Непозиционные системы счисления

Славянская кириллическая нумерация: аддитивная запись	**А** = 1, **В** = 2, **Г** = 3, **Д** = 4… 444 = **УМД**, = 400 (**У**) + 40 (**М**) + 4 (**Д**).
Китайская нумерация: аддитивно-мультипликативная запись	444 = 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.

Алфавит и его подмножества

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* є *X*).

Слово – конечная последовательность знаков (букв) алфавита.

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

Кодирование данных

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

Декодирование — 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 бит / 400 бит = 1,25.
Отношение сжатия = 400 бит / 500 бит = 0,8.

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

Типы и методы 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.

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.

Теорема Шеннона об источнике шифрования

Теорема Шеннона об источнике шифрования устанавливает предел максимального сжатия данных и числовое значение энтропии (меры) Шеннона: невозможно сжать данные настолько, что оценка кода (среднее число бит на символ) меньше, чем энтропия Шеннона исходных данных, без потери точности информации.

$$i(S) = -\sum_{i=1}^{N} p_i \cdot \log_2 p_i$$

P1	P2	Энтропия
0.50	0.50	1.00
0.60	0.40	0.97
0.70	0.30	0.88
0.80	0.20	0.72
0.90	0.10	0.47
0.99	0.01	0.08

Теорема Шеннона об источнике шифрования (2)

Пример. Дан алфавит «ABCDE» с вероятностями встречаемости символов 0,4, 0,2, 0,2, 0,1 и 0,1 соответственно.

Вероятность строки «AAAABBCCDE» = $0{,}4^4 * 0{,}2^2 * 0{,}2^2 * 0{,}1^1 * 0{,}1^1 =$
$= 4{,}096 * 10^{-7}$.

$\log_2 P = -21.21928$.

Наименьшее в среднем число для кодирования строки равно 22 бит.

Энтропийный кодер — кодер, достигающий сжатия максимально близкого к энтропии.

Проблемы округления чисел

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

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

10000 строк

Сумма1 vs Сумма2

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

В среднем (Сумма2 – Сумма1) =
= (10000/100) * 50 коп. =
= 50 руб. переплаты!

Проблемы округления чисел в различных СС

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

Основание 10: **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:

Основание 7: **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 ≈ 4, но 2,5 ≈ 2.
- **К ближайшему нечётному**: 3,5 ≈ 3, но 2,5 ≈ 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.

Пример округления

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

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:

$$00000000000000000000000000000010_{(2)}.$$

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

$$\begin{array}{l} +\ 00000000000000000000000000000010_{(2)} \\ \underline{\ 00000000000000010000000000000010_{(2)}} \\ \ \ 00000000000000010000000000000100_{(2)} \end{array}$$

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

$MIN(a) = 000...000_{(2)} = 0,$

$MAX(a) = 111...111_{(2)} = 2^k - 1.$

$999 = 1000 - 1 = 10^3 - 1$

$111_{(2)} = 1000_{(2)} - 1 = 2^3 - 1$

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

Дополнительный код: пример

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

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

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

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

$$\begin{array}{r} +\ 0000000010_2 \\ \underline{1111111110_2} \\ 10000000000_2 \end{array}$$

– это ноль, т. к. 11-го разряда нет

Почему дополнительный код

$$
\begin{array}{r}
+\ 0\,1\,1\,1_2 \\
\underline{1\,0\,1\,1_2} \\
1\ 0\,0\,1\,0_2
\end{array}
$$

СЗБ	НДСС	ОК	ДК
+7	+3	+7	+7
-3	-9	-4	-5
+2	-2	+2	+2

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

$x_{(2,n)} + \text{inv}(x_{(2,n)}) = \ldots11111111_{(2,n)} = 2^n - 1$. Пример: $0101_{(2,4)} + 1010_{(2,4)} = 1111_{(2,4)} = 2^4 - 1$

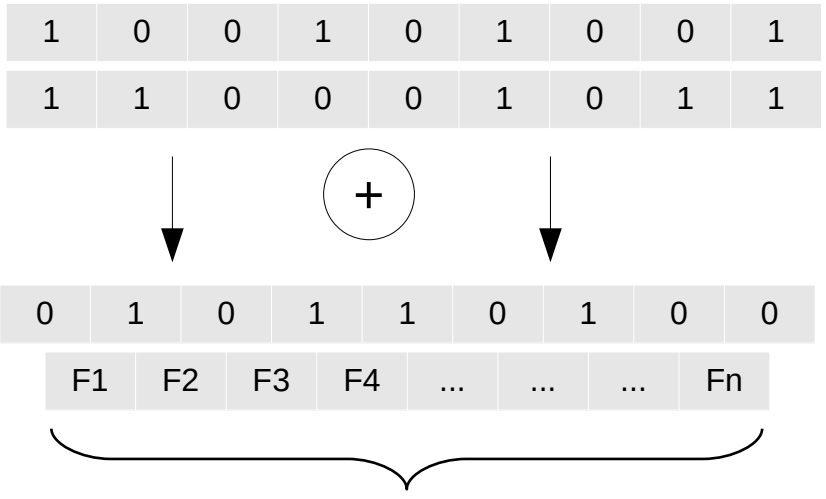
$\text{inv}(x_{(2,n)}) + 1 = 2^n - x_{(2,n)}$

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

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

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.



регистр состояния

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

Флаги состояния процессора

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).

Флаги переполнения и переноса (OF, CF)

***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:

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

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

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

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

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

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

Пример установки флагов состояния процессора

16-разрядный компьютер

Пример 1

$0010.0101.0000.1100_{(2)}$ + 9484$_{(10)}$

\+ $0011.1101.1010.0100_{(2)}$ +15780$_{(10)}$

$0110.0010.1011.0000_{(2)}$ = +25264$_{(10)}$

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

Пример 2

$0110.0010.1010.1001_{(2)}$ +25257$_{(10)}$

\+ $0011.1101.1010.1100_{(2)}$ +15788$_{(10)}$

$1010.0000.0101.0101_{(2)}$ = -24491$_{(10)}$

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

Пример 3

$1110.0111.0110.1000_{(2)}$ - 6296$_{(10)}$

\+ $0110.0010.1011.0000_{(2)}$ +25264$_{(10)}$

$1.0100.1010.0001.1000_{(2)}$ = +18968$_{(10)}$

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