15-213
“The course that gives CMU its Zip!”

Integer Representations
Sep 5, 2000

Topics
  • Numeric Encodings
    – Unsigned & Two’s complement
  • Programming Implications
    – C promotion rules
# Notation

**$W$: Number of Bits in “Word”**

<table>
<thead>
<tr>
<th>C Data Type</th>
<th>Typical 32-bit</th>
<th>Alpha</th>
</tr>
</thead>
<tbody>
<tr>
<td>long int</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>int</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>short</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>char</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

**Integers**
- Lower case
- E.g., $x, y, z$

**Bit Vectors**
- Upper Case
- E.g., $X, Y, Z$
- Write individual bits as integers with value 0 or 1
- E.g., $X = x_{w-1}, x_{w-2}, \ldots, x_0$
  - Most significant bit on left
Encoding Integers

Unsigned

\[ B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i \]

Two’s Complement

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

- C short 2 bytes long

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>15213</td>
<td>3B 6D 00111011 01101101</td>
</tr>
<tr>
<td>y</td>
<td>-15213</td>
<td>C4 93 11000100 10010011</td>
</tr>
</tbody>
</table>

Sign Bit

- For 2’s complement, most significant bit indicates sign
  - 0 for nonnegative
  - 1 for negative
Encoding Example (Cont.)

\[
\begin{align*}
  x &= 15213: 00111011 \ 01101101 \\
  y &= -15213: 11000100 \ 10010011
\end{align*}
\]

<table>
<thead>
<tr>
<th>Weight</th>
<th>15213</th>
<th>-15213</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>64</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>128</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>256</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>512</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1024</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2048</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4096</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8192</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>16384</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-32768</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sum</th>
<th>15213</th>
<th>-15213</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15213</td>
<td>-15213</td>
</tr>
</tbody>
</table>
Other Encoding Schemes

Other less common encodings

- One’s complement: Invert bits for negative numbers
- Sign magnitude: Invert sign bit for negative numbers

short int examples

<table>
<thead>
<tr>
<th></th>
<th>Unsigned</th>
<th>00111011 01101101</th>
</tr>
</thead>
<tbody>
<tr>
<td>15213</td>
<td>Two’s complement</td>
<td>11000100 10010011</td>
</tr>
<tr>
<td>-15213</td>
<td>One’s complement</td>
<td>11000100 10010010</td>
</tr>
<tr>
<td>-15213</td>
<td>Sign magnitude</td>
<td>10111011 01101101</td>
</tr>
</tbody>
</table>

- ISO C does not define what encoding machines use for signed integers, but 99% (or more) use two’s complement.
- For truly portable code, don’t count on it.
Numeric Ranges

Unsigned Values

• $UMin = 0$
  \[000\ldots0\]

• $UMax = 2^w - 1$
  \[111\ldots1\]

Two’s Complement Values

• $TMin = -2^{w-1}$
  \[100\ldots0\]

• $TMax = 2^{w-1} - 1$
  \[011\ldots1\]

Other Values

• Minus 1
  \[111\ldots1\]

Values for $W = 16$

<table>
<thead>
<tr>
<th></th>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMax</td>
<td>65535</td>
<td>FF FF</td>
<td>11111111 11111111</td>
</tr>
<tr>
<td>Tmax</td>
<td>32767</td>
<td>7F FF</td>
<td>01111111 11111111</td>
</tr>
<tr>
<td>Tmin</td>
<td>-32768</td>
<td>80 00</td>
<td>10000000 00000000</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>FF FF</td>
<td>11111111 11111111</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>00 00</td>
<td>00000000 00000000</td>
</tr>
</tbody>
</table>
## Values for Different Word Sizes

<table>
<thead>
<tr>
<th>W</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMax</td>
<td>255</td>
<td>65,535</td>
<td>4,294,967,295</td>
<td>18,446,744,073,709,551,615</td>
</tr>
<tr>
<td>Tmax</td>
<td>127</td>
<td>32,767</td>
<td>2,147,483,647</td>
<td>9,223,372,036,854,775,807</td>
</tr>
<tr>
<td>Tmin</td>
<td>-128</td>
<td>-32,768</td>
<td>-2,147,483,648</td>
<td>-9,223,372,036,854,775,808</td>
</tr>
</tbody>
</table>

### Observations

- \(|TMin| = TMax + 1\)
  - Asymmetric range
- \(UMax = 2 \times TMax + 1\)

### C Programming

- \#include <limits.h>
  - K&R, App. B11
- Declares constants, e.g.,
  - ULONG_MAX
  - LONG_MAX
  - LONG_MIN
- Values platform-specific
Unsigned & Signed Numeric Values

Example Values
- \( W = 4 \)

Equivalence
- Same encodings for nonnegative values

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

⇒ Can Invert Mappings
- \( U2B(x) = B2U^{-1}(x) \)
  - Bit pattern for unsigned integer
- \( T2B(x) = B2T^{-1}(x) \)
  - Bit pattern for two’s comp integer

<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>
Casting Signed to Unsigned

C Allows Conversions from Signed to Unsigned

```c
short int x = 15213;
unsigned short int ux = (unsigned short) x;
short int y = -15213;
unsigned short int uy = (unsigned short) y;
```

Resulting Value

- No change in bit representation
- Nonnegative values unchanged
  - \( ux = 15213 \)
- Negative values change into (large) positive values
  - \( uy = 50323 \)
Relation Between 2’s Comp. & Unsigned

Two’s Complement → T2U → B2U → Unsigned

Maintain Same Bit Pattern

\[ u_x = \begin{cases} x & x \geq 0 \\ x + 2^w & x < 0 \end{cases} \]

\[
\begin{align*}
\quad & \begin{array}{cccccccccccc}
\_ & \_ & \_ & \_ & + & + & + & & & & \\
\_ & \_ & \_ & \_ & + & + & + & & & & \\
\end{array} \\
\quad & \begin{array}{cccccccccccc}
\_ & \_ & \_ & \_ & + & + & + & & & & \\
\_ & \_ & \_ & \_ & + & + & + & & & & \\
\end{array} \\
\quad & u_x \quad 0 \\
\quad & \_ \quad + \quad + \quad \_ \quad \_ \quad \_ \quad + \quad + \quad + \\
\quad & \_ \quad + \quad + \quad \_ \quad \_ \quad \_ \quad + \quad + \quad + \\
\end{align*}
\]

\[ +2^{w-1} - 2^{w-1} = 2 \times 2^{w-1} = 2^w \]
Relation Between Signed & Unsigned

\[ u_y = y + 2 \times 32768 = y + 65536 \]
From Two’s Complement to Unsigned

• $T2U(x)$
  
  $= B2U(T2B(x))$

  $= x + x_{w-1} 2^w$

• What you get in C:
  
  ```c
  unsigned t2u(int x) {
    return (unsigned) x;
  }
  ```

<table>
<thead>
<tr>
<th>$X$</th>
<th>$B2U(X)$</th>
<th>$B2T(X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1010</td>
<td>10</td>
<td>-6</td>
</tr>
</tbody>
</table>

$+16$
From Unsigned to Two’s Complement

- \( U2T(x) \)
  \[ = B2T(U2B(x)) \]
  \[ = x - x_{w-1} 2^w \]

- What you get in C:
  ```
  int u2t(unsigned x) {
    return (int) x;
  }
  ```

<table>
<thead>
<tr>
<th>( X )</th>
<th>B2U(( X ))</th>
<th>B2T(( X ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1010</td>
<td>10</td>
<td>-6</td>
</tr>
</tbody>
</table>

\[ \text{Identity} \]
Signed vs. Unsigned in C

Constants
• By default are considered to be signed integers
• Unsigned if have “U” as suffix
  0U, 4294967259U

Casting
• Explicit casting between signed & unsigned same as U2T and T2U
  ```
  int tx, ty;
  unsigned ux, uy;
  tx = (int) ux;
  uy = (unsigned) ty;
  ```
• Implicit casting also occurs via assignments and procedure calls
  ```
  tx = ux;
  uy = ty;
  ```
**Casting Surprises**

**Expression Evaluation**

- If mix unsigned and signed in single expression, signed values implicitly cast to unsigned
- Including comparison operations $\lt$, $\gt$, $\equiv$, $\leq$, $\geq$
- Examples for $W = 32$

<table>
<thead>
<tr>
<th>Constant$_1$</th>
<th>Constant$_2$</th>
<th>Relation</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0U</td>
<td>$\equiv$</td>
<td>unsigned</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>$&lt;$</td>
<td>signed</td>
</tr>
<tr>
<td>-1</td>
<td>0U</td>
<td>$&gt;$</td>
<td>unsigned</td>
</tr>
<tr>
<td>2147483647</td>
<td>-2147483648</td>
<td>$&gt;$</td>
<td>signed</td>
</tr>
<tr>
<td>2147483647U</td>
<td>-2147483648</td>
<td>$&lt;$</td>
<td>unsigned</td>
</tr>
<tr>
<td>-1</td>
<td>-2</td>
<td>$&gt;$</td>
<td>signed</td>
</tr>
<tr>
<td>(unsigned) -1</td>
<td>-2</td>
<td>$&gt;$</td>
<td>unsigned</td>
</tr>
<tr>
<td>2147483647</td>
<td>2147483648U</td>
<td>$&lt;$</td>
<td>unsigned</td>
</tr>
<tr>
<td>2147483647</td>
<td>(int) 2147483648U</td>
<td>$&gt;$</td>
<td>signed</td>
</tr>
</tbody>
</table>
Explanation of Casting Surprises

2’s Comp. → Unsigned

• Ordering Inversion
• Negative → Big Positive

2’s Comp. Range

$T_{Max}$

$0$

$-1$

$-2$

$T_{Min}$

$UMax$

$UMax - 1$

$T_{Max} + 1$

$T_{Max}$

$0$

Unsigned Range
Sign Extension

Task:
• Given \( w \)-bit signed integer \( x \)
• Convert it to \( w+k \)-bit integer with same value

Rule:
• Make \( k \) copies of sign bit:
• \( X' = \underbrace{x_{w-1}, \ldots, x_{w-1}}_{k \text{ copies of MSB}}, x_{w-2}, \ldots, x_0 \)
Sign Extension Example

short int x = 15213;
int ix = (int) x;
short int y = -15213;
int iy = (int) y;

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>15213</td>
<td>3B 6D 00111011 01101101</td>
</tr>
<tr>
<td>ix</td>
<td>15213</td>
<td>00 00 C4 92 00000000 00000000 00111011 01101101</td>
</tr>
<tr>
<td>y</td>
<td>-15213</td>
<td>C4 93 11000100 10010011</td>
</tr>
<tr>
<td>iy</td>
<td>-15213</td>
<td>FF FF C4 93 11111111 11111111 11000100 10010011</td>
</tr>
</tbody>
</table>

- Converting from smaller to larger integer data type
- C automatically performs sign extension
Justification For Sign Extension

Prove Correctness by Induction on $k$

- Induction Step: extending by single bit maintains value

Key observation:

$-2^{w-1} = -2^w + 2^{w-1}$

Look at weight of upper bits:

$X = -2^{w-1} x_{w-1}$

$X' = -2^w x_{w-1} + 2^{w-1} x_{w-1} = -2^{w-1} x_{w-1}$
Casting Order Dependencies

```c
short int x = 15213;
short int y = -15213;
unsigned iux = (unsigned)(unsigned short) x;
unsigned iuy = (unsigned)(unsigned short) y;
unsigned uix = (unsigned)(int) x;
unsigned uiy = (unsigned)(int) y;
unsigned uuy = y;
```

![Diagram of casting order dependencies]

```
iux = 15213: 00000000 00000000 00111011 01101101
iuy = 50323: 00000000 00000000 11000100 10010011
uix = 15213: 00000000 00000000 00111011 01101101
uiy = 4294952083: 11111111 11111111 11000100 10010011
uuy = 4294952083: 11111111 11111111 11000100 10010011
```
Why Should I Use Unsigned?

Don’t Use Just Because Number Nonzero

- C compiler on Alpha generates less efficient code
  - Comparable code on Intel/Linux
    ```c
    unsigned i;
    for (i = 1; i < cnt; i++)
      a[i] += a[i-1];
    ```
- Easy to make mistakes
  ```c
  for (i = cnt-2; i >= 0; i--)
    a[i] += a[i+1];
  ```

Do Use When Performing Modular Arithmetic

- Multiprecision arithmetic
- Other esoteric stuff

Do Use When Need Extra Bit’s Worth of Range

- Working right up to limit of word size
Negating with Complement & Increment

In C
\[ \sim x + 1 \equiv -x \]

Complement
- Observation: \[ \sim x + x \equiv 1111...11_2 \equiv -1 \]

\[
\begin{array}{c}
x \quad 100111101 \\
+ \quad \sim x \quad 011000100 \\
\hline
-1 \quad 111111111
\end{array}
\]

Increment
- \[ \sim x + x + (\sim x + 1) \equiv -1 + (\sim x + 1) \]
- \[ \sim x + 1 \equiv -x \]

Warning: Be cautious treating \texttt{int}'s as integers
- OK here
## Comp. & Incr. Examples

\( x = 15213 \)

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>15213</td>
<td>3B 6D 00111011 01101101</td>
</tr>
<tr>
<td>( \sim x )</td>
<td>-15214</td>
<td>C4 92 11000100 10010010</td>
</tr>
<tr>
<td>( \sim x + 1 )</td>
<td>-15213</td>
<td>C4 93 11000100 10010011</td>
</tr>
<tr>
<td>( y )</td>
<td>-15213</td>
<td>C4 93 11000100 10010011</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>00 00 00000000 00000000</td>
</tr>
<tr>
<td>( \sim 0 )</td>
<td>-1</td>
<td>FF FF 11111111 11111111</td>
</tr>
<tr>
<td>( \sim 0 + 1 )</td>
<td>0</td>
<td>00 00 00000000 00000000</td>
</tr>
</tbody>
</table>