Character Codes for Modern Computers

This lecture covers the standard ways in which characters are stored in modern computers. There are five main classes of characters.

1. Alphabetic characters: upper case and lower case.
2. Decimal digits.
3. Punctuation.
4. Control characters, which are not usually printed.
5. All other characters.

There are three standard methods for representing characters.

1.	EBCDIC	**E**xtended **B**inary **C**oded **D**ecimal **I**nterchange **C**ode
2.	ASCII	**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
3.	Unicode	A modern extension of ASCII.

Each encodes a character in eight bits, represented as two hexadecimal digits.

EBCDIC: Origins and Rationale

The EBCDIC (pronounced "IPSY–dick") coding system was developed by IBM as an extension for its BCD (Binary Coded Decimal) system.

EBCDIC uses 8 bits to encode each character, for 256 distinct characters.

The BCD system used 6 bits to encode a character; only 64 distinct characters.

Some of the characters represented in BCD were:

1. The 26 upper case alphabetic characters **`"A" – "Z"`**.
2. The ten digits **`"0" – "9"`**.
3. The space character `" "`.
4. The symbols used in arithmetic **`"+"`, `"–"`, `"*"`, `"/"`, `"="`, `"&"`**
5. Punctuation marks **`","`, `"."`, `"("`, `")"`, `":"`**

Note that there are no lower case letters. I have listed 48 of the BCD characters. There is room for only 16 more.

EBCDIC: Origins and Rationale (Part 2)

The International Business Machines Corporation, called “IBM” by everybody, developed the EBCDIC standard at the same time that the ASCII standard was being developed.

The EBCDIC standard was developed for use in the IBM System/360, a revolutionary computing system introduced in 1964.

IBM supported the ASCII standard strongly. This leads to a simple question: “Why did IBM not use ASCII?”

Here is a little–known fact. While the computers in the IBM System/360 line were designed to use the EBCDIC standard, each on had an “ASCII switch” that would cause it to use ASCII.

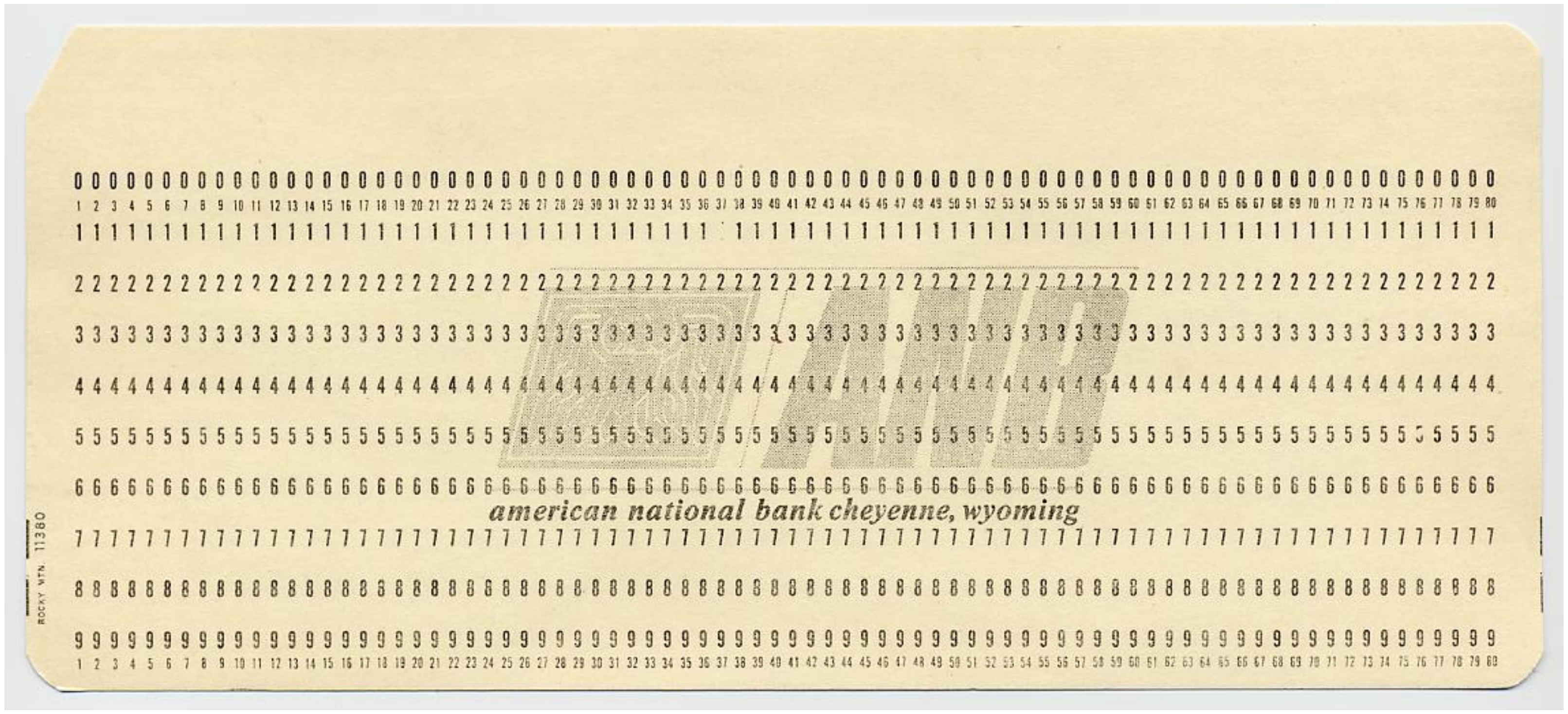
Few system administrators knew of this “ASCII switch” and fewer still used it. When the System/360 evolved to the System/370, the switch was dropped.

IBM used EBCDIC because it was compatible with the existing card codes.

Punched Cards

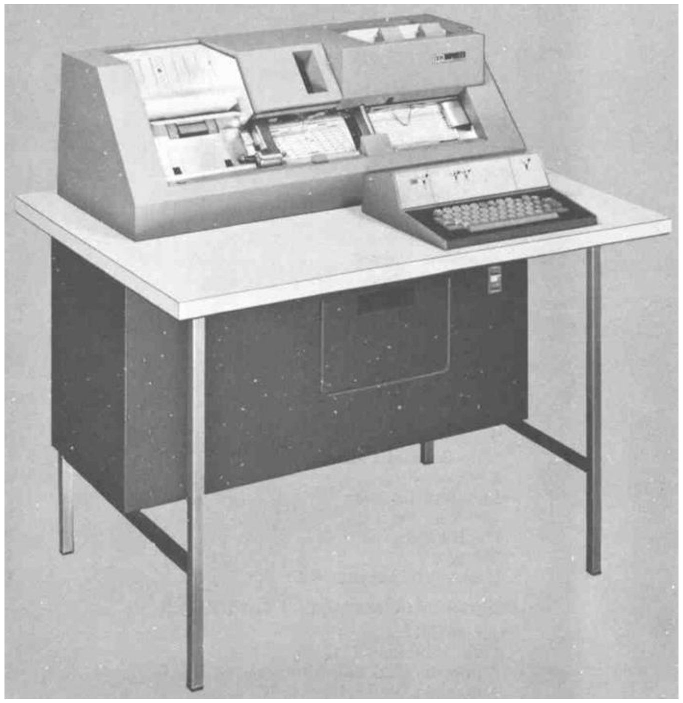
When the IBM 360 was first designed, most data input was from 80–column punched cards. IBM experimented with other formats, but they never caught on.

Here is the picture of a typical 80–column punched card.
It has 12 rows, ten rows labeled 0 – 9; rows 12 and 11 are at the top.



The IBM 029 Key Punch

Here is a picture of the device used to produce punched data cards.



The card feed was at the right.

The card moved right–to–left as it was punched.

The punched cards were stored in a tray at the top left.

IBM 029 Punch Card Codes

Here is a card punched with each of the 64 characters available under this format. Note the lack of lower case letters; they were not used in programming languages of the time.

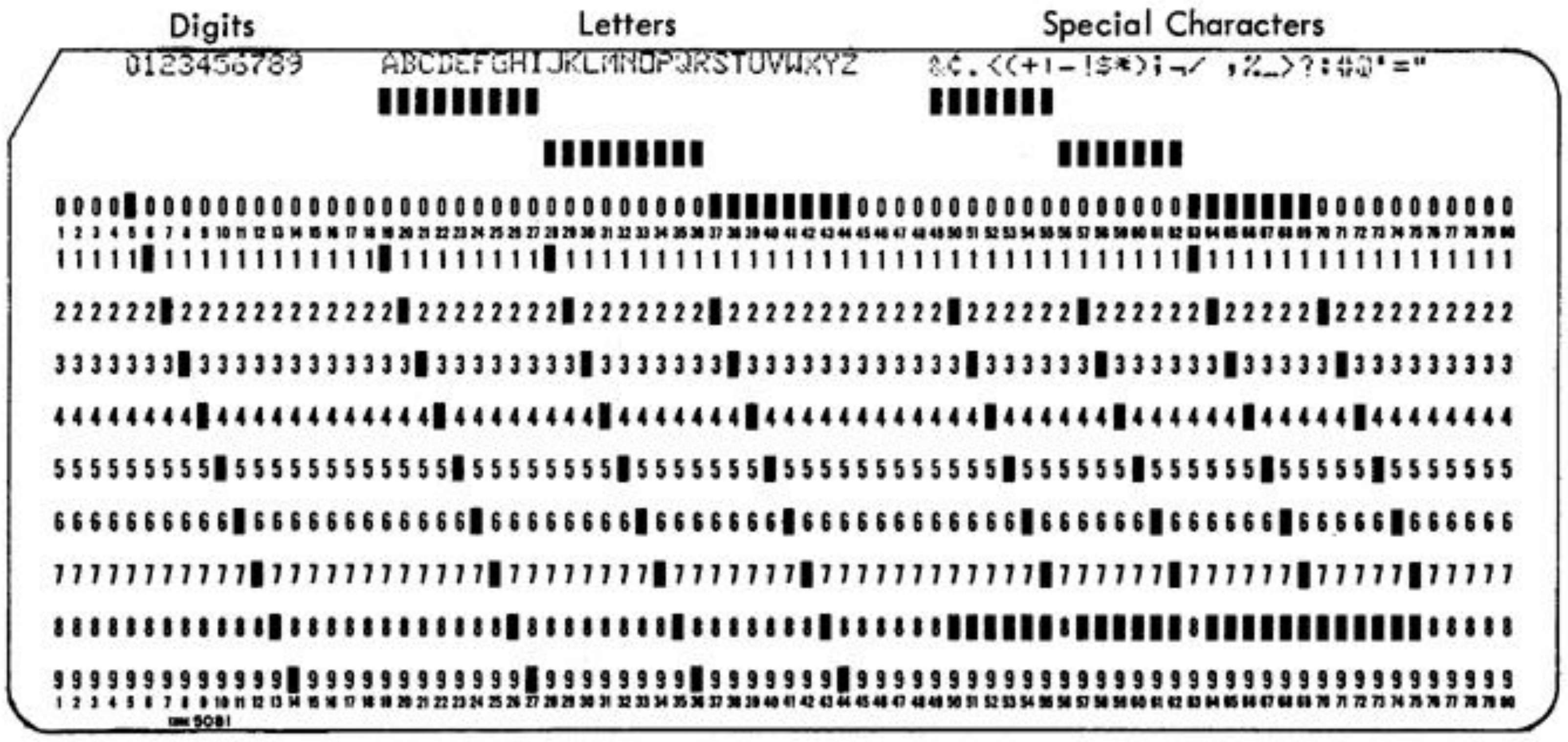


Figure 4.   Card Codes and Graphics for 64-Character Set

More on the Punch Card Codes

Digits were encoded by a single punch in the appropriate row.

A single punch in row 2 encoded a “2”, etc.

Other characters were encoded by two punches in a column.

The letter “A” was encoded as 12–1; a punch in row 12 (the top row), and a punch in row 1.

The letter “K” was encoded as 11–1; a punch in row 11 (next to the top row), and a punch in row 1.

The letter “S” was encoded as 0–2; a punch in row 0 and a punch in row 2.

Back to EBCDIC

Consider the IBM 029 punch codes and compare them to the EBCDIC.

Character	EBCDIC	Punch Card Codes
0 through 9	F0 through F9	0 through 9
A through I	C1 through C9	12–1 through 12–9
J through R	D1 through D9	11–1 through 11–9
S through Z	E2 through E9	0–2 through 0–9

This table explains the design of the EBCDIC system.

1. IBM chose this design for ease in processing input from existing devices, such as the IBM 029 key punch.
2. The gaps in the EBCDIC system: no character from the 64 character set has a non–decimal digit as its second digit.

   Cards did not have rows marked A, B, C, D, E, or F.

Control Characters

In any character set, some codes represent characters and some codes represent control information used to indicate how the data are to be processed.

In EBCDIC, the first 64 codes (with hexadecimal values 0x00 – 0x3F) represent control characters. Here are a few of the codes used for control characters.

Value	**Name**	**Meaning**
0x01	SOH	Start of heading section of a message
0x02	STX	Start of text section of a message
0x03	ETX	End of text section of a message
0x05	HT	Horizontal tab (standard tab on a keyboard)
0x0B	VT	Vertical tab
0x0C	FF	Form feed (commonly moves to another page)
0x0D	CR	Carriage return (moves back to column 0 of the display)
0x25	LF	Line feed (moves directly down to the next line)

Printable EBCDIC Characters

Here are some of the character codes for printable EBCDIC characters.
The row ID contains the first digit of the code, the column ID the second.

Code	0	1	2	3	4	5	6	7	8	9
8		a	b	c	d	e	f	g	h	i
9		j	k	l	m	n	o	p	q	r
A		~	s	t	u	v	w	x	y	z
B										
C	{	A	B	C	D	E	F	G	H	I
D	}	J	K	L	M	N	O	P	Q	R
E	\		S	T	U	V	W	X	Y	Z
F	0	1	2	3	4	5	6	7	8	9

Here, we note that 0xF0 is the code for the digit ‘0’.

Note that there are a lot of gaps in the code. There is no printable character with the code 0xCA.

The ASCII Printable Character Set

ASCII has its own set of control characters, with meanings similar to those used in EBCDIC. Here are the ASCII codes for printable characters.

There are 128 code values in ASCII, ranging from 0x00 – 0x7F.
The value 0x20 is the ASCII code for the space character: “ ”.
The value 0x7F is the ASCII code for the delete character, called “DEL”.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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	{	|	}	~	

Properties of ASCII

ASCII has a number of interesting features that make it appealing to a programmer. Suppose we are examining a value stored in a variable.

If the value falls in the range 0x41 – 0x5A, the value represents an upper case character.

If the value falls in the range 0x71 – 0x7A, the value represents a lower case character.

For each alphabetic character, the code for the upper case and the code for the lower case are strongly related. Only one bit is reset.

Look at the codes for the letter A. We give these in binary.

A	**0100 0001**
a	**0110 0001**

We shall later develop a formula to convert between upper case and lower case.

Unicode as an Extension of ASCII

The ASCII code set and the EBCDIC code set are each sufficient for expressing any idea, as long as it can be expressed in standard Latin characters (the character set used to write in English).

This is not an issue when writing programs, as all programming languages can be expressed in something that looks like English.

Suppose your company wants to market an application in a country (such as Korea, Japan, China, Egypt, or Saudi Arabia) in which English is not the main language. How do you design your GUI (Graphical User Interface) for the screen displays?

One option is to require that everybody learn English, which is almost a de facto requirement anyway.

Suppose that you want to market an application to be used in a small shop, such as a corner market or cobbler shop. Should grandpa learn English?

A better way is to develop a method to represent non–Latin characters.

Code Pages and Unicode

An early modification was to develop what were called “code pages”.

This works for alphabetic languages, such as Arabic and Greek, in which a relatively small alphabet is used. One just replaces the Latin alphabet.

ASCII could be modified for Arabic just by redefining each of the code values 0x41 – 0x5A and 0x61 – 0x7A to stand for an Arabic character.

The main problem with each of ASCII and EBCDIC is the small number of distinct characters that can be represented.

Standard ASCII can represent only 128 distinct characters.

Extended ASCII can represent only 256 distinct characters.

EBCDIC can represent only 256 distinct characters.

Unicode, seen as a 16–bit encoding method, can support 65,536 distinct characters. There seems to be a 32–bit version of Unicode.

Some Unicode Examples

Here are some examples of character sets supported by the Unicode standard. These are taken from the web site http://www.unicode.org/charts/.

The Latin alphabet (used in English)

Greek

Cyrillic (used in the Russian language)

Egyptian hieroglyphs

Arabic

Hebrew

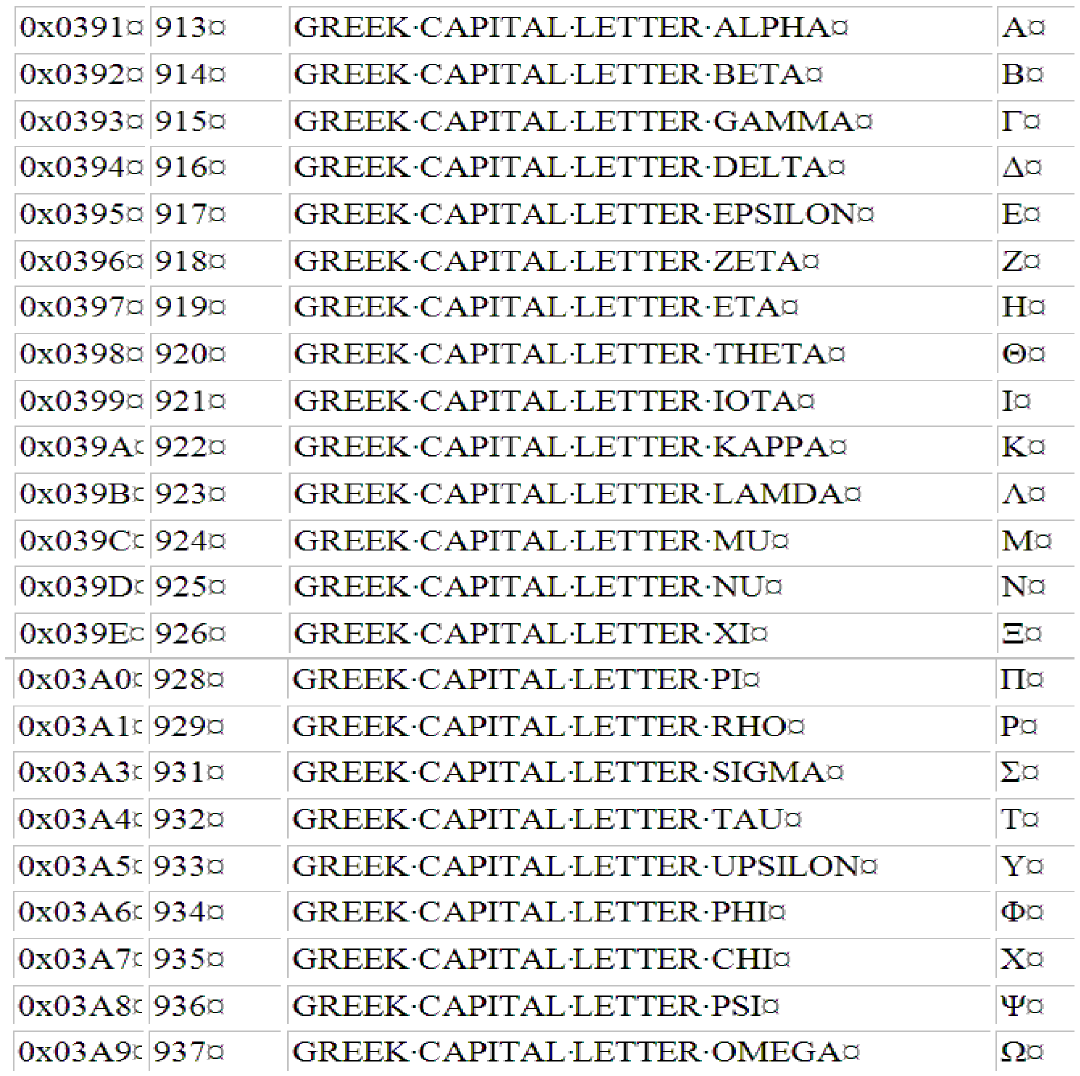
Cuneiform (old Egyptian) and Runic (Norse characters)

Lycian and Lydian (kingdoms in Anatolia during the 4th century BC)

Cherokee (an alphabet developed in the early 19th century)

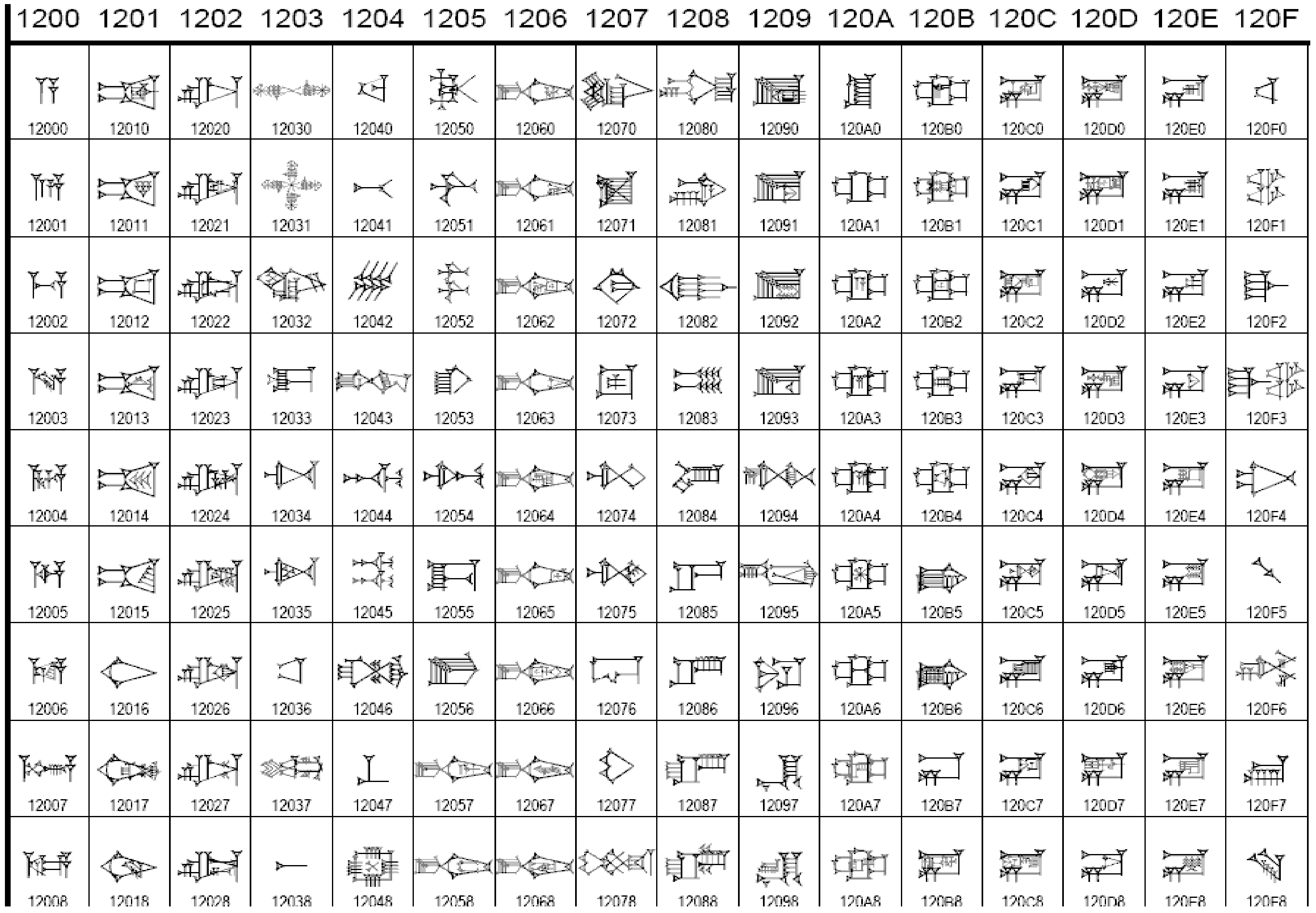
Phoenician, Parthian, Etruscan, and Old Turkic

Unicode Representation of Some Greek Characters



0x0391¤	913¤	GREEK·CAPITAL·LETTER·ALPHA¤	Α¤
0x0392¤	914¤	GREEK·CAPITAL·LETTER·BETA¤	Β¤
0x0393¤	915¤	GREEK·CAPITAL·LETTER·GAMMA¤	Γ¤
0x0394¤	916¤	GREEK·CAPITAL·LETTER·DELTA¤	Δ¤
0x0395¤	917¤	GREEK·CAPITAL·LETTER·EPSILON¤	Ε¤
0x0396¤	918¤	GREEK·CAPITAL·LETTER·ZETA¤	Ζ¤
0x0397¤	919¤	GREEK·CAPITAL·LETTER·ETA¤	Η¤
0x0398¤	920¤	GREEK·CAPITAL·LETTER·THETA¤	Θ¤
0x0399¤	921¤	GREEK·CAPITAL·LETTER·IOTA¤	Ι¤
0x039A	922¤	GREEK·CAPITAL·LETTER·KAPPA¤	Κ¤
0x039B	923¤	GREEK·CAPITAL·LETTER·LAMDA¤	Λ¤
0x039C	924¤	GREEK·CAPITAL·LETTER·MU¤	Μ¤
0x039D	925¤	GREEK·CAPITAL·LETTER·NU¤	Ν¤
0x039E	926¤	GREEK·CAPITAL·LETTER·XI¤	Ξ¤
0x03A0	928¤	GREEK·CAPITAL·LETTER·PI¤	Π¤
0x03A1	929¤	GREEK·CAPITAL·LETTER·RHO¤	Ρ¤
0x03A3	931¤	GREEK·CAPITAL·LETTER·SIGMA¤	Σ¤
0x03A4	932¤	GREEK·CAPITAL·LETTER·TAU¤	Τ¤
0x03A5	933¤	GREEK·CAPITAL·LETTER·UPSILON¤	Υ¤
0x03A6	934¤	GREEK·CAPITAL·LETTER·PHI¤	Φ¤
0x03A7	935¤	GREEK·CAPITAL·LETTER·CHI¤	Χ¤
0x03A8	936¤	GREEK·CAPITAL·LETTER·PSI¤	Ψ¤
0x03A9	937¤	GREEK·CAPITAL·LETTER·OMEGA¤	Ω¤

How About Cuneiform?



1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	120A	120B	120C	120D	120E	120F
12000	12010	12020	12030	12040	12050	12060	12070	12080	12090	120A0	120B0	120C0	120D0	120E0	120F0
12001	12011	12021	12031	12041	12051	12061	12071	12081	12091	120A1	120B1	120C1	120D1	120E1	120F1
12002	12012	12022	12032	12042	12052	12062	12072	12082	12092	120A2	120B2	120C2	120D2	120E2	120F2
12003	12013	12023	12033	12043	12053	12063	12073	12083	12093	120A3	120B3	120C3	120D3	120E3	120F3
12004	12014	12024	12034	12044	12054	12064	12074	12084	12094	120A4	120B4	120C4	120D4	120E4	120F4
12005	12015	12025	12035	12045	12055	12065	12075	12085	12095	120A5	120B5	120C5	120D5	120E5	120F5
12006	12016	12026	12036	12046	12056	12066	12076	12086	12096	120A6	120B6	120C6	120D6	120E6	120F6
12007	12017	12027	12037	12047	12057	12067	12077	12087	12097	120A7	120B7	120C7	120D7	120E7	120F7
12008	12018	12028	12038	12048	12058	12068	12078	12088	12098	120A8	120B8	120C8	120D8	120E8	120F8

A Problem with Unicode

The global Internet will use Unicode to represent the URL (Uniform Resource Locator). The URL for Columbus State University is http://www.columbusstate.edu/

Here is an example taken from a security textbook. The question is as follows: Which of these two URLs references the PayPal service.

www.paypal.com

www.paypal.com

Here is the answer. We look at the word "paypal" and focus on the 16–bit Unicode representation of each of the words.

The first is the correct link. Its encoding is:

`0x0070 0x0061 0x0079 0x0070 0x0061 0x006C`

The second encoding is

`0x0070 0x0430 0x0079 0x0070 0x0061 0x006C`

The second letter is the Cyrillic lower case "a".