

## Data representation

- Bit, byte and word
- Characters and strings, ASCII
- Big-endian, little-endian
- Real numbers
- References:
  - Chapter 2 in Bryant and O'Hallaron
  - Appendix B in Tanenbaum
  - Related links on Web site

## Bit, byte and word (cont.)

- 1 Kilobyte (KB) = 1024 bytes =  $2^{10}$  bytes
- 1 Megabyte (MB) = 1024 KB = 1,048,576 bytes
- 1 Gigabyte (GB) = 1024 MB
- Typical modern computers have millions of bytes of memory (e.g. PC with 512 Megabytes, servers 4-16 Gigabytes)
- CD-ROMs store around 650 Megabytes (MB)
- DVDs store around 4.7 Gigabytes (GB), up to 17 GB
- Both instructions and data are stored in memory

## 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:
  - Byte = 8 bits (e.g. 1010 1010 or 1000 0001)
  - Word = 1, 2, 4 or 8 bytes (usually 2 or 4)
  - Word length depends on processor, operating system, etc. (e.g. 8 bytes for Intel's I-64 or Compaq Alpha architectures)
- Data and machine instructions are stored and accessed in words

## Characters and strings

- Text can be stored in memory by using a number to represent every character
  - ASCII (see *Bryant*, page 40-41), American Standard Code for Information Interchange (7 bits, 128 characters)
  - EBCDIC - Extended Binary-Coded Decimal Interchange Code (8 bits)
  - UCS/UNICODE - attempts to extend ASCII to other languages (65,536 characters)
- A *string* is a sequence of characters, usually terminated with a byte value of 0
- Example: 43 4F 4D 50 32 33 30 30 0<sub>16</sub>

## Big-endian versus little-endian

- Small integers are sometimes stored in one byte
- Most integers are stored in words of more than one byte.  
Question: How are they arranged in memory?

Example: 126540713<sub>10</sub> stored in 4 bytes:  
00000111 10001010 11011011 10101001<sub>2</sub>

Address	Big-endian	Little-endian
FF <sub>16</sub>	10101001	00000111
FE <sub>16</sub>	11011011	10001010
FD <sub>16</sub>	10001010	11011011
FC <sub>16</sub>	00000111	10101001

- Big-endian stores the most significant byte (MSB) at the lowest address in the word, little-endian in the highest

## Real numbers

- Section 2.4 in *Bryant*, related links on Web site
- Fix-point systems:
  - A fixed number of bits for the integer component and a fixed number of bits for the fractional component
  - Needs a large number of bits to represent a useful range.  
Example: Mass of electron is  $9 \times 10^{-28}$  grams, mass of the sun is  $2 \times 10^{33}$ , a range of  $10^{60}$ ! We would need 200 bits (25 bytes) to cover this range.  
We would have 200 bits of precision when dealing with solar masses, but only a couple of bits when working with atomic masses.  
The first is wasteful, the second not accurate enough.

## Big-endian versus little-endian (cont)

- Similar to use of *me@cs.anu.edu.au* OR *me@au.edu.anu.cs* (as was the case with the 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 math easier)
- Big Endian: (SPARC, big IBM) higher byte first gives easy testing of positive or negative number. Numbers store in order they are printed, making binary to decimal conversion easier.
- Confusion in communicating between the two!

## Floating point numbers

- Similar to scientific notation:  
 $n = f \cdot 10^e$  Example:  $3.141 = 0.3141 \times 10^1$
- Floating point generally uses:  $n = s \times m \times 2^e$ 
  - One bit for sign  $s$  (0 positive, 1 negative)
  - Precision is determined by the number of bits used to represent the mantissa  $m$
  - Range is determined by the number of bits used to represent the exponent  $e$
- Floating point numbers are usually normalised

## Floating point formats

- Example: 32-bits (IEEE 754 standard float)
 

s	eeeeeeee	mmmmmmm	mmmmmmmm	mmmmmmmm
1 Bit	8 Bits		23 Bits	
Sign	Exponent		Mantissa	
- IEEE double: 1 bit sign, 11 bits exponent, 52 bits mantissa
- IEEE 754 standard also defines specific formats, behaviours and values, like overflow, infinity (+/-) and Not-A-Number (NaN, e.g. error, not defined variable)
- Many processors work with 80 bits internally, and report the result back at 32 or 64 bit precision

## Floating point caveats

- Not all numbers can be represented:
  - *overflow*: Numbers bigger than  $maximum\_mantissa \times 2^{maximum\_exponent}$
  - *underflow*: Numbers smaller than  $minimum\_mantissa \times 2^{minimum\_exponent}$
- Rounding errors can do accumulate to become significant.  
Example: Patriot Missile System (see link on Web site)

## Floating point extra

- Present a few slides relating to *Bryant 2.4.3*
- Illustrate grainy nature of floating point numbers and concept of machine numbers