COMPUTER MEMORY UNITS

This short chapter introduces some interesting topics that help to explain numeric HTML character entities – basically why characters have numerical codes that uniquely identify them. It also covers the basics of fundamental computer memory storage units like bits, bytes, kilobytes, and so forth. It would be a crime to get out of an entry-level computer science course without being introduced to these concepts.

1. BITS AND BYTES

The term *bit* is short for binary digit. Humans think of binary digits as 0 and 1. Indeed, *binary numbers* such as 10101100 are composed of 0s and 1s. Most people realize that computers store information in binary, but don't realize that 0s and 1s are for just humans. Computer hardware needs a physical representation. For example, on a magnetic hard drive, small sectors either have a negative or positive charge. On a silicone computer chip, a capacitor can have two different levels of electrical charge. These are physical things that represent two-state conditions, but humans conveniently think of the two distinct states as 0 and 1. When a computer stores an 8-digit binary number like 10101100, it's physically storing a sequence of 8 plus/minus charges on a magnetic plate, or a sequence of 8 two-state electrical charges in capacitors on a silicon chip.

A *byte* is a sequence of 8 bits. In modern computers, a byte is the smallest unit of storage that can have a distinct address. Hence, virtually all forms of computer storage is measured in bytes. Computer storage is often called memory in the sense that the computer "remembers" data over a long period of time. How much memory is a byte? Another way to ask that is how many different things can 1 byte represent?

The hardest way to answer that question would simply be to start writing down all the distinct bytes: 00000001, 00000010, 00000100, …, 10000000, 00000011, 00000101, …. If you do that, and are very careful not to leave any out, you will count 256 distinct bytes. That might be good fun on a rainy day, but there are much better ways. For example, since there are 8 digits in a byte, and there are exactly 2 possibilities for each digit, the multiplication principle indicates that there are $2x2x2x2x2x2x2x2 = 2^8 = 256$ distinct possibilities.

Yet another way to deduce that is to convert 8-digit binary numbers into base 10 numbers. Base 10 numbers are the ones we use every day. In base 10, each digit (from right to left) represents increasing powers of 10: $10^0 = 1$, $10^1 = 10$, $10^2 = 100$, $10^3 = 1000$, and so forth. It works the same way in binary, expect the base is 2. Going from right to left in base 2, the digits are valued as $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, and so forth. Clearly the smallest byte is 00000000 which converts to 0 in base 10 since there are 0 of each power of 2. The largest byte is 11111111 which converts to 255 in base 10 since there is 1 of each power of 2 (1+2+4+8+16+32+64+128 = 255). So bytes cover all numbers from 0-255 when converted into base 10. That's is 256 distinct bytes.

We have concluded (in 3 different ways) that there are 256 distinct bytes. That means if each distinct byte represents something, then 1 byte can hold 256 different "somethings." Depending upon what the computer storage is being used for, those 256 different somethings might be 256 different keyboard characters, 256 different colors, or simply the numbers from 0-255. Recall that IP addresses are of the form 164.168.21.170, where each of the numbers must be in the range 0-255. That means each number is 1 byte, and IP addresses are a 4-byte addressing system!

2. CHARACTER ENCODINGS

An example provided with this Lesson introduced HTML Character Entities such as * * which is a non-breaking space (software won't soft-wrap around it). Character entities can also be expressed using a numerical code. For example, both * * and * * create exactly the same non-breaking space character, and both *©* and *©* create the exact same copyright symbol ©. It is then natural to wonder what it means for characters to have numerical codes in addition to names like *©* that are easier for humans to remember. The answer to that explains why *bytes* are the fundamental units used to describe the storage capacity of most modern computer devices, including hard drives.

When you type a keyboard character, it has to be converted into a number to store it in computer memory. Consider the following examples for the capital A and lower case a characters.

Character		**Code**		**Binary (byte)**
A	⟷	65	⟷	01000001
a	⟷	97	⟷	01100001

When you type a character, the computer looks up its numerical code, and then converts that into binary for storage. Suppose, for example that you type a capital A and it is stored as the byte shown above as part of a text file. Now suppose that text file is sent to a different computer. It's absolutely imperative that the other computer converts character #65 back into a cap A. That is, if two different computers think character #65 is two different characters, then they would not be able to share text data because it would get scrambled.

The *American Standard Code for Information Interchange (ASCII)*, was created in the 1960s to standardize storage of characters most commonly found on English language keyboards in the USA. The ASCII *character encoding* ensured that text typed on one computer would be interpreted as the same characters on another computer as long as they were both ASCII compliant. For example, a byte stored as 01000001 in text data would always be translated to the cap A character regardless of the computer system reading the data.

Original ASCII standardized 128 characters, thus requiring only 7 bits. But most computer systems stored data in byte sized chunks (8 bits), allowing for a total of 256 characters. The 256-character encodings (128 ASCII characters plus 128 additional characters) were usually called *Extended ASCII*. By the 1980s, there were a few different Extended ASCII character encodings with

differences only among the 128 characters that extended ASCII. Since Microsoft was the dominant desktop operating system, their Extended ASCII character encoding was widely used.

There is a graphic on the course web site that shows a typical Extended ASCII character encoding. Characters #0-31 are mostly non-printing characters that are mostly obsolete today, except for characters like the horizontal tab (#9). Characters #32-127 are the original ASCII characters where you can find the lower and cap A codes provided above, for example. Characters #128-255 are mostly European language characters and other useful symbols like ©. This facilitated use of keyboards configured for European languages, for example. Notice that #160 is the non-breaking space (HTML character entity * *), but original ASCII also provided a normal, breaking space as character #32. Most people are surprised to learn that there are two distinct blank space characters with different functionality.

In the 1990s, a new standard called *Unicode Transformation Format* was introduced. It's usually just called *Unicode* or *UTF-8*. UTF-8 is backwards compatible with ASCII, but can encode over 1 million different characters, including characters from Arabic and Hebrew, and the symbolic languages like Chinese, Korean, and Japanese which can have 10s of thousands of characters each. It even includes ancient characters from languages like Babylonian Cuneiform and Egyptian Hieroglyphs, and new ones like Emojis where new ones are still being created. UTF-8 still stores ASCII characters in one byte, but UTF-8 uses up to 4 bytes per character for larger character numbers.

You can find some examples of character encodings for some of the larger numbers in the HTML character entities example included with this Lesson. The numbers for the HTML character entities are the same as the UTF-8 character numbers. If you are an astute observer, you may have noticed that the basic HTML 5 document provided by *W3Schools.com* has the following line in the head section: *<meta charset="UTF-8">* This tells the browser that the bytes saved in the HTML file represent UTF-8 characters. That means HTML files are *plain text* files – the bytes in the file represent UTF-8 character codes. Many types of computer files are not plain text files, like spreadsheets, powerpoint, music, video, etc. Those are sometimes called *binary blobs* because it's much harder to describe what the bytes in the file represent.

This section explains why bytes are the fundamental memory unit. Hard drives, computer RAM and cell data usage, just to name a few, are still all measured in bytes. This stems from the early 8-bit character encodings such as ASCII. Since keyboard characters were stored as bytes, it made sense to measure computer memory in bytes – basically how many keyboard characters could be stored.

In contrast, network throughput speeds (data transfer rates) are generally measured in bits per second. For example, the speed of the Internet service you buy from an ISP, or the speed of a Wi-FI network will be listed in bits per second (not bytes). Basically, that's a measure of how fast raw data can be moved.

3. KILOBYTES, MEGABYTES, GIGABYTES, ETC

The *International System of Units* defines several prefixes that were historically derived from the Greek language. The ones you are most likely familiar are *kilo* ($1{,}000 = 10^3$), *mega* ($1{,}000{,}000 = 10^6$), and *giga* ($1{,}000{,}000{,}000 = 10^9$). Using these prefixes, a *KiloByte* (KB) is 1,000 bytes, a *MegaByte* (MB) is 1,000 KiloBytes, and a *GigaByte* (GB) is 1,000 MegaBytes. When the context is clear, people often use K instead of KB, M instead of MB, and so forth.

It can be hard to wrap your brain around what it means to talk about millions (mega) and billions (giga) of bytes. There are even larger prefixes like *tera*, *peta*, and *exa* that have become more relevant as computer storage capacities have gotten larger, and the amount of data the world produces has skyrocketed. In terms of computer memory capacity, the larger prefixes were mostly not even relevant not that long ago, like in the 1990s when the WWW was invented.

There is a principle called *Moore's Law*, named after a scientist that co-founded the Intel corporation that's famous for making computer processors. Very loosely stated, Moore's law says that the amount of data computer chips can process doubles about every 2 years as new chips get smaller and faster. The same principle has roughly predicted the growth of computer memory capacity as well. This principle has held for the most part since 1975, although it has begun to slow down in recent years. The next big breakthrough that will enable smaller devices to hold radically more data is likely to be quantum computing, which is basically computing at the atomic level.

The table below lists a few examples so that you can wrap your brain around the order of magnitude involved with each prefix. Advances in technology that have roughly followed Moore's law have increasingly made prefixes like tera and peta more relevant. The Internet of Things has even made the exa prefix relevant in today's world.

Unit	**Very Rough Estimates to Show Order or Magnitude**
KiloByte 10^3 (thousand)	• IP Packet – 1.5K • This PDF file – 150K • Typical web Page (HTML file + Graphics) – a few K to a few hundred K
MegaByte 10^6 (million)	• MP3 file – about 1M per minute • Uncompressed Digital Picture from Typical Smartphone – 5M to 10M • Traditional CD (Compact Disk) – 700M
GigaByte 10^9 (billion)	• DVD Disk Capacity (per side) or Standard HD Movie – 5G • Ultra HD Movie (4K pixel resolution) – 20G • Typical Cell Phone Hard Drive – 32G to 128G • Typical Laptop Hard Drive – 500G
TeraByte 10^{12} (trillion)	• Typical External Hard Drive – 1T to 5T • Annual Data Generated by Hubble Space Telescope – 10T • Amount of Video Data Uploaded to YouTube Per Day in 2016 – 24T
PetaByte 10^{15} (quadrillion)	• Standard HD Movie that's 50 Years Long (need lots of popcorn) – 1P • Human Brain Capacity – 2P or 3P (based on common estimates) • Amount of Data Processed by Google Every Day – 50P • Entire Written Work of Mankind from Beginning of Written History (All Languages) – 50P
Exabyte 10^{18} (quintillion)	• Internet Traffic Per Day – 3E • Total Estimated Data held by Google – 15E • Total Computer Data on Saved on Earth – A few thousand E (that's a few *ZettaBytes*)

In computing, things are generally done in powers of 2. You know, that binary thing. Technically speaking, a KiloByte was originally $1024 = 2^{10}$ bytes. The prefix Kilo was used for convenience since 1000 is a reasonably close approximation to 1024. So technically speaking, a MegaByte was originally 1024 K, and a GigaByte was 1024 M, and so forth. This was a source of confusion for a long time since these suffixes are used for many other purposes, like a KiloMeter. In non-computer contexts, Kilo always means 1000, and the larger prefixes are always exact powers of 10.

To alleviate confusion, the International System of Units introduced some new units around the year 2000. For example, a *KibbiByte* is 1024 bytes, a *Mebibyte* is 1024 KibbiBytes, a *Gibibyte* is 1024 Mebibytes, and so forth. These odd-sounding prefixes are used in certain contexts in computer science. But when you buy computer equipment, or talk about computer storage in every day conversation, a KiloByte means 1000 bytes, and so forth. So for example, if your phone has 64G of memory, then that's the common use of Giga and means 64,000,000,000 bytes (not GibiBytes).