Byte Ordering

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Memory Model

- **Physical memory**
  - DRAM chips can read/write 4, 8, 16 bits.
  - DRAM modules can read/write 64 bits.

- **Programmer’s view of memory**
  - Conceptually very large array of bytes
  - Stored-program computers: keeps program codes and data in memory.
  - Running programs share the physical memory
  - OS handles memory allocation and management
Machine Words

- Machine has “word size”
  - Nominal size of integer-valued data
    - Including addresses (= pointer size)
  - Most current machines use 32 bits (4 bytes) words
    - Limits addresses to 4GB
    - Becoming too small for memory-intensive applications
  - High-end systems use 64 bits (8 bytes) words
    - Potential address space ≈ 1.8 × 10^{19} bytes
    - x86-64 machines support 48-bit addresses: 256 Terabytes
  - Machines support multiple data formats
    - Fractions or multiples of word size
    - Always integral number of bytes
### Sizes of C Objects (in bytes)

<table>
<thead>
<tr>
<th>C Data Type</th>
<th>Typical 32-bit</th>
<th>Intel IA-32</th>
<th>x86-64</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>long long</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>long double</td>
<td>8</td>
<td>10/12</td>
<td>10/16</td>
</tr>
<tr>
<td>char *</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

or any other pointer
- Addresses specify byte locations
  - Address of first byte in word
  - Addresses of successive words differ by 4 (32-bit) or 8 (64-bit)
  - Usually, addresses should be aligned to the word boundary
Byte Ordering

- How should bytes within multi-byte word be ordered in memory?

- Conventions
  - **Big Endian**: Sun, PowerPC Mac, Internet
  - **Little Endian**: x86

- **Note**:
  - Alpha and PowerPC can run in either mode, with the byte ordering convention determined when the chip is powered up.
  - Problem when the binary data is communicated over a network between different machines.
Byte Ordering Example (1)

- **Big endian**
  - Least significant byte has highest address

- **Little endian**
  - Least significant byte has lowest address
### Byte Ordering Example (2)

- **Disassembly**
  - Text representation of binary machine code
  - Generated by program that reads the machine code

- **Example fragment**

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction Code</th>
<th>Assembly Rendition</th>
</tr>
</thead>
<tbody>
<tr>
<td>8048365:</td>
<td>5b</td>
<td>pop %ebx</td>
</tr>
<tr>
<td>8048366:</td>
<td>81 c3 ab 12 00 00</td>
<td>add $0x12ab,%ebx</td>
</tr>
<tr>
<td>804836c:</td>
<td>83 bb 28 00 00 00 00</td>
<td>cmpl $0x0,0x28(%ebx)</td>
</tr>
</tbody>
</table>

- **Deciphering numbers**
  - Value: 0x12ab
  - Pad to 32 bits: 0x000012ab
  - Split into bytes: 00 00 12 ab
  - Reverse: ab 12 00 00
What is the output of this program?

- Solaris/SPARC: ?
- Linux/x86: ?

```c
#include <stdio.h>

union {
    int i;
    unsigned char c[4];
} u;

int main () {
    u.i = 0x12345678;
    printf ("%x %x %x %x\n", u.c[0], u.c[1], u.c[2], u.c[3]);
}
```
Representing Integers

\[
\text{int } A = 15213; \\
\text{int } B = -15213; \\
\text{long int } C = 15213;
\]

<table>
<thead>
<tr>
<th>IA32, x86-64 A</th>
<th>Sun A</th>
<th>IA32 C</th>
<th>x86-64 C</th>
<th>Sun C</th>
</tr>
</thead>
<tbody>
<tr>
<td>6D</td>
<td>00</td>
<td>6D</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>3B</td>
<td>00</td>
<td>3B</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>00</td>
<td>6D</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IA32, x86-64 B</th>
<th>Sun B</th>
<th>IA32 C</th>
<th>x86-64 C</th>
<th>Sun C</th>
</tr>
</thead>
<tbody>
<tr>
<td>93</td>
<td>FF</td>
<td>6D</td>
<td>6D</td>
<td>00</td>
</tr>
<tr>
<td>C4</td>
<td>FF</td>
<td>3B</td>
<td>3B</td>
<td>00</td>
</tr>
<tr>
<td>FF</td>
<td>FF</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>FF</td>
<td>C4</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>FF</td>
<td>93</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

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

Two’s complement representation
Representing Pointers

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

<table>
<thead>
<tr>
<th>Sun P</th>
<th>IA32 P</th>
<th>x86-64 P</th>
</tr>
</thead>
<tbody>
<tr>
<td>EF</td>
<td>D4</td>
<td>0C</td>
</tr>
<tr>
<td>FF</td>
<td>F8</td>
<td>89</td>
</tr>
<tr>
<td>FB</td>
<td>FF</td>
<td>EC</td>
</tr>
<tr>
<td>2C</td>
<td>BF</td>
<td>FF</td>
</tr>
</tbody>
</table>

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 = 0x00

- **Compatibility**
  - Byte ordering not an issue

```c
char S[6] = "15213";
```
Main Points

- It’s all about bits & bytes
  - Numbers, programs, text, ...

- Different machines follow different conventions
  - Word size
  - Byte ordering
  - Representations (Integer, Floating-Point)

- When programming, be aware of
  - Type casting & mixed signed/unsigned expressions
  - Overflow
  - Error propagation
  - Byte ordering