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>‘2’</th>
<th>‘0’</th>
<th>‘0’</th>
<th>‘9’</th>
<th>‘s’</th>
<th>‘k’</th>
<th>‘k’</th>
<th>‘u’</th>
</tr>
</thead>
<tbody>
<tr>
<td>(int)</td>
<td>959459378</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(double)</td>
<td>0.41170812257764958060866393978917759243058270049622... x $10^{258}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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: 00000000₂ to 11111111₂
  - Octal: 000₈ to 377₈
    - An integer constant that begins with 0 is an octal number in C
  - Decimal: 0₁₀ to 255₁₀
    - First digit must not be 0 in C
  - Hexadecimal: 00₁₆ to FF₁₆
    - Base 16 number representation
    - Use characters ‘0’ to ‘9’ and ‘A’ to ‘F’
    - Write FA1D37B₁₆ in C as 0xFA1D37B or 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
    \[
    \begin{array}{c|cc}
    & 0 & 1 \\
    \hline
    0 & 0 & 0 \\
    1 & 1 & 1 \\
    \end{array}
    \]

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

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

- 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>0</td>
<td>X</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>~X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X &amp; Y</td>
<td>~ (X → Y)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X^Y</td>
<td>; XOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>; OR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>~ (X</td>
<td>Y)</td>
<td>; NOR</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>~ (X^Y)</td>
<td>; X-NOR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>~ X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>~ (X &amp; Y)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>~ (X → Y)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>~ (X &amp; Y)</td>
<td>; NAND</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Constant 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

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

- X^Y = (X & ~Y) | (~X & Y)
- X→Y = ~X | Y

A complete set: NAND = ~ (X & Y)

- ~X
- X & Y
- X | Y
- X → Y
- ~ (X & Y)
Combinational Logic

- Adder

Full Adder

1-bit Full Adder

4-bit Ripple Carry Adder
Sequential Logic

- **Flip-flops**

![Diagram of edge triggered D flip-flop]

- Shifter

- 4-bit register
Transistors (1)

- **Transistor = Electronic switch**
  - Controlled by voltages
    - e.g., Logic 1 = 5V, Logic 0 = 0V
  - NMOS transistor

![Diagram of Transistor]

- Closed switch ($V_G = 5V$)
- Open switch ($V_G = 0V$)
Transistors (2)

- NOT logic built with NMOS technology

\[ \text{Diagram of NOT gate built with NMOS technology} \]
Transistors (3)

- NAND logic built with NMOS technology

\[
\begin{array}{ccc}
  x_1 & x_2 & f \\
  0 & 0 & 1 \\
  0 & 1 & 1 \\
  1 & 0 & 1 \\
  1 & 1 & 0 \\
\end{array}
\]
Transistors (4)

- NOT logic built with CMOS technology
Transistors (5)

- NAND logic built with CMOS technology
• Boolean algebra is a mathematical foundation for modern digital systems
• Boolean algebra provides an effective means of describing circuits built with switches
  – Claude Shannon in the late 1930’s.
• You can build any digital systems with NAND gates
• A NAND gate can be easily built with CMOS transistors
• The transistor is the basic building block for digital systems
Bit-Level Operations in C

- Operations &, |, ~, ^ Available in C
  - Apply to any "integral" data type
    - long, int, short, char, unsigned
  - View arguments as bit vectors
  - Arguments applied bit-wise

- Examples (Char data type)
  - ~0x41 --> 0xBE
    ~01000001₂ --> 10111110₂
  - ~0x00 --> 0xFF
    ~00000000₂ --> 11111111₂
  - 0x69 & 0x55 --> 0x41
    01101001₂ & 01010101₂ --> 01000001₂
  - 0x69 | 0x55 --> 0x7D
    01101001₂ | 01010101₂ --> 01111101₂
  - 0x69 ^ 0x55 --> 0x4C
    01101001₂ ^ 01010101₂ --> 00111100₂
Logic Operations in C

- Contrast to logical operators
  - &&, ||, !
  - View 0 as “False”
  - Anything nonzero as “True”
  - Always return 0 or 1
  - Early termination

- Examples (char data type)
  - !0x41  -->  0x00
  - !0x00  -->  0x01
  - !!0x41 -->  0x01
  - 0x69  &&  0x55  -->  0x01
  - 0x69  ||  0x55  -->  0x01
  - if (p && *p)  (avoids null pointer access)
Shift Operations

- **Left shift:** \(x << y\)
  - Shift bit-vector \(x\) left \(y\) positions
    - Throw away extra bits on left
    - Fill with 0’s on right

- **Right shift:** \(x >> y\)
  - Shift bit-vector \(x\) right \(y\) positions
    - Throw away extra bits on right
  - Logical shift
    - Fill with 0’s on left
  - Arithmetic shift
    - Replicate MSB on right
    - Useful with two’s complement integer representation

- **Undefined if** \(y < 0\) or \(y \geq \text{word size}\)

<table>
<thead>
<tr>
<th>Argument (x)</th>
<th>01100010</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;= 3)</td>
<td>00010000</td>
</tr>
<tr>
<td>Log. &gt;&gt; 2</td>
<td>00011000</td>
</tr>
<tr>
<td>Arith. &gt;&gt; 2</td>
<td>00011000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Argument (x)</th>
<th>10100010</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;= 3)</td>
<td>00010000</td>
</tr>
<tr>
<td>Log. &gt;&gt; 2</td>
<td>00101000</td>
</tr>
<tr>
<td>Arith. &gt;&gt; 2</td>
<td>11101000</td>
</tr>
</tbody>
</table>