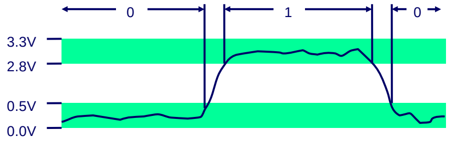
Lecture 8

Bit Representations

CPSC 275
Introduction to Computer Systems

Binary Representations

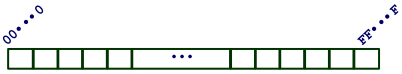


Encoding Byte Values

- Byte = 8 bits
  - Binary 00000000_2 to 11111111_2
  - Decimal: 0_{10} to 255_{10}
  - Hexadecimal 00_{16} to FF_{16}
    - Base 16 number representation
    - Use characters ‘0’ to ‘9’ and ‘A’ to ‘F’
    - Write $FA1D37B_{16}$ in C as
      0xFA1D37B or 0xfa1d37b

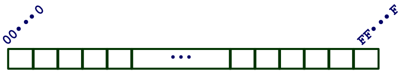
Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Byte-Oriented Memory Organization



- Programs Refer to *Virtual Addresses*
  - Conceptually very large array of bytes
  - Actually implemented with hierarchy of different memory types
  - System provides address space private to particular *process*
    - Program being executed
    - Process can access its own data, but not that of others

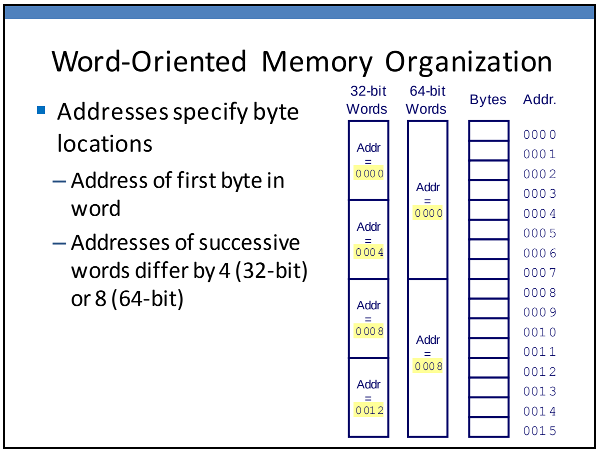
Memory Organization, cont’d



- Compiler + Run-Time Allocation
  - Where different program objects should be stored
  - All allocation within single virtual address space

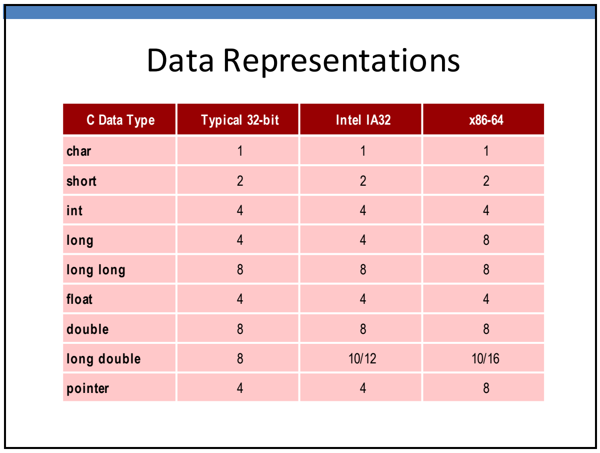
Machine Words

- Machine has *word* size
  - Nominal size of integer-valued data
    - Including addresses
  - Most older machines use 32 bits (4 bytes) words
    - Limits addresses to 4GB
    - Becoming too small for memory-intensive applications
  - Most current machines use 64 bits (8 bytes) words
    - Potential address space ≈ 1.8 X 10^{19} bytes
    - x86-64 machines support 48-bit addresses: 256 Terabytes



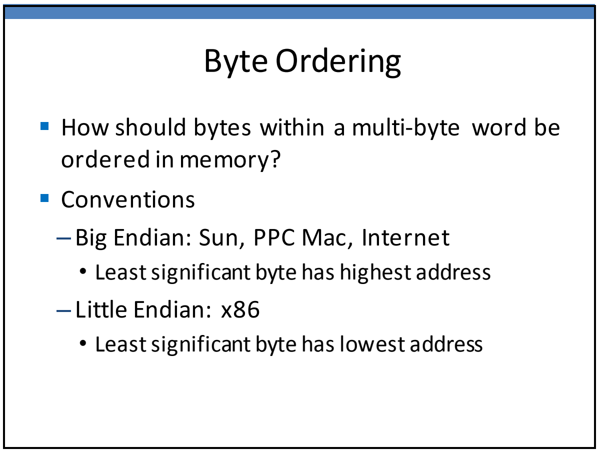
Word-Oriented Memory Organization

- Addresses specify byte locations
  - Address of first byte in word
  - Addresses of successive words differ by 4 (32-bit) or 8 (64-bit)



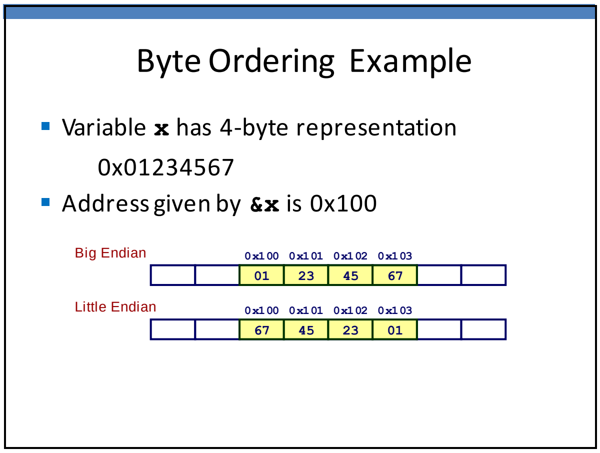
Data Representations

C Data Type	Typical 32-bit	Intel IA32	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	4	8
long long	8	8	8
float	4	4	4
double	8	8	8
long double	8	10/12	10/16
pointer	4	4	8



Byte Ordering

- How should bytes within a multi-byte word be ordered in memory?
- Conventions
  - Big Endian: Sun, PPC Mac, Internet
    - Least significant byte has highest address
  - Little Endian: x86
    - Least significant byte has lowest address



Byte Ordering Example

- Variable **x** has 4-byte representation

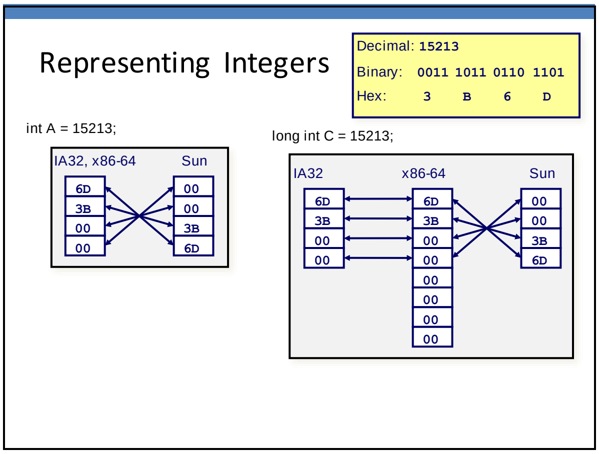
  0x01234567
- Address given by **&x** is 0x100

Big Endian

		0x100	0x101	0x102	0x103		
		01	23	45	67		

Little Endian

		0x100	0x101	0x102	0x103		
		67	45	23	01		

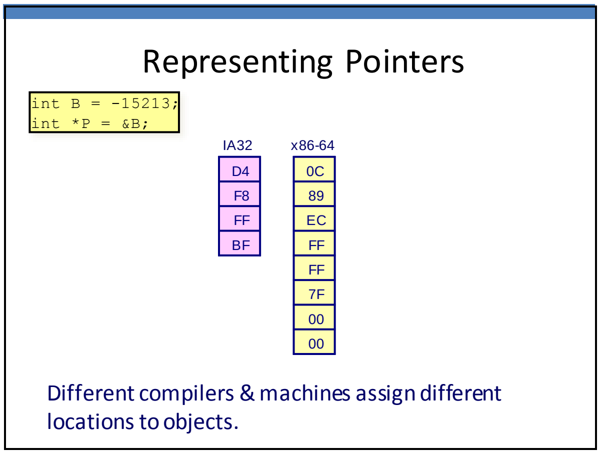


Representing Integers

Decimal: 15213
Binary: 0011 1011 0110 1101
Hex: 3 B 6 D

int A = 15213;

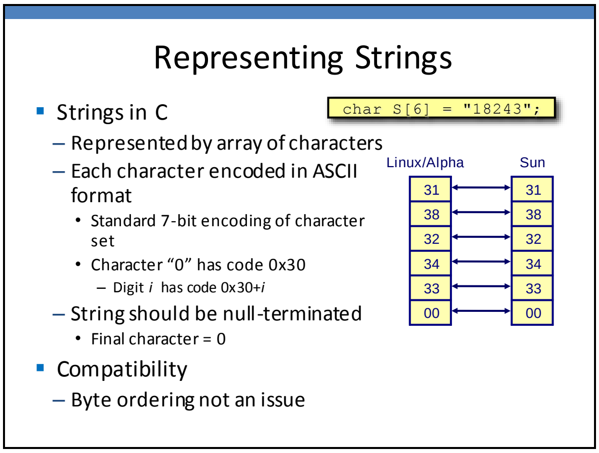
long int C = 15213;



Representing Pointers

```
int B = -15213;
int *P = &B;
```

Different compilers & machines assign different locations to objects.

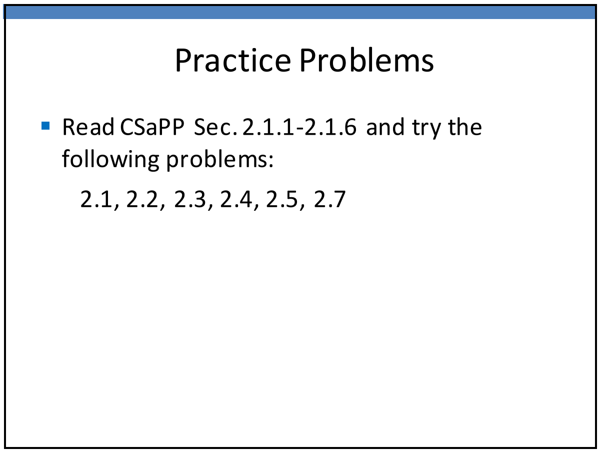


Representing Strings

- Strings in C
  - Represented by array of characters
  - Each character encoded in ASCII format
    - Standard 7-bit encoding of character set
    - Character “0” has code 0x30
      - Digit *i* has code 0x30+*i*
  - String should be null-terminated
    - Final character = 0
- Compatibility
  - Byte ordering not an issue

```
char S[6] = "18243";
```

Linux/Alpha	Sun
31	31
38	38
32	32
34	34
33	33
00	00



Practice Problems

- Read CSaPP Sec. 2.1.1-2.1.6 and try the following problems:

  2.1, 2.2, 2.3, 2.4, 2.5, 2.7