Bitwise AND

The & operator

- takes two values (1, 2, 4, 8 bytes), treats as sequence of bits
- performs logical AND on each corresponding pair of bits
- result contains same number of bits as inputs

Example:

\[
\begin{array}{c|c}
00100111 & \text{AND} \\
\hline
& 11100011 \\
\hline
& \text{|} \\
& --- \\
\hline
& 00100011 \\
\end{array}
\]

Used for e.g. checking whether a bit is set
Exercise: Checking for odd numbers

One obvious way to check for odd numbers in C

```c
int isOdd(int n) {
    return n % 2 == 1;
}
```

Could we use & to achieve the same thing? How?

Aside: an alternative to the above

```c
int isOdd(int n) {
    return n & 1;
}
```
Bitwise OR

The | operator
- takes two values (1, 2, 4, 8 bytes), treats as sequence of bits
- performs logical OR on each corresponding pair of bits
- result contains same number of bits as inputs

Example:

<table>
<thead>
<tr>
<th>00100111</th>
<th>OR</th>
<th>0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>11100011</td>
<td>----</td>
<td>-----</td>
</tr>
<tr>
<td>--------</td>
<td>0 0</td>
<td>1 1</td>
</tr>
</tbody>
</table>

Used for e.g. ensuring that a bit is set
Bitwise NEG

The operator

- takes a single value (1, 2, 4, 8 bytes), treats as sequence of bits
- performs logical negation of each bit
- result contains same number of bits as input

Example:

<table>
<thead>
<tr>
<th>~ 00100111</th>
<th>NEG</th>
<th>0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>11011000</td>
<td></td>
<td>1 0</td>
</tr>
</tbody>
</table>

Used for e.g. creating useful bit patterns
Bitwise Operations in C

- everything is ultimately a string of bits
- e.g. unsigned char = 8-bit value
- e.g. literal bit-string 0b01110001
- e.g. literal hexadecimal 0x71
- & = bitwise AND
- | = bitwise OR
- = bitwise NEG
Bitwise XOR

The ^ operator

- takes two values (1,2,4,8 bytes), treats as sequence of bits
- performs logical XOR on each corresponding pair of bits
- result contains same number of bits as inputs

Example:

```
00100111 XOR | 0 1
^ 11100011 ----|-----
-------- 0 | 0 1
11000100 1 | 1 0
```

Used in e.g. generating hashes, graphic operation, cryptography
The `<<` operator

- takes a single value (1,2,4,8 bytes), treats as sequence of bits
- and a small positive integer $x$
- moves (shifts) each bit $x$ positions to the left
- left-end bit vanishes; right-end bit replaced by zero
- result contains same number of bits as input

Example:

<table>
<thead>
<tr>
<th></th>
<th><code>00100111 &lt;&lt; 2</code></th>
<th><code>00100111 &lt;&lt; 8</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0010011100</td>
<td>0000000000</td>
</tr>
</tbody>
</table>
Right Shift

The $\gg$ operator
- takes a single value (1, 2, 4, 8 bytes), treats as sequence of bits
- and a small positive integer $x$
- moves (shifts) each bit $x$ positions to the right
- right-end bit vanishes; left-end bit replaced by zero**
- result contains same number of bits as input

Example:

```
00100111 >> 2 00100111 >> 8
-------- --------
00001001 00000000
```

Beware: shifts involving negative values are not portable (implementation defined) - use unsigned values to be safe/portable.
Exercise: Bitwise Operations

Given the following variable declarations:

// a signed 8-bit value
unsigned char x = 0x55;
unsigned char y = 0xAA;

What is the value of each of the following expressions:

• \( (x \& y) \) \( (x \oplus y) \)
• \( (x \ll 1) \) \( (y \ll 1) \)
• \( (x \gg 1) \) \( (y \gg 1) \)
Exercise: Bit-manipulation

Assuming 8-bit quantities and writing answers as 8-bit bit-strings: What are the values of the following:

- 25, 65, 0, 1, 0xFF, 0xFF
- \((01010101 \& 10101010)\), \((01010101 \mid 10101010)\)
- \((x \& x)\), \((x \mid x)\)

How can we achieve each of the following:

- ensure that the 3rd bit from the RHS is set to 1
- ensure that the 3rd bit from the RHS is set to 0