Today

• Review representations
• Floating point and hexadecimal
Unsigned integer representation

• With n bits, max value that can be represented: $2^n - 1$
Binary to Decimal conversion

\[
\begin{array}{ccccccc}
1 & 1 & 0 & 1 & 0 & 0 & 0 \\
2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
32 & 16 & + & 4 & & \\
=52 & & & & & \\
\end{array}
\]

6 bits, so max number possible is \(2^6 - 1 = 63\)
Binary to Decimal conversion

0 0 1 1 0 1

\[
\begin{array}{cccccc}
2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
8 & + & 4 & & + & 1 \\
\hline
=13
\end{array}
\]

6 bits, so max number possible is \(2^6 - 1 = 63\)
Binary to Decimal conversion (8 bits)

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
\begin{array}{cccccccc}
2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
128 & 64 & + & 8 & + & 4 & + & 1 \\
\end{array}
\]

=205

8 bits, so max number possible is \(2^8 - 1 = 255\)
Binary to Decimal conversion (8 bits)

1 1 1 1 1 1 1 1

\[\begin{array}{cccccccc}
2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \\
\end{array}\]

\[128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255\]

8 bits, so max number possible is \(2^8-1 = 255\)
Decimal to Binary

1. Find number of bits required
   \[ \text{Floor}(\log_2 \text{number}) + 1 \]

2. For each bit-position, starting from highest, Repeatedly check if number greater or equal to \(2^{\text{bit-position}}\), and set bit to 0 or 1 accordingly
Number of bits required

• **52**: \( \log_2(52) = 5.7; \) \( \text{floor}(5.7) = 5; \) # bits = 6
  - Check 1: \( 2^6 - 1 \leq 52; \) 63 < 52 (YES)
  - Check 2: \( 2^5 - 1 > 52; \) 32 > 52 (YES)

• **102**: \( \log_2(102) = 6.67; \) \( \text{floor}(6.67) = 6; \) # bits = 7
  - Check 1: \( 2^7 - 1 \leq 102; \) 127 < 107 (YES)
  - Check 2: \( 2^6 - 1 > 102; \) 63 > 107 (YES)

• **276**: \( \log_2(276) = 8.10; \) \( \text{floor}(8.10) = 8; \) # bits = 9
  - Check 1: \( 2^9 - 1 \leq 276; \) 511 < 276 (YES)
  - Check 2: \( 2^8 - 1 > 276; \) 255 > 276 (YES)
Number of bits required

• 64: \( \log_2(64) = 6.0; \, floor(6.0) = 6; \) # bits = 7
  - Check 1: \( 2^7 - 1 \leq 64; \, 127 < 64 \) (YES)
  - Check 2: \( 2^6 - 1 > 64; \, 63 > 64 \) (YES)
### Decimal to Binary

52; # bits = 6

Bit positions start at 0, so 6 bits means $2^0$ to $2^5$

<table>
<thead>
<tr>
<th>Bit position</th>
<th>Power of 2</th>
<th>Number</th>
<th>&gt;=</th>
<th>Remainder</th>
<th>Bit value</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>32</td>
<td>52</td>
<td>Yes</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
We can check

\[
\begin{array}{ccccccc}
1 & 1 & 0 & 1 & 0 & 0 & 0 \\
\hline
2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
32 & 16 & + & 4 \\
=52
\end{array}
\]
### Decimal to Binary

37; # bits = 6  
Bit positions start at 0

<table>
<thead>
<tr>
<th>Bit position</th>
<th>Power of 2</th>
<th>Number</th>
<th>&gt;=</th>
<th>Remainder</th>
<th>Bit value</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>32</td>
<td>37</td>
<td>Yes</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
We can check

\[
\begin{array}{ccccccc}
1 & 0 & 0 & 1 & 0 & 1 & 1 \\
2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
32 & 16 & 8 & 4 & 2 & 1 \\
\hline
32 & + & 4 & & +1 \\
37 & \\
\end{array}
\]
## Decimal to Binary

### 37; # bits = 6
### Bit positions start at 0

<table>
<thead>
<tr>
<th>Bit position</th>
<th>Power of 2</th>
<th>Number</th>
<th>&gt;=</th>
<th>Remainder</th>
<th>Bit value</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>32</td>
<td>37</td>
<td>Yes</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2’s complement representation

- Allows representing negative numbers
- **Arithmetic operations can be done by operating on individual bits**

- 0 is always all bits 0
- Most negative number: \(-2^{n-1}\)
- Most positive number: \(+2^{n-1} - 1\)
### 2’s complement range

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit</td>
<td>This is weird: -1 to 0</td>
<td></td>
</tr>
<tr>
<td>2 bits</td>
<td>-2</td>
<td>To</td>
</tr>
<tr>
<td>3 bits</td>
<td>-4</td>
<td>To</td>
</tr>
<tr>
<td>4 bits</td>
<td>-8</td>
<td>To</td>
</tr>
<tr>
<td>5 bits</td>
<td>-16</td>
<td>To</td>
</tr>
<tr>
<td>6 bits</td>
<td>-32</td>
<td>To</td>
</tr>
<tr>
<td>7 bits</td>
<td>-64</td>
<td>To</td>
</tr>
<tr>
<td>8 bits</td>
<td>-128</td>
<td>To</td>
</tr>
<tr>
<td>9 bits</td>
<td>-256</td>
<td>To</td>
</tr>
<tr>
<td>10 bits</td>
<td>-512</td>
<td>To</td>
</tr>
</tbody>
</table>
Decimal to Binary (2’s comp)

- First get number of bits
  \[ \text{Floor}(\log_2 |\text{number}|) + 2 \]
- If positive number, then use process we developed before and you are done
- If negative number,
  - First get representation of the absolute value
  - Then invert all bits
  - Then add +1 to the inverted bits
Decimal to Binary

• 52
1. # bits = 7
2. Positive number; 0110100

All 7 bits. Note that the MSB will always be zero in this step
Decimal to Binary

-52
1. # bits = 7
2. Negative number
   a) Representation of +52 = 0110100

b) Invert all bits: 1001011

c) Add +1: +0000001
   =1001100

All 7 bits. Note that the MSB will always be zero in this intermediate step

All 7 bits. Note that the MSB will always be ONE for negative numbers at the very end
Decimal to Binary

-101

1. # bits = 8
2. Negative number
   a) Representation of -101 = 01100101

   b) Invert all bits:
      10011010

c) Add +1:
   10011011

All 7 bits. Note that the MSB will always be zero in this step

All 7 bits. Note that the MSB will always be ONE for negative numbers at the very end
Decimal to Binary

-64
1. # bits = 7
2. Negative number
   a) Representation of +64 = 1000000

b) Invert all bits: 0111111
c) Add +1:
   0000001
   = 1000000

All 7 bits. Note that the MSB will always be zero in this intermediate step.

All 7 bits. Note that the MSB will always be ONE for negative numbers at the very end.
2’s complement binary to decimal

- If MSB is 0, same as unsigned
- If MSB is 1, reverse steps:
  a) Invert all bits
  b) Add +1
  c) Now determine magnitude
     Remember it is a negative number
Binary to decimal

- 1001100
- MSB is 1
  a) Invert all bits: 0110011
  b) Add +1: +00000001
     \[ \begin{array}{ccccccccc}
     \text{2^6} & \text{2^5} & \text{2^4} & \text{2^3} & \text{2^2} & \text{2^1} & \text{2^0} \\
     0 & 0 & 0 & 0 & 0 & 1 & 1
     \end{array} \]
     \[ 0110100 \]
\[ = 32 + 16 + 4 = 52 \]
  c) -52
  d) Go back to 2’s complement range and check
Binary to decimal

• 10011011
• MSB is 1
  
  a) Invert all bits: 01100100
  
  b) Add +1: +000000001

\[
\begin{array}{cccccccc}
2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}
\]

\[01100101 \]

= 64+32+4+1 = 101

b) Add +1: +000000001

\[
\begin{array}{cccccccc}
2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}
\]

\[01100101 \]

= 64+32+4+1 = 101

c) -101

d) Go back to 2’s complement range and check
2’s complement arithmetic
It’s bitwise addition!

• $52 + (-101) = -49$

  \[
  \begin{array}{c}
  00110100 \\
  +10011011 \\
  \hline
  11001111 \\
  \end{array}
  \]

• Let’s check what this value is
Check value

- 11001111
- MSB is 1

a) Invert all bits: 00110000

b) Add +1: +000000001

   00110000
   +000000001
   00110001

   \[ 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0 \]

   = 32+16+1 = 49

c) -49

d) Go back to 2’s complement range and check
2’s complement extension

-52 in 7 bits
-52 in 8 bits
-52 in 9 bits
-52 in 10 bits

-52

1. # bits = 7
2. Negative number
   a) Representation of +52 = 0110100
   b) Invert all bits: 1001011
   c) Add +1: +0000001
      = 1001100
2’s complement extension

-52

7 bits

a) Representation of +52 = 0110100

b) Invert all bits: 1001011

c) Add +1: +0000001

=1001100
Extension rule: 2s complement

1001100     7 bits
11001100     8 bits
111001100     9 bits

• To take a number represented in X bits can get its representation in Y bits, (Y > X), copy the **MSB** into the “new” bit positions
Fixed point

- After the decimal point negative powers of 2
- 0.001

\[
\begin{array}{cccccc}
1 & . & 1 & 0 & 1 & 0 \\
2^0 & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} \\
\end{array}
\]

2.625
Conversion from binary to decimal

- 0.43

<table>
<thead>
<tr>
<th>Power of 2</th>
<th>Weight</th>
<th>Number</th>
<th>&gt;=</th>
<th>Remainder</th>
<th>Bit value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.5</td>
<td>0.43</td>
<td>No</td>
<td>0.43</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
<td>0.43</td>
<td>Yes</td>
<td>0.18</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0.125</td>
<td>0.18</td>
<td>Yes</td>
<td>0.055</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>0.0625</td>
<td>0.055</td>
<td>No</td>
<td>0.055</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>0.03125</td>
<td>0.055</td>
<td>Yes</td>
<td>0.02375</td>
<td>1</td>
</tr>
<tr>
<td>64</td>
<td>0.015625</td>
<td>0.02375</td>
<td>Yes</td>
<td>0.008125</td>
<td>1</td>
</tr>
<tr>
<td>128</td>
<td>0.0078125</td>
<td>0.008125</td>
<td>Yes</td>
<td>0.0003125</td>
<td>1</td>
</tr>
</tbody>
</table>

Represented value 0.4296875
<table>
<thead>
<tr>
<th>Power of 2</th>
<th>Weight</th>
<th>Number</th>
<th>&gt;=</th>
<th>Remainder</th>
<th>Bit value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.5</td>
<td>0.17</td>
<td>No</td>
<td>0.17</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
<td>0.17</td>
<td>No</td>
<td>0.17</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0.125</td>
<td>0.17</td>
<td>Yes</td>
<td>0.045</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>0.0625</td>
<td>0.045</td>
<td>No</td>
<td>0.045</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>0.03125</td>
<td>0.045</td>
<td>Yes</td>
<td>0.01375</td>
<td>1</td>
</tr>
<tr>
<td>64</td>
<td>0.015625</td>
<td>0.01375</td>
<td>No</td>
<td>0.01375</td>
<td>0</td>
</tr>
<tr>
<td>128</td>
<td>0.0078125</td>
<td>0.01375</td>
<td>Yes</td>
<td>0.0059375</td>
<td>1</td>
</tr>
</tbody>
</table>

Represented value 0.1640625
<table>
<thead>
<tr>
<th>Power of 2</th>
<th>Weight</th>
<th>Number</th>
<th>&gt;=</th>
<th>Remainder</th>
<th>Bit value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.5</td>
<td>0.14</td>
<td>No</td>
<td>0.14</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
<td>0.14</td>
<td>No</td>
<td>0.14</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0.125</td>
<td>0.14</td>
<td>Yes</td>
<td>0.015</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>0.0625</td>
<td>0.015</td>
<td>No</td>
<td>0.015</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>0.03125</td>
<td>0.015</td>
<td>No</td>
<td>0.015</td>
<td>0</td>
</tr>
<tr>
<td>64</td>
<td>0.015625</td>
<td>0.015</td>
<td>No</td>
<td>0.015</td>
<td>0</td>
</tr>
<tr>
<td>128</td>
<td>0.0078125</td>
<td>0.015</td>
<td>Yes</td>
<td>0.0071875</td>
<td>1</td>
</tr>
</tbody>
</table>

Represented value: 0.1328125
<table>
<thead>
<tr>
<th>Power of 2</th>
<th>Weight</th>
<th>Number</th>
<th>&gt;=</th>
<th>Remainder</th>
<th>Bit value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.5</td>
<td>0.09</td>
<td>No</td>
<td>0.09</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
<td>0.09</td>
<td>No</td>
<td>0.09</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0.125</td>
<td>0.09</td>
<td>No</td>
<td>0.09</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0.0625</td>
<td>0.09</td>
<td>Yes</td>
<td>0.0275</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>0.03125</td>
<td>0.0275</td>
<td>No</td>
<td>0.0275</td>
<td>0</td>
</tr>
<tr>
<td>64</td>
<td>0.015625</td>
<td>0.0275</td>
<td>Yes</td>
<td>0.011875</td>
<td>1</td>
</tr>
<tr>
<td>128</td>
<td>0.0078125</td>
<td>0.011875</td>
<td>Yes</td>
<td>0.0040625</td>
<td>1</td>
</tr>
</tbody>
</table>

Represented value 0.0859375
<table>
<thead>
<tr>
<th>Power of 2</th>
<th>Weight</th>
<th>Number</th>
<th>&gt;=</th>
<th>Remainder</th>
<th>Bit value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.5</td>
<td>0.01</td>
<td>No</td>
<td>0.01</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
<td>0.01</td>
<td>No</td>
<td>0.01</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0.125</td>
<td>0.01</td>
<td>No</td>
<td>0.01</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0.0625</td>
<td>0.01</td>
<td>No</td>
<td>0.01</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>0.03125</td>
<td>0.01</td>
<td>No</td>
<td>0.01</td>
<td>0</td>
</tr>
<tr>
<td>64</td>
<td>0.015625</td>
<td>0.01</td>
<td>No</td>
<td>0.01</td>
<td>0</td>
</tr>
<tr>
<td>128</td>
<td>0.0078125</td>
<td>0.01</td>
<td>Yes</td>
<td>0.0021875</td>
<td>1</td>
</tr>
</tbody>
</table>

Represented value 0.0078125
<table>
<thead>
<tr>
<th>Power of 2</th>
<th>Weight</th>
<th>Number</th>
<th>&gt;=</th>
<th>Remainder</th>
<th>Bit value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.5</td>
<td>0.01</td>
<td>No</td>
<td>0.01</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
<td>0.01</td>
<td>No</td>
<td>0.01</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0.125</td>
<td>0.01</td>
<td>No</td>
<td>0.01</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0.0625</td>
<td>0.01</td>
<td>No</td>
<td>0.01</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>0.03125</td>
<td>0.01</td>
<td>No</td>
<td>0.01</td>
<td>0</td>
</tr>
<tr>
<td>64</td>
<td>0.015625</td>
<td>0.01</td>
<td>No</td>
<td>0.01</td>
<td>0</td>
</tr>
<tr>
<td>128</td>
<td>0.0078125</td>
<td>0.01</td>
<td>Yes</td>
<td>0.0021875</td>
<td>1</td>
</tr>
<tr>
<td>256</td>
<td>0.00390625</td>
<td>0.002188</td>
<td>No</td>
<td>0.0021875</td>
<td>0</td>
</tr>
<tr>
<td>512</td>
<td>0.001953125</td>
<td>0.002188</td>
<td>Yes</td>
<td>0.000234375</td>
<td>1</td>
</tr>
</tbody>
</table>

Represented value 0.009765625
Various conversions

- 1001001
- Decimal value interpreted as unsigned representation?
- Decimal value interpreted as 2’s complement representation?
- 0.1001001 = ?
Floating point

- Allows representing very large and very small numbers
- Standard used in all machines today
  - Much interesting theory behind it
- It is equivalent to scientific notation but done in binary.
- Think:
  - $0.145 = 1.45 \times 10^{-2}$
  - $0.0090897 = 9.0897 \times 10^{-3}$
  - $145 = 1.45 \times 10^{+2}$
- How do we do this with binary?
Floating Point Standard
IEEE-754 Standard
Single Precision Representation

1 bit          8 bits                   23 bits

| S | Exponent | Fraction |

- \( N = -1^S \times 1.fraction \times 2^{\text{exponent}-127} \)
  
  *when* \( 1 \leq \text{exponent} \leq 254 \)

- \( N = -1^S \times 0.fraction \times 2^{-126} \)
  
  *when* \( \text{exponent} == 0 \)
Example

110000001100100010000000000000000000000
1 100000001 10010000000000000000000000

S = 1; therefore negative number
exponent = 129
fraction = 1001000000000000000000000000

\[ N = -1^1 \times 1.1001 \times 2^{129-127} \]

\[ N = -1^1 \times 1.1001 \times 2^2 \]

\[ N = -1^1 \times 110.01 \]

\[ N = 6.25 \]
Example 2

010000101100100010000000000000000000000
0 10000101 10010000000000000000000000

S = 1; therefore negative number
exponent = 133
fraction = 10010000000000000000000000

\[ N = -1^1 \times 1.1001 \times 2^{133-127} \]

\[ N = -1^1 \times 1.1001 \times 2^5 \]

\[ N = -1^1 \times 110010 \]

\[ N = 50 \]
Hexadecimal notation

• Would be nice to not write so many bits!
• Simple scheme to represent 16 bits with one symbol
• 16 is a power of 2, in fact it is 2 raised 4.
• So one hexadecimal “symbol” represent 4 bits.
• The symbols are:
  0 through 9, a, b, c, d, e, f
<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Decimal</th>
<th>Binary (8 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0100 0010 1100 1000 1000 0000 0000 0000</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>1101</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>1110</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>
Unsigned integer representation

- With n bits, max value that can be represented: \(2^n - 1\)

Decimal to Binary

- 52; # bits = 6
- Bit positions start at 0, so 6 bits means \(2^5\) to \(2^0\)

<table>
<thead>
<tr>
<th>Bit position</th>
<th>Power of 2</th>
<th>Number</th>
<th>(\times)</th>
<th>Remainder</th>
<th>Bit value</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>32</td>
<td>52</td>
<td>Yes</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>20</td>
<td>Yes</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>8</td>
<td>No</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>Yes</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>Yes</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>No</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Decimal to Binary

- \(-52\)
  1. # bits = 7
  2. Negative number
     a) Representation of \(+52 = 0110100\)
     b) Invert all bits: \(1001011\)
     c) Add +1:
        \(+0000001\)
        \(-1001100\)

Extension rule: 2's complement

- To take a number represented in X bits can get its representation in Y bits, \(Y > X\), copy the MSB into the "new" bit positions
- All 7 bits, note that the MSB will always be ones in this intermediate step

2's complement binary to decimal

- If MSB is 0, same as unsigned
- If MSB is 1, reverse steps:
  a) Invert all bits
  b) Add +1
  c) Now determine magnitude
     Remember it is a negative number

2's complement representation

- Allows representing negative numbers
- Arithmetic operations can be done by operating on individual bits
- 0 is always all bits 0
- Most negative number: \(-2^{n-1}\)
- Most positive number: \(+2^{n-1} - 1\)

Binary to Decimal conversion

- 110100
- \(2^5 = 32\)
- \(2^4 = 16\)
- \(2^3 = 8\)
- \(2^2 = 4\)
- \(2^1 = 2\)
- \(2^0 = 1\)
- \(32 \times 2 + 16 + 4 = 52\)
- 6 bits, so max number possible is \(2^6 - 1 = 63\)

2's complement range

1. Find number of bits required
   \(\text{Floor}(\log_2 \text{number}) + 1\)
2. For each bit-position, starting from highest, repeatedly check if number greater or equal to \(2^{\text{bit-position}}\), and set bit to 0 or 1 accordingly

Number of bits required

- 52: \(\log_2(52) = 5.7; \text{Floor}(5.7) = 5; \# \text{bits} = 6\)
- Check 1: \(2^6 = 1 \leq 52; 63 < 52\) (YES)
- Check 2: \(2^5 = 1 > 52; 32 < 52\) (YES)
- 102: \(\log_2(102) = 6.67; \text{Floor}(6.67) = 6; \# \text{bits} = 7\)
- Check 1: \(2^7 = 1 \leq 102; 127 < 102\) (YES)
- Check 2: \(2^6 = 1 > 102; 63 > 107\) (YES)
- 276: \(\log_2(276) = 8.10; \text{Floor}(8.10) = 8; \# \text{bits} = 9\)
- Check 1: \(2^9 = 1 \leq 276; 511 \leq 276\) (YES)
- Check 2: \(2^8 = 1 > 276; 255 > 276\) (YES)

Decimal to Binary (2's comp)

- First get number of bits
  \(\text{Floor}(\log_2 \text{abs(number)}) + 2\)
- If positive number, then use process we developed before and you are done
- If negative number,
  - First get representation of the absolute value
  - Then invert all bits
  - Then add +1 to the inverted bits

2's complement arithmetic

- It's bitwise addition!
- 52 + (-101) = -49
- 00110100
  + 10011011
  \(-11001111\)

Steps:

1. Set S = 0
2. Exponent = 127
3. MSB = \((-1)^n\) * fraction * \(2^{exponent - 127}\) when \(1 \leq \text{exponent} \leq 254\)
4. MSB = \((-1)^n\) * fraction * \(2^{exponent - 112}\) when exponent = 0

Floating Point Standard

IEEE-754 Standard

Single-Precision Representation

<table>
<thead>
<tr>
<th>S</th>
<th>Exponent</th>
<th>Fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit</td>
<td>8 bits</td>
<td>23 bits</td>
</tr>
</tbody>
</table>

\(N = -1^S \cdot \text{fraction} \cdot 2^{\text{exponent} - 127}\)

\(N = -1^S \cdot \text{fraction} \cdot 2^{\text{exponent} - 112}\)
Representation

• Binary
• Integers
  – Sign magnitude representation
  – 2’s complement
• Decimal numbers (fixed point)
• Floating point
• Boolean Logic
• Hexadecimal digits
Review Number Representation

With n bit, the max value that can be presented is $2^n - 1$

Binary to Decimal Conversion

$110100 = 1*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 0*2^0 = 32 + 16 + 4 = 52$

$001101 = 0*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 8 + 4 + 1 = 13$

More bits means more weights

$11001101 = 1*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 128 + 64 + 8 + 4 + 1 = 205$

Decimal to Binary Conversion

Find the number of bits required

$\lfloor \log_2(\text{number}) \rfloor + 1$

For each bit-position, starting from highest, and set bit to 1 if greater than that position value

Number of bits required:

- 52: log base 2 of 52 = 5.7, so need 6 bits
- 102: log base 2 of 102 = 6.67, so need 7 bits
- 276: log base 2 of 276 = 8.10, so need 9 bits

Number Conversion:

52, 6 bits, $= 32 + 16 + 4 = 1*32 + 1*16 + 0*8 + 1*4 + 0*2 + 0*1 = 110100$

37, 6 bits, $= 32 + 4 + 1 = 1*32 + 0*16 + 0*8 + 1*4 + 0*2 + 1*1 = 100101$

2's complement representation: allows representing negative number

arithmetic operations can be done by operating on individual bits

most negative: $-2^{n-1}$

most positive: $2^{n-1} - 1$

2's complement range

1 bit: -1 to 0
2 bits: -2 to 1
3 bits: -4 to 3
4 bits: -8 to 7
5 bits: -16 to 15
6 bits: -32 to 31
7 bits: -64 to 53
8 bits: -128 to 127
9 bits: -256 to 255
10 bits: -512 to 511
Decimal to Binary (2’s complement)

If not negative

number of bits = \( \lfloor \log_2(\text{number}) \rfloor + 2 \)

Convert the same method as before

If negative

number of bits = \( \lfloor \log_2(\text{number} - 1) \rfloor + 2 \)  \( \text{(not mentioned in slides)} \)

Convert by

1. get positive value representation
2. invert all bits
3. add 1

52: 7 bits, 0110100 using the same method as before

-52: 7 bits, start with abs value 0110100; then invert all bits: 1001011; and add 1: 1001100

-101: 8 bits, start with abs value: 01100101; then invert all bits: 10011010; and add 1: 10011011

-64: 7 bits, start with abs value: 1000000; then invert all bits: 1011111; and add 1: 1000000

If the most significant bit (MSB) is 0, then same as unsigned

If MSB is 1, reverse steps to find decimal number

1001100: MSB is 1, so invert all bits 0110011; add 1: 0110100; convert to decimal: -52

In 2’s complement, arithmetic is bitwise addition!

52+(-101)=-49: 00110100 + 10011011 = 11001111

Homework: (-101) + (-163) = -264

10011011 + 10100011 = 1011111000

2’s complement sign extension:

-52 in 7 bits: 1001100
-52 in 8 bits: 11001100

52 in 8 bits is 00110100; invert all bits: 11001011; add 1: 11001100

-52 in 9 bits: 111001100
-52 in 10 bits: 1111001100

simply add more of the most significant bits

Fixed Point Representation

After the binary point, it presents negative powers of 2

1.1010 is \(1*2^0 + 1*2^-1 + 0*2^-2 + 1*2^-3 + 0*2^-4 = 1.625\)

Conversion from binary to decimal

0.43 is represented as 0.4296875 = 0.0110111 as shown in the table below

<table>
<thead>
<tr>
<th>Power of 2</th>
<th>weight</th>
<th>number</th>
<th>&gt;=</th>
<th>remainder</th>
<th>bit value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.5</td>
<td>0.43</td>
<td>no</td>
<td>0.43</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
<td>0.43</td>
<td>yes</td>
<td>0.18</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0.125</td>
<td>0.18</td>
<td>yes</td>
<td>0.055</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>0.0625</td>
<td>0.055</td>
<td>no</td>
<td>0.02375</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>0.03125</td>
<td>0.055</td>
<td>yes</td>
<td>0.008125</td>
<td>1</td>
</tr>
<tr>
<td>64</td>
<td>0.015625</td>
<td>0.02375</td>
<td>yes</td>
<td>0.0003125</td>
<td>1</td>
</tr>
</tbody>
</table>

0.17 is represented as 0.1640625 = 0.0010101

0.01 is represented as 0.0078125 = 0.00000001 or 0.00976525 = 0.000000101

Closer approximation as we increase the number of bits
Floating Point Representation

Allow representing very large and very small numbers
Standard used in all machines today
It is scientific notation done in binary

\[ 0.145 = 1.45 \times 10^{-2} \]
\[ 0.0090897 = 9.0897 \times 10^{-3} \]
\[ 145 = 1.45 \times 10^{2} \]

Format: 1 signed bit, 8 bits exponent, and 23 bits fraction, with an implicit 1 at the left of the binary point

If \( exponent = 0 \)
\[ N = (-1)^{s} \times 0.fraction \times 2^{-126} \]

Else
\[ N = (-1)^{s} \times 1.fraction \times 2^{exponent - 127} \]

Example 1
\[ \begin{align*}
1_{\text{10000001}} & \quad 1_{\text{00100000000000000000000000000000}} \\
S &= 1; \text{ therefore negative number} \\
exponent &= 129 \\
fraction &= 10010000000000000000000000000000 \\
N &= (-1)^{s} \times 1.1001 \times 2^{(129 - 127)} = 6.25
\end{align*} \]

Example 2
\[ \begin{align*}
0_{\text{1000101}} & \quad 0_{\text{10010000000000000000000000000000}} \\
S &= 1; \text{ therefore positive number} \quad \text{(typo in slides)} \\
exponent &= 133 \\
fraction &= 10010000000000000000000000000000 \\
N &= (-1)^{s} \times 0.1001 \times 2^{(133 - 127)} = 50
\end{align*} \]

Example 3
\[ 743.2 \]
\[ \begin{align*}
\text{fraction} &= 0110 \ 0111 \\
\text{exponent} &= 136 = 10001000 \\
\text{final representation} &= 0 \ 1001000 \ 0111 \ 0011 \ 1000 \ 0000 \ 0000 \ 0000
\end{align*} \]

Hexadecimal Representation

Convenience sake to avoid writing so many bits
Has nothing to do with how machines interpret numbers
4 bits is represented as 1 symbol: 0-9 and A-F
\[ \begin{align*}
0100 & \quad 0010 \quad 1100 \quad 1000 \quad 0000 \quad 0000 \quad 0000 = 4 \ 2 \ C \ 8 \ 8 \ 0 \ 0 \ 0 \\
\text{prefixed with 0x, 0X, 0h, or 0H} \\
\text{side note: binary numbers can be prefixed with 0b or 0B}
\end{align*} \]