Number systems

Decimal (base 10):
why 10^2;
digits: 0..9
8031 = 3 \times 10^3 + 0 \times 10^2 + 3 \times 10^1 + 1 \times 10^0

Binary (base 2):
why 2^2;
digits: 0, 1
10112 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 11

Byte = 8 Bits; 1011 00112 value e 50..255

Conversion decimal -> binary: division with rest with 2
example: 11 = 5 \times 2 + 1
5 = 2 \times 2 + 1
2 = 1 \times 2 + 0
1 = 0 \times 2 + 1

result

Hexadecimal (base 16):
why 16^2;
digits: 0..9 A..F
B16 = 1110, 10B2

1 Byte = 2 hex. digits 00..FF
0x2AB1 = 2AB116 = 2 \times 16^3 + 10 \times 16^2 + 11 \times 16^1 + 1

Conversion decimal -> hex.: division with rest using 16
(Virtual) memory is divided into bytes:

Every byte has an address.

Size of address = size of pointer (e.g. char *) = word size

32-bit architecture: 4 bytes → $2^{32}$ = 4G bytes
virtual address space

64-bit architecture: 8 bytes → $2^{64}$ bytes

Data types and sizes:

<table>
<thead>
<tr>
<th>Type</th>
<th>ia 32</th>
<th>x86-64</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>long</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>double</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>char *</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

Addressing:

- chars: 0x001004 0x0100
- int: 0x0100
- double: 0x0100

Byte ordering:

\[ \text{int } x = 0x01234567 \]

Little endian (ia32, x86-64):

67 45 23 01

Big endian (Sun):

01 23 45 67

8x: depends on machine/compiler
**Boole, 1850, foundation of logic:**

- $0$ = false
- $1$ = true

**Commutative**
- $\land$
- $\lor$
- $\land$
- $\lor$

**Distributive**
- $a \lor (b \land c)$
- $a \land (b \lor c)$

**Relationship with integers**

- $(\mathbb{Z}, +, \cdot)$
- $(\mathbb{Q}, 1, \cdot, 1)$
- $(\mathbb{Q}, 1, \land, 1)$

$= (\mathbb{Z}, 1, \cdot, 1)$

**Set operations in C**

- $\&, \land$, $\lor$ - available for integral data types char, int, ...
- $\neg$ - bitwise operation

$\neg 0x97 = \neg 0b01010111_2 = 0b11001000_2 = 0x68$

$0x97 \& 0x7c = 0b01010111_2 \& 0b01111100_2 = 0b01010000_2$ masks out bits

**Application:** Set representation, $S = \{0, \ldots, 7\}$

- $\subseteq$ $\land$
- $\cup$
- $\cap$
- $\neg$ (complement)
Application 2:

```c
p = malloc(...)
```
change `p` to the next address divisible by 16
(array is 16-byte aligned)

1. set last 4 bits to zero
   
   ```c
   p = p & 0xFFFFFFF0
   ```

2. if `p` = `p` ✓
   ```c
   else p = p + 0x10
   ```

3. make it independent of word size
   ```c
   0xFFFFFFF0 ⇔ ~0x00F
   ```

Logical operations in C (different from bit operations)

```c
and, or, not ⇔ &, |, !
```

0 = false
1 = true
nonzero = true
always returns 0 or 1
early termination

Shift operations in C

```
left shift by k: x << k
```

```c
[x_{n-1}, x_{n-2}, ..., x_0] \_2 \rightarrow [x_{n-1}, ..., x_0, 0, ..., 0] _2
```

Right shift by k: x >> k

logical ```c
[x_{n-1}, x_{n-2}, ..., x_0] _2 \rightarrow [0, ..., 0, x_{n-1}, ..., x_k] _2
```
(for unsigned data)

arithmetic

```
(x_{n-1}, x_{n-2}, ..., x_k) _2 \rightarrow [x_{n-1} - x_{n-1}, x_{n-1}, ..., x_k] _2
```
usually for signed data