ASCII and EBCDIC Compared

Updated November 2004

What is ASCII?

ASCII is the American Standard Code for Information Interchange, also known as ANSI X3.4. There are many variants of this standard, typically to allow different code pages for language encoding, but they all basically follow the same format. ASCII is quite elegant in the way it represents characters, and it is very easy to write code to manipulate upper/lowercase and check for valid data ranges.

ASCII is essentially a 7-bit code which allows the 8th most significant bit (MSB) to be used for error checking, however most modern computer systems tend to use ASCII values of 128 and above for extended character sets.

For a detailed **ASCII** table click here. ASCII table

spc	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{	¦	}	~	del

ASCII table summary

What is EBCDIC? top

EBCDIC (Extended Binary Coded Decimal Interchange Code) is a character encoding set used by IBM mainframes. Unlike virtually every computer system in the world which uses a variant of ASCII, IBM mainframes and midrange systems such as the AS/400 tend to use a wholly incompatible character set primarily designed for ease of use on punched cards. (For an excellent page on punched cards, see **Doug Jones's Punched Card Codes**).

EBCDIC uses the full 8 bits available to it, so parity checking cannot be used on an 8 bit system. Also, EBCDIC has a wider range of control characters than ASCII.

The character encoding is based on Binary Coded Decimal (BCD), so the

contiguous characters in the alphanumeric range are formed up in blocks of up to 10 from 0000 binary to 1001 binary. Non alphanumeric characters are almost all outside the BCD range.

There are four main blocks in the EBCDIC code page: 0000 0000 to 0011 1111 is reserved for control characters; 0100 0000 to 0111 1111 are for punctuation; 1000 0000 to 1011 1111 for lowercase characters and 1100 0000 to 1111 1111 for uppercase characters and numbers.

There are several different dialects of EBCDIC, and these tend to differ in the punctuation coding. The following table uses some fairly common EBCDIC codings.

For a detailed EBCDIC table click **here**. EBCDIC table

```
spc                          ¢ . < ( +
                           ! $ * ) ; |
  &                          , % _ > ?
  - /                      : # @ ' = "
    a b c d e f g h i
    j k l m n o p q r
      s t u v w x y z

    A B C D E F G H I
    J K L M N O P Q R
      S T U V W X Y Z
  0 1 2 3 4 5 6 7 8 9
```

EBCDIC table summary

Why EBCDIC is better than ASCII top

EBCDIC is easier to use on punched cards and included the "cent sign" (¢) character that ASCII does not.

Why ASCII is better than EBCDIC top

Reason 1: Writing code

EBCDIC is a mess. The lack of contiguous character blocks make coding a real pain. Consider a simple program to parse an ASCII data stream and turn it into printable data, ditching anything that isn't printable:

```
DO
 READ DataStreamByte FROM DataStream
 IF (DataStreamByte > 31) AND (DataStreamByte < 127)
   THEN DisplayCharacter(DataStreamByte)
 END IF
```

```
    ' ASCII characters in the range 32 to 126 are printable
 IF DataSteamByte = 13 THEN DisplayNewLine
    ' ASCII 13 is the return character, so display a new line
LOOP UNTIL END_OF_FILE(DataSteam)
```

Method A

In EBCDIC the code might look something like this:

```
DO
 READ DataStreamByte FROM DataStream
 IF (DataStreamByte = 64) OR
       ' Check for space character
    ((DataStreamByte >= 74) AND (DataStreamByte <= 80)) OR
    ((DataStreamByte >= 90) AND (DataStreamByte <= 97)) OR
    ((DataStreamByte >= 107) AND (DataStreamByte <= 111)) OR
    ((DataStreamByte >= 122) AND (DataStreamByte <= 127)) OR
       ' The previous 4 lines check for punctuation
    ((DataStreamByte >= 129) AND (DataStreamByte <= 137)) OR
    ((DataStreamByte >= 145) AND (DataStreamByte <= 153)) OR
    ((DataStreamByte >= 162) AND (DataStreamByte <= 169)) OR
       ' The previous 3 lines check for lowercase
    ((DataStreamByte >= 193) AND (DataStreamByte <= 201)) OR
    ((DataStreamByte >= 209) AND (DataStreamByte <= 217)) OR
    ((DataStreamByte >= 226) AND (DataStreamByte <= 223)) OR
       ' The previous 3 lines check for uppercase
    ((DataStreamByte >= 240) AND (DataStreamByte <= 249))
THEN
       ' Check for numbers
    DisplayCharacter(DataStreamByte)
  END IF
  IF DataSteamByte = 21 THEN DisplayNewLine
LOOP UNTIL END_OF_FILE(DataSteam)
```

Elegant, huh?

Method B

Programmers wishing to take a more Zen-like approach may wish to consider the deeper boolean issues and create 4 by 16 array to act as a mask thus:

$$\begin{pmatrix} 0111111111000000 \\ 0111111111000000 \\ 0011111111000000 \\ 1111111111000000 \end{pmatrix}$$

In theory you should be able to manipulate the array rows with NOT or set them to zero depending which block of the EBCDIC table you are using. In practice this *almost* works except the coding is quite lengthy and different dialects of EBCDIC often put punctuation in odd places which would require some ancillary coding.

Method C

Another alternative is an quick-and-dirty engineering approach using a lookup table. If you are using a programming language that supports the

BASIC-like INSTR (instring) function your code could look like this:

```
DO
 READ DataStreamByte FROM DataStream
 IF (INSTR(" ¢.<(+&!$*);|-
/.%_>?:#@'="""",ChrString(DataSteamByte)) <> 0) OR

(INSTR("abcdefghijklmnopqrstuvwxyz",ChrString(DataSteamByte))
<> 0) OR

(INSTR("ABCDEFGHIJKLMNOPQRSTUVWXYZ",ChrString(DataSteamByte))
<> 0) OR
    (INSTR("0123456789",ChrString(DataSteamByte)) <> 0) OR
 THEN
   DisplayCharacter(DataStreamByte)
 END IF
 IF DataSteamByte = 21 THEN DisplayNewLine
LOOP UNTIL END_OF_FILE(DataSteam)
```

In this example ChString is a function converting the binary value into a text string. The double-quote in the first INSTR function is to signal a quote enclosed in the string. The output of the INSTR function is the position of the second parameter in the first, so INSTR("abc","b") would return 2.

Although the code is more compact and easier to maintain than the other two, the lookup table must potentially do 89 comparision functions per character in the data stream. This could be optimised by a variety of techniques, but this is additional coding. EBCDIC Method A only used 23, so would be faster. The ASCII method uses just 3 comparisons.

Method D

A combination of these methods may work best. A binary lookup table in a 256 element array is the simplest approach, but potentially a 16 by 16 array would be faster with the correct coding. If you need to process the data quickly, you will have to keep a close eye on performance. In any case, the speed at which are able to parse the data stream is always going to be slower with EBCDIC than ASCII.

Reason 2: Talking to the outside world top

Most of the world runs on ASCII. Even in IBM mainframe environments, host PCs, terminals and printers may use ASCII as their native character set.

Standard versions of EBCDIC miss out the ASCII characters **[ ] \ { } ^ ~ ¦** and include the **¢** sign, so there isn't even a direct match between them.

Worse still, some of the missing ASCII characters are in the **UUencoding** range, so it will tend to corrupt standard Internet mail attachments. A process called XXencoding helps with this issue, but this is a rarely used way of encoding attachments. This can cause a problem even if the EBCDIC system is only a mail relay.

Reason 3: EBCDIC just plain sucks top

Although all binary character encoding mechanisms are based on Baudot code (developed as an alternative to Morse code in the late 19th Century), EBCDIC's close link with punched cards (themselves a 19th Century invention) should relegate it to one of history's has-beens. ASCII on the other hand was developed in the late 1960s, and although also based on Baudot code to an extent, is clearly a character set designed with easy of use in mind.

EBCDIC also has dozens of different variants, some of which are essentially incompatible with other variants of EBCDIC. Documentation on EBCDIC is very hard to obtain from IBM, even though they still manufacture EBCDIC systems such as the AS/400 into the twenty-first century.

Web Resources top

- **A Brief History of Character Codes in North America, Europe, and East Asia** at **Tron Web**
- **Problems Encountered Transferring Files from EBCDIC to ASCII** at **IBM Geographic Information Systems**
- A detailed **ASCII-EBCDIC chart** at **Natural Innovations**
- dmoz open directory project related topics at **dmoz**

subj:

home technical diary webmaster stuff orange book shop contact links your privacy