Ceng 111 – Fall 2015
Week 3a

Binary Representation

Credit: Some slides are from the “Invitation to Computer Science” book by G. M. Schneider, J. L. Gersting and some from the “Digital Design” book by M. M. Mano and M. D. Ciletti.
A computer

Devices

Gates

Transistors

Previously on CENG 111!
Decoders

Previously on CENG 111!

<table>
<thead>
<tr>
<th>S1</th>
<th>S0</th>
<th>Q0</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ Q_0 = S_1' S_0' \]
\[ Q_1 = S_1' S_0 \]
\[ Q_2 = S_1 S_0' \]
\[ Q_3 = S_1 S_0 \]
Multiplexers

<table>
<thead>
<tr>
<th>S</th>
<th>D1</th>
<th>D0</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Q = S' D0 + S D1
Data Representation

- Based on 1s and 0s
  - So, everything is represented as a set of binary numbers

- We will now see how we can represent:
  - Integers: 3, 1234435, -12945 etc.
  - Floating point numbers: 4.5, 124.3458, -1334.234 etc.
  - Characters: /, &, +, -, A, a, ^, 1, etc.
  - ...

2015 S. Kalkan & G. Ucoluk & A. Cosar - CEng 111
Binary Representation of Numeric Information (continued)

- Sign/magnitude notation
  
  1 101 = -5 
  0 101 = +5 

- Problems:
  
  - Two different representations for 0:
    - 1 000 = -0
    - 0 000 = +0
  
  - Addition & subtraction require a watch for the sign! Otherwise, you get wrong results:
    - 0 010 (+2) + 1 010 (-2) = 1 100 (-4)
Two’s complement instead of sign-magnitude representation

- Positive numbers have a leading 0.
  - 5 => 0101
- The representation for negative numbers is found by subtracting the absolute value from $2^N$ for an N-bit system:
  - -5 => $2^4 - 5 = 16 - 5 = 11_{10} => 1011_2$

Advantages:

- 0 has a single representation: +0 = 0000, -0 = 0000
- Arithmetic works fine without checking the sign bit:
  - 1011 (-5) + 0110 (6) = 0001 (1)
  - 1011 (-5) + 0011 (3) = 1110 (-2)
Binary Representation of Numeric Information (continued)

- **Shortcut to convert from “two’s complement”:**
  - If the leading bit is zero, no need to convert.
  - If the leading bit is one, invert the number and add 1.

- **What is our range?**
  - With 2’s complement we can represent numbers from $-2^{N-1}$ to $2^{N-1} - 1$ using N bits.
  - 8 bits: -128 to +127.
  - 16 bits: -32,768 to +32,767.
  - 32 bits: -2,147,483,648 to +2,147,483,647.

<table>
<thead>
<tr>
<th>Binary Number</th>
<th>Decimal Value</th>
<th>Value in Two’s Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>-8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>-7</td>
</tr>
<tr>
<td>1010</td>
<td>10</td>
<td>-6</td>
</tr>
<tr>
<td>1011</td>
<td>11</td>
<td>-5</td>
</tr>
<tr>
<td>1100</td>
<td>12</td>
<td>-4</td>
</tr>
<tr>
<td>1101</td>
<td>13</td>
<td>-3</td>
</tr>
<tr>
<td>1110</td>
<td>14</td>
<td>-2</td>
</tr>
<tr>
<td>1111</td>
<td>15</td>
<td>-1</td>
</tr>
</tbody>
</table>
Example:

- We want to compute: $12 - 6$
- $12 \Rightarrow 01100$
- $-6 \Rightarrow -(00110) \Rightarrow (11001)+1 \Rightarrow (11010)$

$12 - 6 =$

01100  
+ 11010

00110 $\Rightarrow$ 6

So, addition and subtraction operations are simpler in the Two’s Complement representation.
Why does Two’s Complement work?

1st Perspective:

We divide the range of unsigned numbers using N bits into two halves:

- 0 to $2^{N-1} - 1$ => Positive numbers
- $2^{N-1}$ to $2^N - 1$ => Negative numbers ($-2^{N-1}$ to -1)

Why?

- $i - j \mod 2^N = i + (2^N - j) \mod 2^N$

Example:

- Consider X and Y are positive numbers.
- $X + (-Y) = X + (2^N - Y) = 2^N - (Y - X) = -(Y - X) = X - Y$

<table>
<thead>
<tr>
<th>Binary Number</th>
<th>Decimal Value</th>
<th>Value in Two’s Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>-8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>-7</td>
</tr>
<tr>
<td>1010</td>
<td>10</td>
<td>-6</td>
</tr>
<tr>
<td>1011</td>
<td>11</td>
<td>-5</td>
</tr>
<tr>
<td>1100</td>
<td>12</td>
<td>-4</td>
</tr>
<tr>
<td>1101</td>
<td>13</td>
<td>-3</td>
</tr>
<tr>
<td>1110</td>
<td>14</td>
<td>-2</td>
</tr>
<tr>
<td>1111</td>
<td>15</td>
<td>-1</td>
</tr>
</tbody>
</table>
Why does Two’s Complement work?

A 2\textsuperscript{nd} perspective:

- Inversion and addition of a 1-bit correspond effectively to subtraction from 0 – i.e., negative of a number.
- Negative of a binary number X: 00...00\textsubscript{2} - X\textsubscript{2}
- Note that 00...00\textsubscript{2} = 11...11\textsubscript{2} + 1\textsubscript{2}
- In other words:
  - 00...00\textsubscript{2} - X\textsubscript{2} = (11...11\textsubscript{2}) - X\textsubscript{2} + 1.  
    (i.e., how we find two’s complement)
Why does Two’s Complement work?

A smart trick used in mechanical calculators

- To subtract $b$ from $a$, invert $b$ and add that to $a$. Then discard the most significant digit.

Today

- Quiz
- Binary representation of real numbers
Binary Representation of Real Numbers

Conversion of the digits after the dot into binary:

1\textsuperscript{st} Way:
- 0.375 \rightarrow 0 \times \frac{1}{2} + 1 \times \frac{1}{4} + 1 \times \frac{1}{8} \rightarrow 011

2\textsuperscript{nd} Way:
- Multiply by 2 and get the integer part until we get ‘0’ after the dot:
  - 0.375 \times 2 = 0.750 = 0 + 0.750
  - 0.750 \times 2 = 1.500 = 1 + 0.500
  - 0.500 \times 2 = 1.000 = 1 + 0.000
Binary Representation of Real Numbers

- **Approach 1:** Use fixed-point
  - Similar to integers, except that there is a decimal point.
  - E.g: using 8 bits:
    
    \[
    \begin{array}{cccccccc}
    1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
    \end{array}
    \]
    
    Assumed decimal point
    
    \[
    \begin{align*}
    1*2^7 + 1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0 + \\
    1*2^{-1} + 1*2^{-2} + 1*2^{-3} + 1*2^{-4} &= 15.9375
    \end{align*}
    \]
Binary Representation of Real Numbers

- Location of the decimal point changes the value of the number.
  - E.g.: using 8 bits:

\[
\begin{array}{ccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}
\]

\[
= 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0
\]

\[
1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} = 31.875
\]

Assumed decimal point
Binary Representation of Real Numbers

- Problems with fixed-point:
  - Limited in the maximum and minimum values that can be represented.
  - For instance, using 32-bits, reserving 1-bit for the sign and putting the decimal point after 16 bits from the right, the maximum positive value that can be stored is slightly less than, $2^{15}$.
  - Allowing larger values gives away from the precision (the decimal part).
Binary Representation of Real Numbers

- Solution: Use scientific notation: \( a \times 2^b \) (or \( \pm M \times B^{\pm E} \))
  - Example: 5.75
    - 5 \( \rightarrow \) 101
    - 0.75 \( \rightarrow \) \( \frac{1}{2} + \frac{1}{4} \rightarrow 2^{-1} + 2^{-2} \rightarrow 11 \)
    - 5.75 \( \rightarrow \) 101.11 \( \times \) 2\(^0\)

- Number is then normalized so that the first significant digit is immediately to the left of the binary point
  - Example: 1.0111 \( \times \) 2\(^2\)

- We take and store the mantissa and the exponent.
Binary Representation of Real Numbers

- This needs some standardization for:
  - where to put the decimal point
  - how to represent negative numbers
  - how to represent numbers less than 1
IEEE 32bit Floating-Point Number Representation

- **M x 2^E**
- **Exponent (E):** 8 bits
  - Add 127 to the exponent value before storing it
  - E can be 0 to 255 with 127 representing the real zero.
- **Fraction (M - Mantissa):** 23 bits
  - \(2^{128} = 1.70141183 \times 10^{38}\)
IEEE 32bit Floating-Point Number Representation

• Example: 12.375
• The digits before the dot:
  • \((12)_{10} \rightarrow (1100)_{2}\)
• The digits after the dot:
  • 1\textsuperscript{st} Way: 0.375 \rightarrow 0 \times \frac{1}{2} + 1 \times \frac{1}{4} + 1 \times \frac{1}{8} \rightarrow 011
  • 2\textsuperscript{nd} Way: Multiply by 2 and get the integer part until 0:
    • 0.375 \times 2 = 0.750 = 0 + 0.750
    • 0.750 \times 2 = 1.50 = 1 + 0.50
    • 0.50 \times 2 = 1.0 = 1 + 0.0
• \((12.375)_{10} = (1100.011)_{2}\)
• NORMALIZE: \((1100.011)_{2} = (1.100011)_{2} \times 2^{3}\)
• Exponent: 3, adding 127 to it, we get 1000 0010
• Fraction: 100011
• Then our number is: 0 10000010 100011000000000000000000
Why add bias to the exponent?

- It helps in comparing the exponents of the same-sign real-numbers without looking out for the sign of the exponent.
IEEE 32bit Floating-Point Number Representation

- **Zero:**
  - Exponent: *All zeros*
  - Fraction: *All zeros*
  - +0 and -0 are different numbers but they are equal!

- **Infinity:**
  - Exponent: *All ones*
  - Fraction: *All zeros*

- **Not a number (NaN):**
  - Exponent: *All ones*
  - Fraction: The most significant bit is one.

http://steve.hollasch.net/cgindex/coding/ieeefloat.html
IEEE 32bit Floating-Point Number Representation

■ What is the maximum positive IEEE floating point value that can be stored?
  ▪ Just less than $2^{129}$.

■ Check out these useful links:
  ▪ [http://steve.hollasch.net/cgindex/coding/ieeefloat.html](http://steve.hollasch.net/cgindex/coding/ieeefloat.html)
IEEE 32bit Floating-Point Number Representation

- Now consider 4.1:
  - 4 => \((100)_2\
  - 0.1 =>
    - \(x \times 2 = 0.2 = 0 + 0.2\)
    - \(x \times 2 = 0.4 = 0 + 0.4\)
    - \(x \times 2 = 0.8 = 0 + 0.8\)
    - \(x \times 2 = 1.6 = 1 + 0.6\)
    - \(x \times 2 = 1.2 = 1 + 0.2\)
    - \(x \times 2 = 0.4 = 0 + 0.4\)
    - \(x \times 2 = 0.8 = 0 + 0.8\)
    - .......

- So,
  - Representing a fraction which is a multiple of \(1/2^n\) is lossless.
  - Representing a fraction which is not a multiple of \(1/2^n\) leads to accuracy loss.