Numbers and Arithmetic

Prof. Hakim Weatherspoon
CS 3410
Computer Science
Cornell University

[Weatherspoon, Bala, Bracy, and Sirer]
Big Picture: Building Processor

Simplified Single-cycle processor
Goals for Today

Binary Operations

- Number representations
- One-bit and four-bit adders
- Negative numbers and two’s compliment
- Addition (two’s compliment)
- Detecting and handling overflow
- Subtraction (two’s compliment)
Number Representations

Recall: Binary

• Two symbols (base 2): true and false; 1 and 0
• Basis of Logic Circuits and all digital computers

So, how do we represent numbers in *Binary* (base 2)?
Number Representations

Recall: Binary
- Two symbols (base 2): true and false; 1 and 0
- Basis of Logic Circuits and all digital computers

So, how do we represent numbers in Binary (base 2)?
- We can represent numbers in Decimal (base 10).
  - E.g. $6 \ 3 \ 7$
    - $10^2 \ 10^1 \ 10^0$

- Can just as easily use other bases
  - Base 2 — Binary
    - $101111101$
    - $2^9 \ 2^8 \ 2^7 \ 2^6 \ 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0$
  - Base 8 — Octal
    - $0o1175$
    - $8^3 \ 8^2 \ 8^1 \ 8^0$
  - Base 16 — Hexadecimal
    - $0x27d$
    - $16^2 \ 16^1 \ 16^0$
Number Representations

Recall: Binary
- Two symbols (base 2): true and false; 1 and 0
- Basis of Logic Circuits and all digital computers

So, how do we represent numbers in *Binary* (base 2)?
- We can represent numbers in Decimal (base 10).
  - E.g. 6 3 7
    \[
    6 \cdot 10^2 + 3 \cdot 10^1 + 7 \cdot 10^0 = 637
    \]
  - Can just as easily use other bases
    - Base 2 — Binary
      \[
      1 \cdot 2^9 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^0 = 637
      \]
    - Base 8 — Octal
      \[
      1 \cdot 8^3 + 1 \cdot 8^2 + 7 \cdot 8^1 + 5 \cdot 8^0 = 637
      \]
    - Base 16 — Hexadecimal
      \[
      2 \cdot 16^2 + 7 \cdot 16^1 + \text{d} \cdot 16^0 = 637
      \]
      \[
      2 \cdot 16^2 + 7 \cdot 16^1 + \text{13} \cdot 16^0 = 637
      \]
Number Representations: Activity #1
Counting

How do we count in different bases?

- **Dec (base 10)** | **Bin (base 2)** | **Oct (base 8)** | **Hex (base 16)**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>1 0000</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>1 0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>1 0010</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0b 1111 1111 = ?
0b 1 0000 0000 = ?
0o 77 = ?
0o 100 = ?
0x ff = ?
0x 100 = ?
Number Representations: Activity #1
Counting

How do we count in different bases?

- **Dec (base 10)** | **Bin (base 2)** | **Oct (base 8)** | **Hex (base 16)**

<table>
<thead>
<tr>
<th>Dec (base 10)</th>
<th>Bin (base 2)</th>
<th>Oct (base 8)</th>
<th>Hex (base 16)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>12</td>
<td>a</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>13</td>
<td>b</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>14</td>
<td>c</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>15</td>
<td>d</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>16</td>
<td>e</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>17</td>
<td>f</td>
</tr>
<tr>
<td>16</td>
<td>10000</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>17</td>
<td>10001</td>
<td>21</td>
<td>11</td>
</tr>
<tr>
<td>18</td>
<td>10010</td>
<td>22</td>
<td>12</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>99</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $0b\ 1111\ 1111 = 255$
- $0b\ 1\ 0000\ 0000 = 256$
- $0o\ 77 = 63$
- $0o\ 100 = 64$
- $0x\ ff = 255$
- $0x\ 100 = 256$
Number Representations

How to convert a number between different bases?

Base conversion via repetitive division

• Divide by base, write remainder, move left with quotient

• $637 \div 8 = 79$ remainder $5$ lsb (least significant bit)
• $79 \div 8 = 9$ remainder $7$
• $9 \div 8 = 1$ remainder $1$
• $1 \div 8 = 0$ remainder $1$ msb (most significant bit)

• $637 = 0o\,1175$

• $637 = 0_o\,1175$

Number Representations

Convert a base 10 number to a base 2 number

Base conversion via repetitive division

- Divide by base, write remainder, move left with quotient

- \( \frac{637}{2} = 318 \) remainder 1
- \( \frac{318}{2} = 159 \) remainder 0
- \( \frac{159}{2} = 79 \) remainder 1
- \( \frac{79}{2} = 39 \) remainder 1
- \( \frac{39}{2} = 19 \) remainder 1
- \( \frac{19}{2} = 9 \) remainder 1
- \( \frac{9}{2} = 4 \) remainder 1
- \( \frac{4}{2} = 2 \) remainder 0
- \( \frac{2}{2} = 1 \) remainder 0
- \( \frac{1}{2} = 0 \) remainder 1

637 = 10 0111 1101 (can also be written as 0b10 0111 1101)

lsb (least significant bit)

msb (most significant bit)
Clicker Question!

Convert the number $657_{10}$ to base 16
What is the least significant digit of this number?

a) D  
b) F  
c) 0  
d) 1  
e) 11
Clicker Question!

Convert the number $657_{10}$ to base 16
What is the least significant digit of this number?

a) D
b) F
c) 0
d) 1
e) 11
Number Representations

Convert a base 10 number to a base 16 number

Base conversion via repetitive division

- Divide by base, write remainder, move left with quotient
  - \( 657 \div 16 = 41 \) remainder \( 1 \)
  - \( 41 \div 16 = 2 \) remainder \( 9 \)
  - \( 2 \div 16 = 0 \) remainder \( 2 \)

Thus, \( 657 = 0x291 \)
Number Representations

Convert a base 10 number to a base 16 number

Base conversion via repetitive division

- Divide by base, write remainder, move left with quotient
- $637 \div 16 = 39$ remainder 13
- $39 \div 16 = 2$ remainder 7
- $2 \div 16 = 0$ remainder 2

$637 = 0x\ 2\ 7\ \underline{13}$ = ?

Thus, $637 = 0x27d$
Number Representations

Convert a base 2 number to base 8 (oct) or 16 (hex)

Binary to Hexadecimal

- Convert each nibble (group of four bits) from binary to hex
- A nibble (four bits) ranges in value from 0…15, which is one hex digit
  - Range: 0000…1111 (binary) => 0x0 …0xF (hex) => 0…15 (decimal)
- E.g. 0b 10 0111 1101
  
  2  7   d → 0x27d

- Thus, 637 = 0x27d = 0b10 0111 1101

Binary to Octal

- Convert each group of three bits from binary to oct
- Three bits range in value from 0…7, which is one octal digit
  - Range: 0000…1111 (binary) => 0x0 …0xF (hex) => 0…15 (decimal)
- E.g. 0b1 001 111 101

  1  1  7  5       → 0o1175

- Thus, 637 = 0o1175 = 0b10 0111 1101
Number Representations Summary

We can represent any number in any base

- **Base 10 — Decimal**
  
  \[
  6 \ 3 \ 7 \quad 10^2 \ 10^1 \ 10^0 \\
  \]
  
  \[6 \cdot 10^2 + 3 \cdot 10^1 + 7 \cdot 10^0 = 637\]

- **Base 2 — Binary**
  
  \[
  1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \\
  2^9 \ 2^8 \ 2^7 \ 2^6 \ 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0 \\
  \]
  
  \[1 \cdot 2^9 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^0 = 637\]

- **Base 8 — Octal**
  
  \[
  0 \ 1 \ 7 \ 5 \\
  8^3 \ 8^2 \ 8^1 \ 8^0 \\
  \]
  
  \[1 \cdot 8^3 + 1 \cdot 8^2 + 7 \cdot 8^1 + 5 \cdot 8^0 = 637\]

- **Base 16 — Hexadecimal**
  
  \[
  0 \ x \ 2 \ 7 \ d \\
  16^2 \ 16^1 \ 16^0 \\
  \]
  
  \[2 \cdot 16^2 + 7 \cdot 16^1 + \text{d} \cdot 16^0 = 637\]
  
  \[2 \cdot 16^2 + 7 \cdot 16^1 + \text{13} \cdot 16^0 = 637\]
Achievement Unlocked!

There are 10 types of people in the world:
Those who understand binary
And those who do not
And those who know this joke was written in base 2
Takeaway

Digital computers are implemented via logic circuits and thus represent *all* numbers in binary (base 2).

We (humans) often write numbers as decimal and hexadecimal for convenience, so need to be able to convert to binary and back (to understand what the computer is doing!).
Today’s Lecture

Binary Operations

• Number representations
• One-bit and four-bit adders
• Negative numbers and two’s compliment
• Addition (two’s compliment)
• Detecting and handling overflow
• Subtraction (two’s compliment)
Next Goal

Binary Arithmetic: Add and Subtract two binary numbers
Binary Addition

How do we do arithmetic in binary?

1

183

+ 254

437

• Addition works the same way regardless of base
  • Add the digits in each position
  • Propagate the carry

Unsigned binary addition is pretty easy
  • Combine two bits at a time
  • Along with a carry
Binary Addition

How do we do arithmetic in binary?
1  
183
+ 254
   ____
   437

- Addition works the same way regardless of base
  - Add the digits in each position
  - Propagate the carry

111
001110
+ 011100
   ____
  101010

Unsigned binary addition is pretty easy
- Combine two bits at a time
- Along with a carry
Binary Addition

- Binary addition requires
  - Add of *two bits* PLUS *carry-in*
  - Also, *carry-out* if necessary
1-bit Adder

Half Adder
- Adds two 1-bit numbers
- Computes 1-bit result and 1-bit carry
- No carry-in

**Clicker Question**
What is the equation for $C_{out}$?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>$C_{out}$</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a) $A + B$

b) $AB$

c) $A \oplus B$

d) $A + !B$

e) $!A!B$
1-bit Adder

Half Adder
- Adds two 1-bit numbers
- Computes 1-bit result and 1-bit carry
- No carry-in

Clicker Question
What is the equation for $C_{out}$?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>$C_{out}$</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

- a) $A + B$
- b) $AB$
- c) $A \oplus B$
- d) $A + !B$
- e) $!A!B$
1-bit Adder

Half Adder
- Adds two 1-bit numbers
- Computes 1-bit result and 1-bit carry
- No carry-in

- \( S = \overline{A}B + A\overline{B} \)
- \( C_{\text{out}} = AB \)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>( C_{\text{out}} )</th>
<th>S</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>
1-bit Adder

Half Adder
- Adds two 1-bit numbers
- Computes 1-bit result and 1-bit carry
- No carry-in

- $S = \overline{A}B + AB = A \oplus B$
- $C_{out} = AB$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>$C_{out}$</th>
<th>S</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>
1-bit Adder with Carry

Full Adder
- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry
- Can be cascaded

Now You Try:
1. Fill in Truth Table
2. Create Sum-of-Product Form
3. Minimization the equation
   1. Karnaugh Maps (coming soon!)
   2. Algebraic minimization
4. Draw the Logic Circuits
1-bit Adder with Carry

Full Adder
- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry
- Can be cascaded

Clicker Question
What is the equation for Cout?
- a) \( A + B + C_{in} \)
- b) \(!A + !B + !C_{in}\)
- c) \(A \oplus B \oplus C_{in}\)
- d) \(AB + AC_{in} + BC_{in}\)
- e) \(ABC_{in}\)
1-bit Adder with Carry

Full Adder
- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry
- Can be cascaded

Clicker Question
What is the equation for $C_{out}$?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>$C_{in}$</th>
<th>$C_{out}$</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</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>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

a) $A + B + C_{in}$
b) $!A + !B + !C_{in}$
c) $A \oplus B \oplus C_{in}$
d) $AB + AC_{in} + BC_{in}$
e) $ABC_{in}$
1-bit Adder with Carry

Full Adder
- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry
- Can be cascaded

S = \overline{A}BC + \overline{A}B\overline{C} + AB\overline{C} + ABC
C_{out} = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C_{in}</th>
<th>C_{out}</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<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>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
1-bit Adder with Carry

Full Adder
- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry
- Can be cascaded

\[
S = \overline{A}BC + \overline{A}B\overline{C} + AB\overline{C} + ABC
\]
\[
C_{out} = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC
\]
1-bit Adder with Carry

Full Adder
- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry
- Can be cascaded

\[ S = \overline{ABC} + \overline{A}B\overline{C} + AB\overline{C} + ABC \]
\[ C_{out} = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC \]
\[ C_{out} = AB + AC + BC \]
1-bit Adder with Carry

Full Adder
- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry
- Can be cascaded

\[
S = \overline{A}BC + \overline{A}B\overline{C} + A\overline{B}C + ABC
\]

\[
S = \overline{A}(BC + B\overline{C}) + A(\overline{B}C + BC)
\]

\[
S = \overline{A}(B \oplus C) + A(\overline{B} \oplus C)
\]

\[
S = A \oplus (B \oplus C)
\]
1-bit Adder with Carry

Full Adder
- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry
- Can be cascaded

\[
S = \overline{A}BC + \overline{A}B\overline{C} + A\overline{B}C + ABC
\]
\[
S = \overline{A}(BC + B\overline{C}) + A(\overline{B}C + BC)
\]
\[
S = \overline{A}(B \oplus C) + A(\overline{B} \oplus \overline{C})
\]
\[
S = A \oplus (B \oplus C)
\]
\[
C_{out} = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC
\]
\[
C_{out} = AB + AC + BC
\]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C_{in}</th>
<th>C_{out}</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<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>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Lab1  1-bit Adder with Carry

\[ S = \overline{A}BC + \overline{A}B\overline{C} + A\overline{B}C + ABC \]
\[ S = \overline{A}(\overline{B}C + B\overline{C}) + A(B\overline{C} + BC) \]
\[ S = \overline{A}(B \oplus C) + A(B \oplus \overline{C}) \]
\[ S = A \oplus (B \oplus C) \]

\[ C_{out} = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC \]
\[ C_{out} = \overline{A}BC + A\overline{B}C + AB(\overline{C} + C) \]
\[ C_{out} = \overline{A}BC + A\overline{B}C + AB \]
\[ C_{out} = (\overline{A}B + A\overline{B})C + AB \]
\[ C_{out} = (A \oplus B)C + AB \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C_{in}</th>
<th>C_{out}</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<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>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
4-bit Adder

4-Bit Full Adder

- Adds two 4-bit numbers and carry in
- Computes 4-bit result and carry out
- Can be cascaded
4-bit Adder

- Adds two 4-bit numbers, along with carry-in
- Computes 4-bit result and carry out
- Carry-out = overflow indicates result does not fit in 4 bits
4-bit Adder

- Adds two 4-bit numbers, along with carry-in
- Computes 4-bit result and carry out
- Carry-out = overflow indicates result does not fit in 4 bits
Takeaway

Digital computers are implemented via logic circuits and thus represent *all* numbers in binary (base 2).

We (humans) often write numbers as decimal and hexadecimal for convenience, so need to be able to convert to binary and back (to understand what computer is doing!).

Adding two 1-bit numbers generalizes to adding two numbers of any size since 1-bit full adders can be cascaded.
Today’s Lecture

Binary Operations

- Number representations
- One-bit and four-bit adders
- Negative numbers and two’s compliment
- Addition (two’s compliment)
- Detecting and handling overflow
- Subtraction (two’s compliment)
Next Goal

How do we subtract two binary numbers?
Equivalent to adding with a negative number

How do we represent negative numbers?
1st Attempt: Sign/Magnitude Representation

- First Attempt: Sign/Magnitude Representation
  - 1 bit for sign (0=positive, 1=negative)
    \[0111 = 7\]
    \[1111 = -7\]
  - N-1 bits for magnitude

Problem?
- Two zero’s: +0 \[0000 = +0\]
  different than -0 \[1000 = -0\]
- Complicated circuits
- \(-2 + 1 = ???\)

IBM 7090, 1959:
“a second-generation transistorized version of the earlier IBM 709 vacuum tube mainframe computers”
Second Attempt: One’s complement

- Second Attempt: One’s complement
  - Leading 0’s for positive and 1’s for negative
  - Negative numbers: complement the positive number

\[
\begin{align*}
0111 & = 7 \\
1000 & = -7
\end{align*}
\]

- Problem?
  - Two zero’s still: +0 different than -0
  - -1 if offset from two’s complement
  - Complicated circuits
    - Carry is difficult

\[
\begin{align*}
0000 & = +0 \\
1111 & = -0
\end{align*}
\]
Two’s Complement Representation

What is used: Two’s Complement Representation

Nonnegative numbers are represented as usual
- 0 = 0000, 1 = 0001, 3 = 0011, 7 = 0111

Leading 1’s for negative numbers
To negate any number:
- complement all the bits (i.e. flip all the bits)
- then add 1
- -1: 1 ⇒ 0001 ⇒ 1110 ⇒ 1111
- -3: 3 ⇒ 0011 ⇒ 1100 ⇒ 1101
- -7: 7 ⇒ 0111 ⇒ 1000 ⇒ 1001
- -8: 8 ⇒ 1000 ⇒ 0111 ⇒ 1000
- -0: 0 ⇒ 0000 ⇒ 1111 ⇒ 0000 (this is good, -0 = +0)
## Two’s Complement

<table>
<thead>
<tr>
<th>Non-negatives (as usual):</th>
<th>Negatives (two’s complement)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0 = 0000</td>
<td>( \overline{0} = 1111 )</td>
</tr>
<tr>
<td>+1 = 0001</td>
<td>( \overline{1} = 1110 )</td>
</tr>
<tr>
<td>+2 = 0010</td>
<td>( \overline{2} = 1101 )</td>
</tr>
<tr>
<td>+3 = 0011</td>
<td>( \overline{3} = 1100 )</td>
</tr>
<tr>
<td>+4 = 0100</td>
<td>( \overline{4} = 1011 )</td>
</tr>
<tr>
<td>+5 = 0101</td>
<td>( \overline{5} = 1010 )</td>
</tr>
<tr>
<td>+6 = 0110</td>
<td>( \overline{6} = 1001 )</td>
</tr>
<tr>
<td>+7 = 0111</td>
<td>( \overline{7} = 1000 )</td>
</tr>
<tr>
<td>+8 = 1000</td>
<td>( \overline{8} = 0111 )</td>
</tr>
<tr>
<td></td>
<td>(-0 = 0000 )</td>
</tr>
<tr>
<td></td>
<td>(-1 = 1111 )</td>
</tr>
<tr>
<td></td>
<td>(-2 = 1110 )</td>
</tr>
<tr>
<td></td>
<td>(-3 = 1101 )</td>
</tr>
<tr>
<td></td>
<td>(-4 = 1100 )</td>
</tr>
<tr>
<td></td>
<td>(-5 = 1011 )</td>
</tr>
<tr>
<td></td>
<td>(-6 = 1010 )</td>
</tr>
<tr>
<td></td>
<td>(-7 = 1001 )</td>
</tr>
<tr>
<td></td>
<td>(-8 = 1000 )</td>
</tr>
</tbody>
</table>
# Two’s Complement

<table>
<thead>
<tr>
<th>Non-negatives (as usual):</th>
<th>Negatives (two’s complement)</th>
</tr>
</thead>
</table>
| +0 = 0000                | \(\overline{0} = 1111\)  
| +1 = 0001                | \(\overline{1} = 1110\)  
| +2 = 0010                | \(\overline{2} = 1101\)  
| +3 = 0011                | \(\overline{3} = 1100\)  
| +4 = 0100                | \(\overline{4} = 1011\)  
| +5 = 0101                | \(\overline{5} = 1010\)  
| +6 = 0110                | \(\overline{6} = 1001\)  
| +7 = 0111                | \(\overline{7} = 1000\)  
| +8 = 1000                | \(\overline{8} = 0111\)  

-0 = 0000  
-1 = 1111  
-2 = 1110  
-3 = 1101  
-4 = 1100  
-5 = 1011  
-6 = 1010  
-7 = 1001  
-8 = 1000
### Two’s Complement vs. Unsigned

<table>
<thead>
<tr>
<th>4 bit Two’s Complement</th>
<th>4 bit Unsigned Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>0</td>
</tr>
<tr>
<td>-7</td>
<td>0100</td>
</tr>
<tr>
<td>-6</td>
<td>0110</td>
</tr>
<tr>
<td>-5</td>
<td>0111</td>
</tr>
<tr>
<td>-4</td>
<td>1000</td>
</tr>
<tr>
<td>-3</td>
<td>1001</td>
</tr>
<tr>
<td>-2</td>
<td>1010</td>
</tr>
<tr>
<td>-1</td>
<td>1011</td>
</tr>
<tr>
<td>0</td>
<td>1100</td>
</tr>
<tr>
<td>1</td>
<td>1101</td>
</tr>
<tr>
<td>2</td>
<td>1110</td>
</tr>
<tr>
<td>3</td>
<td>1111</td>
</tr>
</tbody>
</table>

-1 = 1111 = 15
-2 = 1110 = 14
-3 = 1101 = 13
-4 = 1100 = 12
-5 = 1011 = 11
-6 = 1010 = 10
-7 = 1001 = 9
-8 = 1000 = 8
+7 = 0111 = 7
+6 = 0110 = 6
+5 = 0101 = 5
+4 = 0100 = 4
+3 = 0011 = 3
+2 = 0010 = 2
+1 = 0001 = 1
0 = 0000 = 0
Clicker Question!

What is the value of the 2s complement number 11010?

a) 26  
b) 6  
c) -6  
d) -10  
e) -26
Clicker Question!

What is the value of the 2's complement number 11010?

a) 26  
b) 6  
c) -6  
d) -10  
e) -26

11010

00101 (flip) +1

-6 = 00110
Two’s Complement Facts

Signed two’s complement
   Negative numbers have leading 1’s
   zero is unique: +0 = - 0
   wraps from largest positive to largest negative

N bits can be used to represent

unsigned: range 0…$2^N - 1$
   eg: 8 bits ⇒ 0…255

signed (two’s complement): -(2^{N-1})…(2^{N-1} - 1)
   E.g.: 8 bits ⇒ (1000 000) … (0111 1111)
   -128 … 127
Sign Extension & Truncation

Extending to larger size
  • 1111 = -1
  • 1111 1111 = -1
  • 0111 = 7
  • 0000 0111 = 7

Truncate to smaller size
  • 0000 1111 = 15
  • BUT, 0000 1111 = 1111 = -1
Two’s Complement Addition

- Addition with two’s complement signed numbers
- Addition as usual. Ignore the sign. It just works!
- Examples
  - 1 + -1 =
  - -3 + -1 =
  - -7 + 3 =
  - 7 + (-3) =
Two’s Complement Addition

- Addition with two’s complement signed numbers
- Addition as usual. Ignore the sign. It just works!
- Examples
  - $1 + (-1) = 0001 + 1111 = -1 = 1111 = 15$
  - $-3 + (-1) = 1101 + 1111 = -2 = 1110 = 14$
  - $-7 + 3 = 1001 + 0011 = -3 = 1101 = 13$
  - $7 + (-3) = 0111 + 1101 = -4 = 1100 = 12$
  - $-5 = 1011 = 11$
  - $-6 = 1010 = 10$
  - $-7 = 1001 = 9$
  - $-8 = 1000 = 8$
  - $+7 = 0111 = 7$
  - $+6 = 0110 = 6$
  - $+5 = 0101 = 5$
  - $+4 = 0100 = 4$
  - $+3 = 0011 = 3$
  - $+2 = 0010 = 2$
  - $+1 = 0001 = 1$
  - $0 = 0000 = 0$
Two’s Complement Addition

- Addition with two’s complement signed numbers
- Addition as usual. Ignore the sign. It just works!
- Examples
  - 1 + -1 = 0001 + 1111 = 0000 (0)
  - -3 + -1 = 1101 + 1111 = 1100 (-4)
  - -7 + 3 = 1001 + 0011 = 1100 (-4)
  - 7 + (-3) = 0111 + 1101 = 0100 (4)

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

• Addition with two’s complement signed numbers
• Addition as usual. Ignore the sign. It just works!
• Examples
  • 1 + -1 = 0001 + 1111 = 0000 (0) -1 = 1111 = 15
  • -3 + -1 = 1101 + 1111 = 1100 (-4) -2 = 1110 = 14
  • -7 + 3 = 1001 + 0011 = 1100 (-4) -3 = 1101 = 13
  • 7 + (-3) = 0111 + 1101 = 0100 (4) -4 = 1100 = 12
  • 7 + (-3) = 0111 + 1101 = 0100 (4)

Clicker Question
Which of the following has problems?

a) 7 + 1
b) -7 + -3
c) -7 + -1
d) Only (a) and (b) have problems
e) They all have problems

-1 = 1111 = 15
-2 = 1110 = 14
-3 = 1101 = 13
-4 = 1100 = 12
-5 = 1011 = 11
-6 = 1010 = 10
-7 = 1001 = 9
-8 = 1000 = 8
+7 = 0111 = 7
+6 = 0110 = 6
+5 = 0101 = 5
+4 = 0100 = 4
+3 = 0011 = 3
+2 = 0010 = 2
+1 = 0001 = 1
0 = 0000 = 0
Two’s Complement Addition

• Addition with two’s complement signed numbers
• Addition as usual. Ignore the sign. It just works!
• Examples
  • $1 + (-1) = 0001 + 1111 = 0000 (0)$
  • $-3 + (-1) = 1101 + 1111 = 1100 (-4)$
  • $-7 + 3 = 1001 + 0011 = 1100 (-4)$
  • $7 + (-3) = 0111 + 1101 = 0100 (4)$

Clicker Question
Which of the following has problems?

a) $7 + 1 = 1000$

b) $-7 + -3 = 10110$

c) $-7 + -1 = 1000$

d) Only (a) and (b) have problems

e) They all have problems
Two’s Complement Addition

- Addition with two’s complement signed numbers
- Addition as usual. Ignore the sign. It just works!
- Examples

- $1 + (-1) = 0001 + 1111 = 0000$ (0)
- $-3 + (-1) = 1101 + 1111 = 1100$ (-4)
- $-7 + 3 = 1001 + 0011 = 1100$ (-4)
- $7 + (-3) = 0111 + 1101 = 0100$ (4)

Clicker Question

Which of the following has problems?

- **a)** $7 + 1 = 1000$ overflow
- **b)** $-7 + -3 = 10110$ overflow
- **c)** $-7 + -1 = 1000$ fine
- **d)** Only (a) and (b) have problems
- **e)** They all have problems

-1 = 1111 = 15
-2 = 1110 = 14
-3 = 1101 = 13
-4 = 1100 = 12
-5 = 1011 = 11
-6 = 1010 = 10
-7 = 1001 = 9
-8 = 1000 = 8
+7 = 0111 = 7
+6 = 0110 = 6
+5 = 0101 = 5
+4 = 0100 = 4
+3 = 0011 = 3
+2 = 0010 = 2
+1 = 0001 = 1
0 = 0000 = 0
Next Goal

In general, how do we detect and handle overflow?
Overflow

When can overflow occur?
- adding a negative and a positive?
  - Overflow cannot occur (Why?)
  - Always subtract larger magnitude from smaller
- adding two positives?
  - Overflow can occur (Why?)
  - Precision: Add two positives, and get a negative number!
- adding two negatives?
  - Overflow can occur (Why?)
  - Precision: add two negatives, get a positive number!

Rule of thumb:
- Overflow happens iff carry into msb != carry out of msb
Overflow

When can overflow occur?
• adding a negative and a positive?
  - Overflow cannot occur (Why?)
  - Always subtract larger magnitude from smaller
• adding two positives?
  - Overflow can occur (Why?)
  - Precision: Add two positives, and get a negative number!
• adding two negatives?
  - Overflow can occur (Why?)
  - Precision: add two negatives, get a positive number!

Rule of thumb:
• Overflow happens iff
carry into msb != carry out of msb
Overflow

When can overflow occur?
- adding a negative and a positive?
  - Overflow *cannot occur* (Why?)
  - Always subtract larger magnitude from smaller magnitude
- adding two positives?
  - Overflow *can occur* (Why?)
  - Precision: Add two positives, get a negative number!
- adding two negatives?
  - Overflow *can occur* (Why?)
  - Precision: add two negatives, get a positive number!

Rule of thumb:
- Overflow happens iff carry into msb != carry out of msb
We never thought a video would be watched in numbers greater than a 32-bit integer (=2,147,483,647 views), but that was before we met PSY. "Gangnam Style" has been viewed so many times we had to upgrade to a 64-bit integer (9,223,372,036,854,775,808)!

Hover over the counter in PSY's video to see a little math magic and stay tuned for bigger and bigger numbers on YouTube.
Today’s Lecture

Binary Operations

• Number representations
• One-bit and four-bit adders
• Negative numbers and two’s compliment
• Addition (two’s compliment)
• Detecting and handling overflow
• Subtraction (two’s compliment)
Binary Subtraction

Why create a new circuit?
Just use addition using two’s complement math
How?
Binary Subtraction

- Two’s Complement Subtraction
  - Subtraction is simply addition, where one of the operands has been negated
    - Negation is done by inverting all bits and adding one
      \[ A - B = A + (-B) = A + (\overline{B} + 1) \]
Binary Subtraction

- Two’s Complement Subtraction
- Subtraction is simply addition, where one of the operands has been negated
  - Negation is done by inverting all bits and adding one
    \[ A - B = A + (-B) = A + (\overline{B} + 1) \]

Q: How do we detect and handle overflows?
Q: What if \((-B)\) overflows?
Two’s Complement Adder

Two’s Complement Adder with overflow detection
Two’s Complement Adder

Two’s Complement Adder with overflow detection
Two’s Complement Adder

Two’s Complement Adder with overflow detection

Note: 4-bit adder is drawn for illustrative purposes and may not represent the optimal design.
Two’s Complement Adder

Two’s Complement Adder with overflow detection

Note: 4-bit adder is drawn for illustrative purposes and may not represent the optimal design.
Two’s Complement Adder

Two’s Complement Adder with overflow detection
Two’s Complement Adder

Two’s Complement Adder with overflow detection

Before: 2 inverters, 2 AND gates, 1 OR gate
After: 1 XOR gate
Takeaways

Digital computers are implemented via logic circuits and thus represent *all* numbers in binary (base 2).

We write numbers as decimal or hex for convenience and need to be able to convert to binary and back (to understand what the computer is doing!).

Adding two 1-bit numbers generalizes to adding two numbers of any size since 1-bit full adders can be cascaded.

Using Two’s complement number representation simplifies adder Logic circuit design (0 is unique, easy to negate). Subtraction is adding, where one operand is negated (two’s complement; to negate: flip the bits and add 1).

Overflow if sign of operands A and B != sign of result S. Can detect overflow by testing $C_{in} \neq C_{out}$ of the most significant bit (msb), which only occurs when previous statement is true.
Summary

We can now implement combinational logic circuits

• Design each block
  - Binary encoded numbers for compactness
• Decompose large circuit into manageable blocks
  - 1-bit Half Adders, 1-bit Full Adders,
  \[ n \text{-bit Adders via cascaded 1-bit Full Adders, ...} \]
• Can implement circuits using NAND or NOR gates
• Can implement gates using use PMOS and NMOS-transistors
• And can add and subtract numbers (in two’s compliment)!
• Next time, state and finite state machines…