How it was: ASCII, EBCDIC, ISO, and Unicode

Clive (Max) Maxfield - October 07, 2011

There’s an old engineering joke that says: "Standards are great ... everyone should have one!" The problem is that – very often – everyone does. Consider the case of storing textual data inside a computer, where the computer regards everything as being a collection of numbers. In this case, someone has to (a) decide which characters need to be represented in the first place and (b) decide which numeric values are going to be associated with the various characters. The resulting collection of character-to-number mappings is referred to as a "code".

ASCII Code(s)
Towards the end of the 1950s, the American Standards Association (ASA) began to consider the problem of defining a standard character code mapping that could be used to facilitate the representation, storing, and interchanging of textual data between different computers and peripheral devices. In 1963, the ASA – which changed its name to the American National Standards Institute (ANSI) in 1969 – announced the first version of the American Standard Code for Information Interchange (ASCII).

However, this first version of the ASCII code (which is pronounced "ask-key") left many things – such as the lower case Latin letters – undefined, and it wasn't until 1968 that the currently used ASCII standard of 96 printing characters and 32 control characters was defined as illustrated in Figure 1.
Let’s just pause for a moment to appreciate how tasty this version of the table looks (like all of the images in this article, it was created by yours truly in Visio). But we digress…

Note that code $20$ (which is annotated "SP") is equivalent to a space. Also, as an aside, the terms uppercase and lowercase were handed down to us by the printing industry, from the compositors’ practice of storing the type for capital letters and small letters in two separate trays, or cases. When working at the type-setting table, the compositors invariably kept the capital letters and small letters in the upper and lower cases, respectively; hence, "uppercase" and "lowercase." Prior to this, scholars referred to capital letters as majuscules and small letters as minuscules, while everyone else simply called them capital letters and small letters.

We should also note that one of the really nice things about ASCII is that all of the alpha characters are numbered sequentially; that is, 65 ($41$ in hexadecimal) = 'A', 66 = 'B', 67 = 'C', and so on until the end of the alphabet. Similarly, 97 ($61$ in hexadecimal) = 'a', 98 = 'b', 99 = 'c', and so forth. This means that we can perform cunning programming tricks like saying "char = 'A' + 23" and have a reasonable expectation of ending up with the letter 'X'. Alternatively, if we wish to test to see if a character (called "char") is lowercase and – if so – convert it into its uppercase counterpart, we could use a piece of code similar to the following:
if (char >= 'a') and (char <= 'z') then char = char - 32;

Don't worry as to what computer language this is; the important point here is that the left-hand portion of this statement is used to determine whether or not we have a lowercase character and, if we do, subtracting 32 ($20 \text{ in hexadecimal}$) from that character's code will convert it into its uppercase counterpart.

As can be seen in Figure 1, in addition to the standard alphanumeric characters ('a'...'z', 'A'...'Z' and '0'...'9'), punctuation characters (comma, period, semi-colon, ...) and special characters ( '*', '#', '%', ...), there are an awful lot of strange mnemonics such as EOT, ACK, NAK, and BEL. The point is that, in addition to representing textual data, ASCII was intended for a number of purposes such as communications; hence the presence of such codes as EOT, meaning "End of transmission," and BEL, which was used to ring a physical bell on old-fashioned printers.

Some of these codes are still in use today, while others are, generally speaking, of historical interest only. For those who are interested, a more detailed breakdown of these special codes is presented in Figure 2.

![Figure 2. ASCII control characters.](image)

One final point is that ASCII is a 7-bit code, which means that it only uses the binary values %0000000 through %1111111 (that is, 0 through 127 in decimal or $00$ through $7F \text{ in hexadecimal}$). However, computers store data in multiples of 8-bit bytes, which means that – when using the ASCII code – there's a bit left over. In some systems, the unused, most-significant bit of an 8-bit byte representing an ASCII character is simply set to logic 0. In other systems, the extra 128 codes that can be accessed using this bit might be used to represent simple “chunky graphics” characters. Alternatively, this bit might be used to implement a form of error detection known as a parity check, in which case it would be referred to as the parity bit.

**EBCDIC Code(s)**

The ASCII code discussed above was quickly adopted by the majority of American computer manufacturers, and was eventually turned into an international standard (see also the discussions on ISO and Unicode later in this paper.) However, IBM already had its own six-bit code called BCDIC (Binary Coded Decimal Interchange Code). Thus, IBM decided to go its own way, and it developed a proprietary 8-bit code called the Extended Binary Coded Decimal Interchange Code (EBCDIC).
Pronounced "eb-sea-dick" by some and "eb-sid-ick" by others, EBCDIC was first used on the IBM 360 computer, which was presented to the market in 1964. As was noted in our earlier discussions, one of the really nice things about ASCII is that all of the alpha characters are numbered sequentially. In turn, this means that we can perform programming tricks like saying "char = 'A' + 23" and have a reasonable expectation of ending up with the letter 'X'. To cut a long story short, if you were thinking of doing this with EBCDIC ... don't. The reason we say this is apparent from the table shown in Figure 3.

Figure 3. EBCDIC character codes.

A brief glance at this illustration shows just why EBCDIC can be such a pain to use – the alphabetic characters don't have sequential codes. That is, the letters 'A' through 'I' occupy codes $C1$ to $C9$, 'J' through 'R' occupy codes $D1$ to $D9$, and 'S' through 'Z' occupy codes $E2$ to $E9$ (and similarly for the lowercase letters). Thus, performing programming tricks such as using the expression ('A' + 23) is somewhat annoying with EBCDIC. Another nuisance is that EBCDIC doesn't contain all of the ASCII codes, which makes transferring text files between the two representations somewhat problematical.

Once again, in addition to the standard alphanumerical characters ('a'...'z', 'A'..'Z' and '0'..'9'), punctuation characters (comma, period, semi-colon, ...), and special characters ('!', '#', '%', ...), EBCDIC includes a lot of strange mnemonics, such as ACK, NAK, and BEL, which were designed for communications purposes. Some of these codes are still used today, while others are, generally speaking, of historical interest only. A slightly more detailed breakdown of these codes is presented in Figure 4 for your edification and delight.
As one final point of interest, different countries have different character requirements, such as the á, è, and ü characters. Due to the fact that IBM sold its computer systems around the world, it had to create multiple versions of EBCDIC. In fact, 57 different national variants were eventually wending their way across the planet. (A "standard" with 57 variants! You can only imagine how much fun everybody had when transferring files from one country to another).

**ISO and Unicode**

Upon its introduction, ASCII quickly became a de facto standard around the world. However, the original ASCII didn't include all of the special characters (such as á, è, and ü) that are required by the various languages that employ the Latin alphabet. Thus, the International Organization for Standardization (ISO) in Geneva, Switzerland, undertook to adapt the ASCII code to accommodate other languages.

In 1967, the organization released its recommendation ISO 646. Essentially, this left the original 7-bit ASCII code "as was", except that ten character positions were left open to be used to code for so-called "national variants."

ISO 646 was a step along the way toward internationalization. However, it didn't satisfy everyone's requirements; in particular (as far as the ISO was concerned), it wasn't capable of handling all of the languages in use in Europe, such as the Arabic, Cyrillic, Greek, and Hebrew alphabets. Thus, the ISO created its standard 2022, which described the ways in which 7-bit and 8-bit character codes were to be structured and extended.

The principles laid down in ISO 2022 were subsequently used to create the ISO 8859-1 standard.
Unofficially known as "Latin-1", ISO 8859 is widely used for passing information around the Internet in Western Europe to this day. Full of enthusiasm, the ISO then set about defining an "all-singing all-dancing" 32-bit code called the Universal Coded Character Set (UCS). Now known as ISO/IEC DIS 10646 Version 1, this code was intended to employ escape sequences to switch between different character sets. The result would have been able to support up to 4,294,967,296 characters, which would have been more than sufficient to address the world's (possibly the universe's) character coding needs for the foreseeable future.

However, starting in the early 1980s, American computer companies began to consider their own solutions to the problem of supporting multilingual character sets and codes. This work eventually became known as the Unification Code, or Unicode for short. Many people preferred Unicode to the ISO 10646 offering on the basis that Unicode was simpler. After a lot of wrangling, the proponents of Unicode persuaded the ISO to drop 10646 Version 1 and to replace it with a Unicode-based scheme, which ended up being called ISO/IEC 10646 Version 2.

Further Reading
There are many additional resources available to anyone who is interested in knowing more about this - and related - topics. The following should provide some "food for thought," and please feel free to let me know of any other items you feel should be included in this list (email me at max@CliveMaxfield.com):

1. Two really good starting points are, of course, the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO).
2. Steven J. Searle has created a tremendously useful site entitled A Brief History of Character Codes. This site provides a wealth of information pertaining to the introduction and evolution of character codes in North America, Europe, and East Asia.
3. Jim Price provides a very useful Introduction to ASCII on his website. In addition to information such as IBM PC keyboard scan codes, Jim's FAQ answers questions such as: "What are the ASCII codes for things like the degrees symbol (°), the trademark symbol (®), solid blocks, and other special symbols?"
4. Last but certainly not least, my book How Computers Do Math includes a series of step-by-step interactive laboratories that guide the reader in using ASCII codes to display messages and numerical data on a virtual machine called the DIY Calculator that comes on the CD accompanying the book.

Click Here to see other articles in this "How it was..." series...

Editor's Note: It would be great if - in addition to commenting on my articles - you took the time to write down short stories of your own. I can help in the copy editing department, so you don't need to worry about being "word perfect". All you have to do is to email your offering to me at max@CliveMaxfield.com with "How it was" in the subject line.

I can post your article as “anonymous” if you wish. On the other hand, what would be really cool would be if you wanted to add a few words about yourself - and maybe even provide a couple of “Then and Now” pictures - for example:
On the left we see me as a young sprog – I was still a student at this time, poised on the brink of leaping into my first position at International Computers Limited (ICL). On the right we see me as I am today – a much older and sadder man, beaten down by the pressures of work and bowed by the awesome responsibilities I bear (grin).

If you found this article to be of interest, visit EDA Designline where – in addition to blogs on all sorts of "stuff" – you will find the latest and greatest design, technology, product, and news articles with regard to all aspects of Electronic Design Automation (EDA).

Also, you can obtain a highlights update delivered directly to your inbox by signing up for the EDA Designline weekly newsletter - just Click Here to request this newsletter using the Manage Newsletters tab (if you aren't already a member you'll be asked to register, but it's free and painless so don't let that stop you [grin]).