Data Representation and Manipulation.

- **Humans** - decimal (base 10) number system
- **Computers** - binary (base 2)

**3 representations of numbers**

1. **Unsigned encodings** (for numbers \( \geq 0 \))
2. **2's complement** (for signed numbers)
3. **Floating-point** (for real numbers)

**Information Storage**

*Hexadecimal Notation*

<table>
<thead>
<tr>
<th>00000000</th>
<th>11111111</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 16</td>
<td>FF 16</td>
</tr>
</tbody>
</table>

**Symbols:** 0, 1, 2, ..., 9, A, B, C, D, E, F.

eg. **0x1A0F** = 0001 1010 0000 1111

**Words**

word size \( (w) = 32 \) bits (in most machines).

If \( w = 32 \) bits, a program's virtual address space can access addresses in the range of 0 to \(<2^{32} - 1> \) (ie) \( 2^{32} \) 2 bytes.

\( w = 64 \) bits, common these days, \( 2^{64} \) bytes.
### Data Sizes

<table>
<thead>
<tr>
<th>C type</th>
<th>32-bit</th>
<th>64-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>short int</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>long int</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>char *</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

**short int** - at least 16 bits

**int** - 16 bits

**long int** - 32 bits

**long long int** - 64 bits

---

<table>
<thead>
<tr>
<th></th>
<th>Win 16 API</th>
<th>Unix &amp; Unix-like</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>2 bytes</td>
<td>4 bytes</td>
</tr>
</tbody>
</table>
Addressing and Byte Ordering

```c
int x;
x = 0x12345678
```

**Endian-ness**

<table>
<thead>
<tr>
<th>Little Endian</th>
<th>Big Endian</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x100</td>
<td>0x100</td>
</tr>
<tr>
<td>0x101</td>
<td>0x101</td>
</tr>
<tr>
<td>0x102</td>
<td>0x102</td>
</tr>
<tr>
<td>0x103</td>
<td>0x103</td>
</tr>
</tbody>
</table>

Least Significant Byte (LSB) comes first. "most" Intel machines.

MSB comes first. "most" IBM, Sun Microsystems, machines.

Show code to determine endian-ness of a machine.

Bi-endian - machines than can be configured to work as both little or big endian.

Origin of term "endian-ness"

Gulliver's Travels by Jonathan Swift.


Danny Cohen.

(Networking Protocol.)

Soft boiled Egg

Which side to open?
Problems with endianness:

1. Little endian → Network Byte Order (Big endian)
   htons, htonl, ntohs, ntohl.
   htonl - host to network; ntohs - network to host.

2. Machine-level code
   80483bd: 01 05
   64 94 04 08
   add %eax, 0x8049464
   08 04 94 64

Strings in C

→ encoded by an array of characters terminates by the NULL character.

eg. "ABC" ⇒ 41 42 43 00
    65 in decimal. null character.
To see ASCII code: `man ascii
Unicode standard for text:
- ASCII - only English
- Unicode - Albanian to Xamitanga.

Universal Character Set - 32-bit representation of chars.
- UTF-8 - Universal Coded Character Set
  Transformation Format - 8 bit.
  100,000 characters
  1,112,064 valid code points.
- Java uses Unicode.

Boolean Algebra

George Boole around 1850.

TRUE = 1
FALSE = 0

<table>
<thead>
<tr>
<th>op</th>
<th>logical op</th>
<th>boolean op</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>\overline{p}</td>
<td>\overline{p}</td>
</tr>
<tr>
<td>AND</td>
<td>p \land q</td>
<td>p \land q</td>
</tr>
<tr>
<td>OR</td>
<td>p \lor q</td>
<td>p \lor q</td>
</tr>
<tr>
<td>EXOR</td>
<td>p \oplus q</td>
<td>p \oplus q</td>
</tr>
</tbody>
</table>
Bit Vectors

Strings of 0s and 1s with some fixed length \( n \).

\[
a = 10101 \\
b = 01100 \\
a \land b = 00100
\]

Bit-level Operations in C

\( \sim 0 \times 00 \sim [0000 \ 0000] = [1111 \ 1111] \) (or) \( 0 \times FF \)

\( 0 \times 01 \mid 0 \times 02 = 0000 \ 0001 \\
0000 \ 0010 \\
\overline{0000 \ 0011} = 0 \times 03 \)

Masking Operations

\( 0 \times 89 \ AB \ CD \ EF \)

\( x \ & \ 0 \times FF = 0 \times 00 \ 00 \ 00 \ EF \)

\[ \text{Masking of the } \text{LSB} \]
*Logical Operations in C*

**OR** - `||`
**AND** - `&&`
**NOT** - `!`

TRUE - any non-zero value
FALSE - 0

eg. 

- `!0x41 = 0x00`
- `!0x00 = 0x01`

**short circuit in logical operations.**

- `a && 5/a` → never cause a division by zero.
- `p && *p++` → "" null ptr dereference

*Shift operations in C*

\[ x = 01100011 \]
\[ x << 4 = 00110000 \]
\[ x >> 4 = 00000110 \]
\[ (\text{logical}) \]
\[ x >> 4 = \text{and} \ 0000 \ 0110 \]
\[ (\text{arithmetic}) \]

\[ y = 1001 \ 0101 \]
\[ y << 4 = 0101 \ 0000 \]
\[ y >> 4 = 0000 \ 1001 \]
\[ (\text{logical}) \]
\[ y >> 4 = 1111 \ 1001 \]
\[ (\text{arithmetic}) \]

In C, no precise definition for right shifts.

In Java, `x >> k` means arithmetic and `x >>> k` means logical.
Integer Representations

I. Unsigned encoding

\[ B2U_w(x) = \sum_{i=0}^{w-1} x_i 2^i \]

\[ \begin{array}{c c c c c}
0 & 1 & 0 & 1 \\
\hline
2^3 & 2^2 & 2^1 & 2^0
\end{array} = 1 \times 2^2 + 1 \times 2^0 = 5 \]

\[ w = 4 \]

II. 2's Complement Encodings

\[ B2T_w(x) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i 2^i \]

\[ \begin{array}{c c c c c}
0000 & = & 0 \\
0001 & = & 1 \\
\hline
w = 2
\end{array} \]

\[ \begin{array}{c c c c c}
00 & = & 0 \\
01 & = & 1 \\
10 & = & -2 \\
11 & = & -1 \\
\hline
w = 3
\end{array} \]

\[ B2T_3(1011) = -1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = -8 + 2 + 1 = -5 \]
Note: 1's complement encoding

\[
\begin{align*}
000 &= +0 \\
001 &= +1 \\
010 &= +2 \\
011 &= +3 \\
100 &= -1 \\
101 &= -2 \\
110 &= -3 \\
111 &= -0
\end{align*}
\]

Important Numbers

Word size \( w = 32 \) bits

\[
\begin{align*}
U_{\text{Max}}_{32} &= 0 \times FF \, FF \, FF \, FF \, FF = 4,294,967,295 \\
T_{\text{Max}}_{32} &= 0 \times FF \, FF \, FF \, FF = 2,147,483,647 \\
T_{\text{Min}}_{32} &= 0 \times 80 \, 00 \, 00 \, 00 \, 00 = -2,147,483,648
\end{align*}
\]

\[
\begin{align*}
0 &= 0 \times 00 \, 00 \, 00 \, 00 \, 00 \\
-1 &= 0 \times FF \, FF \, FF \, FF
\end{align*}
\]

\[
\begin{align*}
|T_{\text{Min}}| &= |T_{\text{Max}}| + 1 \\
\Rightarrow & \text{ There is no positive counterpart for } T_{\text{Min}}.
\end{align*}
\]

\[
U_{\text{Max}} = 2 \times T_{\text{Max}} + 1
\]
\langle \text{limits}. h \rangle \quad \text{defines constants with the limits of fundamental integral types for the specific system and compiler/implementation used.}

\text{eg.} \quad \text{INT\_MAX} = +2,147,483,647
\quad \text{INT\_MIN} = -2,147,483,648
\quad \text{UINT\_MAX} = 4,294,967,295

\text{ISO C99 standard - stdint.h}
\quad \text{int}_t \quad \text{and} \quad \text{uint}_t
\quad \text{eg.} \quad \text{int32}_t \quad \text{and} \quad \text{uint32}_t

\text{MACROS:} \quad \text{INTN\_MIN}, \quad \text{INTN\_MAX}, \quad \text{and} \quad \text{UINTN\_MAX}
\quad \text{eg.} \quad \text{INT32\_MIN}

\* \text{Signed - Magnitude form}
\quad B^2 S_w(x) = (-1)^{x_{w-1}} \cdot \left( \sum_{i=0}^{w-2} x_i 2^i \right)

\begin{align*}
\text{00} &= +0 \\
\text{01} &= +1 \\
\text{10} &= -0 \\
\text{11} &= -1
\end{align*}
Two's complement of \( x = 2^w - x \) (a single two).

- \( x \) in two's complement

Ones' complement of \( x = [111\ldots1] - x \) (multiple ones).

- \( x \) in ones' complement

* Conversions between Signed and Unsigned

unsigned \( u = 4,294,967,295 \) u;

\( \text{int } tu = (\text{int}) u; \)

printf("u = %.u, tu = %.d\n", u, tu);

\( u = 4294967295 \), \( tu = -1 \)

\( u \) and \( tu \) in hex = Ox FF FF FF FF FF

- when signed \( \leftrightarrow \) unsigned (of same word size)

\( \Rightarrow \) numeric values might change

but

the bit patterns do not.
Signed vs unsigned in C

- Most numbers are signed by default.
  - Example: 12345 or 0x1A2B
- Unsigned: 12345U or 0x1A2Bu

If signed and unsigned appear in an operation, signed → unsigned.

Problems:

1. \(-1 < 0U \Rightarrow 4,294,967,295U < 0U\)

Sign extension & Zero extension

- Short sx = -12345;
- Unsigned short usx = sx;
- Int x = sx;
- Unsigned ux = usx;

sx = -12345 : CF C7
usx = 53191 : CF C7
x = -12345 : FF FF CF C7
ux = 53191 : 00 00 CF C7

Sign extension → zero extension.
sign extension

\[
\begin{align*}
\text{w=3} & \quad \text{to} \quad \text{w=4} \\
[101] & \quad \begin{bmatrix} 1101 \end{bmatrix} \\
-4+1 = -3 & \quad -8+4+1 = -3
\end{align*}
\]

\[
\begin{bmatrix} 1111 \end{bmatrix} = \begin{bmatrix} 111 \end{bmatrix} = -1
\]

short to unsigned int?

\[
\begin{align*}
\text{short } sx = -12345; & \quad /\!* \text{ CF CF7 x}*/ \\
\text{unsigned uy = sx ;}
\end{align*}
\]

\[
\begin{align*}
\text{CF C7} & \xrightarrow{\text{change size}} \text{ FF FF CF C7} & \xrightarrow{\text{change from signed to unsigned}} 429495495
\end{align*}
\]

if 1. signed \(\rightarrow\) unsigned \[ \Rightarrow \] 53,191. \( \times \) 

\[
\text{Truncating Numbers}
\]

\[
\begin{align*}
\text{int } x = 53191; & \\
\text{short } sx = (\text{short}) x; & \quad /\!* -12345 */\!
\end{align*}
\]

\[
\begin{align*}
\text{int } y \underline{= } sx; & \quad /\!* -12345 */\! \\
& \quad (\text{sign extension})
\end{align*}
\]
Advice on Signed vs. Unsigned

```c
size_t strlen (const char *s);
```

```c
unsigned int
int strlonger (char *s, char *t)
{
    return strlen(s) - strlen(t) > 0;
}
```

When

- \( s > t \), OK
- \( s = t \), OK.
- \( s < t \), NOT OK. \( \Rightarrow 1u - 2u > 0u \)

Connection

```
return strlen(s) > strlen(t);
```

Don't mix unsigned int and int!

(size_t)

Security vulnerability in getpeername.
INTEGER ARITHMETIC

1. Unsigned Addition.
   \[ w = 4 \quad 0000 \text{ to } 1111 \]
   \[ (w+1) = 0 \text{ to } 15 \]
   \[ \text{Max sum } = 15 + 15 = 30 \]
   Word size inflation
   \[ w + w \rightarrow \text{needs (w+1) bits} \]
   \[ (w+1) + (w+1) \rightarrow \text{needs (w+2) bits} \]
   How to do unsigned arithmetic?
   Use modular arithmetic.

   \[ \begin{array}{c}
   10111110 \\
   \div 16 \\
   \hline
   3 \times 16 + 2 \\
   \hline
   0100101 \\
   \end{array} \]

   \[ \begin{array}{c}
   21 \div 16 = 5 \\
   2^4 \\
   \end{array} \]

   21 needs 5 bits

   \[ \begin{array}{c}
   1001 + 1100 \\
   \hline
   10101 \\
   \end{array} \]

   Discard the last bit = subtracting \( 2^w \) from the sum.

   Ununsigned addition

   \[ x + y = \begin{cases} 
   x + y, & x + y < 2^w \\
   x + y - 2^w, & 2^w \leq x + y < 2^{w+1} 
   \end{cases} \]

   Overflow of arithmetic operation - Full integer result cannot fit within the word size. Limits of the data type.
Two's complement addition

Range of \( -2^{w-1} \leq x, y \leq 2^{w-1} \)

\[ x + y = \begin{cases} x + y - 2, & 2^{w-1} \leq x + y \leq 2^w - 1 \\ x + y, & -2^{w-1} \leq x + y < 2^{w-1} \\ x + y + 2, & x + y < 2^{w-1} \end{cases} \]

\[ x \oplus y = \begin{cases} x \oplus y, & 2^{w-1} \leq x \oplus y \leq 2^w - 1 \\ x \oplus y, & -2^{w-1} \leq x \oplus y < 2^{w-1} \end{cases} \]

\[ \begin{array}{c|c|c|c|c|c|c|c} \hline x & y & x+y & x\oplus y \\ \hline 1000 & 1011 & 1101 & 1011 \\ 0010 & 0101 & 1001 & 0101 \\ \hline \end{array} \]

\[ \begin{array}{c|c|c|c|c|c|c|c} \hline x & 4 & x + 4 & y \\ \hline 1000 & 1011 & 0011 & 1011 \\ \hline \end{array} \]

Example:

\[ x = -8, \ y = -5 \]

\[ x + y = -13 \]

\[ -13 + 16 = 3 \]

\[ x + 4 = 3 \]

Positive overflow

Normal

Negative overflow
Two's Complement Negation.

A number $x$'s additive inverse is $-x$.

$$x + (-x) = 0$$

For all $\frac{1}{2} < x < 2^{w-1}$, additive inverse is $-x$.

But for $x = -2^{w-1}$, add. inv. is $-2^{w-1}$.

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

Unsigned Multiplication

$$0 \leq x, y \leq 2^{w-1}$$

$$0 \leq x \cdot y \leq (2^{w-1})^2 = \frac{2^w \cdot 2^{w+1}}{2 - 2 + 1}$$

Solution

Truncation of higher order bits, e.g. if $w = 4$:

$$2^4 - 2^3 + 1 = 16 - 8 + 1 = 9$$

Need 4 bits.

$$\overline{\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}} \times \overline{\begin{bmatrix} 3 \\ 3 \end{bmatrix}} = 9$$

$$9 \div 2 = 4.5$$

$$9 \mod 2 = 1$$
Twos' Complement Multiplication

\[ \text{w} = 3 \]

unsigned
Twos comp.

\[ 5[101] \times 3[011] = 15[001111] \times 7[111] \]

\[ -3[101] \times 3[011] = -9[110111] \times -1[111] \]

* Multiplying by constants

\[ x \times 14\]

14 = \(2^3 + 2^2 + 2^1\)

\[ \Rightarrow (x \ll 3) + (x \ll 2) + (x \ll 1) \text{ ?} \]

\[ \text{Integer multiplication} \]

\[ 1 \times 14 = 14 \]

\[ [0001] \times [1110] = [1110] \]

Also:

\[ 14 = 4 - 2 \]

so,

\[ x \times 14 = (x \ll 4) - (x \ll 1) \]

If \( x = 2[0010] \)

\[ 2 \times 14 = 28 \]

\[ [0010] \times [1110] = [11100] \]

\[ = [0010] \ll 3 + [0010] \ll 2 + [0010] \ll 1 \]

\[ = 10000 + 1000 + 0100 = 11100 = [1100]_{12} \]
Dividing by Powers of Two

Integer division - 30 or more clock cycles.

Unsigned division by \(2^k\) (e) \(x / 2^k\) (logical right shift)

III" for +ve signed number.

Negative signed numbers (arithmetic right shift).

\(-7 / 2 = -3\)

But \([1001] / 2^1 = [1001] \gg 1 (\therefore k=1) = [1100] = -4 (Wrong)

ie if \(x < 0\), then \(x = x + (1 << k) - 1\)

e.g. \(x = -7 < 0\), \(\therefore x = [1001] + [0001] \ll 1) - [0001]

\(-7 = [1001] + [0010] - [0001]

\(= [1001] + [0001] =

\(= [1010]

Now \(x / 2^1 = [1010] \gg 1 = [1101] = -3\) Now