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

Integer Representations
Aug. 31, 1999

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.)

\[
x = 15213: \quad 00111011 \quad 01101101 \\
y = -15213: \quad 11000100 \quad 10010011
\]

<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>

Sum 15213 -15213
Other Encoding Schemes

Other less common encodings

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

<table>
<thead>
<tr>
<th>Short int</th>
<th>Encoding type</th>
<th>Binary representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>15213</td>
<td>Unsigned</td>
<td>00111011 01101101</td>
</tr>
<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 95% (or more) use two’s complement.

For truly portable code, don’t count on it.
### Numeric Ranges

**Unsigned Values**
- $U_{\text{Min}} = 0$
  - $000...0$
- $U_{\text{Max}} = 2^w - 1$
  - $111...1$

**Two's Complement Values**
- $T_{\text{Min}} = -2^{w-1}$
  - $100...0$
- $T_{\text{Max}} = 2^{w-1} - 1$
  - $011...1$

**Other Values**
- Minus 1
  - $111...1$

### 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
- $|T_{min}| = T_{max} + 1$
  - Asymmetric range
- $U_{max} = 2 \times T_{max} + 1$

C Programming
- `#include <limits.h>`
  - Harbison and Steele, 5.1
- 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
Casting Signed to Unsigned

C Allows Conversions from Signed to Unsigned

```
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

Maintain Same Bit Pattern

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

\[
\begin{array}{cccccccccccc}
& & & & & & & & & & & \\
& & & & & & & & & & & \\
& & & & & & & & & & & \\
& & & & & & & & & & & \\
& & & & & & & & & & & \\
& & & & & & & & & & & \\
& & & & & & & & & & & \\
& & & & & & & & & & & \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
+ & + & + & & & & & & & & & \\
+ & + & + & & & & & & & & & \\
+ & + & + & & & & & & & & & \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
 & & & & & & & & & & & \\
- & + & + & & & & & & & & & \\
- & + & + & & & & & & & & & \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
+2^{w-1} & -2^{w-1} &= 2*2^{w-1} &= 2^w
\end{array}
\]
Relation Between Signed & Unsigned

\[ u_y = y + 2 \times 32768 = y + 65536 \]

<table>
<thead>
<tr>
<th>Weight</th>
<th>-15213</th>
<th>50323</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>64</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>128</td>
<td>1</td>
<td>128</td>
</tr>
<tr>
<td>256</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>512</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1024</td>
<td>1</td>
<td>1024</td>
</tr>
<tr>
<td>2048</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4096</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8192</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16384</td>
<td>1</td>
<td>16384</td>
</tr>
<tr>
<td>32768</td>
<td>1</td>
<td>32768</td>
</tr>
</tbody>
</table>

Sum: -15213 50323
From Two’s Complement to Unsigned

- T2U(x)
  = B2U(T2B(x))
  = x + x_{w-1} 2^w

- What you get in 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>

Identity
From Unsigned to Two’s Complement

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

• What you get in C:
  ```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>

-16
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 \(<, >, ==, <=, >=\)
• 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>0(\text{U})</td>
<td>==</td>
<td>unsigned</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>&lt;</td>
<td>signed</td>
</tr>
<tr>
<td>-1</td>
<td>0(\text{U})</td>
<td>&gt;</td>
<td>unsigned</td>
</tr>
<tr>
<td>2147483647</td>
<td>-2147483648</td>
<td>&gt;</td>
<td>signed</td>
</tr>
<tr>
<td>2147483647(\text{U})</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>2147483648(\text{U})</td>
<td>&lt;</td>
<td>unsigned</td>
</tr>
<tr>
<td>2147483647</td>
<td>(int) 2147483648(\text{U})</td>
<td>&gt;</td>
<td>signed</td>
</tr>
</tbody>
</table>
Explanations of Casting Surprises

2’s Comp. → Unsigned

- Ordering Inversion
- Negative → Big Positive
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:
  - $X' = x_{w-1}, \ldots, x_{w-1}, x_{w-2}, \ldots, x_0$
- $k$ copies of MSB
Sign Extension Example

```c
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:

$X \quad -2^{w-1} x_{w-1}$

$X' \quad -2^w x_{w-1} + 2^{w-1} x_{w-1} = -2^{w-1} x_{w-1}$
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
uiy = 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