выполнять такие же действия, как и над данными.
- **Принцип адресуемости памяти** - структурно основная память состоит из пронумерованных ячеек, процессору в произвольный момент времени доступна любая ячейка.
- **Принцип жесткости архитектуры** - неизменяемость в процессе работы топологии, архитектуры, списка команд.
- **Принцип программного управления**:
  1. В начале процессору сообщается адрес первой команды программы (который заносится в специальный **регистр команд**), после этого программа управляет сама собой.
  2. После выполнения команды процессор увеличивает адрес, хранимый в регистре команд, на длину только что выполненной команды, чтобы получить адрес следующей команды. Так можно выполнить цепочку команд из **последовательно** расположенных ячеек памяти.
  3. Существуют специальные **команды переходов**, которые сразу содержат в себе адрес следующей команды. После выполнения таких команд указанный адрес просто заносится в регистр команд. Так можно выполнить цепочку команд из **непоследовательно** расположенных ячеек памяти.

9.2 Классификация архитектур ЭВМ

Архитектурой ЭВМ определяется, как именно в этой ЭВМ происходит обработка и преобразование данных с учетом конкретных принципов взаимодействия технических средств и программного обеспечения.

1. **По способу хранения команд/данных**:

- **Принстонская архитектура**: программы и данные хранятся в одном массиве памяти (микросхеме) и передаются в процессор по одному каналу связи (шине). Проще реализовать (сконструировать), гибкость модификации программ.
- **Гарвардская архитектура**: предусматривает раздельные хранилища и потоки передачи (шины) для команд и данных. Возможность одновременной работы с данными и командами.

2. **По разрядности интерфейсов и машинных слов**: 8-, 16-, 32-, 64-, 128-разрядные.
3. **По особенностям набора команд и регистров**:
   - CISC – Complete Instruction Set Computer
   - RISC – Restricted (Reduced) Instruction Set Computer
   - CRISP – Complex-Reduced-Instruction-Set Processor
   - VLIW – Very Long Instruction Word
4. **По количеству вычислителей**: однопроцессорные, многопроцессорные, одноядерные, многоядерные.

Существуют следующие архитектуры систем команд (рисунок 9.2):

- **Регистровая архитектура** - внутри процессора существует специальная память (регистры) для хранения промежуточных результатов.
- **Аккумуляторная архитектура** - существует один регистр (аккумулятор), в котором хранится результат.
- **Стековая архитектура** - команды не имеют операндов.

В процессоре ограниченное количество команд, как и в языках программирования. И каждую операцию он разбивает на более мелкие и простые микрокоманды. Архитектуры CISC и RISC определяют количество этих микрокоманд. Например, есть всего 10 команд, а все остальные операции можно составить из этих команд - это архитектура RISC. Или на каждую возможную операцию необходима своя команда - это будет архитектура CISC. Рассмотрим подробнее.

9.2.1 Архитектура CISC

CISC - complex instruction set computer - компьютер с полным набором команд.

- много команд;
- мало регистров общего назначения (памяти для хранения операндов арифметико-логических инструкций, а также адресов или отдельных компонентов адресов ячеек памяти) (до 32);
- разнообразие способов адресации (прямая, косвенная);
- много форматов команд различной разрядности;
- обработка совмещается с обращением к памяти.

Доля сложных дополнительных команд CISC в общем объеме программ не превышает 10-20% . Емкость микропрограммной памяти для поддержании сложных команд может увеличиваться на 60% .

9.2.2 Архитектура RISC

RISC - reduced instruction set computer - компьютер с сокращенным набором команд.

- мало команд (только наиболее часто используемые);
- много регистров общего назначения (РОН) (сотни);
- есть только две команды обращения к памяти (все остальные команды могут работать только с РОНами);
- мало форматов команд и способов указания адресов операндов.

9.2.3 Сравнение RISC и CISC

1. Возможность повышения тактовой частоты и упрощения кристалла с высвобождением площади под кэш.
2. Снижение энергопотребления процессора засчет уменьшения числа транзисторов.
3. Доля сложных дополнительных команд CISC в общем объеме программ не превышает 10-20%.
4. Возможность упреждающего выполнения команд.

Результат: большинство современных процессоров являются либо чистыми RISC, либо "CISC-поверх-RISC".

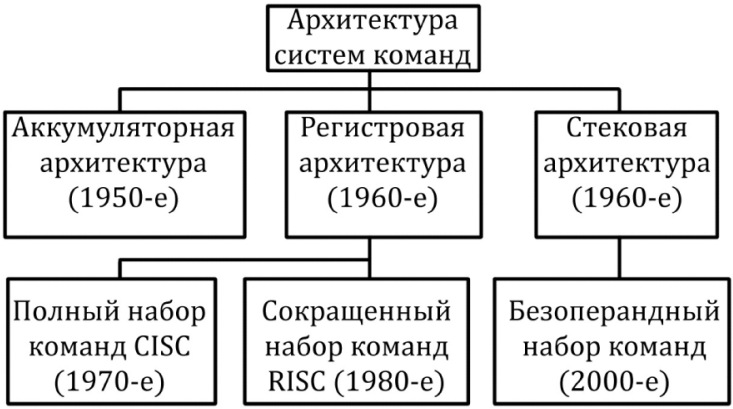


Рис. 9.2: Классификация систем команд

9.3 Команды процессора

Этапы выполнения команд процессором

Полный цикл выполнения команды может включать в себя следующие этапы:

- Вычисление адреса команды (ВАК) - счетчик команд указывает на некоторую ячейку, содержащую команду;
- Выборка команды (ВК) - обращение к ячейке памяти по адресу, указанному в счетчике команд, считывание команды;
- Декодирование команды (ДК) - процессор распознает, что за команда (сложение, вычитание, переход и так далее);
- Вычисление адреса операнда (ВАО) - если команда адресная, то производится вычисление адреса операнда, который содержит команда;
- Выборка операнда (ВО) - обращение к ячейке памяти по адресу, указанному в команде, считывание операнда;
- Выполнение заданной операции (ВЗО);
- Запись результата (ЗР).

Конвейерная обработка команд

Рис. 9.3: Форматы команд процессора: а) многоадресная, б) адресная, в) безадресная

При выполнении почти каждой команды, необходимо осуществить 4-7 действий. При выполнении команд последовательно (как это делали старые процессоры) следующая команда будет ждать, пока полностью осуществится первая. Понятно, что можно выполнять команды по принципу конвейера, что существенно увеличит скорость работы.

IF - ВАК+ВК (вычисление адреса команды и выборка команды);

ID - ДК+ВАО+ВО (декодирование команды, вычисление адреса операнда, выборка операнда);

EX - ВЗО (выполнение заданной операции);

MEM - ЗР (запись результата);

WB - ЗР (запись результата);

Видно, что пока первая команда находится на этапе WB, задействован один узел - записи результата. Тем временем другой узел - выполнения заданной операции, может не простаивать, а выполнять третью команду.

Пример 5-уровневого конвейера в реальном RISC-процессоре:

И таким образом, все отдельные независящие друг от друга узлы, работают одновременно, выполняя этапы разных команд, чем значительно увеличивают производительность.

Глава 10

Организация хранения данных в ЭВМ

10.1 Устройство памяти

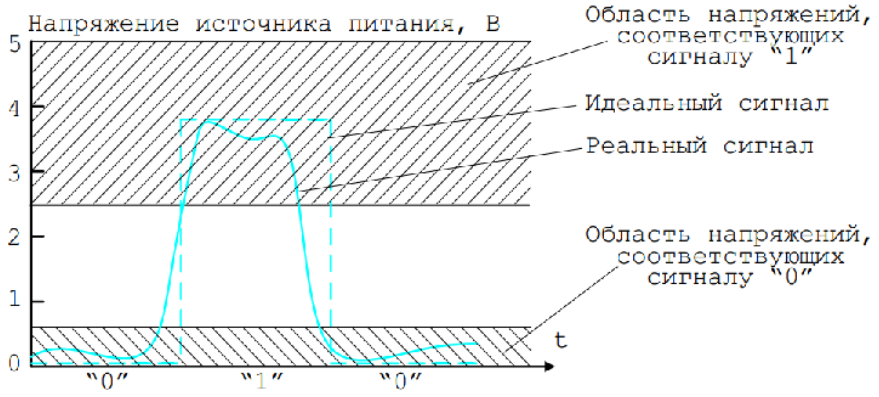
1. Память состоит из адресуемых ячеек (размером 1 - 128 бит). Ячейки в основном организованы по 8 бит и свой собственный адрес имеет только каждый восьмой бит.

2. Ячейки состоят из запоминающих электрических элементов. Это может быть конденсатор, транзистор или попупроводниковый материал, умеющий хранить два состояния. Есть так же элементы, умеющие хранить три состояния, но эти три состояния сложнее считывать.

3. Электрический элемент может находиться в одном из двух устойчивых состояний (для хранения 1 бита):

    - конденсатор заряжен/разряжен;
    - транзистор в проводящем/непроводящем состоянии;
    - полупроводниковый материал имеет высокое/низкое сопротивление.

Одно из таких физических состояний создает высокий уровень выходного напряжения элемента памяти, а другое - низкий. В элементах памяти ряда микроЭВМ это электрические напряжения порядка 4В и 0В соответственно, причем первое обычно принимается за двоичную единицу, а второе - за двоичный нуль.
Если сигнал попадает в незаштрихованную область (между 0,5В и 2,5В)



то сигнал не распознается.

Запоминающим элементом называется элемент, который способен принимать и хранить код двоичной цифры (исходные значения некоторых величин, промежуточные значения обработки и окончательные результаты вычислений).
Триггер - элементарный цифровой автомат, обладающий способностью длительно находиться в одном из двух устойчивых состояний и чередовать их под воздействием внешних сигналов. Состояние 0 на выходе Q соответствует выключенному состоянию, а $Q = 1$ - включенному.
Сущесвуют следующие типы триггеров:

- **RS-триггер** - меняет свое состояние в зависимости от того, на какой из входов была подана единица. При подаче сигнала на вход S (*Set*) на выходе устанавливается единица. При подаче сигнала на вход R (*Reset*) сигнал на выходе пропадает.
- **D-триггер** (*Delay* или *Data*) - запоминает (задерживает) состояние входа на один такт. При кратковременной подаче сигнала на C (*Clock*) (обычно, с тактового генератора), запоминает сигнал на входе D (*Data*) и выдает его на выход до следующей итерации.
- **T-триггер** (*Toggle*) - при подаче сигнала на вход, меняет сигнал на выходе на противоположный.
- **JK-триггер** - аналогичен *RS*-триггеру (J (*Jump*) = *Set*, K (*Kill*) = *Reset*), с одним лишь исключением: при подаче единицы на оба входа, состояние выхода изменяется на противоположное.

Регистры - это узлы ЭВМ, служащие для хранения информации в виде машинных слов или его частей, а так же для выполнения над сло-

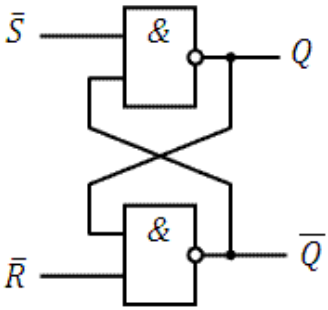


Рис. 10.1: Асинхронный RS-триггер на элементах 2И-НЕ (IEC)

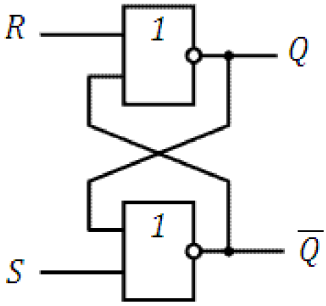


Рис. 10.2: Асинхронный RS-триггер на элементах 2ИЛИ-НЕ (IEC)

вами некоторых логических преобразований.

Счетчики - узлы ЭВМ, которые осуществляют счет и хранение кода числа подсчитанных сигналов.

Дешифратор (избирательная схема) - это узел ЭВМ, в котором каждой комбинации входных сигналов соответствует наличие сигнала на одной вполне определенной шине на выходе.

Шифратор (кодер) - это узел ЭВМ, преобразующий унитарный код в некоторый позиционный код.

Преобразователи кодов - это узлы ЭВМ, предназначенные для кодирования чисел.

Мультиплексоры - это узлы, преобразующие параллельные цифровые коды в последовательные. В этом устройстве выход соединяется с одним из входов в зависимости от значения адресных входов.

Демультиплексоры - это узлы, преобразующие информацию из последовательной формы в параллельную.

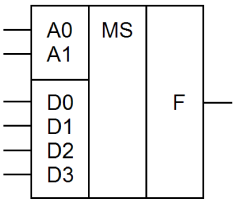


Рис. 10.3: Мультиплексор

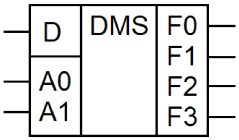


Рис. 10.4: Демультиплексор

Сумматор - это узел, в котором выполняется арифметическая операция суммирования цифровых кодов двух двоичных чисел.

10.2 Характеристики систем памяти

1. **Место расположения:**
   - *Процессорная*, то есть на общем кристалле с центральным процессором (ЦП) (регистры, кэш-память 1-го уровня);
   - *Внутренняя*, то есть на системной плате (основная память (ОП), кэш-память 2-го и последующего уровней);
   - *Внешняя* (медленные запоминающие устройства (ЗУ) большой ёмкости).
2. **Емкость ЗУ** - число бит/байт, которое можно хранить на ЗУ.
3. **Единица пересылки.** Для ОП единица пересылки определяется шириной шины данных, то есть количество бит, передаваемых по линиям шины параллельно. Обычно равна длине слова.
4. **Метод доступа к данным:**
   - *Последовательный доступ* - ЗУ ориентировано на хранение информации в виде последовательности блоков, называемых записями. Для доступа к нужному элементу необходимо прочитать все предшествующие блоки (ЗУ на магнитной ленте);
   - *Прямой доступ* - каждая запись имеет уникальный адрес, отражающий ее физическое размещение на носителе информации. Обращение определяется как адресный доступ к началу записи плюс последующий последовательный доступ к определённой информации внутри записи (магнитные диски);
   - *Произвольный доступ* - каждая ячейка памяти имеет уникальный физический адрес. Обращение к любой ячейке занимает одно и то же время и может водиться в произвольной очередности (ОП);
   - *Ассоциативный доступ* - позволяет выполнять поиск ячеек, содержащих такую информацию, в которой значение отдельных бит совпадает с состоянием одноименных битов в заданном образце. Сравнение осуществляется параллельно для всех ячеек памяти, независимо от ее емкости (кэш-память).
5. **Быстродействие** - один из важнейших показателей:

- *Время доступа ($T_д$)* - интервал времени от момента поступления адреса до момента, когда данные заносятся в память или становятся доступными.
- *Длительность цикла памяти или период обращения ($T_ц$)* - понятие применяется к памяти с произвольным доступом, для которой оно означает минимальное время между двумя последовательными обращениями к памяти. Период обращения включает в себя время доступа плюс некоторое дополнительное время.
- *Скоростъ передачи* - скорость, с которой данные могут передаваться в память или из нее.

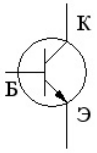
6. **Физический тип**
   - Полупроводниковая память;
   - Память с магнитным носителем информации (используемая в магнитных лентах и дисках);
   - Память с оптическим носителем (оптические диски);
7. **Физические особенности** (например, энергозависимость).
8. **Стоимость** - стоимость хранения одного бита информации.

10.3 Иерархия памяти

Чем меньше время доступа, тем выше стоимость хранения бита. Чем больше емкость, тем ниже стоимость хранения бита, но больше время доступа.

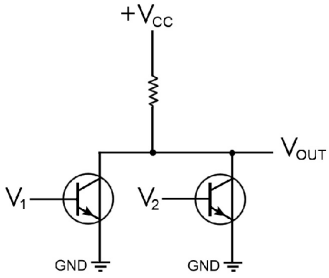
10.4 Физическое устройство памяти

10.4.1 Кэш-память



Рассмотрим биполярный n-p-n транзистор: К - коллектор, Б - база, Э - эмиттер. На коллектор подано напряжение. Если на базу подать напряжение - транзистор откроется и ток с коллектора пойдет на эмиттер.
Небольшая особенность - напряжение на базе должно быть выше, чем на коллекторе (на сколько - зависит от конкретного транзистора, обычно немного. Например, 5В на коллекторе, 6В на базе).

Теперь рассмотрим следующую схему:



V_{CC} - линия питания устройства (например 5В).
GND - линия 0В.
Если оба транзистора закрыты (на базу не подается напряжение, V_1 и V_2 равны 0), то ток уходит напрямую с V_{CC} на V_{OUT}. В результате получается логическая единица.
Если подать напряжение хотя бы на один транзистор (V_1 или V_2), то ток с V_{CC} будет уходить через транзистор в GND и в V_{OUT} не пойдет. В результате получается логический нуль.

Так как напряжение напряжение на базе должно быть выше, чем на коллекторе (в данном случае, на линии питания V_{CC}), а повышенное взять неоткуда, то имеющееся напряжение на V_{CC} занижается с помощью резистора и получается 5В на базе и чуть меньше на коллекторе.

Получается, что данная схема реализует логическую функцию ИЛИ-НЕ (ANSI) (также известную как стрелка Пирса) (где A и B соответственно V_1 и V_2):

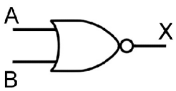
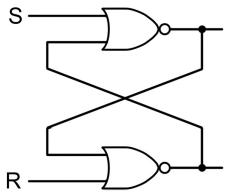


Таблица истинности для ИЛИ-НЕ:

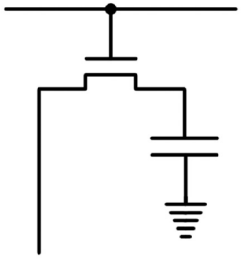
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Объеденим два элемента ИЛИ-НЕ обратной связью. Выход одного элемента ИЛИ-НЕ поступает на вход другого. Получилась самая простая память для хранения 1 бита, использующая 4 транзистора.



В данном случае, изображен асинхронный RS-триггер (изображен на рисунке 10.2). Также, память может состоять и из других триггеров. При подаче единицы на вход S выходное состояние становится равным логической единице. А при подаче единицы на вход R выходное состояние становится равным логическому нулю. Если на оба входа R и S одновременно поданы логические единицы, оба выхода переходят в состояние логического нуля, которое является неустойчивым и переходит в одно из устойчивых состояний при снятии управляющего сигнала с одного из входов, иначе говоря, в ячейку может записаться любое значение.

10.4.2 Оперативная память



В отличие от кэш-памяти, оперативная память устроена намного проще. Она устроена из 1 конденсатора и 1 транзистора, что дешево и занимает мало места. Один конденсатор легче воспринимать как память (конденсатор разряжен - 0, заряжен - 1). Транзистор нужен с одной целью - чтобы не разряжать постоянно конденсатор. Конденсатор необходимо периодически подзаряжать, а заряжается и разряжается он медленно. Переключить транзистор быстрее чем, зарядить или разрядить конденсатор.

Если на базу подано напряжение, транзистор открыт, мы можем считать значение - 1. Если напряжения нет, значение не считывается, записывается 0.

10.5 Локальность памяти

10.5.1 Пространственная локальность памяти

С очень высокой вероятностью адрес очередной команды программы либо следует непосредственно за адресом, по которому была считана текущая команда, либо расположен вблизи него. Такое расположение адресов называется **пространственной локальностью программы**.
Обрабатываемые данные, как правило, структурированы, и такие структуры обычно хранятся в последовательных ячейках памяти. Такая особенность программ называется **пространственной локальностью данных**.

10.5.2 Временна́я локальность памяти

Кроме того, программы содержат множество небольших циклов и подпрограмм. Это означает, что небольшие наборы команд могут многократно повторяться в течение некоторого интервала времени, то есть имеет место **временная локальность**.
Все три вида локальности объединяет понятие **локальность по обращению**. Принцип локальности часто облекают в численную форму и представляют в виде так называемого правила "90/10": 90 % времени работы программы связано с доступом к 10% адресного пространства этой программы.

10.6 Порядок хранения байт в памяти

Существует несколько способов хранения байт в памяти:

- **От старшего к младшему** (англ. *big-endian*): $A_n, \ldots, A_0$ запись начинается со старшего и заканчивается младшим. Этот порядок является стандартным для протоколов TCP/IP, он используется в заголовках пакетов данных и во многих протоколах более высокого уровня, разработанных для использования поверх TCP/IP. Поэтому, порядок байтов от старшего к младшему часто называют сетевым порядком байтов.

- **От младшего к старшему** (англ. *little-endian*): $A_0, \ldots, A_n$ запись начинается с младшего и заканчивается старшим. Этот порядок записи принят в памяти персональных компьютеров с x86-процессорами, в связи с чем иногда его называют интеловский порядок байт (по названию фирмы-создателя архитектуры x86).

- **Переключаемый порядок** (англ. *bi-endian*). Многие процессоры могут работать и в порядке от младшего к старшему, и в обратном. Обычно порядок байтов выбирается программно во время инициализации операционной системы, но может быть выбран и аппаратно перемычками на материнской плате. В этом случае правильнее говорить о порядке байтов операционной системы.

- **Смешанный порядок** (англ. *middle-endian*) иногда используется при работе с числами, длина которых превышает машинное слово. Число представляется последовательностью машинных слов, которые записываются в формате, естественном для данной архитектуры, но сами слова следуют в обратном порядке.

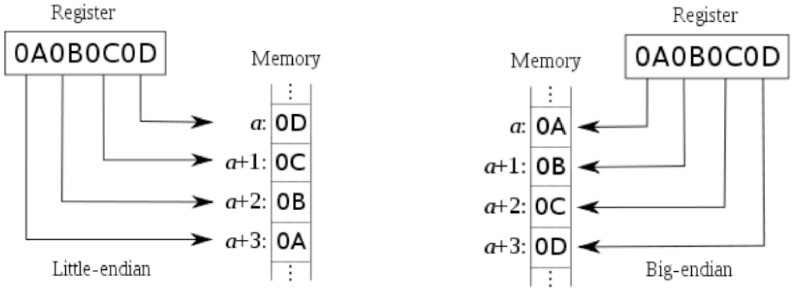


Рис. 10.5: Сравнение порядков от младшего к старшему и от старшего к младшему

Глава 11

Передача данных в компьютерных сетях

11.1 Многоуровневая модель OSI (Open Systems Interconnection)

Процесс передачи данных по компьютерной сети очень сложен, поэтому специалисты International Standards Organization решили разделить его на семь логических независимых уровней. Специалист на одном уровне может работать независимо от специалиста на другом уровне, не мешая друг другу.

№	Название уровня (layer)	Основная функция
7	прикладной (application)	взаимодействие программы пользователя с сетевой подсистемой ОС (API)
6	уровень представления (presentation)	шифрование, сжатие, выбор кодировки
5	сеансовый (session)	установление соединения
4	транспортный (transport)	надежность доставки, реакция на потери
3	сетевой (network)	маршрутизация, объединение разнородных локальных сетей, адресация в глобальной сети (IP)
2	канальный (data link)	связь между узлами одной локальной сети, адресация в локальной сети (MAC-адрес)
1	физический (physical)	физические характеристики каналов связи и передаваемых сигналов

11.1.1 Прикладной уровень

Субъекты взаимодействия: пользовательская программа на передающем/принимающем компьютере; ОС.
Объекты взаимодействия: Пользовательские данные, представленные в "родном"понятном виде для приемной и передающей программы.
Основные функции: Вызов специальных функций ОС для работы с сетью (API). Программист не обязан знать о внутреннем устройстве сети, для него передача данных по сети не отличается от сохранения в файл (просто надо вызвать нужную функцию API ОС).
API (интерфейс программирования приложений, интерфейс прикладного программирования) (англ. *application programming interface*) — набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) для использования во внешних программных продуктах. Используется программистами при написании всевозможных приложений.

11.1.2 Уровень представления

Субъекты взаимодействия: специальное ПО для шифрования, сжатия, кодирования; ОС.
Объекты взаимодействия: Закодированные пользовательские данные (пользовательская программа уже не может работать с такими данными без декодирования).
Основные функции: Шифрование, сжатие, выбор кодировки, выбор способа представления порядка байт (little-endian, big-endian). Каждый этап может выполняться несколько раз разными субъектами.

11.1.3 Сеансовый уровень

Субъекты взаимодействия: ОС на компьютере-передатчике; ОС на компьютере-приемнике.
Объекты взаимодействия: Служебные данные о об установке соединения: логины, пароли, сертификаты, цифровые подписи, пустые пакеты для проверки отсутствия обрывов связи, служебные пакеты с командами типа "запрос соединения "подтверждение соединения "разрыв соединения"(т.е. никакие пользовательские данные на этом уровне не передаются).
Основные функции:

- Установление соединения (с возможной аутентификацией абонентов).

- Отслеживание состояния соединения (возможное автопереподключение при обнаружении ошибок).
- Реагирование на долгую неактивность сеанса связи (например, автоотсоединение по таймауту).
- Принудительный разрыв соединения при окончании передачи (попутно освобождаются ресурсы ОС, которые хранят информацию о состоянии сеанса).

11.1.4 Транспортный уровень

Субъекты взаимодействия: ОС; драйвер сетевой карты.
Объекты взаимодействия: Пользовательские данные, снабженные служебными заголовками для обнаружения проблем передачи (контрольная сумма, порядковые номера фрагментов), служебные пакеты-подтверждения.
Основные функции:

- Отслеживание проблемных пакетов: искаженных, потерянных, пришедших в неверном порядке или дубликатов.
- Реакция на обнаружение проблемных пакетов (запрос повторной передачи или игнорирование, сбор целых пакетов из пришедших в разном порядке фрагментов).
- Реализация механизма повторной передачи (передается весь файл целиком или только проблемные части).

11.1.5 Сетевой уровень

Субъекты взаимодействия: ОС; драйвер сетевой карты.
Объекты взаимодействия: Данные, нарезанные на фрагменты, которые можно передавать в конкретной локальной сети (например, в проводных сетях Fast Ethernet предельный размер фрагмента ≈ 1500 байт, а в сетях Wi-Fi он равен ≈ 8000 байт). Каждый фрагмент снабжается глобальным адресом (например, IP-адресом), который понятен в любой локальной сети, но при этом уникален для всей глобальной сети.
Основные функции: Маршрутизация в большой сети; обеспечение возможности объединить несколько разнородных локальных сетей в одну сеть.

11.1.6 Канальный уровень

Субъекты взаимодействия: драйвер сетевой карты; модуль сетевой карты, который генерирует физические сигналы (ток, радиоволна, пучок

света).
Объекты взаимодействия: Набор битов, полностью готовых к передаче от одного компьютера локальной сети к другому (без выхода в глобальную сеть). Помимо данных пользователя, в этот набор включают адреса приёмника и передатчика внутри локальной сети (например, MAC-адреса).
Основные функции:

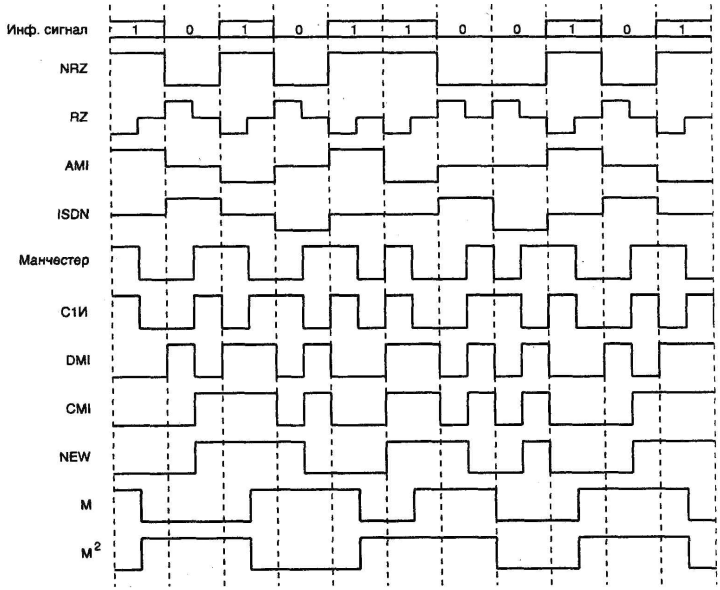
- Проверка доступности (свободности) канала связи, если он общий для нескольких абонентов. Например, в Wi-Fi-канал является общим для нескольких устройств в радиусе действия базовой станции, поэтому он не всегда доступен для передачи и каждому устройству приходится ждать своей очереди.
- Передача данных и адресация осуществляются только внутри локальной сети (MAC-адрес имеет смысл только в пределах локальной сети, так как он не передаётся в глобальную сеть).

11.1.7 Физический уровень

Субъекты взаимодействия: модуль сетевой карты, который генерирует физические сигналы (ток, радиоволна, пучок света); проводник сигнала (медный кабель, оптоволокно, радиоэфир).
Объекты взаимодействия: Физические сигналы (ток, пучок света, радиоволна).
Основные функции: Выбор носителя сигнала (ток, свет, радиоволна). Выбор свойств проводника сигнала (материал: медь, оптоволокно; диаметр сечения, сопротивление, предельно допустимая длина). Выбор способа представления цифровых данных в виде физического сигнала (кодирование, модуляция).

Кодирование

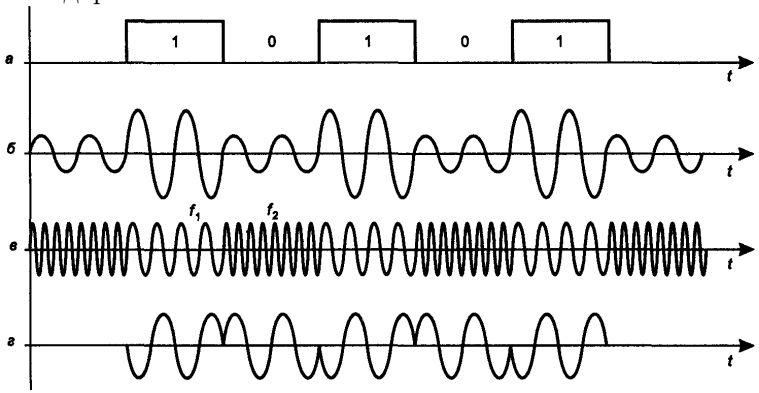
0 и 1 можно представить в виде разного напряжения электрического тока. Самый интуитивно-понятный способ называется NRZ. Однако существует много других способов, устраняющих недостатки NRZ (например, проблему вырождения переменного сигнала в постоянный ток, если передаются много единиц подряд).



Различные системы кодирования данных

Модуляция

Если сетевая карта умеет генерировать физический сигнал в виде синусоиды, то управляя амплитудой/частотой/фазой этой синусоиды, можно кодировать 0 и 1.



а) информационный сигнал, б) амплитудная модуляция (AM), в) частотная модуляция (FM), г) фазовая модуляция (PM)

11.1.8 Адекватность OSI-модели

Не существует ни одной сетевой технологии, в которой бы была идеально реализована вся OSI-модель с четким разделением уровней. Модель OSI далека от реальности, ее назначение - быть идеальной абстракцией.

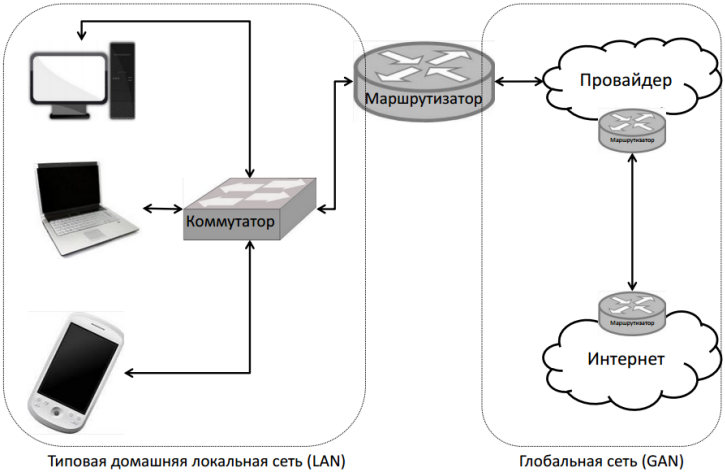
Реальность	OSI-уровни
Skype	7,6,5
FTP	7,3
TCP	7,5,4,3
IP	3,4
Wi-Fi	1,2
Fast-Ethernet	1,2

11.2 Отличие TCP от UDP

Свойство	TCP	UDP
Установка соединения	✓	×
Разрыв соединения	✓	×
Подтверждение доставки	✓	×
Проверка контрольной суммы	✓	✓
Обнаружение искаженных пакетов	✓	✓
Обнаружение потерянных пакетов	✓	×
Повторная передача потерянных/искаженных	✓	×

TCP применяют, если необходимо удостовериться, что все данные дошли корректно, получив об этом подтверждение и организовав повторную передачу поврежденных данных (пример: передача почты).
UDP применяют либо если канал связи абсолютно надежен, либо если нет смысла повторно передавать потерянные/искаженные пакеты (пример: видео-звонок), но при этом хочется сэкономить на передаче ненужных служебных данных, используемых в TCP.

11.3 Сетевые устройства



Сравнение коммутатора и маршрутизатора

Свойство	Коммутатор (switch)	Маршрутизатор (router)
Наличие MAC-адреса	Нет	Много (ровно по по одному на каждый порт/антенну)
Наличие IP-адреса	Нет	Много (минимум по по одному на каждый порт/антенну)
Уровни OSI-модели	1,2	1,2,3
Умение выбирать маршруты	Нет (так как в локальной сети всегда только один маршрут	Да
Назначение	Обмен данными между компьютерами внутри локальной сети	Обмен данными между несколькими локальными сетями

Примечание: существуют гибридные устройства, совмещающие в себе коммутатор и маршрутизатор (они используются у большинства пользователей домашнего интернета, однако в корпоративных сетях применяются реже).