# Reading Review

- Terminology:
  - word size, byte-oriented memory
  - address, address space
  - most-significant bit (MSB), least-significant bit (LSB)
  - big-endian, little-endian
  - pointer
- Questions from the Reading?





# Review Questions

- By looking at the bits stored in memory, I can tell what a particular 4 bytes is being used to represent.

  **A. True** **B. False**

- To fetch a piece of data from memory, all we need to know is its address.

  **A. True** **B. False**

- Which of the following bytes have a most-significant bit (MSB) of 1?

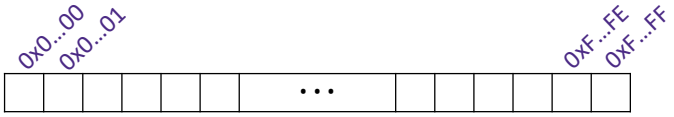
  **A. 0x63** **B. 0x90** **C. 0xCA** **D. 0xF**





# Bits and Bytes and Things (Review)

- 1 byte = 8 bits
- $n$ bits can represent up to $2^n$ things
  - Sometimes (oftentimes?) those "things" **are also bytes**!
- If addresses are $a$-bits wide, how many distinct addresses are there?
- What does each address refer to?

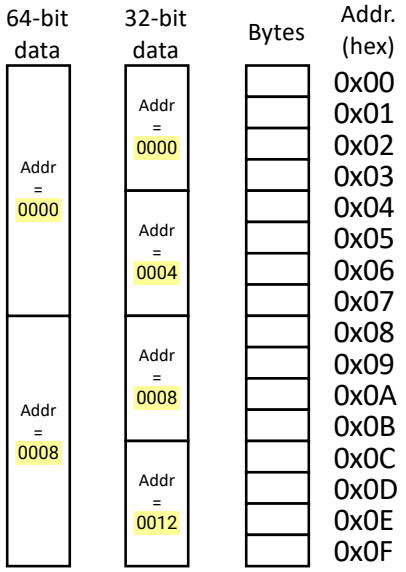






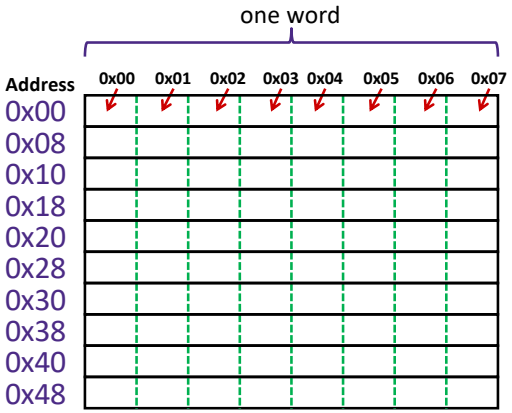

# Alignment

- The address of a chunk of memory is considered aligned if its address is a multiple of its size
  - View memory as a series of consecutive chunks of this particular size and see if your chunk doesn't cross a boundary



# A Picture of Memory (64-bit view)

- A "64-bit (8-byte) word-aligned" view of memory:
  - In this type of picture, each row is composed of 8 bytes
  - Each cell is a byte
  - An aligned, 64-bit chunk of data will fit on one row

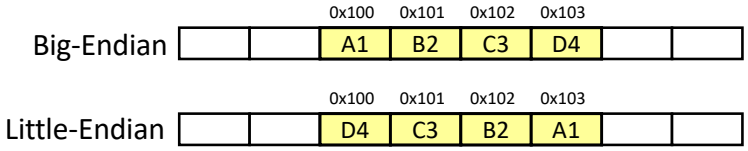







# Byte Ordering

- Big-endian (SPARC, z/Architecture)
  - Least significant byte has highest address (at the "big end")
- Little-endian (x86, x86-64)
  - Least significant byte has lowest address (at the "little end")
- Bi-endian (ARM, PowerPC)
  - Endianness can be specified as big or little
- **Example:** 4-byte data 0xA1B2C3D4 at address 0x100








# Polling Question

- We store the value `0x 01 02 03 04` as a ***word*** at address 0x100 in a big-endian, 64-bit machine
- What is the ***byte of data*** stored at address 0x104?
  - Vote on Ed Lessons

A. 0x04
B. 0x40
C. 0x01
D. 0x10
E. We're lost…

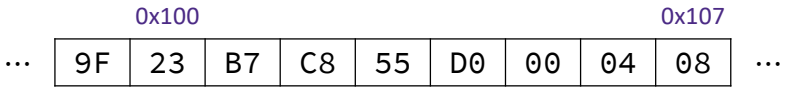
| 0x100 | 0x101 | 0x102 | 0x103 | 0x104 | 0x105 | 0x106 | 0x107 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |





# Exploration Question

- Assume the state of memory is as shown below for a **little-endian** machine.




- If we (1) *read* the value of an `int` at address 0x102, (2) add 8 to it, and then (3) store the new value as an `int` at address 0x104, which of the following addresses retain their original value?

A. 0x102 B. 0x104 C. 0x105 D. 0x107