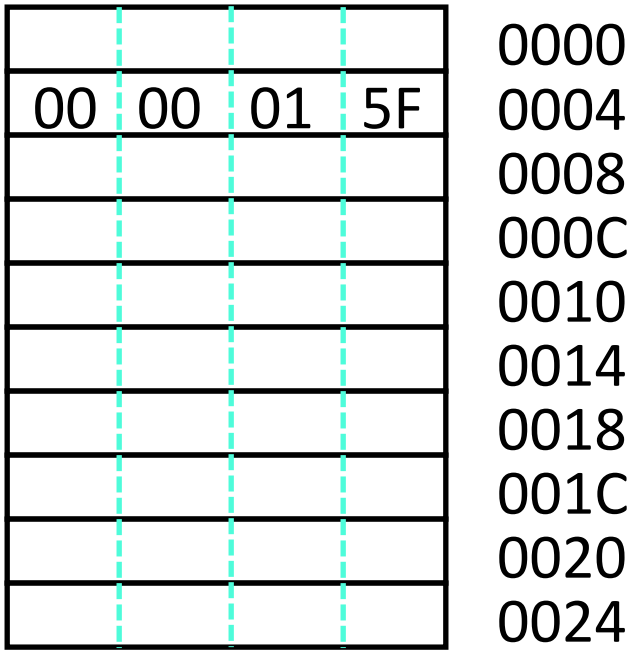
# Section 1: Memory, Data, and Addressing

- Preliminaries
- Representing information as bits and bytes
- <u>Organizing and addressing data in memory</u>
- Manipulating data in memory using C
- Boolean algebra and bit-level manipulations

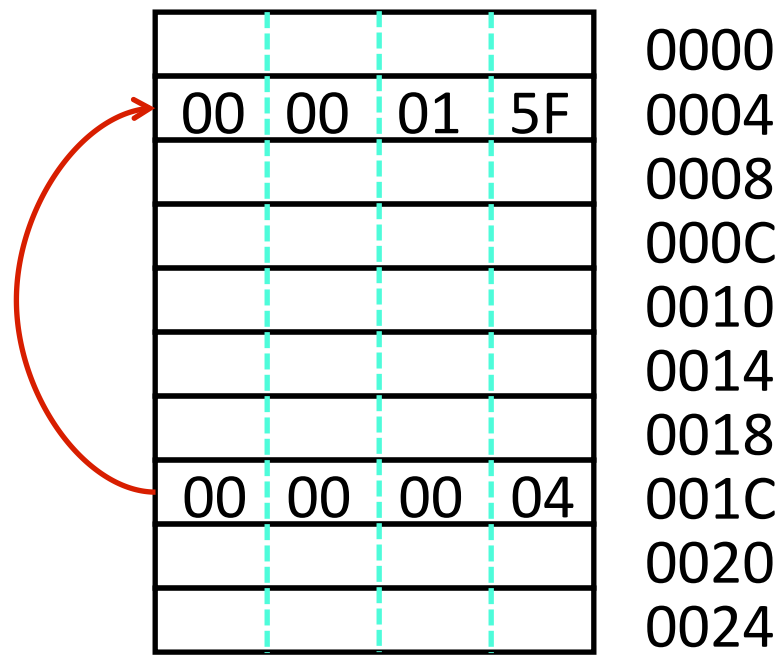
# Addresses and Pointers

- **Address is a *location* in memory**
- **Pointer is a data object that *contains an address***
- **Address 0004 stores the value 351 (or $15F_{16}$)**



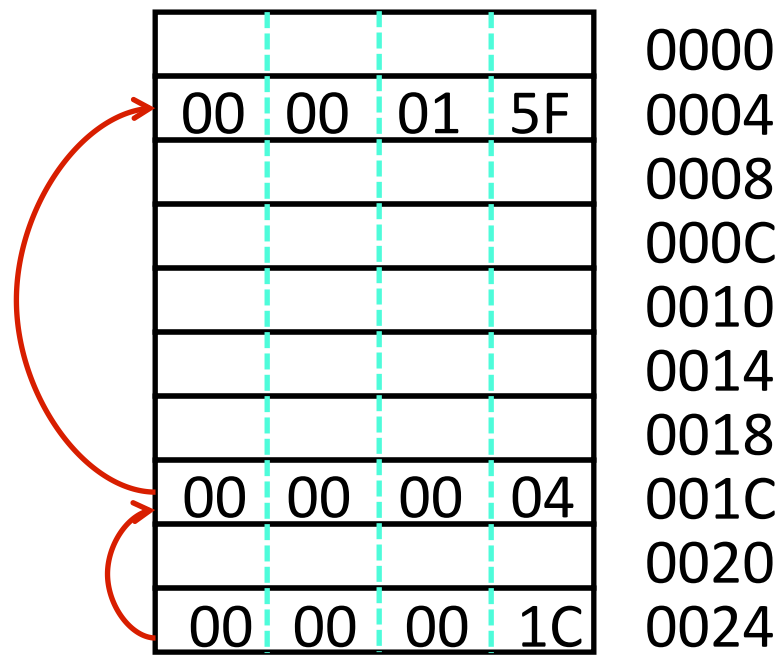
# Addresses and Pointers

- Address is a *location* in memory
- Pointer is a data object that *contains an address*
- Address 0004 stores the value 351 (or $15F_{16}$)
- Pointer to address 0004 stored at address 001C



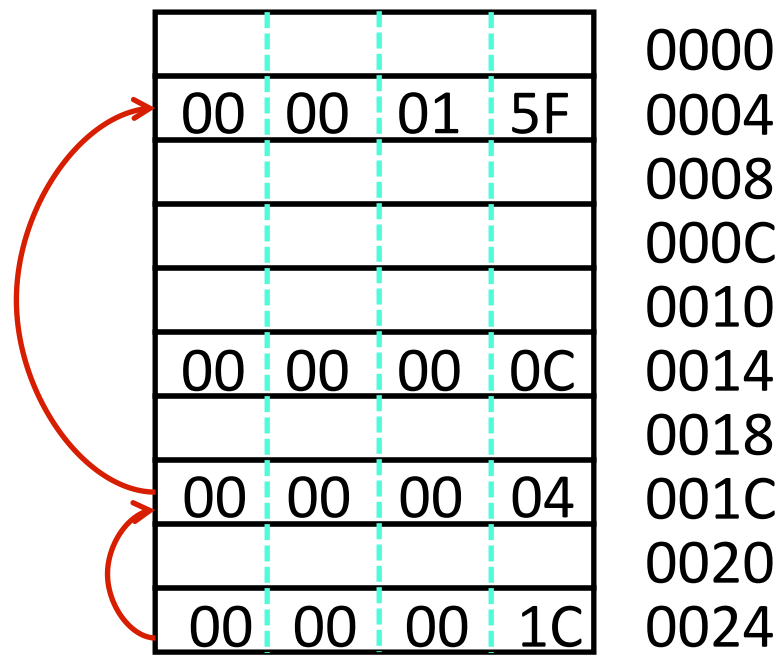
# Addresses and Pointers

- Address is a *location* in memory
- Pointer is a data object that *contains an address*
- Address 0004 stores the value 351 (or $15F_{16}$)
- Pointer to address 0004 stored at address 001C
- Pointer to a pointer in 0024



# Addresses and Pointers

- **Address is a *location* in memory**
- **Pointer is a data object that *contains an address***
- **Address 0004 stores the value 351 (or $15F_{16}$)**
- **Pointer to address 0004 stored at address 001C**
- **Pointer to a pointer in 0024**
- **Address 0014 stores the value 12**
  - Is it a pointer?



# Data Representations

- **Sizes of objects (in bytes)**

| Java data type | C data type | Typical 32-bit | x86-64 |
|---|---|---|---|
| boolean | *bool* | 1 | 1 |
| byte | char | 1 | 1 |
| char | | 2 | 2 |
| short | short int | 2 | 2 |
| int | int | 4 | 4 |
| float | float | 4 | 4 |
| | long int | 4 | 8 |
| double | double | 8 | 8 |
| long | long long | 8 | 8 |
| | long double | 8 | 16 |
| (reference) | pointer * | 4 | 8 |

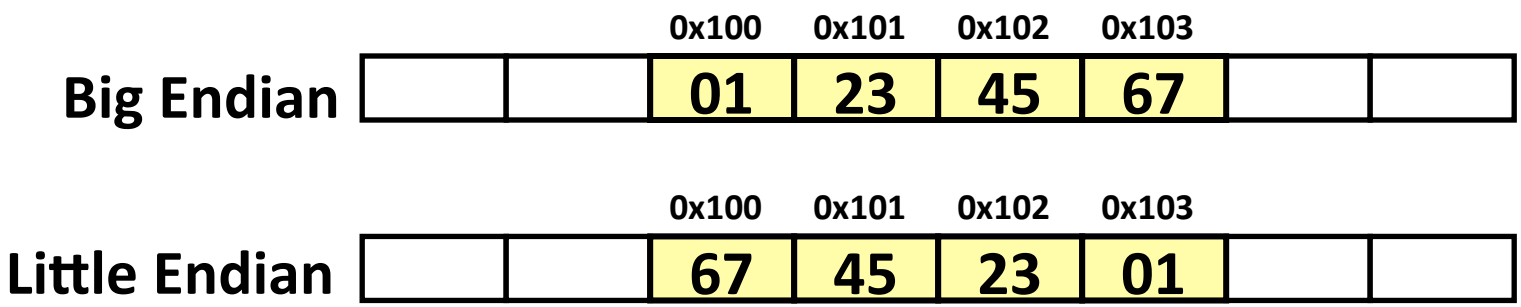
# Byte Ordering

- **How should bytes within multi-byte word be ordered in memory?**
- **Say you want to store the 4-byte word 0xaabbccdd**
  - What order will the *bytes* be stored?

- ***Endianness:* big endian vs. little endian**
  - Two different conventions, used by different architectures
  - Origin: *Gulliver's Travels* (see CS:APP2 textbook, section 2.1)

# Byte Ordering Example

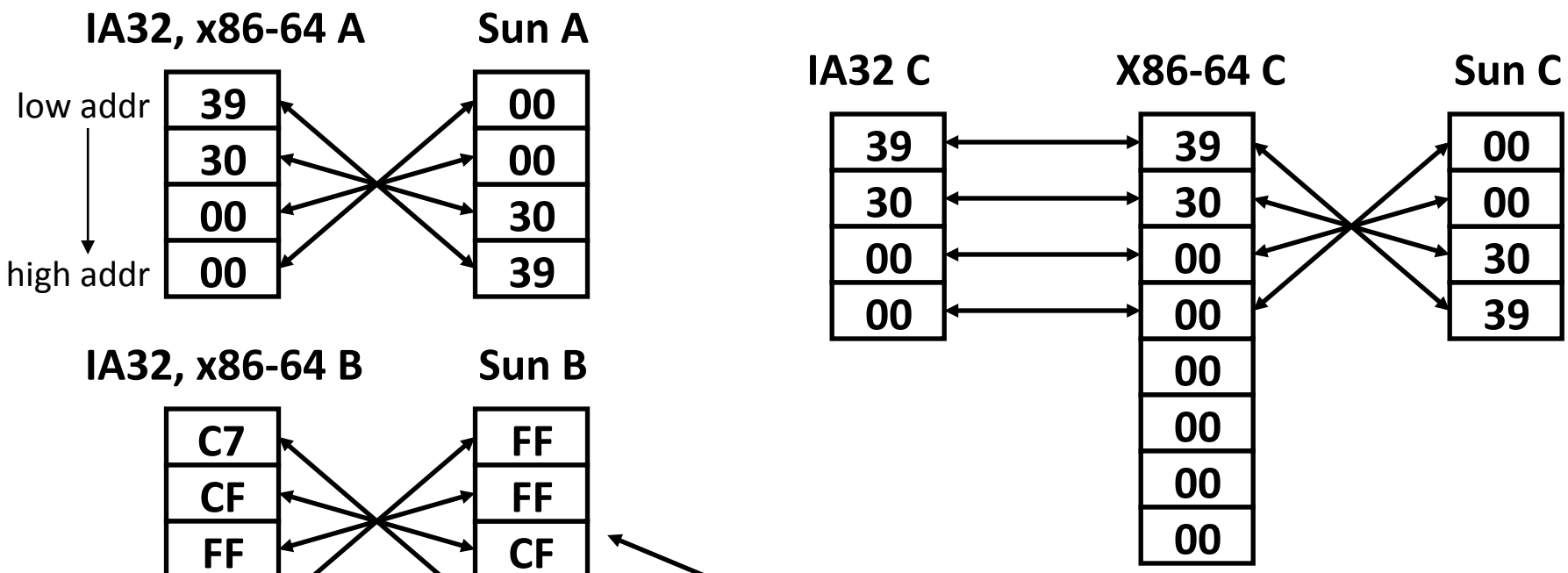
- **Big endian** (PowerPC, Sun, Internet)
  - Big end first: most-significant byte has lowest address
- **Little endian** (x86)
  - Little end first: least-significant byte has lowest address
- **Example**
  - Variable has 4-byte representation `0x01234567`
  - Address of variable is `0x100`



# Representing Integers

- `int A = 12345;`
- `int B = -12345;`
- `long int C = 12345;`

| | |
|---|---|
| **Decimal:** | 12345 |
| **Binary:** | 0011 0000 0011 1001 |
| **Hex:** | 3 0 3 9 -> 0x00003039 |




Two's complement representation for negative integers