Data Representation: Objectives

- refs: [O'H&Bryant, sect 2.1,2.4], [Tanembaum, appendix B], related web links
- understand how binary data is organized on a computer, and the units for counting them
- understand how characters and strings are encoded as bit patterns
- understand the difference between the big- and little-endian storage conventions

Bit, Byte and Word

- basic unit of memory is the bit (Binary digIT)
- one bit is too small to be generally useful, so bits are grouped together:
  - nibble: 4 bits (cf. a hex digit)
  - byte: 8 bits (e.g. 1010 1010 or 1000 0001)
  - word: 1, 2, 4 or 8 bytes (usually 4 or 8)
  - word length depends on processor, operating system, etc. (e.g. 8 bytes for Intel/AMD's x86-64 and most modern architectures)
- data and machine instructions are normally stored and accessed in words (in memory)
  - main memory can be thought of as a large array of bytes
  - modern computers can access memory in units of bytes and words (and data sizes in-between)
    - data of a certain size must be properly aligned

Kilobyte, Megabyte, Gigabyte, ...

There are a number of different standards about the size of data – care must be taken (depends on the context). (figure from Wikipedia)

Multiples of bytes				
SI decimal prefixes		Binary usage	IEC binary prefixes	
Name (Symbol)	Value		Name (Symbol)	Value
kilobyte (kB/KB)	10^3	2^{10}	kibibyte (KiB)	2^{10}
megabyte (MB)	10^6	2^{20}	mebibyte (MiB)	2^{20}
gigabyte (GB)	10^9	2^{30}	gibibyte (GiB)	2^{30}
terabyte (TB)	10^{12}	2^{40}	tebibyte (TiB)	2^{40}
petabyte (PB)	10^{15}	2^{50}	pebibyte (PiB)	2^{50}
exabyte (EB)	10^{18}	2^{60}	exbibyte (EiB)	2^{60}
zettabyte (ZB)	10^{21}	2^{70}	zebibyte (ZiB)	2^{70}
yottabyte (YB)	10^{24}	2^{80}	yobibyte (YiB)	2^{80}
See also: Multiples of bits · Orders of magnitude of data				

- my current desktop machine has:
  - 3072 KiB cache memory, 8.0 GiB main memory
  - 282 GiB hard disk, with 10.6 GiB swap space
- CD-ROMs store around 703 MiB
- DVDs store around 4.3 to 8.0 GiB
- mass storage systems: $\approx$ 1 PB

Characters and Strings

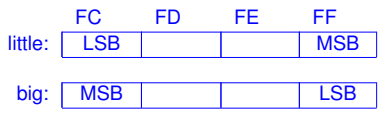
- text can be stored in memory by using a number to represent every character
  - ASCII (American Standard Code for Information Interchange): 7 bits, 128 characters (see `man ascii`)
  - EBCDIC: Extended Binary-Coded Decimal Interchange Code (8 bits)
  - UCS/UNICODE: attempts to extend ASCII to other languages (65,536 characters) (even tengwar is there!)
  - UTF-8 is a variable width standard that can represent Unicode characters and is backward compatible with ASCII
- a string is a sequence of characters, usually terminated with a byte value of 0
- example: 43 4F 4D 50 32 33 30 30 00_{16}

Big-endian versus Little-endian Integers

- small (how small?) integers are sometimes stored in one byte
- most integers are stored in (longer) words; how are they arranged in memory?

  Example: 126540713_{10} stored in 4 bytes: $00000111\ 10001010\ 11011011\ 10101001_2$

Address	big-endian	little-endian
FF_{16}	10101001	00000111
FE_{16}	11011011	10001010
FD_{16}	10001010	11011011
FC_{16}	00000111	10101001



- big-endian stores the most significant byte (MSB) at the lowest address in the word, little-endian in the highest (see also endian.c. Origin of term endian)
- similar to use of `me@cs.anu.edu.au` OR `me@au.edu.anu.cs` (e.g. JANET network in the UK)
- little endian: (Intel) more natural and consistent way to pick up 1, 2, 4, or longer byte integers (making multi-precision arithmetic easier)
- big endian: (SPARC, IBM) MSB first gives easy testing of +ve/-ve. Numbers stored in order they are printed, making binary to decimal conversion easier
- potential confusion in communication with differing endianness!