

Представление данных в компьютере

АКОС, МФТИ

19 сентября, 2024



Как работают целые числа

$$137_{10} =$$


$$= \begin{array}{|c|c|c|c|c|c|c|c|} \hline +128 & +64 & +32 & +16 & +8 & +4 & +2 & +1 \\ \hline \end{array} =$$

$$= \begin{array}{cccccccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{array}_2$$

Целочисленные типы в C

- `char` : 1 байт (или `CHAR_BIT` бит) данных;
- `short` и `int` : не менее 16 бит данных;
- `long` : не менее 32 бит данных;
- `long long` : не менее 64 бит данных.

Типы фиксированной длины (`#include <stdint.h>`):

- `int8_t` и `uint8_t` : строго 8 бит;
 - `int16_t` и `uint16_t` : строго 16 бит;
 - `int32_t` и `uint32_t` : строго 32 бита;
 - `int64_t` и `uint64_t` : строго 64 бита.
- 

Как работают знаковые числа

$$-23_{10} =$$

$$= \begin{array}{|c|c|c|c|c|c|c|c|} \hline -128 & +64 & +32 & +16 & +8 & +4 & +2 & +1 \\ \hline \end{array} =$$

$$= \begin{array}{cccccccc} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}_2$$

- ⚠ : Любое отрицательное число начинается с **1** и наоборот;
- ⚠ : Конвертация знаковых типов друг к другу становится менее тривиальным.

Знаковые и беззнаковые типы в С

- `char` : не определено стандартом.
- `short` , `int` , `long` и `long long` : по умолчанию **знаковые**;
- Любой из типов выше можно сделать:
 - **Знаковым** (напр., `signed char`);
 - **Беззнаковым** (напр., `unsigned int`).

Типы фиксированной длины (`#include <stdint.h>`):

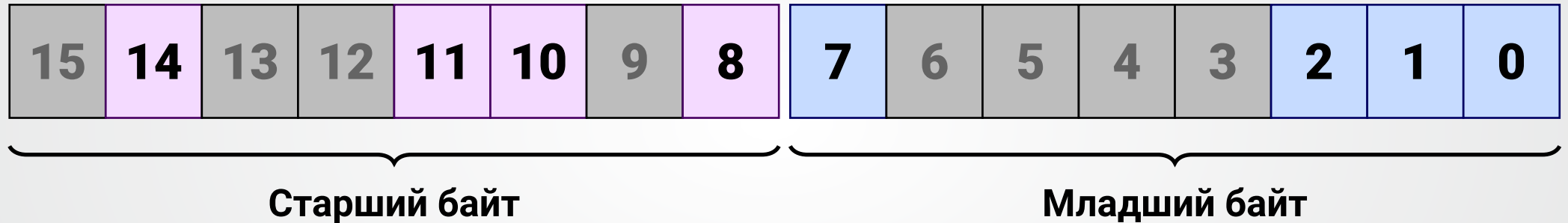
- `int16_t` : **знаковое** 16-битное число;
- `uint16_t` : **беззнаковое** 16-битное число (`u` в начале от слова `unsigned`);



: Знаковые типы **нельзя переполнять** - в Си это UB. Беззнаковые – можно.

Как хранить длинные типы?

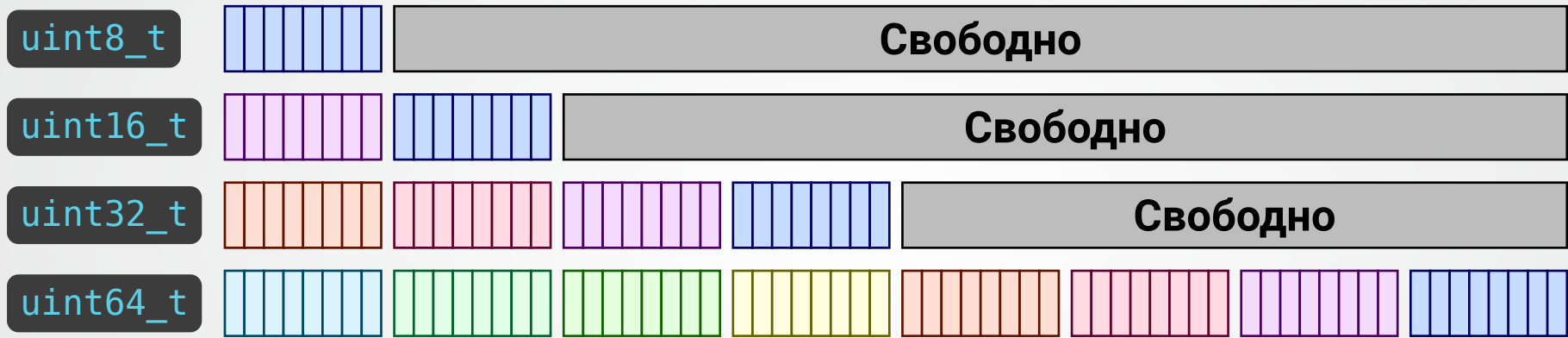
```
(uint16_t) 19847 =
```



Не будет ли проблем с такой схемой...

Что может пойти не так?

```
1  int main() {  
2      uint64_t my_long = 42;  
3  
4      printf("%d\n", &my_long); // Что выведет?  
5  }
```



: Младший **синий** байт оказывается в разных местах.

Сложности приведения типов

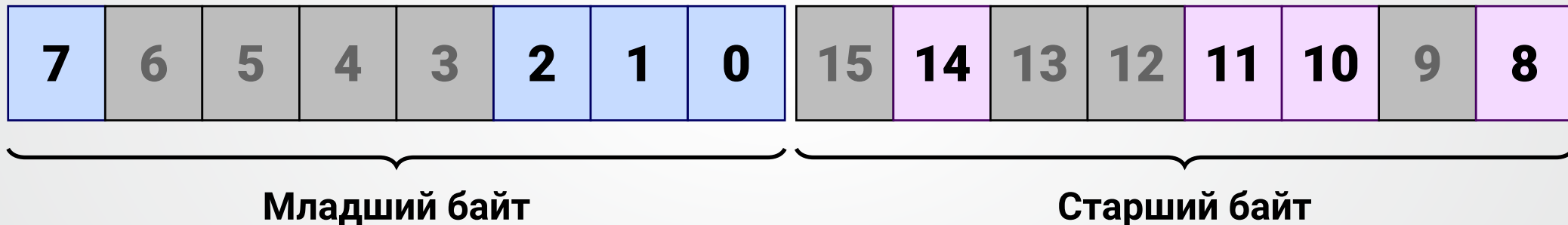
```
1  uint8_t a_byte = 42;
2
3  uint16_t a_16b = a_byte; // Перенесёт 42 во второй байт
4  uint32_t a_32b = a_byte; // Перенесёт 42 в четвёртый байт
5  uint64_t a_64b = a_byte; // Перенесет 42 в восьмой байт
```



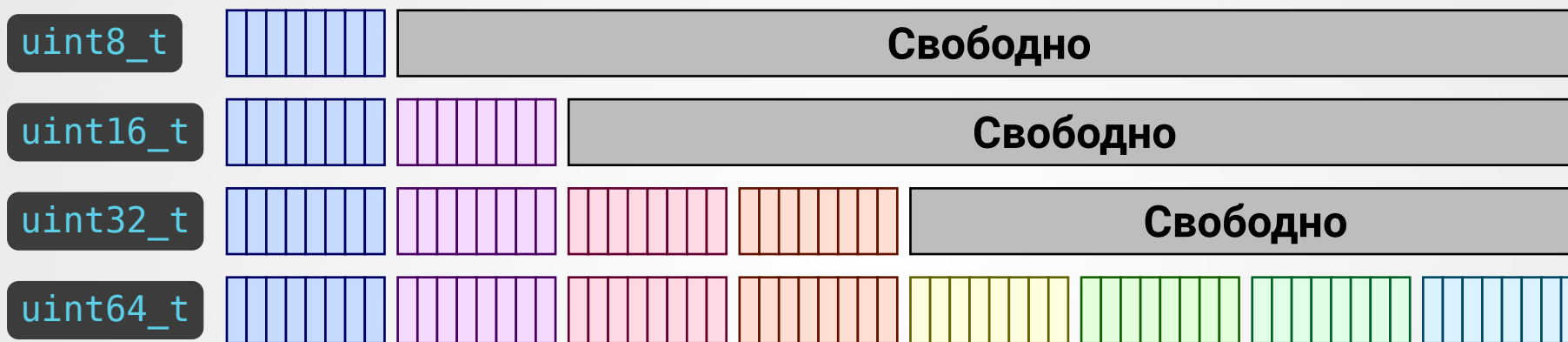
: В схеме Big-Endian **каждый целочисленный каст** требует **менять порядок байт**:

Что, если хранить байты наоборот?

```
(uint16_t) 19847 =
```



Что, если хранить байты наоборот?



✓ В такой схеме (Little-Endian) приведение целочисленных типов не требует перемещения байт.

Схема Big-Endian (BE, сначала старший байт)

- + Удобная для человека. Порядок бит как в десятичной записи;
- + Позволяет ускорять `strcmp` и `memcmp` ;
- Иногда сложнее приводить типы.

Схема Little-Endian (LE, сначала младший байт)

- + Легко кастуется туда-обратно;
- Чуть-чуть ломает мозг.

К чему пришли люди:

- x86 всегда LE. ARM по умолчанию LE, но поддерживает оба варианта.
- В Big-Endian вводятся двоичные литералы: `0b1000000000000000` `==` `32768` , а не `128`
- А еще Big-Endian используется при передачи данных по сети.

Дробные числа

Дробные числа

$$10.675_{10} =$$

$$= \begin{array}{|c|c|c|c|c|c|c|c|} \hline +16 & +8 & +4 & +2 & +1 & +0.5 & +0.25 & +0.125 \\ \hline \end{array} =$$

$$= 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1_2$$

Несколько младших разрядов можно зарезервировать под дробную часть. Получится **fixed-point**.

Недостатки **fixed-point**

- Для каждой задачи **нужно подбирать оптимальное количество дробных бит**.
- При неоптимальном порядке представления либо **теряет относительную точность**, либо **быстро переполнится**.

К чему пришли люди:

- Давайте **менять число дробных бит на ходу**;
- Выделим несколько бит под **счётчик**;
- Назовём это ✨ **floating-point** ✨.

floating-point в IEEE 754

$$\pi \approx 3.125 =$$

$$= 1 \cdot \begin{array}{|c|c|c|c|c|c|c|c|} \hline \cdot -1 & \cdot 2^4 & \cdot 2^2 & \cdot 2^1 & +\frac{1}{2} & +\frac{1}{4} & +\frac{1}{8} & +\frac{1}{16} \\ \hline \end{array} \cdot 2^{-3} =$$

$$= \mathbf{0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1}_2$$

(Вообще, в IEEE 754 минимум 16 бит, но суть та же)



Интерактивная визуализация работы

float

$$-1^{\boxed{0}} \cdot 2^{\boxed{100}}_2^{-3} \cdot 1.\boxed{1001}_2$$

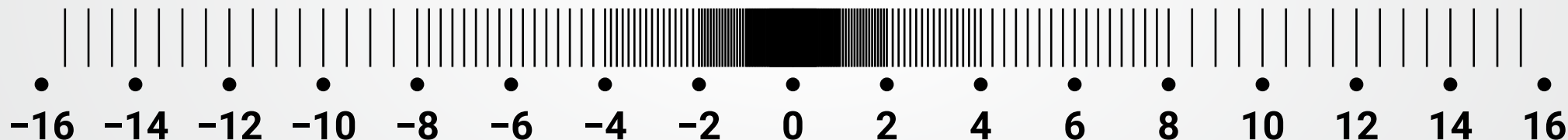
На самом деле, двоичная аналогия этого:

$$1.616255 \cdot 10^{-35}$$

fixed-point (знаковый, 8 бит, 3-б. дробь)



floating-point (3-б. экспонента, 4-б. мантисса)



- +** Макс. погрешность на (1, 15) стала **3%** против **5.6%**;
- **32 из 256 значений (12%)** уходят на служебные (**NaN**, **Inf**).

Спец. значения **floating-point** в IEEE 754

+Inf	0	1	1	1	0	0	0	0
-Inf	1	1	1	1	0	0	0	0
qNaN	0	1	1	1	1	0	0	0
sNaN	0	1	1	1	0	1	0	0
0	0	0	0	0	0	0	0	0
-0	1	0	0	0	0	0	0	0

- При нулевой экспоненте включается **денормализованный режим**. Он заменяет старшую единицу на ноль. Так сохраняется плотность значений близко к нулю.

Сложение floating-point

$$A = 4.5$$

$$= -1 \boxed{0} \cdot 2^{\boxed{101}_2 - 3} \cdot 1.\boxed{0010}_2$$

0	1	0	1	0	0	1	0
---	---	---	---	---	---	---	---

$$B = -4$$

$$= -1 \boxed{1} \cdot 2^{\boxed{101}_2 - 3} \cdot 1.\boxed{0000}_2$$

1	1	0	1	0	0	0	0
---	---	---	---	---	---	---	---

$$A + B =$$

Сложение floating-point

$$A = 4.5$$

$$= -1 \boxed{0} \cdot 2^{\boxed{101}_2 - 3} \cdot 1.\boxed{0010}_2$$



$$B = -4$$

$$= -1 \boxed{1} \cdot 2^{\boxed{101}_2 - 3} \cdot 1.\boxed{0000}_2$$



$$A + B = 2^{101_2 - 3} \cdot (1.0010_2 - 1.0000_2)$$

Сложение floating-point

$$A = 4.5$$

$$= -1 \boxed{0} \cdot 2^{\boxed{101}_2 - 3} \cdot 1.\boxed{0010}_2$$



$$B = -4$$

$$= -1 \boxed{1} \cdot 2^{\boxed{101}_2 - 3} \cdot 1.\boxed{0000}_2$$

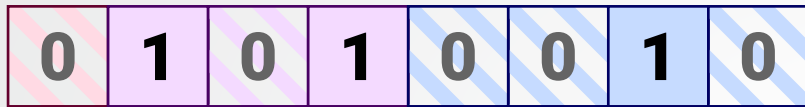


$$A + B = 2^{101_2 - 3} \cdot (1.0010_2 - 1.0000_2) \approx$$
$$\approx 2^{101_2 - 3} \cdot 0.0010_2$$

Сложение floating-point

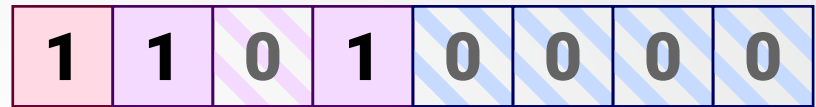
$$A = 4.5$$

$$= -1 \boxed{0} \cdot 2^{\boxed{101}_2 - 3} \cdot 1.\boxed{0010}_2$$



$$B = -4$$

$$= -1 \boxed{1} \cdot 2^{\boxed{101}_2 - 3} \cdot 1.\boxed{0000}_2$$



$$A + B = 2^{101_2 - 3} \cdot (1.0010_2 - 1.0000_2) \approx$$

$$\approx 2^{101_2 - 3} \cdot 0.0010_2 =$$

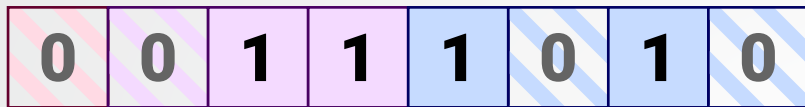
$$= -1 \boxed{0} \cdot 2^{\boxed{010}_2 - 3} \cdot 1.\boxed{0000}_2 = 0.5$$



Сложение floating-point разных порядков

$$A = 1.625$$

$$= -1 \boxed{0} \cdot 2^{\boxed{011}} \cdot 1.\boxed{1010}_2$$



$$B = 4.5$$

$$= -1 \boxed{0} \cdot 2^{\boxed{101}} \cdot 1.\boxed{0010}_2$$

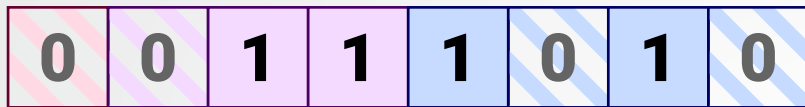


$$A + B =$$

Сложение floating-point разных порядков

$$A = 1.625$$

$$= -1 \boxed{0} \cdot 2^{\boxed{011}}_2 \cdot 1.\boxed{1010}_2$$



$$B = 4.5$$

$$= -1 \boxed{0} \cdot 2^{\boxed{101}}_2 \cdot 1.\boxed{0010}_2$$



$$A + B = 2^{101_2-3} \cdot (1.1010_2 \cdot 2^{-2} + 1.0010_2)$$

Сложение floating-point разных порядков

$$A = 1.625$$

$$= -1 \boxed{0} \cdot 2^{\boxed{011}}_2 \cdot 1.\boxed{1010}_2$$



$$B = 4.5$$

$$= -1 \boxed{0} \cdot 2^{\boxed{101}}_2 \cdot 1.\boxed{0010}_2$$



$$\begin{aligned} A + B &= 2^{101_2-3} \cdot (1.1010_2 \cdot 2^{-2} + 1.0010_2) = \\ &= 2^{101_2-3} \cdot (0.011010_2 + 1.0010_2) \end{aligned}$$

Сложение floating-point разных порядков

$$A = 1.625$$

$$= -1 \boxed{0} \cdot 2^{\boxed{011}} \cdot 1.\boxed{1010}_2$$



$$B = 4.5$$

$$= -1 \boxed{0} \cdot 2^{\boxed{101}} \cdot 1.\boxed{0010}_2$$

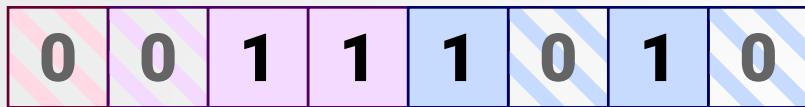


$$\begin{aligned} A + B &= 2^{101_2-3} \cdot (1.1010_2 \cdot 2^{-2} + 1.0010_2) = \\ &= 2^{101_2-3} \cdot (0.011010_2 + 1.0010_2) \approx 2^{101_2-3} \cdot 1.1001_2 \end{aligned}$$

Сложение floating-point разных порядков

$$A = 1.625$$

$$= -1 \boxed{0} \cdot 2^{\boxed{011}}_{2^{-3}} \cdot 1.\boxed{1010}_2$$

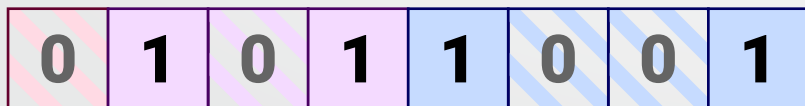


$$B = 4.5$$

$$= -1 \boxed{0} \cdot 2^{\boxed{101}}_{2^{-3}} \cdot 1.\boxed{0010}_2$$



$$\begin{aligned} A + B &= 2^{101_2-3} \cdot (1.1010_2 \cdot 2^{-2} + 1.0010_2) = \\ &= 2^{101_2-3} \cdot (0.011010_2 + 1.0010_2) \approx 2^{101_2-3} \cdot 1.1001_2 = \\ &= -1 \boxed{0} \cdot 2^{\boxed{101}}_{2^{-3}} \cdot 1.\boxed{1001}_2 = 6.25 \end{aligned}$$



Денормализованные floating-point

$$A = 0.5$$

$$= -1 \overset{\boxed{0}}{\cdot} 2^{\overset{\boxed{010}}{-3}} \cdot 1.\overset{\boxed{0000}}{2}$$



$$B = -0.03125$$

$$= -1 \overset{\boxed{1}}{\cdot} 2^{\overset{\boxed{000}}{-2}} \cdot 0.\overset{\boxed{0010}}{2}$$



$$A + B =$$

Денормализованные floating-point

$$A = 0.5$$

$$= -1 \overset{\text{sign}}{0} \cdot 2^{\overset{\text{exponent}}{010} - 3} \cdot 1.\overset{\text{fraction}}{0000}_2$$



$$B = -0.03125$$

$$= -1 \overset{\text{sign}}{1} \cdot 2^{\overset{\text{exponent}}{000} - 2} \cdot 0.\overset{\text{fraction}}{0010}_2$$



$$A + B = 2^{10_2 - 3} \cdot (1.0000_2 - 0.0010_2 \cdot 2^{-1})$$

Денормализованные floating-point

$$A = 0.5$$

$$= -1 \boxed{0} \cdot 2^{\boxed{010}}_{2^{-3}} \cdot 1.\boxed{0000}_2$$



$$B = -0.03125$$

$$= -1 \boxed{1} \cdot 2^{\boxed{000}}_{2^{-2}} \cdot 0.\boxed{0010}_2$$



$$\begin{aligned} A + B &= 2^{10_2-3} \cdot (1.0000_2 - 0.0010_2 \cdot 2^{-1}) = \\ &= 2^{10_2-3} \cdot (1.0000_2 - 0.00010_2) \end{aligned}$$

Денормализованные floating-point

$$A = 0.5$$

$$= -1 \boxed{0} \cdot 2^{\boxed{010}}_{2^{-3}} \cdot 1.\boxed{0000}_2$$



$$B = -0.03125$$

$$= -1 \boxed{1} \cdot 2^{\boxed{000}}_{2^{-2}} \cdot 0.\boxed{0010}_2$$



$$\begin{aligned} A + B &= 2^{10_2-3} \cdot (1.0000_2 - 0.0010_2 \cdot 2^{-1}) = \\ &= 2^{10_2-3} \cdot (1.0000_2 - 0.00010_2) \approx 2^{10_2-3} \cdot 0.1111_2 \end{aligned}$$

Денормализованные floating-point

$$A = 0.5$$

$$= -1 \overset{\text{sign}}{0} \cdot 2^{\overset{\text{exponent}}{010}_2 - 3} \cdot 1.\overset{\text{fraction}}{0000}_2$$

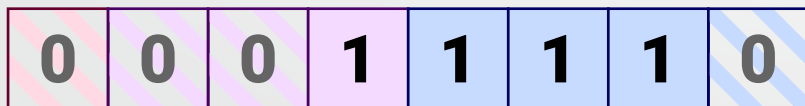


$$B = -0.03125$$

$$= -1 \overset{\text{sign}}{1} \cdot 2^{\overset{\text{exponent}}{000}_2 - 2} \cdot 0.\overset{\text{fraction}}{0010}_2$$



$$\begin{aligned} A + B &= 2^{10_2 - 3} \cdot (1.0000_2 - 0.0010_2 \cdot 2^{-1}) = \\ &= 2^{10_2 - 3} \cdot (1.0000_2 - 0.00010_2) \approx 2^{10_2 - 3} \cdot 0.1111_2 = \\ &= -1 \overset{\text{sign}}{0} \cdot 2^{\overset{\text{exponent}}{001}_2 - 3} \cdot 1.\overset{\text{fraction}}{1110}_2 = 0.46875 \end{aligned}$$



Умножение floating-point

$$A = 3.125 \approx \pi$$

$$= -1^{\boxed{0}} \cdot 2^{\boxed{100}}_2 \cdot 1.\boxed{1001}_2$$



$$B = 2.625 \approx e$$

$$= -1^{\boxed{0}} \cdot 2^{\boxed{100}}_2 \cdot 1.\boxed{0101}_2$$



$$A \cdot B = -1^{0+0} \cdot 2^{100_2-3+100_2-3} \cdot 1.1001_2 \cdot 1.0101_2$$

Умножение floating-point

$$A = 3.125 \approx \pi$$

$$= -1^{\boxed{0}} \cdot 2^{\boxed{100}}_2^{-3} \cdot 1.\boxed{1001}_2$$



$$B = 2.625 \approx e$$

$$= -1^{\boxed{0}} \cdot 2^{\boxed{100}}_2^{-3} \cdot 1.\boxed{0101}_2$$



$$A \cdot B = -1^{0+0} \cdot 2^{100_2-3+100_2-3} \cdot 1.1001_2 \cdot 1.0101_2 =$$

$$= -1^0 \cdot 2^{101_2-3} \cdot 10.00001101_2$$

Умножение floating-point

$$A = 3.125 \approx \pi$$

$$= -1^{\boxed{0}} \cdot 2^{\boxed{100}}_2^{-3} \cdot 1.\boxed{1001}_2$$



$$B = 2.625 \approx e$$

$$= -1^{\boxed{0}} \cdot 2^{\boxed{100}}_2^{-3} \cdot 1.\boxed{0101}_2$$



$$A \cdot B = -1^{0+0} \cdot 2^{100_2-3+100_2-3} \cdot 1.1001_2 \cdot 1.0101_2 =$$

$$= -1^0 \cdot 2^{101_2-3} \cdot 10.00001101_2 \approx$$

$$\approx -1^{\boxed{0}} \cdot 2^{\boxed{110}}_2^{-3} \cdot 1.\boxed{0000}_2 = 8$$



Размеры реальных **floating-point**

	Экспонента	Мантисса
float	8 бит	23 бита
double	11 бит	52 бита



Further reading

 [Как переводить строку в `float`](#)

 [Как переводить `float` в строку](#)

Нћљџєћїљџ

Кодировки

	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
0x00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
0x10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0x20	␣	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
0x30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0x40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0x50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
0x60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0x70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Это ASCII-таблица.

American Standard Code for Information Interchange

Кратно старше каждого в этой аудитории

	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
0x00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
0x10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0x20	▯	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
0x30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0x40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0x50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
0x60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0x70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
0x80	ђ	ѓ	Ѡ	ѓ	„	…	†	‡	€	‰	љ	‹	њ	ќ	ћ	џ
0x90	ђ	‘	’	“	”	•	—	—		™	љ	›	њ	ќ	ћ	џ
0xa0		Ў	ў	Ј	ѡ	Ѓ	Ѕ	Ї	Љ	©	€	«	¬		®	Ї
0xb0	°	±	І	і	ґ	μ	¶	·	ё	№	€	»	ј	Ѕ	ѕ	ї
0xc0	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
0xd0	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
0xe0	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
0xf0	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

CP1251 - Наша, родная кодировка.

	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
0x00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
0x10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0x20	␣	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
0x30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0x40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0x50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
0x60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0x70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
0x80	€	پ	,	f	„	...	†	‡	^	%	ٹ	<	Œ	چ	ژ	ڈ
0x90	گ	'	'	“	”	•	—	—	ک	™	ڑ	>	œ			ں
0xa0		,	¢	£	¤	¥	¦	§	¨	©	ھ	«	¬	SHY	®	-
0xb0	°	±	²	³	´	µ	¶	·	¸	¹	؛	»	¼	½	¾	؟
0xc0	ہ	ء	آ	أ	ؤ	إ	ئ	ا	ب	ة	ت	ث	ج	ح	خ	د
0xd0	ذ	ر	ز	س	ش	ص	ض	×	ط	ظ	ع	غ	—	ف	ق	ك
0xe0	à	ل	â	م	ن	ه	و	ç	è	é	ê	ë	ى	ي	î	ï
0xf0	=	و	=	-	ô	و	-	÷	س	ù	°	û	ü	LRM	RLM	ے

CP1256 - арабская кодировка.

	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
0x00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
0x10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0x20	␣	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
0x30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0x40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0x50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
0x60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0x70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
0x80	Ç	ü	é	â	ä	à	å	ç	ê	ë	è	ï	î	ì	Ä	Å
0x90	É	§	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	¢	£	¥	ℳ	ƒ
0xa0	á	í	ó	ú	ñ	Ñ	ª	º	¿	¬	½	¼	¡	«	»	
0xb0	⌘	⌘	⌘		⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞
0xc0	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞
0xd0	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞	⌞
0xe0	α	β	Γ	π	Σ	σ	μ	τ	Φ	Θ	Ω	δ	∞	φ	ε	η
0xf0	≡	±	≥	≤	∫	∫	÷	≈	°	•	•	√	ⁿ	²	■	NBSP

CP437 - стандартная кодировка IBM PC. Также известна как Alt Codes

Кодовая страница 437

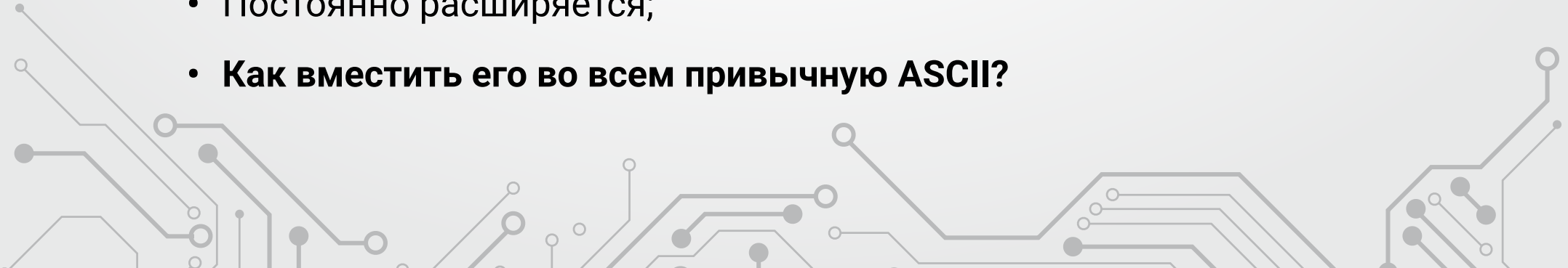
- Покрывает алфавиты нескольких языков: Английского, Немецкого и Шведского;
- Содержит **символы национальных валют** (¢, £, ¥, f, Pt);
- Позволяет рисовать **таблички псевдографикой**;
- **Очень пытается быть универсальной**, но нельзя объять необъятное.



Его Величество

Unicode

- Максимально полный стандарт;
- Содержит **155 063** символов (по состоянию на 10 сент. 2024);
- Постоянно расширяется;
- **Как вместить его во всем привычную ASCII?**

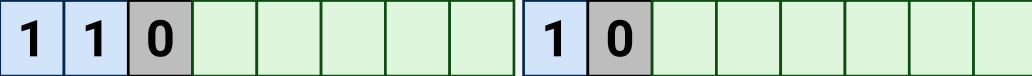


	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
0x00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
0x10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0x20	␣	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
0x30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0x40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0x50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
0x60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0x70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
0x80	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT
0x90	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT
0xa0	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT
0xb0	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT	CNT
0xc0	2-0	2-0	2-0	2-0	2-0	2-0	2-0	2-0	2-0	2-0	2-0	2-0	2-0	2-0	2-0	2-0
0xd0	2-0	2-0	2-0	2-0	2-0	2-0	2-0	2-0	2-0	2-0	2-0	2-0	2-0	2-0	2-0	2-0
0xe0	3-0	3-0	3-0	3-0	3-0	3-0	3-0	3-0	3-0	3-0	3-0	3-0	3-0	3-0	3-0	3-0
0xf0	4-0	4-0	4-0	4-0	4-0	4-0	4-0	4-0	INV	INV	INV	INV	INV	INV	INV	INV

UTF-8. Кодировка символов последовательно от 1 до 4 байт.

Кодировка UTF-8

 – Обычный ASCII-символ (7 бит, 128 символов)







- В зависимости от длины кода, используется разная кодировка.
- Всего можно закодировать **1114111** символов.

Спасибо за внимание!

 github.com/JakMobius/courses/tree/main/mipt-os-basic-2024