Integer Encodings
Integers

- C integer data types
- Unsigned encoding
- Signed encodings
- Conversions
# Integer data types in C99

<table>
<thead>
<tr>
<th>C data type</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Storage Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>[signed] char</td>
<td>-127</td>
<td>127</td>
<td>1 byte</td>
</tr>
<tr>
<td>unsigned char</td>
<td>0</td>
<td>255</td>
<td>1 byte</td>
</tr>
<tr>
<td>short</td>
<td>-32,767</td>
<td>32,767</td>
<td>2 bytes</td>
</tr>
<tr>
<td>unsigned short</td>
<td>0</td>
<td>65,535</td>
<td>2 bytes</td>
</tr>
<tr>
<td>int</td>
<td>-32,767</td>
<td>32,767</td>
<td>2 bytes</td>
</tr>
<tr>
<td>unsigned int</td>
<td>0</td>
<td>65,535</td>
<td>2 bytes</td>
</tr>
<tr>
<td>long</td>
<td>-2,147,483,647</td>
<td>2,147,483,647</td>
<td>4 bytes</td>
</tr>
<tr>
<td>unsigned long</td>
<td>0</td>
<td>4,294,967,295</td>
<td>4 bytes</td>
</tr>
<tr>
<td>int32_t</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
<td>4 bytes</td>
</tr>
<tr>
<td>uint32_t</td>
<td>0</td>
<td>4,294,967,295</td>
<td>4 bytes</td>
</tr>
<tr>
<td>int64_t</td>
<td>-9,223,372,036,854,775,808</td>
<td>9,223,372,036,854,775,807</td>
<td>8 bytes</td>
</tr>
<tr>
<td>uint64_t</td>
<td>0</td>
<td>18,446,744,073,709,551,615</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2.11** Guaranteed ranges for C integral data types. The C standards require that the data types have at least these ranges of values.
Integer data types on stu

All sizes in bytes; sizes in red are larger than mandated by C99

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
</tr>
<tr>
<td>unsigned char</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
</tr>
<tr>
<td>unsigned short</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>unsigned int</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
</tr>
<tr>
<td>unsigned long</td>
<td>8</td>
</tr>
<tr>
<td>long long</td>
<td>8</td>
</tr>
<tr>
<td>unsigned long long</td>
<td>8</td>
</tr>
</tbody>
</table>

char 1
unsigned char 1
short 2
unsigned short 2
int 4
unsigned int 4
long 8
unsigned long 8
long long 8
unsigned long long 8

int8_t 1
uint8_t 1
bool 1
int16_t 2
uint16_t 2
int32_t 4
uint32_t 4
int64_t 8
uint64_t 8
size_t 8
Unsigned integer encoding

- Bit \( i \) represents the value \( 2^i \)
  - Bits typically written from most to least significant (i.e., \( 2^3 2^2 2^1 2^0 \))
  - This is the same encoding we saw last time!
  - No representation of negative numbers

\[
\begin{align*}
1 &= 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = [0001] \\
5 &= 4 + 1 = 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = [0101] \\
11 &= 8 + 2 + 1 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = [1011] \\
15 &= 8 + 4 + 2 + 1 = 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = [1111]
\end{align*}
\]
Unsigned integer encoding

- Textbook’s notation
  - Each bar represents a bit
  - Add together bars to represent the contributions of each bit value to the overall value

**Figure 2.12**
Unsigned number examples for \( w = 4 \).
When bit \( i \) in the binary representation has value 1, it contributes \( 2^i \) to the value.
Signed integer encodings

- Sign magnitude
  - Most natural/intuitive but hardest to implement
- Ones’ complement
  - Cleaner arithmetic but less intuitive
- Two’s complement
  - Cleanest arithmetic but most complicated
  - Most modern signed integer types use this!
Sign magnitude

- **Sign magnitude**
  - Interpret most-significant bit as a **sign bit**
  - Interpret remaining bits as unsigned number $x$ (the **magnitude**)
    - If negative, absolute value is $x$
  - To negate: **flip the sign bit**
- **Disadvantages**:
  - Two zeros: -0 and +0 [1000 and 0000]
  - Less useful for arithmetic because the sign bit has no relationship with the magnitude--cannot use unsigned arithmetic logic!

\[
\begin{array}{ll}
0011 & = 3 \\
1011 & = -3 \\
0111 & = 7 \\
\end{array}
\]

\[
\begin{array}{ll}
0111 & (7) \\
1011 & (-3) \\
? & 010
\end{array}
\]
• What is the negation of 10110 in sign magnitude?
  - A) 10110
  - B) 10111
  - C) 01001
  - D) 01011
  - E) 01010
  - F) 00110
Question

Which of the following are negative numbers if interpreted as a sign magnitude integer?

- A) 10110
- B) 10111
- C) 01001
- D) 01011
- E) 01010
- F) 00110
Ones’ complement

- Interpret most-significant bit as a sign bit
- Interpret ALL bits as unsigned integer $x$
  - If negative, absolute value is $[1111\ldots1] - x$
- To negate: **flip all the bits** (binary NOT)
- Disadvantages:
  - Still have two representations of zero (1111 and 0000)
  - Also, less useful for arithmetic than two’s complement
    - Must “end-around carry” to preserve results

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 011</td>
<td>3</td>
<td>0 111</td>
<td>7</td>
</tr>
<tr>
<td>1 100</td>
<td>-3</td>
<td>1 100</td>
<td>-3</td>
</tr>
<tr>
<td>0 100</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$10 011 + 1 \text{ (end-around carry) } 0 100$
Question

What is the negation of 10110 in ones’ complement?

- A) 10110
- B) 10111
- C) 01001
- D) 01011
- E) 01010
- F) 00110
Question

Which of the following are negative numbers if interpreted as a ones’ complement integer?

- A) 10110
- B) 10111
- C) 01001
- D) 01011
- E) 01010
- F) 00110
Two’s complement

- Interpret most-significant bit as a **sign bit**
- Interpret ALL bits as unsigned integer \( x \)

  - If negative, absolute value is \( 2^N - x \) where \( N \) is the number of bits
  - To negate: **subtract value from** \( 2^N \) where \( N \) is the number of bits
  - One zero; positive numbers wrap to negative ones halfway through

<table>
<thead>
<tr>
<th>2's Comp.</th>
<th>Unsigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>1111</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>-7</td>
<td>1001</td>
</tr>
<tr>
<td>-8</td>
<td>1000</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
</tr>
</tbody>
</table>
Two’s complement advantage: uses unsigned arithmetic logic
- (ignore carries out of the sign bit for now)
- Ex: $5 - 3 = 5 + (-3) = 0101 + 1101 = 0010$ (2)
- Ex: $1 - 3 = 1 + (-3) = 0001 + 1101 = 1110$ (-2)
- Ex: $-2 - 3 = (-2) + (-3) = 1110 + 1101 = 1011$ (-5)

$0011 = 3$
$1100 = -3$
$0111 = 7$
$1101 = -3$
$0100 = 4$
$0111 = 7$
Two’s complement

- Alternate interpretation: value of most significant bit is negated
  - i.e., start at most negative number and build back up towards zero

**Figure 2.12**
Unsigned number examples for $w = 4$.
When bit $i$ in the binary representation has value 1, it contributes $2^i$ to the value.

**Figure 2.16**
Comparing unsigned and two's-complement representations for $w = 4$.
The weight of the most significant bit is $-8$ for two's complement and $+8$ for unsigned, yielding a net difference of 16.
Two’s complement trick

- Alternate way to negate in two’s complement
  - **Flip the bits** (binary NOT) then **add one**

Ex: 5 = 0101 → (binary NOT) → 1010 → (add one) → 1011 = -5 (-8 + 2 + 1)

*Aside*: Why does this work? The sum of a number x and ~x is all ones (or $2^N-1$ where N is the number of bits), so ~x can be expressed as $2^N-1 - x$. Because negating x in two’s complement is equivalent to subtracting x from $2^N$, if we add one to ~x the results are equal:

$$~x + 1 = (2^N-1 - x) + 1 = 2^N - x$$
What is the negation of 10110 in two’s complement?
- A) 10110
- B) 10111
- C) 01001
- D) 01011
- E) 01010
- F) 00110
Question

Which of the following are negative numbers if interpreted as a two’s complement integer?

- A) 10110
- B) 10111
- C) 01001
- D) 01011
- E) 01010
- F) 00110
Ones’ vs. Two’s

- **Ones’ complement**
  - Interpret all bits as unsigned integer $x$
    - Value is $[1111\ldots1] - x$
    - I.e., the complement with respect to ones

- **Two’s complement**
  - Interpret all bits as unsigned integer $x$
    - Value is $2^N - x$ where $N$ is the number of bits
    - I.e., the complement with respect to a power of two
Ones’ complement and two’s complement are both an operation and an encoding.

- E.g., “perform two’s complement” vs “the number is stored in two’s complement”

The operation represents the action necessary to negate a number in that encoding.

- E.g., performing two’s complement (ones’ complement and add one) negates a number in two’s complement encoding

If you have a value in a particular encoding:

- If the sign bit is not set, it’s a positive number
- If it is set, perform the operation to recover the positive value

We will avoid using the operation terminology in this course!
Integer encodings

• Information = Bits + Context
  – What does “1011” mean?  It depends!

  Unsigned:  11
  Sign magnitude:  -3
  Ones' complement:  -4
  Two's complement:  -5
We’ll see one more signed integer encoding next week: “offset binary” / “biased” / “excess”

- For now, here’s a comparison (for 1-byte integers):

<table>
<thead>
<tr>
<th>Binary</th>
<th>Unsigned</th>
<th>Sign Mag</th>
<th>Ones’ C</th>
<th>Two’s C</th>
<th>Offset -127</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111 1111</td>
<td>255</td>
<td>-127</td>
<td>-0</td>
<td>-1</td>
<td>128</td>
</tr>
<tr>
<td>1111 1111</td>
<td>254</td>
<td>-126</td>
<td>-1</td>
<td>-2</td>
<td>127</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1000 0001</td>
<td>129</td>
<td>-1</td>
<td>-126</td>
<td>-127</td>
<td>2</td>
</tr>
<tr>
<td>1000 0000</td>
<td>128</td>
<td>-0</td>
<td>-127</td>
<td>-128</td>
<td>1</td>
</tr>
<tr>
<td>0111 1111</td>
<td>127</td>
<td>127</td>
<td>127</td>
<td>127</td>
<td>0</td>
</tr>
<tr>
<td>0111 1110</td>
<td>126</td>
<td>126</td>
<td>126</td>
<td>126</td>
<td>-1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0000 0001</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-126</td>
</tr>
<tr>
<td>0000 0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-127</td>
</tr>
</tbody>
</table>
• Which of the following are guaranteed to be “safe” (i.e., the value will always be preserved)?
  - A) Smaller unsigned → larger unsigned
  - B) Smaller two’s comp. → larger two’s comp.
  - C) Larger → smaller (unsigned or two’s comp.)
  - D) Unsigned → two’s comp.
  - E) Two’s comp. → unsigned
Conversions

- Smaller unsigned → larger unsigned
  - Safe; zero-extend to preserve value
  \[
  \text{0101 (5)} \rightarrow \text{0000 0101 (5)}
  \]

- Smaller two’s comp. → larger two’s comp.
  - Safe; sign-extend to preserve value
  \[
  \text{1101 (-3)} \rightarrow \text{1111 1101 (-3)}
  \]

- Larger → smaller (unsigned or two’s comp.)
  - Overflow if new type isn’t large enough to fit (truncate)
  \[
  \text{0000 0101 (5)} \rightarrow \text{0101 (5)}
  \text{0011 0101 (53)} \rightarrow \text{0101 (5)}
  \]

- Unsigned → two’s comp.
  - Overflow if first bit is non-zero (otherwise, no change)
  \[
  \text{0101 (5)} \rightarrow \text{0101 (5)}
  \text{1101 (13)} \rightarrow \text{1101 (-3)}
  \]

- Two’s comp. → unsigned
  - Overflow if value is negative (otherwise, no change)
  \[
  \text{0101 (5)} \rightarrow \text{0101 (5)}
  \text{1101 (-2)} \rightarrow \text{1101 (13)}
  \]