Number Systems and Circuits for Addition

CS 231
Dianna Xu

Decimal Number System
(Base-10 number system)

123
= 1 × 100 + 2 × 10 + 3
= 1 × 10^2 + 2 × 10^1 + 3 × 10^0
= 123_{10}

Binary Number System
(Base-2 number system)

\[101_2\]
= 1 × 2^2 + 0 × 2^1 + 1 × 2^0
= 1 × 4 + 0 × 2 + 1
= 5_{10}

Capacity of Binary Numbers

- 1 bit can distinguish 2 states (0 or 1).
- An n-bit binary number can distinguish \(2^n\) states.

Octal Number System
(Base-8 number system)

173_8
= 1 × 8^2 + 7 × 8^1 + 3 × 8^0
= 1 × 64 + 7 × 8 + 3
= 123_{10}

Hexadecimal Number System
(Base-16 number system)

9AB_{16}
= 9 × 16^2 + 10 × 16^1 + 11 × 16^0
= 9 × 256 + 10 × 16 + 11
= 2475_{10}
Hexadecimal

- Often written with a '0x' prefix
  - 0x10 is 10₁₀, or 16₁₆
  - 0x100 is 100₁₀, or 256₁₆
- Binary numbers easily translate:
  - In blocks of 4

### Table 1.3.3

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>4-Bit Binary Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>1010</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>1011</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>1100</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>1101</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>1110</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>1111</td>
</tr>
</tbody>
</table>

Byte

- 1 Byte
  - 8 bits
  - Representable by 2 hexadecimal characters
  - Can distinguish 256 ($2^8$) states

Base-$k$-to-Decimal Conversion

- $101_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
- $= 1 \times 4 + 0 \times 2 + 1$
- $= 5_{10}$

More generally, for $\mathbb{A} = \{0, 1, \ldots, k-1\}$

$$\sum_{i=0}^{\log k} c_i k^i$$

- $\nabla \otimes k = k^2 + \nabla \otimes k + \nabla \otimes k^0$
- $\Rightarrow ?_{10}$

Decimal-to-Binary Conversion

```
2 \times 5
2 \times 2 \ldots 1 \leftarrow \text{the rightmost bit (LSB)}
1 \ldots 0 \leftarrow \text{the second bit from the right}
\uparrow
\text{the most significant bit (MSB)}
MSB = \text{Most Significant Bit}
LSB = \text{Least Significant Bit}
```

Decimal to hexadecimal

```
16) 2475
16) 154 \ldots 11 \leftarrow \text{the rightmost bit (LSB)}
9 \ldots 10 \leftarrow \text{second bit from the right}
\uparrow
\text{the most significant bit (MSB)}
```

Decimal to Base-$k$ conversions work the same way
How to add binary numbers

• Consider adding two 1-bit binary numbers \( x \) and \( y \)
  - \( 0 + 0 = 0 \)
  - \( 0 + 1 = 1 \)
  - \( 1 + 0 = 1 \)
  - \( 1 + 1 = 10 \)

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>Carry</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

• Carry is \( x \) AND \( y \)
• Sum is \( x \) XOR \( y \)
• The circuit to compute this is called a half-adder

Using half adders

• We can then use a half-adder to compute the sum of two binary numbers?

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( \text{Carry} )</th>
<th>( \text{Sum} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

How to fix this

• We need to create an adder that can take a carry bit as an additional input
  - Inputs: \( x \), \( y \), carry in
  - Outputs: sum, carry out
• This is called a full adder
  - Will add \( x \) and \( y \) with a half-adder
  - Will add the sum of that to the carry in
• What about the carry out?
  - Final CO is 1 if:
    - \( x+y = 10 \)
    - \( x+y = 01 \) and carry in = 1

The full adder

• The “HA” boxes are half-adders

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( c )</th>
<th>( S )</th>
<th>( C_1 )</th>
<th>( C_2 )</th>
<th>( \text{Carry} )</th>
<th>( \text{Sum} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The full adder

• The full circuitry of the full adder
Adding bigger binary numbers

• Just chain full adders together

| \(x_0\) | \(y_0\) | \(S_0\) |
| \(x_1\) | \(y_1\) | \(S_1\) |
| \(x_2\) | \(y_2\) | \(S_2\) |
| \(x_3\) | \(y_3\) | \(S_3\) |
| \(\ldots\) | \(\ldots\) | \(C\) |

Parallel adders and number of gates

• A half adder has 4 logic gates
• A full adder has two half adders plus an OR gate
  – Total of 9 logic gates
• To add \(n\) bit binary numbers,
  – \(1 \text{ HA} + n-1 \text{ FAs}\)
• To add 32 bit binary numbers,
  – \(1 \text{ HA} + 31 \text{ FA} = 4 + 9 \cdot 31 = 283\) logic gates
• To add 64 bit binary numbers,
  – \(1 \text{ HA} + 63 \text{ FA} = 4 + 9 \cdot 63 = 571\) logic gates

More about logic gates

• To implement a logic gate in hardware, you use a transistor
• Transistors are all enclosed in an “IC”, or integrated circuit
• 1971 – Intel’s first microprocessor (4004): 2300 transistors
• 1993 – Intel Pentium processor: 3.1 million
• 2006 – Dual-core Itanium 2: 1.7 billion
• 2011 – 10-core Xeon Westmere-Ex: 2.6 billion
• 2015 – SPARC M7: 10 billion

Two’s Complement

• Given a positive integer \(a\), the two’s complement of \(a\) is the \(n\)-bit representation of \(2^n - a\)
• \(2^8 - 35 = 256 - 35 = 221 = 11011101_2\)
• \(a\)’s two’s complement represents \(-a\)
• Always relative to a fixed bit length
• Bit length of 32 and 64 are most commonly used

One’s Complement

• An easier way to calculate two’s complement
• \(2^8 - a = (2^8 - 1) - a + 1\)
• \(2^8 - 1 = 11111111_2\)
• Subtracting any binary number from all 1’s is equivalent to negating all bits, i.e. taking the one’s complement

Example

\[
2^8 - 35 = (2^8 - 1) - 35 + 1 = 11111111_2 - 00100011_2 = 11011100_2 + 1 = 11011101_2 = 221
\]
Two’s Complement Again

- To find the two’s complement of a positive integer $a$:
  - Write the $n$-bit binary representation for $a$
  - Negate all bits
  - Add 1 to the resulting binary notation

8-bit Representations

<table>
<thead>
<tr>
<th>Integer</th>
<th>8-bit Binary</th>
<th>2’s complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>127</td>
<td>01111111</td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>01111110</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>00000000</td>
<td>$2^8 - 1$</td>
</tr>
<tr>
<td>-1</td>
<td>11111111</td>
<td>$2^8 - 2$</td>
</tr>
<tr>
<td>-2</td>
<td>11111110</td>
<td>$2^8 - 3$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>-127</td>
<td>10000000</td>
<td>$2^8 - 127$</td>
</tr>
<tr>
<td>-128</td>
<td>10000000</td>
<td>$2^8 - 128$</td>
</tr>
</tbody>
</table>

Addition with Negative Numbers

$64 - 15 = 64 + (-15) = 01000000_2 + ((11111111_2 - 00001111_2) + 1_2) = 01000000_2 + 11110001_2 + 01000000_2 + 11110001_2 + 00110011_2 = 49$