Representing and Manipulating Integers
Part I

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Introduction

- The advent of the digital age
  - Analog vs. digital?
  - Compact Disc (CD)
    - 44.1 KHz, 16-bit, 2-channel
  - MP3
    - A digital audio encoding with lossy data compression
**Representing Information**

- **Information = Bits + Context**
  - Computers manipulate representations of things.
  - Things are represented as binary digits.
  - What can you represent with N bits?
    - \(2^N\) things
    - Numbers, characters, pixels, positions, source code, executable files, machine instructions, ...
    - Depends on what operations you do on them.

<table>
<thead>
<tr>
<th>(char)</th>
<th>‘s’</th>
<th>‘k’</th>
<th>‘k’</th>
<th>‘u’</th>
<th>‘s’</th>
<th>‘e’</th>
<th>‘m’</th>
<th>‘i’</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>01110011 01101011 01101011 01110101 01110011 01100101 01101101 01101001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| (int) | 1969974131 | 1768777075 |

| (double) | 7.031689903291708178... x 10^{199} |
Binary Representations

- Why not base 10 representation?
  - Easy to store with bistable elements
  - Straightforward implementation of arithmetic functions
  - Reliably transmitted on noisy and inaccurate wires

- Electronic implementation
Encoding Byte Values

- **Byte = 8 bits**
  - **Binary:** $00000000_2$ to $11111111_2$
  - **Octal:** $000_8$ to $377_8$
    - An integer constant that begins with $0$ is an octal number in C
  - **Decimal:** $0_{10}$ to $255_{10}$
    - First digit must not be $0$ in C
  - **Hexadecimal:** $00_{16}$ to $FF_{16}$
    - Base 16 number representation
    - Use characters ‘$0$’ to ‘$9$’ and ‘$A$’ to ‘$F$’
    - Write $FA1D37B_{16}$ in C as $\texttt{0xFA1D37B}$
      or $\texttt{0xfa1d37b}$
**Boolean Algebra (1)**

- **Developed by George Boole in 1849**
  - Algebraic representation of logic
    - Encode “True” as 1 and “False” as 0

- **And**
  - $A \& B = 1$ when both $A=1$ and $B=1$
    - \[
    \begin{array}{c|cc}
    & 0 & 1 \\
    \hline
    0 & 0 & 0 \\
    1 & 0 & 1 \\
    \end{array}
    \]

- **Not**
  - $\sim A = 1$ when $A=0$
    - \[
    \begin{array}{c|c}
    \sim & 0 \\
    \hline
    0 & 1 \\
    1 & 0 \\
    \end{array}
    \]

- **Or**
  - $A|B = 1$ when either $A=1$ or $B=1$
    - \[
    \begin{array}{c|cc}
    & 0 & 1 \\
    \hline
    0 & 0 & 1 \\
    1 & 1 & 1 \\
    \end{array}
    \]

- **Exclusive-Or (Xor)**
  - $A^B = 1$ when either $A=1$ or $B=1$, but not both
    - \[
    \begin{array}{c|cc}
    ^ & 0 & 1 \\
    \hline
    0 & 0 & 1 \\
    1 & 1 & 0 \\
    \end{array}
    \]
### Boolean Algebra (2)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Constant 0

- $X \& Y$; AND
- $\neg (X \rightarrow Y)$
- $\neg (Y \rightarrow X)$
- $X \oplus Y$; XOR
- $X \mid Y$; OR
- $\neg (X \mid Y)$; NOR
- $\neg (X \oplus Y)$; X-NOR
- $\neg Y$
- $Y \rightarrow X$
- $\neg X$
- $X \rightarrow Y$
- $\neg (X \& Y)$; NAND
- Constant 1

#### Constant 1

- $X \rightarrow Y = \neg X \mid Y$

**Basic operations:** AND(&), OR(|), NOT(~)

- $X \oplus Y = (X \& \neg Y) \mid (\neg X \& Y)$
- $X \rightarrow Y = \neg X \mid Y$

**A complete set:** NAND = $\neg (X \& Y)$

![Diagram of Boolean functions](attachment:image.png)
Unsigned Integers

- Encoding unsigned integers

\[ B = [b_{w-1}, b_{w-2}, \ldots, b_0] \]

\[ D(B) = \sum_{i=0}^{w-1} b_i \cdot 2^i \]

\[ x = 0000\ 0111\ 1101\ 0011_2 \]

\[ D(x) = 2^{10} + 2^9 + 2^8 + 2^7 + 2^6 + 2^4 + 2^1 + 2^0 \]

\[ = 1024 + 512 + 256 + 128 + 64 + 16 + 2 + 1 \]

\[ = 2003 \]

- What is the range for unsigned values with \( w \) bits?
Signed Integers (1)

- Encoding positive numbers
  - Same as unsigned numbers

- Encoding negative numbers
  - Sign-magnitude representation
  - Ones’ complement representation
  - Two’s complement representation
Signed Integers (2)

- **Sign-magnitude representation**

  \[ S(B) = (-1)^{b_{w-1}} \cdot \left( \sum_{i=0}^{w-2} b_i \cdot 2^i \right) \]

  - **Two zeros**
    - [000...00], [100...00]
  - **Used for floating-point numbers**
Signed Integers (3)

- Ones’ complement representation

\[ O(B) = -b_{w-1}(2^{w-1} - 1) + \left( \sum_{i=0}^{w-2} b_i \cdot 2^i \right) \]

- Easy to find \(-n\)
- Two zeros
  - [000...00], [111...11]
- No longer used
Signed Integers (4)

- Two’s complement representation

\[ O(B) = -b_{w-1} \cdot 2^{w-1} + \left( \sum_{i=0}^{w-2} b_i \cdot 2^i \right) \]

- Unique zero
- Easy for hardware
  - leading 0 ≥ 0
  - leading 1 < 0
- Used by almost all modern machines
Signed Integers (5)

- **Two’s complement representation (cont’d)**
  - Following holds for two’s complement
    \[
    \sim x + 1 = -x
    \]
  - Complement
    - Observation: \(\sim x + x = 1111\ldots11_2 = -1\)
  - Increment
    \[
    \sim x + x = -1
    \sim x + x + (\sim x + 1) = -1 + (\sim x + 1)
    \sim x + 1 = -x
    \]
Numeric Ranges (1)

- **Unsigned values**
  - **UMin** = 0 [000...00]
  - **UMax** = $2^w - 1$ [111...11]

- **Two’s complement values**
  - **TMin** = $-2^{w-1}$ [100...00]
  - **TMax** = $2^{w-1} - 1$ [011...11]

### Values for w = 16

<table>
<thead>
<tr>
<th></th>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UMax</strong></td>
<td>65535</td>
<td>FF FF</td>
<td>11111111 11111111</td>
</tr>
<tr>
<td><strong>TMax</strong></td>
<td>32767</td>
<td>7F FF</td>
<td>01111111 11111111</td>
</tr>
<tr>
<td><strong>TMin</strong></td>
<td>-32768</td>
<td>80 00</td>
<td>10000000 00000000</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>00 00</td>
<td>00000000 00000000</td>
</tr>
</tbody>
</table>
### Numeric Ranges (2)

#### Values for different word sizes

<table>
<thead>
<tr>
<th></th>
<th>w = 8</th>
<th>w = 16</th>
<th>w = 32</th>
<th>w = 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
- $|\text{TMin}| = \text{TMax} + 1$ (Asymmetric range)
- $\text{UMax} = 2 \times \text{TMax} + 1$

#### In C programming
- `#include <limits.h>`
- `INT_MIN, INT_MAX, LONG_MIN, LONG_MAX, UINT_MAX, ...`
- Values platform-specific
Type Conversion (1)

- **Unsigned**: \( w \) bits \( \rightarrow \) \( w+k \) bits
  - Zero extension: just fill \( k \) bits with 0’s

```
unsigned short x = 2003;
unsigned ix = (unsigned) x;
```

![Diagram showing type conversion](image)
Type Conversion (2)

- **Signed:** $w$ bits $\rightarrow w+k$ bits
  - Given $w$-bit signed integer $x$
  - Convert it to $w+k$-bit integer with same value

- **Sign extension**
  - Make $k$ copies of sign bit
### Sign extension example

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

```c
short int x = 2003;
int ix = (int) x;
short int y = -2003;
int iy = (int) y;
```

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>2003</td>
<td>07 D3 00000111 11010011</td>
</tr>
<tr>
<td>ix</td>
<td>2003</td>
<td>00 00 07 D3 00000000 00000000 00000111 11010011</td>
</tr>
<tr>
<td>y</td>
<td>-2003</td>
<td>F8 2D 11111000 00101101</td>
</tr>
<tr>
<td>iy</td>
<td>-2003</td>
<td>FF FF F8 2D 11111111 11111111 11111000 00101101</td>
</tr>
</tbody>
</table>
Type Conversion (4)

- **Unsigned & Signed:** $w+k$ bits $\rightarrow$ $w$ bits
  - Just truncate it to lower $w$ bits
  - Equivalent to computing $x \mod 2^w$

```
unsigned int   x = 0xcafebabe;
unsigned short ix = (unsigned short) x;
int            y = 0x2003beef;
short          iy = (short) y;
```

<table>
<thead>
<tr>
<th></th>
<th>Decimal</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>3405691582</td>
<td>CA</td>
<td>FE</td>
<td>BA</td>
<td>BE</td>
<td>11001010</td>
</tr>
<tr>
<td>ix</td>
<td>47806</td>
<td>BA</td>
<td>BE</td>
<td></td>
<td></td>
<td>10111010</td>
</tr>
<tr>
<td>y</td>
<td>537116399</td>
<td>20</td>
<td>03</td>
<td>BE</td>
<td>EF</td>
<td>00100000</td>
</tr>
<tr>
<td>iy</td>
<td>-16657</td>
<td>BE</td>
<td>EF</td>
<td></td>
<td></td>
<td>10111110</td>
</tr>
</tbody>
</table>
Type Conversion (5)

- **Unsigned $\rightarrow$ Signed**
  - The same bit pattern is interpreted as a signed number

$$U2T_w(x) = \begin{cases} 
  x, & x < 2^{w-1} \\
  x - 2^w, & x \geq 2^{w-1} 
\end{cases}$$

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>2003</td>
<td>07 D3 00000111 11010011</td>
</tr>
<tr>
<td>ix</td>
<td>2003</td>
<td>07 D3 00000111 11010011</td>
</tr>
<tr>
<td>y</td>
<td>47806</td>
<td>BA BE 10111010 10111110</td>
</tr>
<tr>
<td>iy</td>
<td>-17730</td>
<td>BA BE 10111010 10111110</td>
</tr>
</tbody>
</table>
Type Conversion (6)

- Signed → Unsigned
  - Ordering inversion
  - Negative → Big positive

\[ T2U_w(x) = \begin{cases} 
  x + 2^w, & x < 0 \\
  x, & x \geq 0 
\end{cases} \]

<table>
<thead>
<tr>
<th></th>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td>2003</td>
<td>07 D3</td>
<td>00000111 11010011</td>
</tr>
<tr>
<td>(ix)</td>
<td>2003</td>
<td>07 D3</td>
<td>00000111 11010011</td>
</tr>
<tr>
<td>(y)</td>
<td>-2003</td>
<td>F8 2D</td>
<td>11111000 00101101</td>
</tr>
<tr>
<td>(iy)</td>
<td>63533</td>
<td>F8 2D</td>
<td>11111000 00101101</td>
</tr>
</tbody>
</table>

Example:
- Short \(x = 2003\);
- Unsigned short \(ix = (\text{unsigned short}) x\);
- Short \(y = -2003\);
- Unsigned short \(iy = (\text{unsigned short}) y\);
Type Casting in C (1)

- **Constants**
  - By default, considered to be signed integers
  - Unsigned if have "U" or "u" as suffix
    - 0U, 12345U, 0x1A2Bu

- **Type casting**
  - Explicit casting

  ```c
  int tx, ty;
  unsigned ux, uy;
  tx = (int) ux;
  uy = (unsigned) ty;
  ```

  ```c
  int f(unsigned);
  tx = ux;
  f(ty);
  ```

  - Implicit casting via
    - Assignments
    - Procedure calls
Type Casting in C (2)

- **Expression evaluation**
  - If mix unsigned and signed in single expression, signed values implicitly cast to **unsigned**.
  - Including comparison operations \(<, >, ==, <=, >=\)

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 == 0U</td>
<td>unsigned</td>
<td>True</td>
</tr>
<tr>
<td>-1 &lt; 0</td>
<td>signed</td>
<td>True</td>
</tr>
<tr>
<td>-1 &lt; 0U</td>
<td>unsigned</td>
<td>?</td>
</tr>
<tr>
<td>-1 &gt; -2</td>
<td>signed</td>
<td>?</td>
</tr>
<tr>
<td>(unsigned) -1 &gt; -2</td>
<td>unsigned</td>
<td>?</td>
</tr>
<tr>
<td>2147483647 &gt; -2147483647-1</td>
<td>signed</td>
<td>?</td>
</tr>
<tr>
<td>2147483647U &gt; -2147483647-1</td>
<td>unsigned</td>
<td>?</td>
</tr>
<tr>
<td>2147483647 &gt; (int) 2147483648U</td>
<td>signed</td>
<td>?</td>
</tr>
</tbody>
</table>
Type Casting in C (3)

- Example 1

```c
int main ()
{
    unsigned i;
    for (i = 10; i >= 0; i--)
        printf ("%u\n", i);
}
```

- Example 2

```c
void copy_mem1 (char *src, char *dest, unsigned len)
{
    unsigned i;
    for (i = 0; i < len; i++)
        *dest++ = *src++;
}
```
Type Casting in C (4)

- Example 3

```c
int sum_array (int a[], unsigned len)
{
    int i;
    int result = 0;

    for (i = 0; i <= len - 1; i++)
        result += a[i];

    return result;
}
```
Type Casting in C (5)

- Example 4

```c
#include <stdio.h>

int main ()
{
    unsigned char c;

    while ((c = getchar()) != EOF)
        putchar (c);
}
```
Type Casting in C (6)

- Lessons
  - There are many tricky situations when you use unsigned integers – hard to debug
  - Do not use just because numbers are nonnegative
  - Use only when you need collections of bits with no numeric interpretation ("flags")
  - Few languages other than C support unsigned integers