CMPT 295
Unit - Data Representation
Lectures 3 and 4 – Representing integers in memory – Unsigned and signed
Last Lecture

- Von Neumann architecture
  - Its components: CPU, memory, bus, input and output peripherals
  - One of its characteristics: Data and code stored in memory

- A look at bits in memory
  - Reason for using 0's and 1's (two values) when representing information
  - Word size (w): size of bit vectors we manipulate, also size of machine words (see Section 2.1.2)
  - Algorithm for converting binary to hexadecimal (hex)?
    1. Partition bit vector into groups of 4 bits, starting from the least significant byte (LSB) on the right.
    2. When reach the most significant byte (MSB) on the left, pad it left-hand side with 0's, if necessary.
    3. Translate each group of 4 bits into its hex value.
  - What do bits represent? Encoding schemes give meaning to bits.

- Bit manipulation – regardless of what bit vectors represent
  - Boolean algebra: bitwise operations - and (&), or (|), xor (^), not (~)
  - Shift operations: left shift, right logical shift and right arithmetic shift
    - Logical shift: Fill with 0’s on the left
    - Arithmetic shift: Replicate most significant bit (i.e., sign bit) on the left

C logical operators and C bitwise (bit-level) operators behave differently! Watch out for && versus & , || versus | . ...
Today’s Menu

- Representing information as bits
  - “Under the Hood” - A look at memory
    - Bits in memory
    - Encoding scheme
    - Endian
  - Bit manipulation
    - Boolean algebra + Shifting
- Representing integers in memory
  - Unsigned and signed
  - Converting, expanding and truncating
  - Arithmetic operations
- Representing real numbers in memory
  - IEEE floating point representation
  - Floating point in C – casting, rounding, addition, …
What do bits represent?

Question: What could these bits represent?

Answer: Unsigned integer

\( w = 8 \)

Let’s try: \( M[0x0000] = \)

- **Range:**
- **For any** \( w \), **range of unsigned values:**

**Conclusion:**

\[
B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i
\]
Positional notation: expand and sum all terms

Example: \( 246_{10} = 2 \times 10^2 + 4 \times 10^1 + 6 \times 10^0 \)

\[
\sum_{i=0}^{w-1} x_i \cdot 2^i
\]

\( B2U(X) \) Conversion (Encoding Scheme)
U2B(X) Conversion

**Method 1 - Using subtraction:**
- Subtracting decreasing power of 2 until reach 0

\[
\begin{align*}
246 &\Rightarrow 246 - 128 = 118 \quad \Rightarrow 128 = 2^7 \\
118 &- 64 = 54 \quad \Rightarrow 64 = 2^6 \\
54 &- 32 = 22 \quad \Rightarrow 32 = 2^5 \\
22 &- 16 = 6 \quad \Rightarrow 16 = 2^4 \\
6 &- 4 = 2 \quad \Rightarrow 4 = 2^2 \\
2 &- 2 = 0 \quad \Rightarrow 2 = 2^1 
\end{align*}
\]

\[
246 \Rightarrow 11110110_2
\]

**Method 2 - Using division:**
- Dividing by 2 until reach 0

\[
\begin{align*}
w & = 8 \\
246 \Rightarrow 
\end{align*}
\]
**U2B(X) Conversion – A few tricks**

- **Decimal -> binary**
  - Trick: When decimal number is $2^n$, then its binary representation is 1 followed by n zero’s
  - Let’s try:

- **Decimal -> hex**
  - Trick: $n = i + 4j$, where $0 \leq i \leq 3$, so $x = 2^i$ followed by $j$ zero’s
  - Let try:
What do bits represent?

Question: What could these bits represent?

Answer: Signed integer

Let’s try: \( M[0\times0002] = \)

- Range:
- For any \( w \), range of unsigned values:

\[
B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i
\]

- Same conclusion as unsigned encoding
Examples of “Show your work”

**T2B(X) Conversion -> Two’s Complement**

**Method 1**
If \( x < 0 \),
\[
\text{Method 1: } (\neg\text{U2B}(\lvert x \rvert)) + 1
\]

**Method 2**
If \( x < 0 \),
\[
\text{Method 2: } \text{U2B}(x + 2^w)
\]

9
Properties of unsigned & signed conversions

- **Equivalence**
  - Both encoding scheme produce same bit patterns for nonnegative values

- **Uniqueness**
  - Every bit pattern produced represents unique integer value
  - Each representable integer has unique bit pattern

<table>
<thead>
<tr>
<th>$X$</th>
<th>$B2U(X)$</th>
<th>$B2T(X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>-8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>-7</td>
</tr>
<tr>
<td>1010</td>
<td>10</td>
<td>-6</td>
</tr>
<tr>
<td>1011</td>
<td>11</td>
<td>-5</td>
</tr>
<tr>
<td>1100</td>
<td>12</td>
<td>-4</td>
</tr>
<tr>
<td>1101</td>
<td>13</td>
<td>-3</td>
</tr>
<tr>
<td>1110</td>
<td>14</td>
<td>-2</td>
</tr>
<tr>
<td>1111</td>
<td>15</td>
<td>-1</td>
</tr>
</tbody>
</table>
Converting between signed & unsigned

- Mappings between unsigned and signed numbers: keep same bit pattern and reinterpret it
Converting signed $\leftrightarrow$ unsigned with $w = 4$

$T2U(X)$  
$U2T(X)$
Visualizing relationship between signed & unsigned

 Signed (2’s Complement) Range

 Unsigned Range

 TMax

 TMin
Sign extension

- Converting two unsigned (or signed) of different size
- Sign extension
  - Small data type -> larger
    - Unsigned: zero extension
    - Signed: sign bit extension
- Let’s try:
Truncation

- Converting two unsigned (or signed) of different size
- Truncation
  - Large data type -> smaller
  - May alter original value
    - A form of overflow
- Let’s try:
Summary

- Interpretation of bit patterns into either unsigned or signed values
  - $B2U(X)$ and $U2B(X)$ encoding schemes
- Conversions from unsigned <-> signed values - $B2T(X)$ and $T2B(X)$
- Signed value expressed as two’s complement
- Implication in C: when converting (implicitly and explicitly via casting):
  - Sign:
    - Unsigned <-> signed (of same size) -> Bit pattern is maintained, but reinterpreted
    - Can have unexpected effects: adding or subtracting $2^w$
  - Size:
    - Small -> large (both signed or both unsigned - e.g., short to int)
      - sign extension: Unsigned -> zeros extension, signed -> sign bit extension
      - Both yield expected result
    - Large -> small (e.g., unsigned to unsigned short)
      - truncation: Unsigned/signed -> most significant bits are truncated
      - May alter original value
- Both (sign and size): 1) size then 2) sign
Next Lecture

- Representing information as bits
  - “Under the Hood” - A look at memory
    - Bits in memory
    - Encoding scheme
    - Endian
  - Bit manipulation
    - Boolean algebra + Shifting
- Representing integers in memory
  - Unsigned and signed
  - Converting, expanding and truncating
  - Arithmetic operations
- Representing real numbers in memory
  - IEEE floating point representation
  - Floating point in C – casting, rounding, addition, …