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

 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 + ... + x_0
  – Most significant bit on left

Encoding Integers

Unsigned

Two’s Complement

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

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

short int x = 15213;
short int y = -15213;

• C short 2 bytes long

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>15213</td>
<td>3B 6D</td>
<td>00111011 01101101</td>
</tr>
<tr>
<td>-15213</td>
<td>C4 93</td>
<td>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: 00111011 01101101
y = -15213: 11000100 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 0</td>
<td>1 2</td>
</tr>
<tr>
<td>4</td>
<td>1 4</td>
<td>0 0</td>
</tr>
<tr>
<td>8</td>
<td>1 8</td>
<td>0 0</td>
</tr>
<tr>
<td>16</td>
<td>0 0</td>
<td>1 16</td>
</tr>
<tr>
<td>32</td>
<td>1 32</td>
<td>0 0</td>
</tr>
<tr>
<td>64</td>
<td>1 64</td>
<td>0 0</td>
</tr>
<tr>
<td>128</td>
<td>0 0</td>
<td>1 128</td>
</tr>
<tr>
<td>256</td>
<td>1 256</td>
<td>0 0</td>
</tr>
<tr>
<td>512</td>
<td>1 512</td>
<td>0 0</td>
</tr>
<tr>
<td>1024</td>
<td>0 0</td>
<td>1 1024</td>
</tr>
<tr>
<td>2048</td>
<td>1 2048</td>
<td>0 0</td>
</tr>
<tr>
<td>4096</td>
<td>1 4096</td>
<td>0 0</td>
</tr>
<tr>
<td>8192</td>
<td>1 8192</td>
<td>0 0</td>
</tr>
<tr>
<td>16384</td>
<td>0 0</td>
<td>1 16384</td>
</tr>
<tr>
<td>-32768</td>
<td>0 0</td>
<td>-1 32768</td>
</tr>
</tbody>
</table>

Sum 15213 -15213

class03.ppt
CS 213 F’00
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>Number</th>
<th>Signed</th>
<th>Unsigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>15213</td>
<td>-15213</td>
<td>00110110 01101101</td>
</tr>
<tr>
<td>15213</td>
<td>-15213</td>
<td>11000100 10010011</td>
</tr>
<tr>
<td>15213</td>
<td>-15213</td>
<td>11000100 10010010</td>
</tr>
<tr>
<td>15213</td>
<td>-15213</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
- UMax = $2^{w} - 1$

Two’s Complement Values

- TMin = $-2^{w-1}$
- TMax = $2^{w-1} - 1$

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>

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 ('x')
  - Bit pattern for unsigned integer
- T2B(x) = B2T ('x')
  - Bit pattern for two’s comp integer

Observations

- $|TMin| = TMax + 1$
  - Asymmetric range
- $UMax = 2 \cdot TMax + 1$

C Programming

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

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 ('x')
  - Bit pattern for unsigned integer
- T2B(x) = B2T ('x')
  - Bit pattern for two’s comp integer
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 Signed & Unsigned

<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>-32768</td>
<td>32768</td>
</tr>
</tbody>
</table>

- $uy = y + 2 ^ 16 = 65536$

Relation Between 2's Comp. & Unsigned

Two's Complement

$$ux = \begin{cases} x & x \geq 0 \\ x + 2^w & x < 0 \end{cases}$$

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;
}
```
From Unsigned to Two’s Complement

- \( \text{U2T}(x) = \text{B2T}(\text{U2B}(x)) = x - 2^{w-1} \cdot x \downarrow_w \)

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

<table>
<thead>
<tr>
<th>( x )</th>
<th>( \text{B2U}(x) )</th>
<th>( \text{B2T}(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1010</td>
<td>10</td>
<td>-6</td>
</tr>
</tbody>
</table>

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 \( \text{U2T} \) and \( \text{T2U} \)
  ```c
  int tx, ty;
  unsigned ux, uy;
  tx = (int) ux;
  uy = (unsigned) ty;
  ```
- Implicit casting also occurs via assignments and procedure calls
  ```c
  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>0U</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>0U</td>
<td>&gt;</td>
<td>unsigned</td>
</tr>
<tr>
<td>2147483647</td>
<td>-2147483648</td>
<td>&lt;</td>
<td>signed</td>
</tr>
<tr>
<td>2147483647U</td>
<td>-2147483648</td>
<td>&gt;</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. \( \rightarrow \) Unsigned
- Ordering Inversion
- Negative \( \rightarrow \) 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:
  \[ X' = x_{w-1}, \ldots, x_{w-1}, x_{w-1}, x_{w-2}, \ldots, x_0 \]
  \( k \) copies of MSB

Justification For Sign Extension

Prove Correctness by Induction on \( k \)
• Induction Step: extending by single bit maintains value

Casting Order Dependencies

Sign Extension Example

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>15213</td>
<td>00111011 01101101</td>
</tr>
<tr>
<td>( ix )</td>
<td>15213</td>
<td>00000000 00000000 00111011 01101101</td>
</tr>
<tr>
<td>( y )</td>
<td>-15213</td>
<td>11000100 10010011</td>
</tr>
<tr>
<td>( iy )</td>
<td>-15213</td>
<td>11111111 11111111 11000100 10010011</td>
</tr>
</tbody>
</table>

- Converting from smaller to larger integer data type
- C automatically performs sign extension
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

```c
~x + 1 == -x
```

Complement

• Observation: \( \neg x + x = 1\overline{1}\overline{1}\overline{1} \ldots = -1 \)
  ```c
  x 1 0 0 1 1 0 1 1
  + ~x 0 1 0 0 0 1 0
  ---------
  -1 1 1 1 1 1 1 1
  ```

Increment

• \( \neg x + x + (\neg x + 1) = x + (\neg x + \neg x) \)
• \( \neg x + 1 = -x \)

Warning: Be cautious treating int’s as integers

• OK here

Comp. & Incr. Examples

\( x = 15213 \)

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

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