

# Representations in memory, pointers, strings

Computer Systems  
4th Lecture, Sep 11 2019

Troels Henriksen

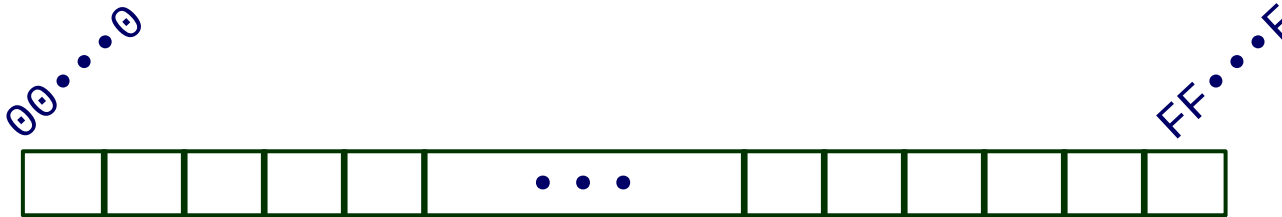
**Based on slides by:**

Randal E. Bryant and David R. O'Hallaron

# Today: Bits, Bytes, and Integers

- Representations in memory, pointers, strings

# Byte-Oriented Memory Organization



- **Programs refer to data by address**
  - Conceptually, envision it as a very large array of bytes
    - In reality, it's not, but can think of it that way
  - An address is like an index into that array
    - and, a pointer variable stores an address
  
- **Note: system provides private address spaces to each “process”**
  - Think of a process as a program being executed
  - So, a program can clobber its own data, but not that of others

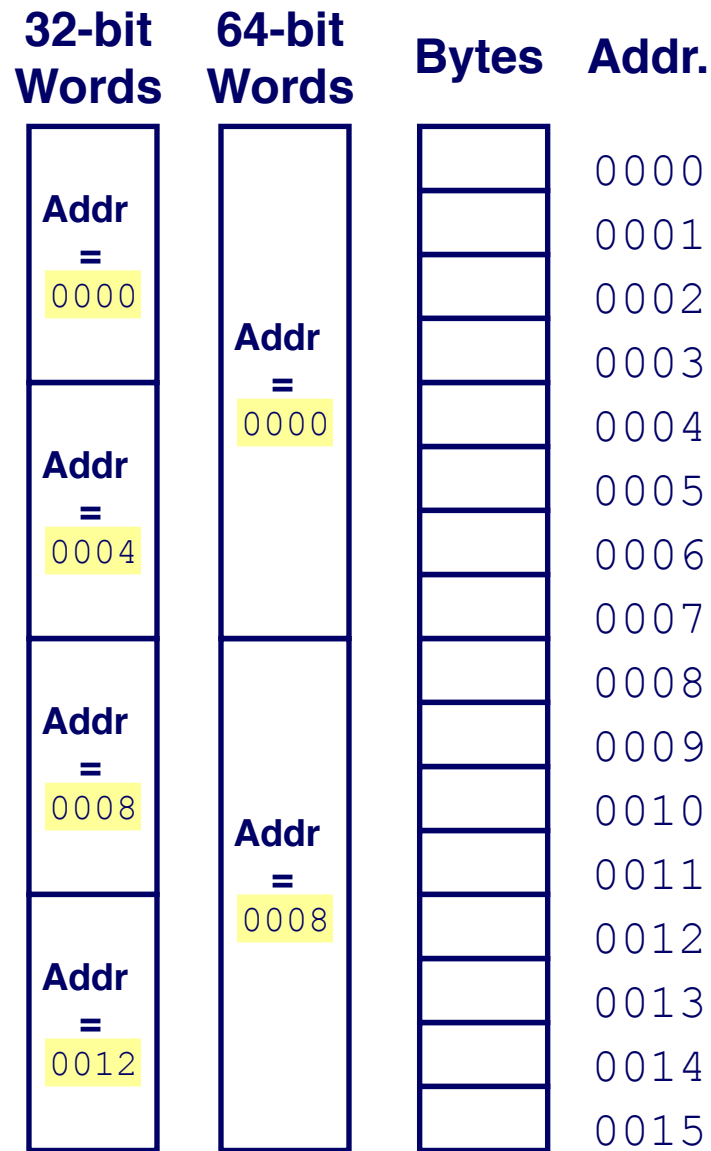
# Machine Words

- **Any given computer has a “Word Size”**
  - Nominal size of integer-valued data
    - and of addresses
  - Until recently, most machines used 32 bits (4 bytes) as word size
    - Limits addresses to 4GB ( $2^{32}$  bytes)
  - Increasingly, machines have 64-bit word size
    - Potentially, could have 18 EB (exabytes) of addressable memory
    - That's  $18.4 \times 10^{18}$
    - Current machines only use lower 48 bits.
  - Machines still support multiple data formats
    - Fractions or multiples of word size
    - Always integral number of bytes

# 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)



# Example Data Representations

C Data Type	Typical 32-bit	Typical 64-bit	x86-64
<code>char</code>	1	1	1
<code>short</code>	2	2	2
<code>int</code>	4	4	4
<code>long</code>	4	8	8
<code>float</code>	4	4	4
<code>double</code>	8	8	8
<code>long double</code>	–	–	10/16
<code>pointer</code>	4	8	8

# Byte Ordering

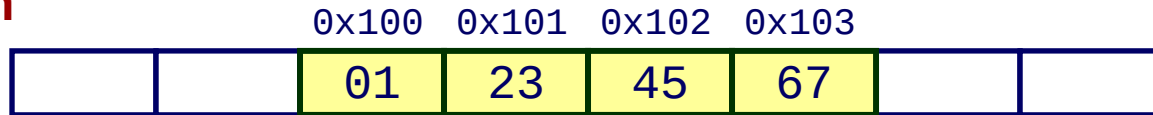
- So, how are the bytes within a multi-byte word ordered in memory?
- Conventions
  - Big Endian: Sun, PPC Mac, Internet
    - Least significant byte has highest address
  - Little Endian: x86, ARM processors running Android, iOS, and Windows
    - Least significant byte has lowest address

# Byte Ordering Example

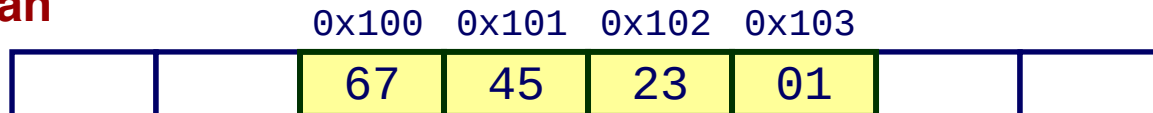
## ■ Example

- Variable x has 4-byte value of 0x01234567
- Address given by &x is 0x100

### Big Endian



### Little Endian





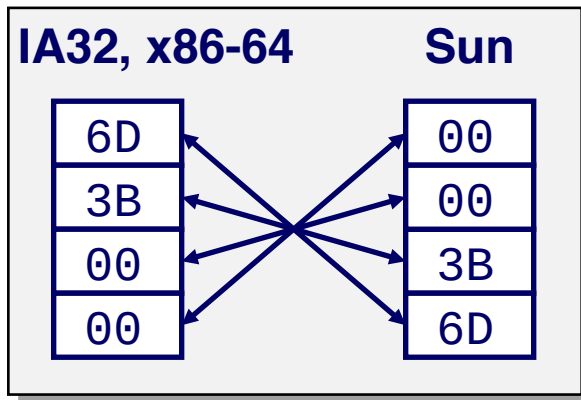
# Representing Integers

**Decimal:** 15213

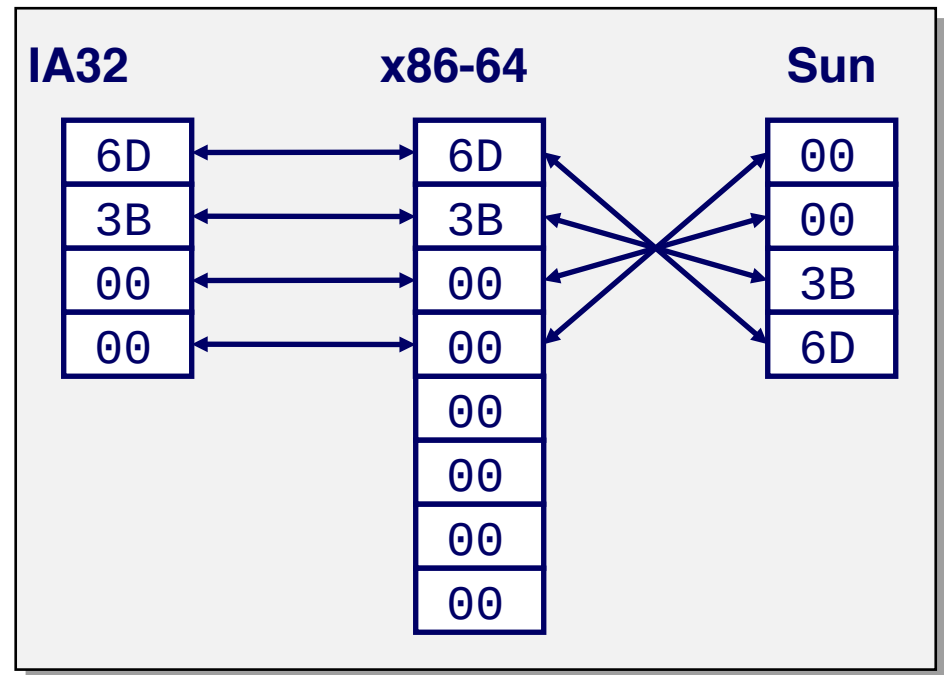
**Binary:** 0011 1011 0110 1101

**Hex:** 3 B 6 D

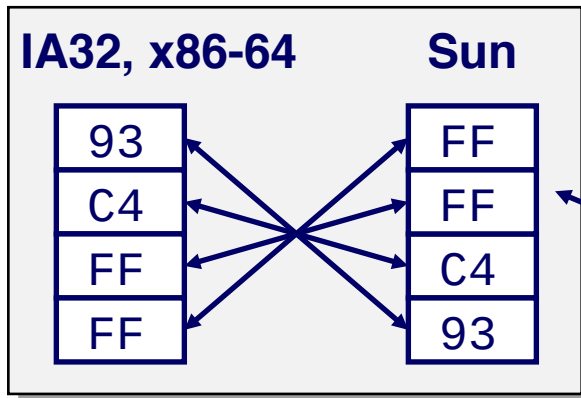
`int A = 15213;`



`long int C = 15213;`



`int B = -15213;`



Two's complement representation

# Examining Data Representations

## ■ Code to Print Byte Representation of Data

- Casting pointer to unsigned char \* allows treatment as a byte array

```
void show_bytes(unsigned char* start, size_t len) {  
    size_t i;  
    for (i = 0; i < len; i++)  
        printf("%p\t0x%.2x\n", start+i, start[i]);  
    printf("\n");  
}
```

### Printf directives:

%p:     Print pointer  
%x:     Print Hexadecimal

# show\_bytes Execution Example

```
int a = 15213;  
printf("int a = 15213;\n");  
show_bytes((pointer) &a, sizeof(int));
```

## Result (Linux x86-64):

```
int a = 15213;  
0x7fffb7f71dbc 6d  
0x7fffb7f71dbd 3b  
0x7fffb7f71dbe 00  
0x7fffb7f71dbf 00
```

# Representing Pointers

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

**Sun**

EF
FF
FB
2C

**IA32**

AC
28
F5
FF

**x86-64**

3C
1B
FE
82
FD
7F
00
00

Different compilers & machines assign different locations to objects.

Even get different results each time run program.

# Representing Strings

```
char S[6] = "18213";
```

## ■ 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

