When is byte ordering an issue?
1. Communications over a network between different machines
2. Representation of integer/real numeric data

3. Circumvention of normal type system
   - Using a “cast” to allow an object to be referenced according to a different data type from which it was created
     - Use and even necessary for system-level programming
   - Can cast such that the value is a sequence of bytes rather than an object of the original data type

```c
#include <stdio.h>
void main()
{
    int x = 0x12345678, i;
    unsigned char *xptr = &x;
    printf("the integer x is 0x%x\n",x);
    for (i = 0; i < 4; i++)
        printf("byte %d is %.2x\n",i+1, *(xptr+i));
}
```

(castex.c)
Systems can have different...

- word size
- byte sizes for each type
- endian-ness (for numeric values)
- representation of pointers
- character encoding schemes (ascii, ebcdic, unicode)
- instruction formats (again, just a sequence of bytes)
- ETC
Bit operations

1 and 3 \(\rightarrow\) exclusive OR (\(^\wedge\))
2 and 4 \(\rightarrow\) and (\(\&\))
5 \(\rightarrow\) or (\(|\))

1100 \hspace{0.5em} \text{carry}
0110 \hspace{0.5em} a
0111 \hspace{0.5em} b
1101 \hspace{0.5em} a+b
Boolean Algebra

- Boolean algebra has many of the same properties as arithmetic over integers
  - *+ and &|
    - Multiplication distributes over addition
      - $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
    - Boolean operation & distributes over |
      - $a \& (b \mid c) = (a \& b) \mid (a \& c)$
    - Boolean operation | distributes over &
      - $a \mid (b \& c) = (a \mid b) \& (a \mid c)$
    - CANNOT distribute addition over multiplication
      - $a + (b \cdot c) \neq (a + b) \cdot (a + c) \ldots$ for all integers

- Boolean ring – commonalities with integers
  - Every value has an additive inverse $-x$, such that $x + -x = 0$
  - $a^a = 0$ each element is its own additive inverse
    - $(a^b)^a = b$ above holds even in different ordering
    - Consider (swap):
      - $*y = *x \wedge *y$;
      - $*x = *x \wedge *y$;
      - $*y = *x \wedge *y$;
Bit-level operations in C

- Can be applied to any “integral” data type
  - One declared as type char or int
    - with or without qualifiers (ex. short, long, unsigned)

- How use?
  - Expand hex arguments to their binary representations
  - Perform binary operation
  - Convert back to hex

- NOTE: the expression ~0 will yield a mask of all ones, regardless of the word size of the machine; same as 0xFFFFFFFF for a 32-bit machine, but such code is not portable.

<table>
<thead>
<tr>
<th>value of x</th>
<th>machine rep</th>
<th>mask</th>
<th>type of x and mask</th>
<th>c expr</th>
<th>result</th>
<th>note</th>
</tr>
</thead>
<tbody>
<tr>
<td>153 (base 10)</td>
<td>0b10011001 == 0x99</td>
<td>0b10000000 == 0x80</td>
<td>char</td>
<td>x &amp; mask</td>
<td>0b10000000 == 0x80</td>
<td>2^7 = 128</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>mask &gt;&gt; 1</td>
<td>x &amp; mask</td>
<td>0b01000000 == 0x40</td>
<td>2^6 = 64 (etc)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0b01000000 == 0x40</td>
<td>x &amp; mask</td>
<td>0b00000000 == 0x00</td>
<td></td>
</tr>
<tr>
<td>153 (base 10)</td>
<td>0b10011001 == 0x99</td>
<td>0b10000000 == 0x80</td>
<td>int</td>
<td>x &amp; mask</td>
<td>0b10000000 == 0x80</td>
<td>same</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>x &lt;&lt; 1</td>
<td></td>
<td></td>
<td>???</td>
</tr>
</tbody>
</table>