Section 1: Memory, Data, and Addressing

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

Encoding Byte Values

- **Binary** 00000000_2 -- 11111111_2
  - Byte = 8 bits (binary digits)
  - Example: $00101011_2 = 32+8+2+1 = 43_{10}$
  - Example: $26_{10} = 16+8+2 = 00101010_2$
- **Decimal** 0_{10} -- 255_{10}
- **Hexadecimal** 00_{16} -- FF_{16}
  - Groups of 4 binary digits
  - Byte = 2 hexadecimal (hex) or base 16 digits
  - Base-16 number representation
  - Use characters '0' to '9' and 'A' to 'F' to represent
  - Write $FA1D37B_{16}$ in C code as a 4-byte value:

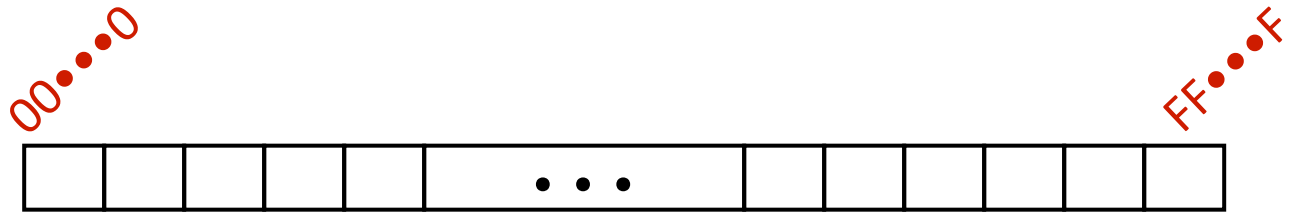
    `0xFA1D37B` or `0xfa1d37b`

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

How is memory organized?

- How do we find data in memory?

Byte-Oriented Memory Organization



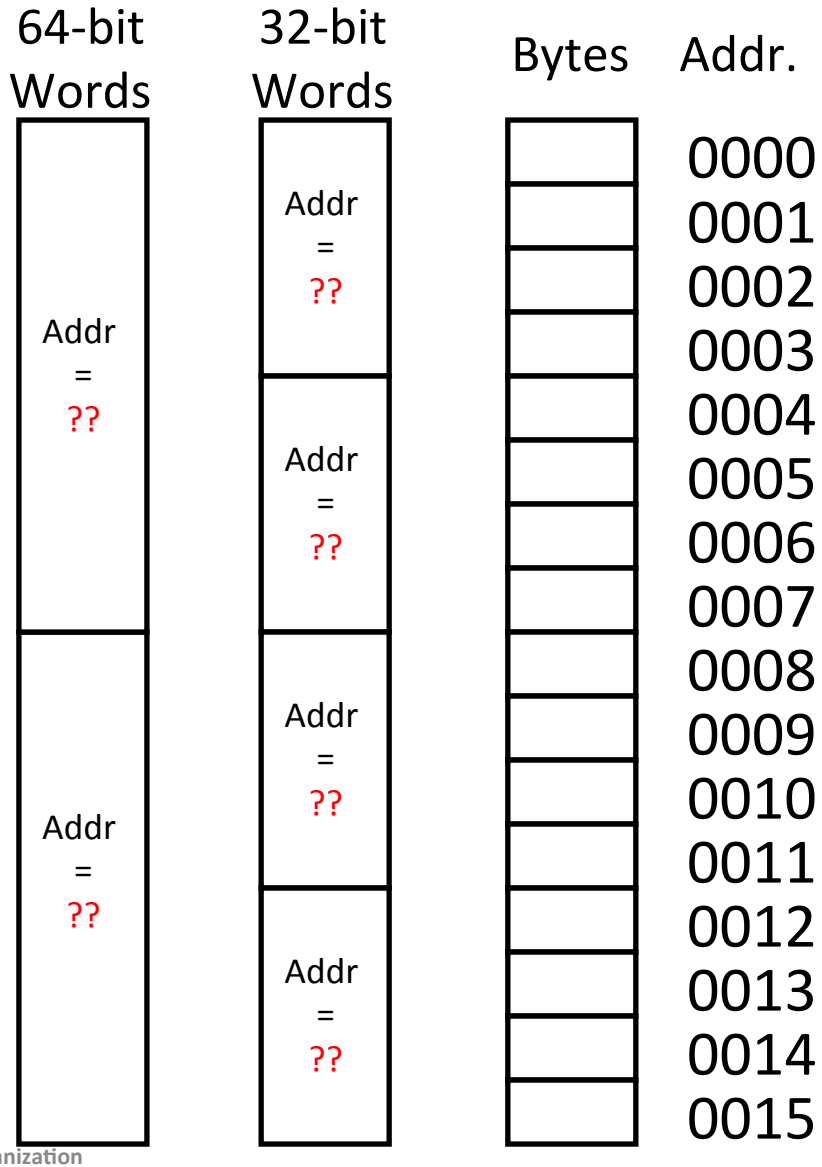
- **Programs refer to addresses**
  - Conceptually, a very large array of bytes, each with an *address* (index)
  - Operating system provides an address space private to each "process"
    - Process = program being executed + its data + its "state"
    - Program can modify its own data, but not that of others
    - Clobbering code or "state" often leads to crashes (or security holes)
- **Compiler + run-time system control memory allocation**
  - Where different program objects should be stored
  - All allocation within a single address space

Machine Words

- **Machine has a "word size"**
  - Nominal size of integer-valued data
    - *Including addresses*
  - Until recently, most machines used 32-bit (4-byte) words
    - Limits addresses to 4GB
    - Became too small for memory-intensive applications
  - Most current x86 systems use 64-bit (8-byte) words
    - Potential address space: $2^{64} \approx 1.8 \text{ X } 10^{19}$ bytes (18 EB – exabytes)
  - For backward-compatibility, many CPUs support different word sizes
    - Always a power-of-2 in the number of bytes: 1, 2, 4, 8, …

Word-Oriented Memory Organization

- **Addresses specify locations of bytes in memory**
  - Address of first byte in word
  - Addresses of successive words differ by 4 (32-bit) or 8 (64-bit)
  - Address of word 0, 1, .. 10?



Word-Oriented Memory Organization

- **Addresses specify locations of bytes in memory**
  - Address of first byte in word
  - Addresses of successive words differ by 4 (32-bit) or 8 (64-bit)
  - Address of word 0, 1, .. 10?