Представление символов, таблицы кодировок

Содержание

Представление символов в вычислительных машинах

В вычислительных машинах символы не могут храниться иначе, как в виде последовательностей бит (как и числа). Для передачи символа и его корректного отображения ему должна соответствовать уникальная последовательность нулей и единиц. Для этого были разработаны таблицы кодировок.

Количество символов, которые можно задать последовательностью бит длины n, задается простой формулой $C(n) = 2^n$. Таким образом, от нужного количества символов напрямую зависит количество используемой памяти.

Таблицы кодировок

На заре компьютерной эры на каждый символ было отведено по пять бит. Это было связано с малым количеством оперативной памяти на компьютерах тех лет. В эти 32 символа входили только управляющие символы и строчные буквы английского алфавита.

С ростом производительности компьютеров стали появляться таблицы кодировок с большим количеством символов. Первой семибитной кодировкой стала ASCII7. В нее уже вошли прописные буквы английского алфавита, арабские цифры, знаки препинания. Затем на ее базе была разработана ASCII8, в которым уже стало возможным хранение 256 символов: 128 основных и еще столько же расширенных. Первая часть таблицы осталась без изменений, а вторая может иметь различные варианты (каждый имеет свой номер). Эта часть таблицы стала заполняться символами национальных алфавитов.

Но для многих языков (например, арабского, японского, китайского) 256 символов недостаточно, поэтому развитие кодировок продолжалось, что привело к появлению UNICODE.

Кодировки стандарта ASCII

Определение:

ASCII — таблицы кодировок, в которых содержатся основные символы (английский алфавит, цифры, знаки препинания, символы национальных алфавитов(свои для каждого региона), служебные символы) и длина кода каждого символа $n = 8$ бит.

7 бит:

- **ASCII7** — первая кодировка, пригодная для работы с текстом. Помимо маленьких букв английского алфавита и служебных символов, содержит большие буквы английского языка, цифры, знаки препинания и другие символы.

Кодировки стандарта ASCII (8 бит):

- **ASCII** — первая кодировка, в которой стало возможно использовать символы национальных алфавитов.
- **КОИ8-R** — первая русская кодировка. Символы кириллицы расположены не в алфавитном порядке. Их разместили в верхнюю половину таблицы так, чтобы позиции кириллических символов соответствовали их фонетическим аналогам в английском алфавите. Это значит, что даже при потере старшего бита каждого символа, например, при проходе через устаревший семибитный модем, текст остается "читаемым".
- **CP866** — русская кодировка, использовавшаяся на компьютерах IBM в системе DOS.

- **Windows-1251** — русская кодировка, использовавшаяся в русскоязычных версиях операционной системы Windows в начале 90-х годов. Кириллические символы идут в алфавитном порядке. Содержит все символы, встречающиеся в типографике обычного текста (кроме знака ударения).

Структурные свойства таблицы

- Цифры 0-9 представляются своими двоичными значениями (например, $5 = 0101_2$), перед которыми стоит 0011_2. Таким образом, двоично-десятичные числа (BCD) превращаются в ASCII-строку с помощью простого добавления слева 0011_2 к каждому двоично-десятичному полубайту.
- Буквы A-Z верхнего и нижнего регистров различаются в своём представлении только одним битом, что упрощает преобразование регистра и проверку на диапазон. Буквы представляются своими порядковыми номерами в алфавите, записанными в двоичной системе счисления, перед которыми стоит 0100_2 (для букв верхнего регистра) или 0110_2 (для букв нижнего регистра).

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

Кодировки стандарта UNICODE

Юникод или **Уникод** (англ. *Unicode*) — это промышленный стандарт обеспечивающий цифровое представление символов всех письменностей мира, и специальных символов.

Стандарт предложен в 1991 году некоммерческой организацией «Консорциум Юникода» (англ. *Unicode Consortium, Unicode Inc.*). Применение этого стандарта позволяет закодировать очень большое число символов из разных письменностей. Стандарт состоит из двух основных разделов: **универсальный набор символов** (англ. *UCS, universal character set*) и семейство кодировок (англ. *UTF, Unicode transformation format*). **Универсальный набор символов** задаёт однозначное соответствие символов кодам — элементам кодового пространства, представляющим неотрицательные целые числа.Семейство кодировок определяет машинное представление последовательности кодов UCS.

Коды в стандарте Unicode разделены на несколько областей. Область с кодами от U+0000 до U+007F содержит символы набора ASCII с соответствующими кодами. Далее расположены области знаков различных письменностей, знаки пунктуации и технические символы. Под символы кириллицы выделены области знаков с кодами от U+0400 до U+052F, от U+2DE0 до U+2DFF, от U+A640 до U+A69F. Часть кодов зарезервирована для использования в будущем.

Кодовое пространство

Хотя формы записи UTF-8 и UTF-32 позволяют кодировать до 2^{31} $(2\ 147\ 483\ 648)$ кодовых позиций, было принято решение использовать лишь $1\ 112\ 064$ для совместимости с UTF-16. Впрочем, даже и этого на текущий момент более чем достаточно — в версии 6.0 используется чуть менее $110\ 000$ кодовых позиций ($109\ 242$ графических и 273 прочих символов).

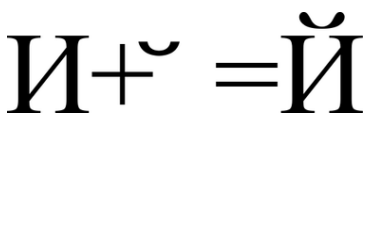
Кодовое пространство разбито на 17 плоскостей (англ. *planes*) по 2^{16} $(65\ 536)$ символов. Нулевая плоскость называется базовой, в ней расположены символы наиболее употребительных письменностей. Первая плоскость используется, в основном, для исторических письменностей, вторая — для для редко используемых иероглифов китайского письма, третья зарезервирована для архаичных китайских иероглифов. Плоскости 15 и 16 выделены для частного употребления.

Для обозначения символов Unicode используется запись вида «U+xxxx» (для кодов $0000_{16}..FFFF_{16}$) или «U+xxxxx» (для кодов $10000_{16}..FFFFF_{16}$) или «U+xxxxxx» (для кодов $100000_{16}..10FFFF_{16}$), где xxx — шестнадцатеричные цифры. Например, символ «я» (U+044F) имеет код $044F_{16} = 1103_{10}$.

Плоскости Юникода		
Плоскость	**Название**	**Диапазон символов**
Plane 0	Basic multilingual plane (BMP)	U+0000…U+FFFF
Plane 1	Supplementary multilingual plane (SMP)	U+10000…U+1FFFF
Plane 2	Supplementary ideographic plane (SIP)	U+20000…U+2FFFF
Planes 3-13	Unassigned	U+30000…U+DFFFF
Plane 14	Supplementary special-purpose plane (SSP)	U+E0000…U+EFFFF
Planes 15-16	Supplementary private use area (S PUA A/B)	U+F0000…U+10FFFF

Модифицирующие символы

Графические символы в Юникоде делятся на протяжённые и непротяжённые. Непротяжённые символы при отображении не занимают дополнительного места в строке. К примеру, к ним относятся знак ударения. Протяжённые и непротяжённые символы имеют собственные коды, но последние не могут встречаться самостоятельно. Протяжённые



символы называются базовыми (англ. *base characters*), а непротяженные — модифицирующими (англ. *combining characters*). Например символ «Й» (U+0419) может быть представлен в виде базового символа «И» (U+0418) и модифицирующего символа «◌̆» (U+0306).

Способы представления

Юникод имеет несколько форм представления (англ. *Unicode Transformation Format, UTF*): UTF-8, UTF-16 (UTF-16BE, UTF-16LE) и UTF-32 (UTF-32BE, UTF-32LE). Была разработана также форма представления UTF-7 для передачи по семибитным каналам, но из-за несовместимости с ASCII она не получила распространения и не включена в стандарт.

UTF-8

UTF-8 — представление Юникода, обеспечивающее наилучшую совместимость со старыми системами, использовавшими 8-битные символы. Текст, состоящий только из символов с номером меньше 128, при записи в UTF-8 превращается в обычный текст ASCII. И наоборот, в тексте UTF-8 любой байт со значением меньше 128 изображает символ ASCII с тем же кодом. Остальные символы Юникода изображаются последовательностями длиной от двух до шести байт (на деле, только до четырех байт, поскольку в Юникоде нет символов с кодом больше $10FFFF_{16}$, и вводить их в будущем не планируется), в которых первый байт всегда имеет вид $11xxxxxx$, а остальные — $10xxxxxx$.

Символы UTF-8 получаются из Unicode следующим образом:

Unicode	UTF-8	Представленные символы
`0x00000000` — `0x0000007F`	`0xxxxxxx`	ASCII, в том числе английский алфавит, простейшие знаки препинания и арабские цифры
`0x00000080` — `0x000007FF`	`110xxxxx 10xxxxxx`	кириллица, расширенная латиница, арабский алфавит, армянский алфавит, греческий алфавит, еврейский алфавит и коптский алфавит; сирийское письмо, тана, нко; Международный фонетический алфавит; некоторые знаки препинания
`0x00000800` — `0x0000FFFF`	`1110xxxx 10xxxxxx 10xxxxxx`	все другие современные формы письменности, в том числе грузинский алфавит, индийское, китайское, корейское и японское письмо; сложные знаки препинания; математические и другие специальные символы
`0x00010000` — `0x001FFFFF`	`11110xxx 10xxxxxx 10xxxxxx 10xxxxxx`	музыкальные символы, редкие китайские иероглифы, вымершие формы письменности
	`111111xx`	служебные символы c, d, e, f

Несмотря на то, что UTF-8 позволяет указать один и тот же символ несколькими способами, только наиболее короткий из них правильный. Остальные формы, называемые overlong sequence, отвергаются по соображениям безопасности.

Принцип кодирования

Правила записи кода одного символа в UTF-8

1. Если размер символа в кодировке UTF-8 = 1 байт

    Код имеет вид (0aaa aaaa), где «0» — просто ноль, остальные биты «a» — это код символа в кодировке ASCII;

2. Если размер символа в кодировке в UTF-8 > 1 байт (то есть от 2 до 6):

    2.1 Первый байт содержит количество байт символа, закодированное в **единичной** системе счисления;

```
2 – 11
3 – 111
4 – 1111
5 – 1111 1
6 – 1111 11
```

    2.2 «0» — бит терминатор, означающий завершение кода размера

    2.3 далее идут значащие байты кода, которые имеют вид (10xx xxxx), где «10» — биты признака продолжения, а «x» — значащие биты.

В общем случае варианты представления **одного символа** в кодировке UTF-8 выглядят так:

```
(1 байт)  0aaa aaaa
(2 байта) 110x xxxx 10xx xxxx
(3 байта) 1110 xxxx 10xx xxxx 10xx xxxx
(4 байта) 1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx
(5 байт)  1111 10xx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx
(6 байт)  1111 110x 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx
```

Определение длины кода в UTF-8

Количество байт UTF-8	Количество значащих бит
1	7
2	11
3	16
4	21
5	26
6	31

В общем случае количество значащих бит C, кодируемых n байтами UTF-8, определяется по формуле:

$C = 7$ при $n = 1$

$C = n \cdot 5 + 1$ при $n > 1$

UTF-16

UTF-16 — один из способов кодирования **символов** (англ. *code point*) из Unicode в виде последовательности 16-битных **слов** (англ. *code unit*). Данная кодировка позволяет записывать символы Юникода в диапазонах U+0000..U+D7FF и U+E000..U+10FFFF (общим количеством $1\ 112\ 064$), причем 2-байтные символы представляются как есть, а более длинные — с помощью суррогатных пар (англ. *surrogate pair*), для которых и вырезан диапазон $D800_{16}..DFFF_{16}$.

В UTF-16 символы кодируются двухбайтовыми словами с использованием всех возможных диапазонов значений (от 0000_{16} до $FFFF_{16}$). При этом можно кодировать символы Unicode в диапазонах $0000_{16}..D7FF_{16}$ и $E000_{16}..10FFFF_{16}$. Исключенный отсюда диапазон $D800_{16}..DFFF_{16}$ используется как раз для кодирования так называемых суррогатных пар — символов, которые кодируются двумя 16-битными словами. Символы Unicode до $FFFF_{16}$ включительно (исключая диапазон для суррогатов) записываются как есть 16-битным словом. Символы же в диапазоне $10000_{16}..10FFFF_{16}$ (больше 16 бит) уже кодируются парой 16-битных слов. Для этого их код арифметически сдвигается до нуля (из него вычитается минимальное число 10000_{16}). В результате получится значение от нуля до $FFFFF_{16}$, которое занимает до 20 бит. Старшие 10 бит этого значения идут в лидирующее (первое) слово, а младшие 10 бит — в последующее (второе). При этом в обоих словах старшие 6 бит используются для обозначения суррогата. Биты с 11 по 15 имеют значения 11011_2, а 10-й бит содержит 0 у лидирующего слова и 1 — у последующего. В связи с этим можно легко определить к чему относится каждое слово.

UTF-16LE и UTF-16BE

Один символ кодировки UTF-16 представлен последовательностью двух байт или двух пар байт. Который из двух байт в словах идёт впереди, старший или младший, зависит от порядка байт. Подробнее об этом будет сказано ниже.

UTF-32

UTF-32 — один из способов кодирования символов из Юникод, использующий для кодирования любого символа ровно 32 бита. Остальные кодировки, UTF-8 и UTF-16, используют для представления символов переменное число байт. Символ UTF-32 является прямым представлением его кодовой позиции (англ. *code point*).

Главное преимущество UTF-32 перед кодировками переменной длины заключается в том, что символы Юникод непосредственно индексируемы. Получение n-ой кодовой позиции является операцией, занимающей одинаковое время. Напротив, коды с переменной длиной требует последовательного доступа к n-ой кодовой позиции. Это делает замену символов в строках UTF-32 простой, для этого используется целое число в качестве индекса, как обычно делается для строк ASCII.

Главный недостаток UTF-32 — это неэффективное использование пространства, так как для хранения символа используется четыре байта. Символы, лежащие за пределами нулевой (базовой) плоскости кодового пространства редко используются в большинстве текстов. Поэтому удвоение, в сравнении с UTF-16, занимаемого строками в UTF-32 пространства не оправдано.

Хотя использование неменяющегося числа байт на символ удобно, но не настолько, как кажется. Операция усечения строк реализуется легче в сравнении с UTF-8 и UTF-16. Но это не делает более быстрым нахождение конкретного смещения в строке, так как смещение может вычисляться и для кодировок фиксированного размера. Это не облегчает вычисление отображаемой ширины строки, за исключением ограниченного числа случаев, так как даже символ «фиксированной ширины» может быть получен комбинированием обычного символа с модифицирующим, который не имеет ширины. Например, буква «й» может быть получена из буквы «и» и диакритического знака «крючок над буквой». Сочетание таких знаков означает, что текстовые редакторы не могут рассматривать 32-битный код как единицу редактирования. Редакторы, которые ограничиваются работой с языками с письмом слева направо и составными символами (англ. *Precomposed character*), могут использовать символы фиксированного размера. Но такие редакторы вряд ли поддержат символы, лежащие за пределами нулевой (базовой) плоскости кодового пространства и вряд ли смогут работать одинаково хорошо с символами UTF-16.

Порядок байт

В современной вычислительной технике и цифровых системах связи информация обычно представлена в виде последовательности байт. В том случае, если число не может быть представлено одним байтом, имеет значение в каком порядке байты записываются в памяти компьютера или передаются по линиям связи. Часто выбор порядка записи байт произволен и определяется только соглашениями.

В общем случае, для представления числа M, большего 255 (здесь $255 = 2^8 - 1$ — максимальное целое число, записываемое одним байтом), приходится использовать несколько байт. При этом число M записывается в позиционной системе счисления по основанию 256:

$$M = \sum_{i=0}^{n} A_i \cdot 256^i = A_0 \cdot 256^0 + A_1 \cdot 256^1 + A_2 \cdot 256^2 + \cdots + A_n \cdot 256^n.$$

Набор целых чисел $A_0, \ldots, A_n$, каждое из которых лежит в интервале от 0 до 255, является последовательностью байт, составляющих M. При этом A_0 называется младшим байтом, а A_n — старшим байтом числа M.

Варианты записи

Порядок от старшего к младшему

Порядок 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 байт от старшего к младшему часто называют сетевым порядком байт (англ. *network byte order*). Этот порядок байт используется процессорами IBM 360/370/390, Motorola 68000, SPARC (отсюда третье название — порядок байт Motorola, Motorola byte order).

В этом же виде (используя представление в десятичной системе счисления) записываются числа индийско-арабскими цифрами в письменностях с порядком знаков слева направо (латиница, кириллица). Для письменностей с обратным порядком (арабская) та же запись числа воспринимается как «от младшего к старшему».

Порядок байт от старшего к младшему применяется во многих форматах файлов — например, PNG, FLV, EBML.

Порядок от младшего к старшему

Порядок 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).

В противоположность порядку big-endian, соглашение little-endian поддерживают меньше кросс-платформенных протоколов и форматов данных; существенные исключения: USB, конфигурация PCI, таблица разделов GUID, рекомендации FidoNet.

Переключаемый порядок

Многие процессоры могут работать и в порядке от младшего к старшему, и в обратном, например, ARM, PowerPC (но не PowerPC 970), DEC Alpha, MIPS, PA-RISC и IA-64. Обычно порядок байт выбирается программно во время инициализации операционной системы, но может быть выбран и аппаратно перемычками на материнской плате. В этом случае правильнее говорить о порядке байт операционной системы. Переключаемый порядок байт иногда называют англ. *bi-endian*.

Смешанный порядок

Смешанный порядок байт (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.

Классический пример middle-endian — представление 4-байтных целых чисел на 16-битных процессорах семейства PDP-11 (известен как PDP-endian). Для представления двухбайтных значений (слов) использовался порядок little-endian, но 4-хбайтное двойное слово записывалось от старшего слова к младшему.

В процессорах VAX и ARM используется смешанное представление для длинных вещественных чисел.

Различия

Существенным достоинством little-endian по сравнению с big-endian порядком записи считается возможность «неявной типизации» целых чисел при чтении меньшего объёма байт (при условии, что читаемое число помещается в диапазон). Так, если в ячейке памяти содержится число 00000022_{16}, то прочитав его как int16 (два байта) мы получим число 0022_{16}, прочитав один байт — число 22_{16}. Однако, это же может считаться и недостатком, потому что провоцирует ошибки потери данных.

Обратно, считается что у little-endian, по сравнению с big-endian есть «неочевидность» значения байт памяти при отладке (последовательность байт (A1, B2, C3, D4) на самом деле значит $D4C3B2A1_{16}$, для big-endian эта последовательность (A1, B2, C3, D4) читалась бы «естественным» для арабской записи чисел образом: $A1B2C3D4_{16}$). Наименее удобным в работе считается middle-endian формат записи; он сохранился только на старых платформах.

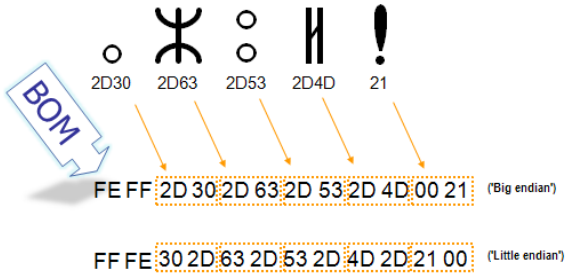
Для записи длинных чисел (чисел, длина которых существенно превышает разрядность машины) обычно предпочтительнее порядок слов в числе little-endian (поскольку арифметические операции над длинными числами производятся от младших разрядов к старшим). Порядок байт в слове — обычный для данной архитектуры.

Маркер последовательности байт

Для определения формата представления Юникода в начало текстового файла записывается сигнатура — символ U+FEFF (неразрывный пробел с нулевой шириной), также именуемый маркером последовательности байт (англ. *byte order mark (BOM)*). Это позволяет различать UTF-16LE и UTF-16BE, поскольку символа U+FFFE не существует.

Представление BOM в кодировках

Кодирование	Представление (Шестнадцатеричное)
UTF-8	`EF BB BF`
UTF-16 (BE)	`FE FF`
UTF-16 (LE)	`FF FE`
UTF-32 (BE)	`00 00 FE FF`
UTF-32 (LE)	`FF FE 00 00`



В кодировке UTF-8, наличие BOM не является существенным, поскольку, нет альтернативной последовательности байт. Когда BOM используется на страницах или редакторах для контента закодированного в UTF-8, иногда он может представить пробелы или короткие последовательности символов, имеющие странный вид (такие как ï»¿). Именно поэтому, при наличии выбора, для совместимости, как правило, лучше упустить BOM в UTF-8 контенте.Однако BOM могут еще встречаться в тексте закодированном в UTF-8, как побочный продукт перекодирования или потому, что он был добавлен редактором. В этом случае BOM часто называют подписью UTF-8.

Когда символ закодирован в UTF-16, его 2 или 4 байта можно упорядочить двумя разными способами (little-endian или big-endian). Изображение справа показывает это. Byte order mark указывает, какой порядок используется, так что приложения могут немедленно расшифровать контент. UTF-16 контент должен всегда начинатся с BOM.

BOM также используется для текста обозначенного как UTF-32. Аналогично UTF-16 существует два варианта четырёхбайтной кодировки — UTF-32BE и UTF-32LE. К сожалению, этот способ не позволяет надёжно различать UTF-16LE и UTF-32LE, поскольку символ U+0000 допускается Юникодом

Проблемы Юникода

В Юникоде английское «a» и польское «a» — один и тот же символ. Точно так же одним символом (но отличающимся от «a» латинского) считаются русское «a» и сербское «a». Такой принцип кодирования не универсален; по-видимому, решения «на все случаи жизни» вообще не может существовать.

Примеры

Если записать строку 'hello мир' в файл exampleBOM, а затем сделать его hex-дамп, то можно убедиться в том, что разные символы кодируются разным количеством байт. Например, английские буквы,пробел, знаки препинания и пр. кодируются одним байтом, а русские буквы - двумя

Код на python

```
#!/usr/bin/env python
#coding:utf-8
import codecs
f = open('exampleBOM','w')
b = u'hello мир'
f.write(codecs.BOM_UTF8)
f.write(b.encode('utf-8'))
f.close()
```

hex-дамп файла exampleBOM

Символ	BOM			h	e	l	l	o	Пробел	м		и		р	
Код в UNICODE	EF	BB	BF	68	65	6C	6C	6F	20	D0	BC	D0	B8	D1	80
Код в UTF-8	11101111	10111011	10111111	01101000	01100101	01101100	01101100	01101111	00100000	11010000	10111100	11010000	10111000	11010001	10000000

См. также

- Представление целых чисел: прямой код, код со сдвигом, дополнительный код
- Представление вещественных чисел

Источники информации

- Wikipedia — таблица ASCII (http://ru.wikipedia.org/wiki/ASCII)
- Wikipedia — стандарт UNICODE (http://ru.wikipedia.org/wiki/%D0%AE%D0%BD%D0%B8%D0%BA%D0%BE%D0%B4)
- Wikipedia — Byte order mark (http://ru.wikipedia.org/wiki/Byte_order_mark)
- Wikipedia — Порядок байтов (http://ru.wikipedia.org/wiki/Порядок_байтов)
- Wikipedia — Юникод (http://ru.wikipedia.org/wiki/%D0%AE%D0%BD%D0%B8%D0%BA%D0%BE%D0%B4)
- Wikipedia — Windows-1251 (http://ru.wikipedia.org/wiki/CP1251)
- Wikipedia — UTF-8 (http://ru.wikipedia.org/wiki/UTF-8)
- Wikipedia — UTF-16 (http://ru.wikipedia.org/wiki/UTF-16)
- Wikipedia — UTF-32 (http://ru.wikipedia.org/wiki/UTF-32)

Источник — «http://neerc.ifmo.ru/wiki/index.php?title=Представление_символов,_таблицы_кодировок&oldid=85381»
