



Славянская кириллическая нумерация:

аддитивная запись

$A = 1, B = 2, \Gamma = 3, Д = 4...$

$444 = \text{УМД}, = 400 (\text{У}) + 40 (\text{М}) + 4 (\text{Д}).$

Китайская нумерация:

аддитивно-мультипликативная запись

$444 = \text{四百四十四} =$
 $= 4 * 100 + 4 * 10 + 4$

Римская (латинская) нумерация:

аддитивно-субтрактивная запись

$LX = 50 + 10 = 60$

$XL = 50 - 10 = 40$

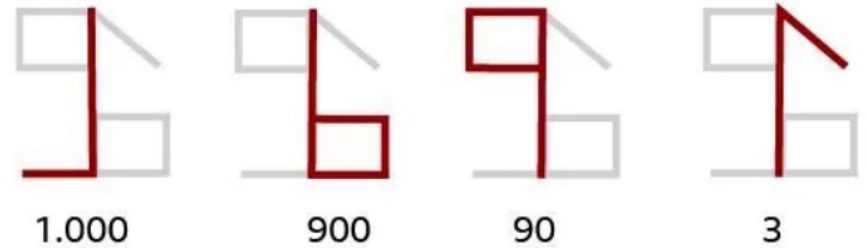
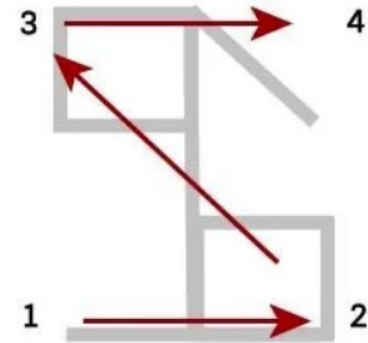
Недостатки непозиционных систем счисления по сравнению с позиционными:

- Сложно выполнять арифметические операции с большими числами.
- Длина записи числа (т. е. количество цифр) немонотонно зависит от его величины.

Цистерианская система счисления



1	2	3	4	5	6	7	8	9
10	20	30	40	50	60	70	80	90
100	200	300	400	500	600	700	800	900
1.000	2.000	3.000	4.000	5.000	6.000	7.000	8.000	9.000



= 1.993



Классическое правило округления – к ближайшему целому:

	Число	Округл.
	1,1	1,0
	2,9	3,0
	5,0	5,0
	3,4	3,0
	8,6	9,0
Сумма	21,0	21,0

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

Пример накопленной ошибки округления



10000 строк

Копеечная часть зарплаты	Округление до целых рублей
0,33	0
0,51	1
0,89	1
0,49	0
...	...
0,50	1
0,73	1
0,20	0

Сумма1 vs Сумма2

В первом столбце из 100 возможных значений только одно приводит к накоплению ошибки в 50 коп., поэтому

В среднем $(\text{Сумма2} - \text{Сумма1}) =$
 $= (10000/100) * 50 \text{ коп.} =$
 $= 50 \text{ руб. переплаты!}$



В системах счисления с **чётным** основанием накапливается ошибка округления:

<u>Основание 10:</u> 1, 2, 3, 4,	← округление в меньшую сторону
5, 6, 7, 8, 9,	← округление в бóльшую сторону
0	← нет ошибки округления

В системах счисления с **нечётным** основанием этой проблемы нет:

<u>Основание 7:</u> 1, 2, 3	← округление в меньшую сторону
4, 5, 6	← округление в бóльшую сторону
0	← нет ошибки округления

Актуальна ли проблема накопления ошибки округления для симметричных СС?

Решение проблемы с округлением в СС с чётным основанием



Суть решения — использовать неклассические правила округления:

- **Случайное округление:** используется датчик случайных чисел при принятии решения о том, в бóльшую или меньшую сторону следует округлять.
- **Банковское округление** (к ближайшему чётному): $3,5 \approx 4$, но $2,5 \approx 2$.
- **К ближайшему нечётному:** $3,5 \approx 3$, но $2,5 \approx 3$. Аналогично: $4,3_{(6)} \approx 5_{(6)}$.
- **Чередующееся:** направление округления меняется на противоположное при каждой операции округления (необходимо «помнить» о предыдущем округлении).

Примечание. Каждое из правил можно применять как полностью универсально, так и комбинировано с классическим правилом округления, дополняя его лишь при округлении пограничных значений.



Number, NS(2)	Math	To odd	To even	Random Coin test	Striped
10,1	11	11	10	11	10
00,1	01	01	00	01	01
01,0	01	01	01	01	01
10,0	10	10	10	10	10
11,0	11	11	11	11	11
00,1	01	01	00	00	00
01,1	10	01	10	01	10
00,0	00	00	00	00	00
Sum					
1011	1101	1100	1010	1011	1011

Пример округления (2)



Number, NS(3)	Math	To odd	To even	Random Coin test	Striped
2.1	2.0	10.0	2.0	10.0	2.0
0.2	1.0	1.0	0.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0
2.0	2.0	2.0	2.0	2.0	2.0
10.0	10.0	10.0	10.0	10.0	10.0
0.1	0.0	1.0	0.0	0.0	0.0
1.2	2.0	1.0	2.0	1.0	2.0
0.0	0.0	0.0	0.0	0.0	0.0
Sum					
102.01	102	110	101	102	102



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 – конечная последовательность знаков (букв) алфавита.

Словарь (словарный запас) – множество различных слов над алфавитом.



Кодирование (модуляция) данных — процесс преобразования символов алфавита X в символы алфавита Y.

Декодирование (демодуляция) — процесс, обратный кодированию.

Символ — наименьшая единица данных, рассматриваемая как единое целое при кодировании/декодировании.

Кодовое слово – последовательность символов из алфавита Y, однозначно обозначающая конкретный символ алфавита X.

Средняя длина кодового слова – это величина, которая вычисляется как взвешенная вероятностями сумма длин всех кодовых слов.

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

Если все кодовые слова имеют одинаковую длину, то код называется **равномерным** (фиксированной длины).

Если встречаются слова разной длины, то – **неравномерным** (переменной длины).



Сжатие данных — процесс, обеспечивающий уменьшение объёма данных путём сокращения их избыточности.

Сжатие данных — частный случай кодирования данных.

Коэффициент сжатия — отношение размера входного потока к выходному потоку.

Отношение сжатия — отношение размера выходного потока ко входному потоку.

Пример. Размер входного потока равен 500 бит, выходного равен 400 бит.

Коэффициент сжатия = $500 \text{ бит} / 400 \text{ бит} = 1,25$.

Отношение сжатия = $400 \text{ бит} / 500 \text{ бит} = 0,8$.

Случайные данные невозможно сжать, так как в них нет никакой избыточности.



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.

Статистические методы — кодирование с помощью усреднения вероятности появления элементов в закодированной последовательности.

Словарные методы — использование статистической модели данных для разбиения данных на слова с последующей заменой на их индексы в словаре.

Современная разработка от FaceBook*:

<https://engineering.fb.com/2021/09/13/core-data/superpack/>



Причины:

- Альфа-частицы от примесей в чипе микросхемы.
- Нейтроны из фонового космического излучения.

Частота единичных битовых ошибок (на 1 GB):

- От 1 раза в час до 1 раза в тысячелетие (по данным исследования Google получилось ~1 раз в сутки).

Способы обработки данных:

- Использовать полученные данные без проверки на ошибки.
- Обнаружить ошибку, выполнить запрос повторной передачи поврежденного блока.
- Обнаружить ошибку и отбросить поврежденный блок.
- Обнаружить и исправить ошибку.
- Тройная модульная избыточность.

Текущая реализация:

- Самый мощный процессор, защищённый от излучения — BAE RAD5545.



Помехоустойчивые коды — коды, позволяющие обнаружить и (или) исправить ошибки в кодовых словах, которые возникают при передаче по каналам связи.

- 1) Блочные — фиксированные блоки длиной i символов преобразуются в блоки длиной n символов:
 - Неравномерные — редко используемые символы кодируются большим количеством символов (имеют большую длину).
 - Равномерные — длина блока (символа) постоянна:
 - а) Неразделимые — коды с постоянной плотностью единиц.
 - б) Разделимые — можно отделить (выделить) служебные биты r от информационных битов i .
- 2) Непрерывные (свёрточные) — передаваемая информационная последовательность не разделяется на блоки.

Коэффициент избыточности — отношение числа проверочных разрядов (r) к общему числу разрядов (n).

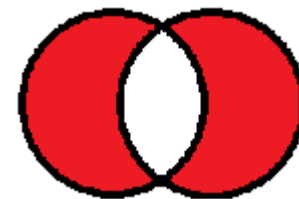
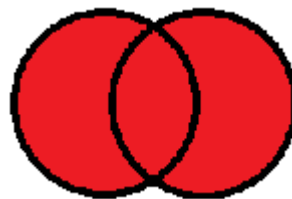
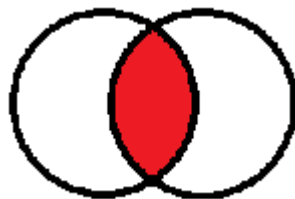


Контрольная сумма — некоторое число, рассчитанное путем применения определенного алгоритма к набору данных и используемое для проверки целостности этого набора данных при их передаче или хранении.

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

Сумма по модулю 2 — исключающее «ИЛИ» (для двух операндов), логическое сложение или битовое сложение, разность двух/трёх множеств.

$$A \bmod 2 B = A \oplus B = (\neg(A \wedge B)) \wedge (A \vee B) = \neg((A \wedge B) \vee (\neg A \vee \neg B))$$





Пример. Есть 1 информационный бит $i = 1$.
 К нему идёт один бит чётности r_1 .
 $i = r_1, i \oplus r_1 = 0$.

i исх	r_1 исх	i рез	r_1 рез	$i \text{ рез} \oplus r_1 \text{ рез}$
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

A	B	C	$A \oplus B \oplus C$
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



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

Синдром последовательности S — набор контрольных сумм информационных и проверочных разрядов.



Ричард
Уэсли
Хэмминг
(1915–1998)

Пример. Есть 1 информационный бит $i = 1$.
К нему идут два бита чётности r_1 и r_2 .

$$i = r_1 = r_2, s_1 = i \oplus r_1, s_2 = i \oplus r_2.$$

i исх	r_1 исх	r_2 исх	i рез	r_1 рез	r_2 рез	s_1	s_2
1	1	1	0	0	0	0	0
1	1	1	0	0	1	0	1
1	1	1	0	1	0	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	0	1	1
1	1	1	1	0	1	1	0
1	1	1	1	1	0	0	1
1	1	1	1	1	1	0	0

r_1	r_2	r_3	i_1	i_2	i_3	i_4
-------	-------	-------	-------	-------	-------	-------

$$r_1 = i_1 \oplus i_2 \oplus i_4$$

$$r_2 = i_1 \oplus i_3 \oplus i_4$$

$$r_3 = i_2 \oplus i_3 \oplus i_4$$

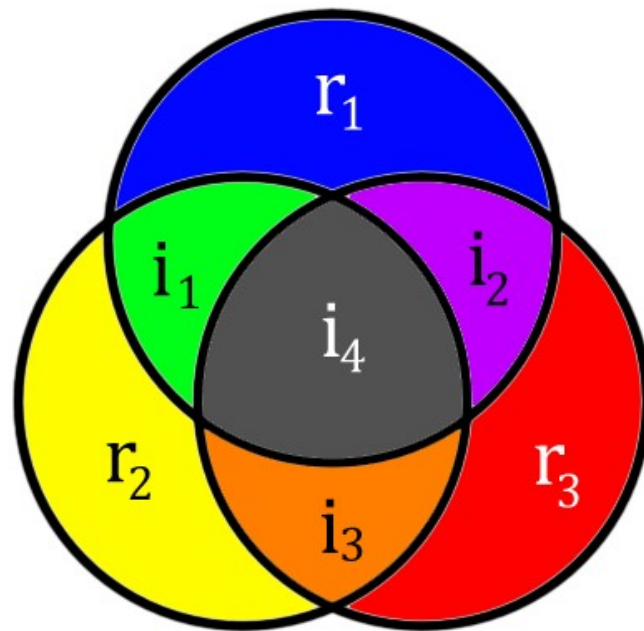
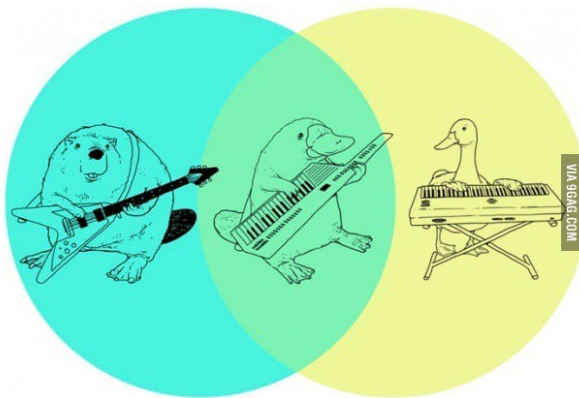


Таблица кода Хэмминга

	1	2	3	4	5	6	7	
2^x	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

$$r_1 = i_1 \oplus i_2 \oplus i_4$$

$$r_2 = i_1 \oplus i_3 \oplus i_4$$

$$r_3 = i_2 \oplus i_3 \oplus i_4$$

$$s_1 = r_1 \oplus i_1 \oplus i_2 \oplus i_4$$

$$s_2 = r_2 \oplus i_1 \oplus i_3 \oplus i_4$$

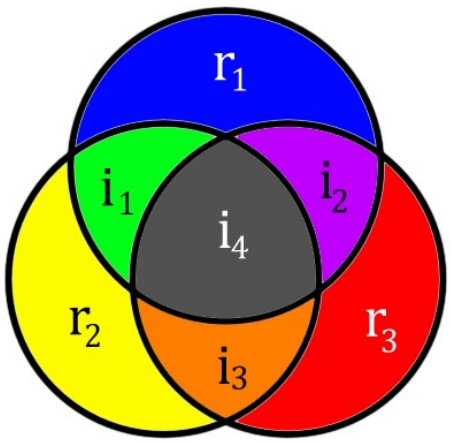
$$s_3 = r_3 \oplus i_2 \oplus i_3 \oplus i_4$$

Синдром S (s1, s2, s3)	000	001	010	011	100	101	110	111
Конфигурация ошибок (позиция в сообщении)	НЕТ	0001000	0100000	0000010	1000000	0000100	0010000	0000001
Ошибка в символе	НЕТ	r_3	r_2	i_3	r_1	i_2	i_1	i_4



Таблица кода Хэмминга (2)


	1	2	3	4	5	6	7	
Пример полученного сообщения	1	1	1	0	0	0	1	
2^x	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



$$s_1 = r_1 \oplus i_1 \oplus i_2 \oplus i_4 = 1 \oplus 1 \oplus 0 \oplus 1 = 1$$
$$s_2 = r_2 \oplus i_1 \oplus i_3 \oplus i_4 = 1 \oplus 1 \oplus 0 \oplus 1 = 1$$
$$s_3 = r_3 \oplus i_2 \oplus i_3 \oplus i_4 = 0 \oplus 0 \oplus 0 \oplus 1 = 1$$

Ошибка в бите i_4 .

Таблица кода Хэмминга (3)



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
2^x	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}	i_{11}	S
1	X		X		X		X		X		X		X		X	s_1
2		X	X			X	X			X	X			X	X	s_2
4				X	X	X	X					X	X	X	X	s_3
8								X	X	X	X	X	X	X	X	s_4

По таблице видно, за какие информационные биты отвечает каждый проверочный бит: контрольный бит с номером N контролирует все последующие N бит через каждые N бит, начиная с позиции N .

Аналогично с ошибочным битом.

Пример. Имеем синдром S (0,0,1,1). Проверяем, за какой бит отвечают только r_3 и r_4 .

Ответ: i_8 (12-й символ сообщения).

Диаграмма Венна для кода Хэмминга $r=4$

Где 5? Где 10?

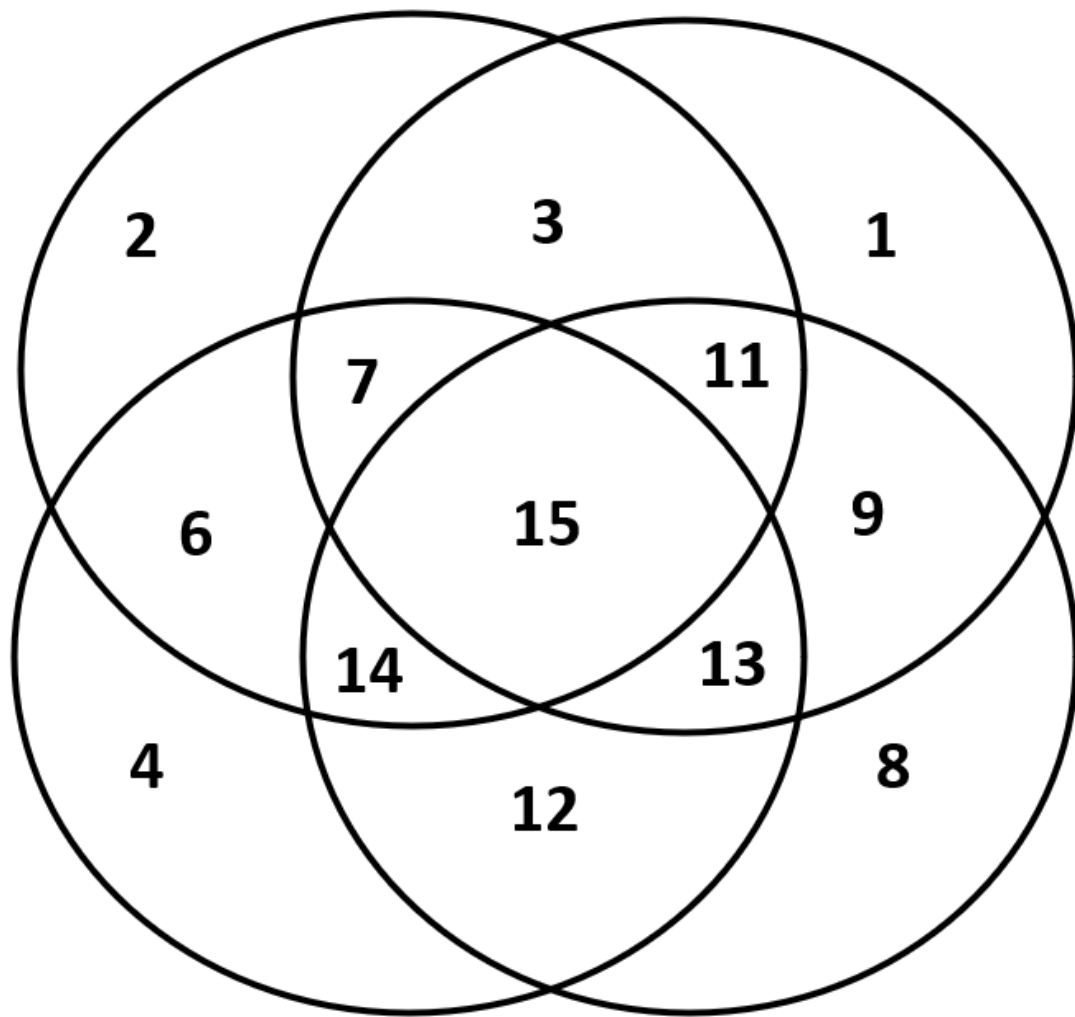


Схема декодирования кода Хэмминга

7-ми символьное
кодированное слово

4-х символьное
Информационное слово

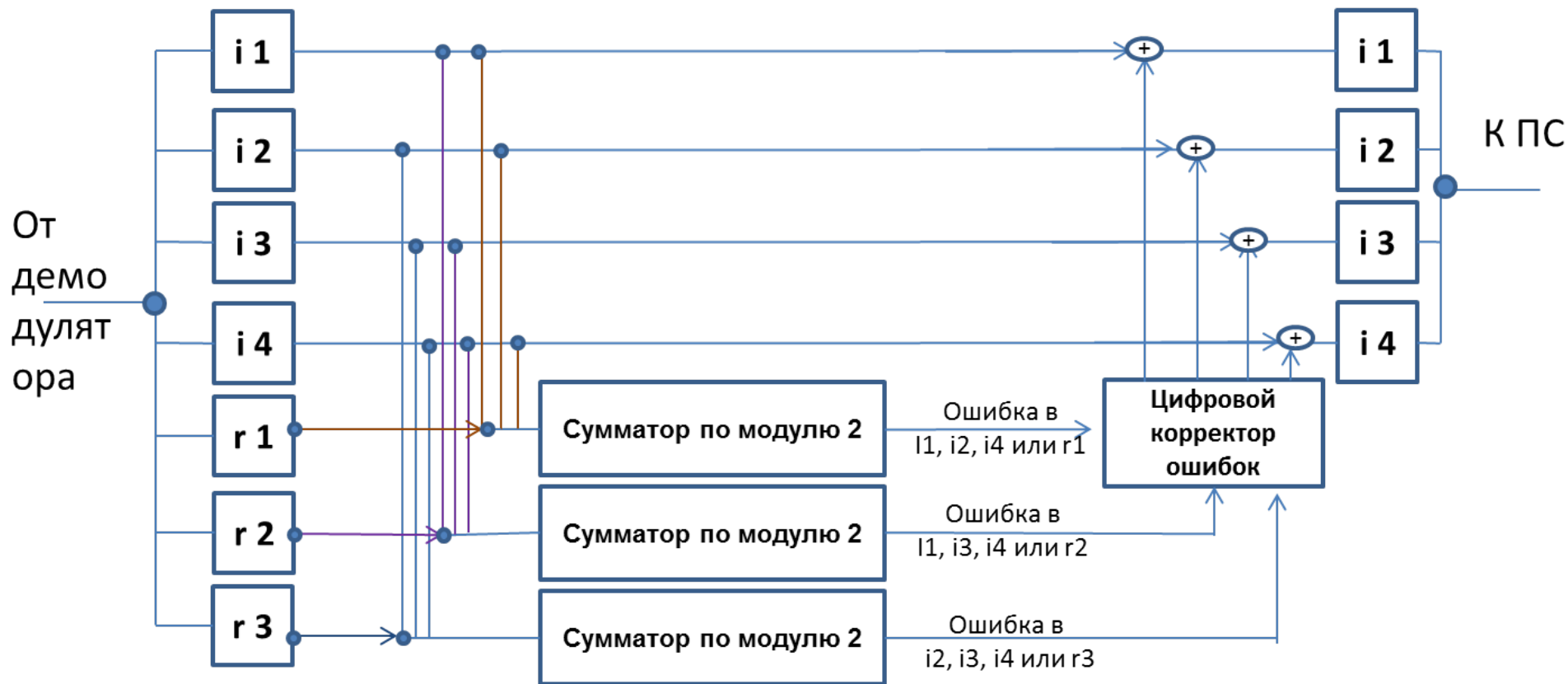
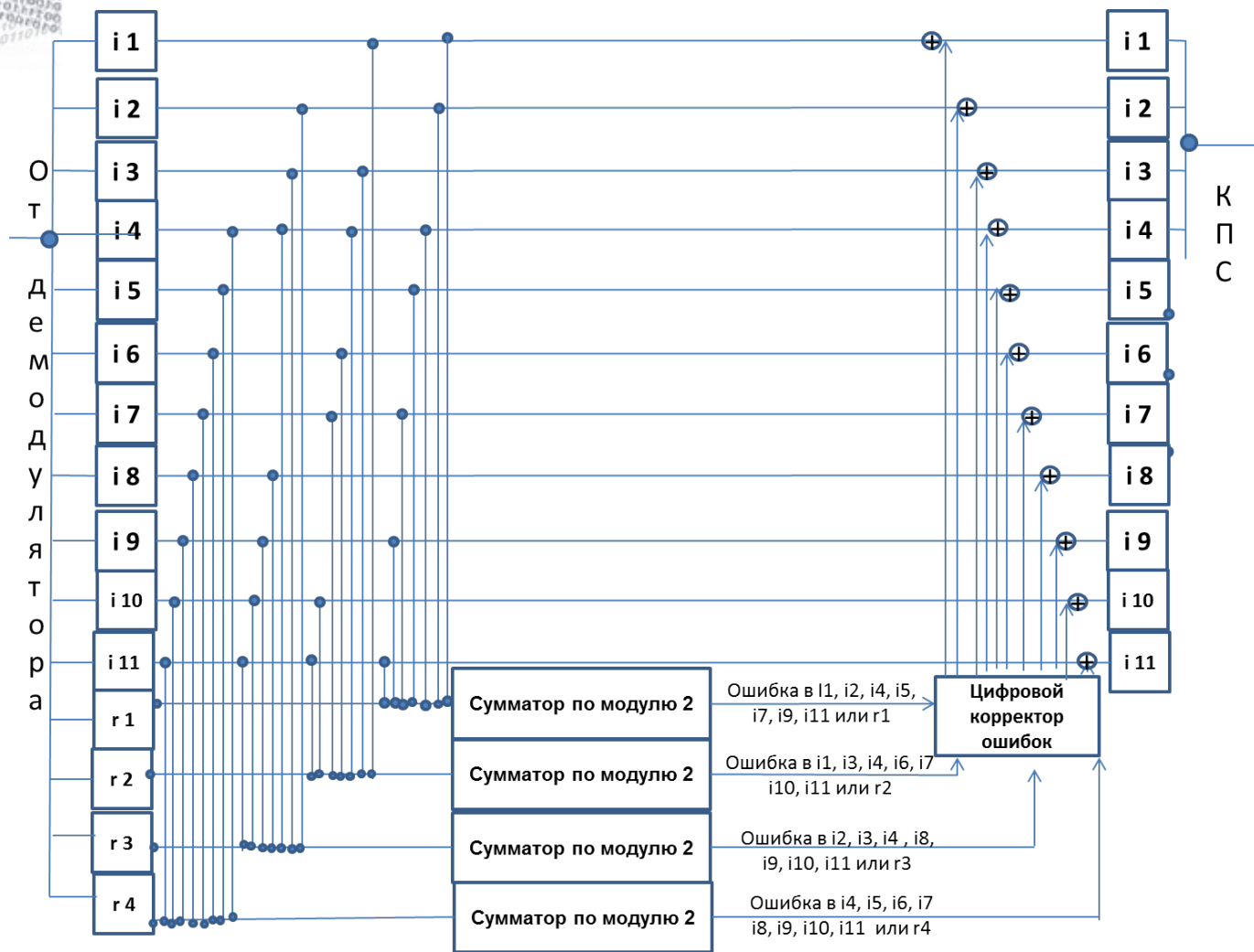


Схема декодирования кода Хэмминга (2)





Определение минимального числа контрольных разрядов: $2^r \geq r + i + 1$.

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

Диапазон информационных разрядов, i	Минимальное число контрольных разрядов, r
1	2
2-4	3
5-11	4
12-26	5
27-57	6

Коэффициент избыточности — отношение числа проверочных разрядов (r) к общему числу разрядов ($n = i + r$).



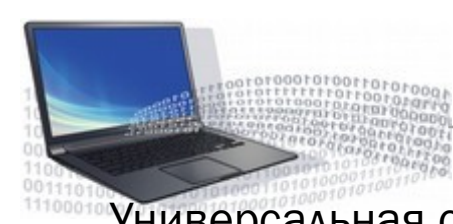
Расстояние Хэмминга (PX) — число символов, которое надо изменить в одном слове (разрешённой комбинации), чтобы получить другое (тоже разрешённую ситуацию).

Кодовое расстояние (d) — минимальное PX среди всех слов алфавита в коде постоянной длины, т.е. минимальное число искажённых символов для перехода.

Число обнаруживаемых ошибок = $d-1$

Число минимально обнаруживаемых и исправляемых ошибок (g): $d \geq 2g + 1$

r_1	r_2	i_1	r_3	i_2	i_3	i_4
0	0	0	0	0	0	0
0	1	0	1	0	0	1
1	1	0	0	0	1	0
...	
1	1	1	1	1	1	1



Универсальная система коррекции ошибок от MIT:

<https://news.mit.edu/2021/grand-decoding-data-0909>

Необычный подход к «шифрованию» данных от Sandia National Laboratories:

<https://techxplre.com/news/2021-08-error-secret-language.html>

Представление целых чисел в ограниченной двоичной разрядной сетке (РС) компьютера



Для хранения целой переменной в памяти компьютера используется фиксированное заранее известное число бит. Например, для хранения $a=2$ в компьютерную память будет записано следующее двоичное число, если используется 32-разрядный компьютер:

0000000000000000000000000000000010₍₂₎.

Процессор за один такт работы выполняет операцию сразу со всеми 32-мя битами:

$$\begin{array}{r} + 0000000000000000000000000000000010_{(2)} \\ \underline{00000000000000000000100000000000010_{(2)}} \\ 00000000000000000000100000000000100_{(2)} \end{array}$$

Пусть для хранения целого неотрицательного числа в переменной a используется k бит.

$$\text{MIN}(a) = 000\dots000_{(2)} = 0,$$

$$\text{MAX}(a) = 111\dots111_{(2)} = 2^k - 1.$$

$$\begin{array}{l} 999 = 1000 - 1 = 10^3 - 1 \\ 111_{(2)} = 1000_{(2)} - 1 = 2^3 - 1 \end{array}$$

Диапазон представления целых неотрицательных чисел в k -разрядной сетке: **от 0 до 2^k-1 .**



Представление целых чисел со знаком в компьютере

В ЭВМ нет способа обозначить в двоичной СС знак «МИНУС» перед числом. Способы решения этой проблемы с примерами для 4-разрядного компьютера:

- **Специальный знаковый бит (СЗБ)**
 $+5 = 0101_2$, $-5 = 1101_2$ (первый бит означает знак числа)
- **Фиксированное смещение влево (ФСВ)**
 $-5 = 0000_2$, $-4 = 0001_2$, ..., $+10 = 1111_2$ (все числа уменьшены на 5)
- **Нега-двоичная система счисления (НДСС)**
 $-5 = 1111_{-2}$, $+5 = 0101_{-2}$ (основание СС равно «-2»)
- **Обратный/инверсный код (ОК)**
 $+5 = 0101_2$, $-5 = 1010_2$ (инвертируются все биты)
- **Дополнительный код (ДК)**
 $+5 = 0101_2$, $-5 = 1011_2$ (инвертировать все биты и прибавить 1)



Целые числа со знаком в трёхразрядном коде

Для сравнения – диапазон представления целых **неотрицательных** чисел в трёхразрядной сетке: от $000_{(2)}$ до $111_{(2)}$, т. е. от 0 до 7.

Трёхразрядный код	СЗБ	ФСВ (5)	НДСС	ОК	ДК
000	+0	-5	0	+0	0
001	1	-4	1	1	1
010	2	-3	-2	2	2
011	3	-2	-1	3	3
100	-0	-1	4	-3	-4
101	-1	0	5	-2	-3
110	-2	1	2	-1	-2
111	-3	2	3	-0	-1
Диапазон	-3..+3	-5..+2	-2..+5	-3..+3	-4..+3



Целые числа со знаком в n -разрядном компьютере

Имея n -разрядный двоичный регистр, можно закодировать 2^n разных символов. Для кодирования целых чисел без знака используется диапазон от 0 до $2^n - 1$. Каков диапазон хранимых чисел со знаком в n -разрядном регистре?

- 1. Специальный знаковый бит (СЗБ):
от $-(2^{n-1} - 1)$ до $+(2^{n-1} - 1)$.
- 2. Фиксированное смещение влево (ФСВ):
от $(-S)$ до $(2^n - 1 - S)$, где S – смещение.
- 3. Нега-двоичная система счисления (НДСС):
чётное n : от $-(2^n - 1) * 2/3$ до $(2^n - 1)/3$,
нечётное n : от $-(2^{n-1} - 1) * 2/3$ до $(2^{n+1} - 1)/3$,
любое n : от $-(2^{n-(n \bmod 2)} - 1) * 2/3$ до $(2^{n+(n \bmod 2)} - 1)/3$.
- 4. Обратный/инверсный код (ОК):
от $-(2^{n-1} - 1)$ до $+(2^{n-1} - 1)$.
- 5. Дополнительный код (ДК):
от (-2^{n-1}) до $(2^{n-1} - 1)$.

min→	1	1	1	1	...	1	1	1	1
max→	0	1	1	1	...	1	1	1	1
	0	0	0	0	...	0	0	0	0
	1	1	1	1	...	1	1	1	1
	...	1	0	1	0	1	0	1	0
	...	0	1	0	1	0	1	0	1
	1	0	0	0	...	0	0	0	0
	0	1	1	1	...	1	1	1	1
	1	0	0	0	...	0	0	0	0
	0	1	1	1	...	1	1	1	1

Как хранится число «-2» в памяти десятиразрядного компьютера?

Решение

1 шаг: записать число «+2», используя все доступные разряды

$$2_{10} = 0000000010_2$$

2 шаг: инвертировать каждый бит полученного числа:

$$0000000010_2 \rightarrow 1111111101_2$$

3 шаг: прибавить один

$$\begin{array}{r} 1111111101_2 \\ + 0000000001_2 \\ \hline 1111111110_2 \end{array}$$

4 шаг: радоваться результату: $-2_{10} = 1111111110_2$ (обратный перевод выполняется так же)

Иллюстрация эффекта $2 + (-2) = 0 \rightarrow$

$$\begin{array}{r} 0000000010_2 \\ + 1111111110_2 \\ \hline 1000000000_2 \end{array}$$

– это ноль, т. к. 11-го разряда нет



$$\begin{array}{r}
 + \quad 0111_2 \\
 \quad \underline{1011_2} \\
 \mathbf{1}0010_2
 \end{array}$$

$$\begin{array}{r}
 \text{СЗБ} \\
 +7 \\
 \quad \underline{-3} \\
 +2
 \end{array}$$

$$\begin{array}{r}
 \text{НДСС} \\
 +3 \\
 \quad \underline{-9} \\
 -2
 \end{array}$$

$$\begin{array}{r}
 \text{ОК} \\
 +7 \\
 \quad \underline{-4} \\
 +2
 \end{array}$$

$$\begin{array}{r}
 \text{ДК} \\
 +7 \\
 \quad \underline{-5} \\
 +2
 \end{array}$$

Как придумали правило ДК? Почему нужно инвертировать биты и прибавлять 1?

$$x_{(2,n)} + \text{inv}(x_{(2,n)}) = \dots 11111111_{(2,n)} = 2^n - 1. \text{ Пример: } 0101_{(2,4)} + 1010_{(2,4)} = 1111_{(2,4)} = 2^4 - 1$$

$$\text{inv}(x_{(2,n)}) + 1 = 2^n - x_{(2,n)}$$

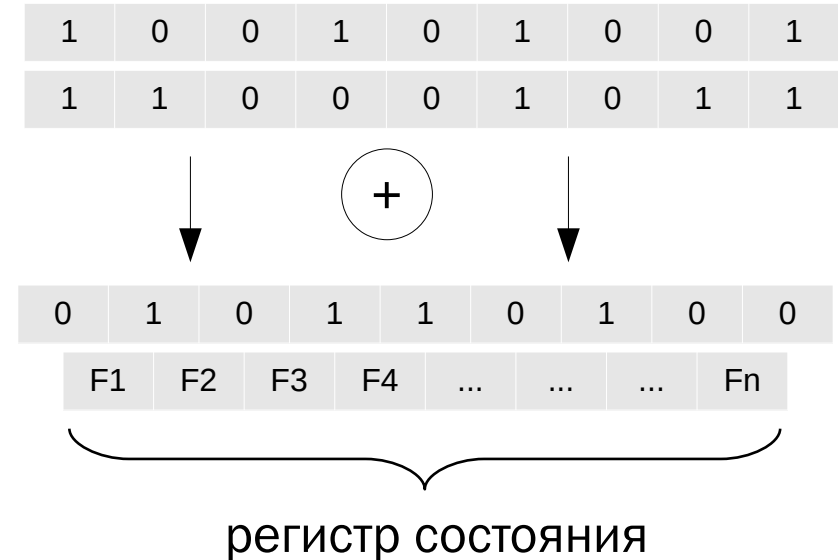
$$\text{inv}(x_{(2,n)}) + 1 = -x_{(2,n)}$$

$$a_{(2,n)} - b_{(2,n)} = a_{(2,n)} + (-b_{(2,n)}) = a_{(2,n)} + (2^n - b_{(2,n)}) = a_{(2,n)} + (\text{inv}(b_{(2,n)}) + 1)$$

Арифметические операции в ограниченной разрядной сетке



- После любой арифметической операции процессор автоматически без явной команды от программиста устанавливает флаги, характеризующие состояние процессора.
- Совокупность этих флагов называется регистром состояния.
- Программист может анализировать содержимое регистра состояния процессора для принятия решений в программе.



if (F1 == 0) then ... else ...;



SF – Sign Flag. Равен 1, если результат операции отрицателен, иначе – 0.

ZF – Zero Flag. Равен 1, если результат операции равен нулю.

PF – Parity Flag. Равен 1, если младший байт результата выполнения операции содержит чётное число единиц.

AF – Adjust Flag. Равен 1, если произошёл заём или перенос между первым и вторым полубайтом (нибблом).

CF – Carry Flag. Равен 1, если происходит перенос за пределы разрядной сетки или заём извне.

OF – Overflow Flag. Равен 1, если результат операции не помещается разрядную сетку (при использовании дополнительного кода).



OF – Overflow Flag. Принимает значение 1, если в результате выполнения операции со знаковыми числами появляется одна из ошибок:

- 1) складываем положительные числа, получаем неположительный результат;
- 2) складываем отрицательные числа, получаем неотрицательный результат.

Примеры для 4-разрядного компьютера:

$$0100_{(2)} + 0001_{(2)} = 0101_{(2)} \text{ (CF=0, OF=0) : } +4 + 1 = +5$$

$$0110_{(2)} + 1001_{(2)} = 1111_{(2)} \text{ (CF=0, OF=0) : } +6 - 7 = -1 \text{ (1111}_2 \text{ в доп. коде это } -1_{10})$$

$$1000_{(2)} + 0001_{(2)} = 1001_{(2)} \text{ (CF=0, OF=0) : } -8 + 1 = -7$$

$$1100_{(2)} + 1100_{(2)} = 1000_{(2)} \text{ (CF=1, OF=0) : } -4 - 4 = -8$$

$$1000_{(2)} + 1000_{(2)} = 0000_{(2)} \text{ (CF=1, OF=1) : } -8 - 8 = 0$$

$$0101_{(2)} + 0100_{(2)} = 1001_{(2)} \text{ (CF=0, OF=1) : } +5 + 4 = -7$$

Пример установки флагов состояния процессора

16-разрядный компьютер

Пример 1

$$\begin{array}{r} 0010.0101.0000.1100_{(2)} \quad + \quad 9484_{(10)} \\ + \quad 0011.1101.1010.0100_{(2)} \quad +15780_{(10)} \\ \hline 0110.0010.1011.0000_{(2)} \quad = \quad +25264_{(10)} \end{array}$$

CF=0, OF=0, ZF=0, AF=1, SF=0, PF=0

Пример 2

$$\begin{array}{r} 0110.0010.1010.1001_{(2)} \quad +25257_{(10)} \\ + \quad 0011.1101.1010.1100_{(2)} \quad +15788_{(10)} \\ \hline 1010.0000.0101.0101_{(2)} \quad = \quad -24491_{(10)} \end{array}$$

CF=0, OF=1, ZF=0, AF=1, SF=1, PF=1

Пример 3

$$\begin{array}{r} 1110.0111.0110.1000_{(2)} \quad - \quad 6296_{(10)} \\ + \quad 0110.0010.1011.0000_{(2)} \quad +25264_{(10)} \\ \hline 1.0100.1010.0001.1000_{(2)} \quad = \quad +18968_{(10)} \end{array}$$

CF=1, OF=0, ZF=0, AF=0, SF=0, PF=1