Integer Numbers

The Number Bases of Integers

Textbook Chapter 2 +
Number Systems

- Unary, or marks:
  - ////////////// = 7
  - ////////////// + ////////////// = /////////////////

- Grouping lead to Roman Numerals:
  - VII + V = XVII = XII

- Better: Arabic Numerals:
  - 7 + 5 = 12 = 1 \cdot 10 + 2
Positional Number System

- The value represented by a digit depends on its *position* in the number.
- Ex: 1832
  - 1000
  - 100
  - 10
  - 1
Sexagesimal: A Positional Number System

- First used over 4000 years ago in Mesopotamia
- Base 60 (Sexagesimal), alphabet: 1..60, written as 60 different groups
- But the Babylonians used only two symbols, 1 and 10, and didn’t have the zero
  - Needed context to tell 1 from 60!
- Example
  - $5,45_{60} = 5 \times 60^1 + 45 \cdot 60^0 = 300 + 45 = 345$
# Babylonian Numbers

<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>11</td>
<td>21</td>
<td>31</td>
<td>41</td>
<td>51</td>
<td>61</td>
<td>71</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>22</td>
<td>32</td>
<td>42</td>
<td>52</td>
<td>62</td>
<td>72</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>23</td>
<td>33</td>
<td>43</td>
<td>53</td>
<td>63</td>
<td>73</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>24</td>
<td>34</td>
<td>44</td>
<td>54</td>
<td>64</td>
<td>74</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>25</td>
<td>35</td>
<td>45</td>
<td>55</td>
<td>65</td>
<td>75</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>26</td>
<td>36</td>
<td>46</td>
<td>56</td>
<td>66</td>
<td>76</td>
</tr>
<tr>
<td>7</td>
<td>17</td>
<td>27</td>
<td>37</td>
<td>47</td>
<td>57</td>
<td>67</td>
<td>77</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
<td>28</td>
<td>38</td>
<td>48</td>
<td>58</td>
<td>68</td>
<td>78</td>
</tr>
<tr>
<td>9</td>
<td>19</td>
<td>29</td>
<td>39</td>
<td>49</td>
<td>59</td>
<td>69</td>
<td>79</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
</tr>
</tbody>
</table>
Positional Number Systems

- Select a number as the base $b$
- Define an alphabet of $b-1$ symbols plus a symbol for zero to represent all numbers $0, 1-9$
- Use an ordered sequence of 1 or more digits $d$ to represent numbers
- The represented number is the sum of all digits, each multiplied by $b$ to the power of the digit’s position $p$

$$\text{Number} = \sum_{p=0}^{\text{num digits}} (d_p \cdot b^p)$$
Arabic/Indic Numerals

• Base (or radix): 10 (decimal)
  – The alphabet (digits or symbols) is 0..9
• Ours based on the Arabic symbols
  – Has the ZERO!!!
• Numerals introduced to Europe by Leonardo Fibonacci in his Liber Abaci
  – In 1202
  – So useful!
Arabic/Indic Numerals

- The Italian mathematician Leonardo Fibonacci
- Also known for the Fibonacci sequence
  - 1, 1, 2, 3, 5, 8, 13, 21

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>European</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arabic-Indic</td>
<td></td>
<td>٠</td>
<td>١</td>
<td>٢</td>
<td>٣</td>
<td>٤</td>
<td>٥</td>
<td>٦</td>
<td>٧</td>
<td>٨</td>
</tr>
<tr>
<td>Eastern Arabic-Indic</td>
<td></td>
<td>٠</td>
<td>١</td>
<td>٢</td>
<td>٣</td>
<td>٤</td>
<td>٥</td>
<td>٦</td>
<td>٧</td>
<td>٨</td>
</tr>
<tr>
<td>Devanagari (Hindi)</td>
<td>०</td>
<td>१</td>
<td>२</td>
<td>३</td>
<td>४</td>
<td>५</td>
<td>६</td>
<td>७</td>
<td>८</td>
<td>९</td>
</tr>
<tr>
<td>Tamil</td>
<td>ம</td>
<td>ஒ</td>
<td>ஒ</td>
<td>ஒ</td>
<td>ஒ</td>
<td>ஒ</td>
<td>ஒ</td>
<td>ஒ</td>
<td>ஒ</td>
<td>ஒ</td>
</tr>
</tbody>
</table>
Numbers for Computers

- There are many ways to represent a number
- Representation does not affect computation result
- Representation affects *difficulty* of computing results
- Computers need a representation that works with (fast) electronic circuits
- Computers generally only have 2 states
Binary Number System

- Base (radix): 2
- Digits (symbols): 0, 1
- Binary Digits, or bits

Example:

\[1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 1 = 9_{10}\]

\[1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 16 + 8 = 24_{10}\]
Knowing The Powers Of Two

- Know them in your sleep

<table>
<thead>
<tr>
<th>$2^0$</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^1$</td>
<td>2</td>
</tr>
<tr>
<td>$2^2$</td>
<td>4</td>
</tr>
<tr>
<td>$2^3$</td>
<td>8</td>
</tr>
<tr>
<td>$2^4$</td>
<td>16</td>
</tr>
<tr>
<td>$2^5$</td>
<td>32</td>
</tr>
<tr>
<td>$2^6$</td>
<td>64</td>
</tr>
<tr>
<td>$2^7$</td>
<td>128</td>
</tr>
<tr>
<td>$2^8$</td>
<td>256</td>
</tr>
<tr>
<td>$2^9$</td>
<td>512</td>
</tr>
<tr>
<td>$2^{10}$</td>
<td>1024</td>
</tr>
</tbody>
</table>
Octal Number System

- Base (radix): 8
- Digits (symbols): 0, 1, 2, 3, 4, 5, 6, 7
- \[345_8 = 3 \times 8^2 + 4 \times 8^1 + 5 \times 8^0\]
  \[= 3 \times 64 + 4 \times 8 + 5 \times 1 = 192 + 32 + 5 = 229_{10}\]
- \[1001_8 = 1 \times 8^3 + 0 \times 8^2 + 0 \times 8^1 + 1 \times 8^0\]
  \[= 1 \times 512 + 0 \times 64 + 0 \times 8 + 1 \times 1 = 512 + 1 = 513_{10}\]
- In C, octal numbers are represented with a leading 0 (0345 or 01001).
Hexadecimal Number System

- Base (radix): 16
- Digits (symbols): 0-9, A–F (a–f)
- In C: leading “0x” (e.g., 0xa3)
- In LC-3: leading “x” (e.g., “x3000”)
- Hexadecimal is also known as “hex” for short

<table>
<thead>
<tr>
<th>Hex</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
</tr>
<tr>
<td>C</td>
<td>12</td>
</tr>
<tr>
<td>D</td>
<td>13</td>
</tr>
<tr>
<td>E</td>
<td>14</td>
</tr>
<tr>
<td>F</td>
<td>15</td>
</tr>
</tbody>
</table>

F is 15
Examples of Converting Hex to Decimal

- $A3_{16} = A \times 16^1 + 3 \times 16^0$
  
  $= 10 \times 16 + 3 \times 1$
  
  $= 160 + 3$
  
  $= 163_{10}$

- $3E8_{16} = 3 \times 16^2 + E \times 16^1 + 8 \times 16^0$
  
  $= 3 \times 256 + 14 \times 16 + 8 \times 1$
  
  $768 + 224 + 8$
  
  $= 100010_{10}$
Base Conversion

Three cases:

I. From any base $b$ to base 10
II. From base 10 to any base $b$
III. From any base $b$ to any other base $c$
From Base $b$ to Base 10

- Base (radix): $b$
- Digits (symbols): 0 ... $(b - 1)$
- $S_{n-1}S_{n-2}...S_2S_1S_0$

Value = $\sum_{i=0}^{n-1} (S_i b^i)$

Use summation to transform any base to decimal
From Base $b$ to Base 10

- Example: $1234_5 = ?_{10}$

$$1 \cdot 5^3 + 2 \cdot 5^2 + 3 \cdot 5^1 + 4 \cdot 5^0$$

$$= 1 \cdot 125 + 2 \cdot 25 + 3 \cdot 5 + 4 \cdot 1$$

$$= 194_{10}$$
From Base 10 to Base $b$ – Method 1

- Use successive divisions
- Remember the remainders
- Divide again with the quotients
From Base 10 to Base $b$ – Method 1

- Example: $2010_{10} = ?_5$

2010/5 = 402 R 0
402/5 = 80 R 2
80/5 = 16 R 0
16/5 = 3 R 1
3/5 = 0 R 3

31020

Flowchart:

- $i = 0$
- $q = N/b$
- $r = N \% b$
- $++i$
- $N = q$
- $r = \text{ith digit of } Nb$
- $q == 0$
- $\text{END}$
From Base 10 to Base $b$ – Method 2

- Know your powers of the second base
- Subtract out the largest power of second base that fits
- Multiple by scalar, in case of binary, only a 1, so easy
- Put 1 in position for binary, scalar in position for other bases
- Repeat with remainder
From Base 10 to Base $b$ – Method 2

- Example: $210_{10} = ?_2$
  \[210 - 128 = 82\]
  \[82 - 64 = 18\]
  \[18 - 16 = 2\]
  \[2 - 2 = 0\]

- Example: $57_{10} = ?_3$
  \[57 - 27 = 30 - 27 = 3\]
  \[3 - 3 = 0\]
From Base $b$ to Base $c$

- Use a known intermediate base
- The easiest way is to convert from base $b$ to base 10 first, and then from 10 to $c$
- Or, in some cases, it is easier to use base 2 as the intermediate base (we’ll see them soon)
Decimal To Binary Conversion: Method 1

- Divide decimal value by 2 until the value is 0
- Example: $444_{10}$
  - Divide 444 by 2; what is the remainder?
  - Divide 222 by 2; what is the remainder?
  - ...
  - Result is 0: done
- Write the remainders starting from the least significant position (the right to the left)
Decimal To Binary Conversion: Method 2

- Know your powers of two and subtract...
  ... 256 128  64  32  16  8  4  2  1

- Example: \(61_{10}\)
  - What is the biggest power of two that fits?
  - What is the remainder?
  - What fits?
  - What is the remainder?
  - ...
  - What is the binary representation?
Binary to Octal Conversion

- Group into 3 starting at least significant bit
  - Why 3?
  - Add leading 0 as needed
    - Why not trailing 0s?
- Write one octal digit for each group
Binary to Octal Conversion: Examples

- 100 010 111 (binary) 4 2 7 (octal)
- 010 101 110 (binary) 2 5 6 (octal)
Octal to Binary Conversion

- Write down the 3-bit binary code for each octal digit
- Example:
  - 047

<table>
<thead>
<tr>
<th>Octal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
</tbody>
</table>
Binary to Hex Conversion

1111 = 15

- Group into 4 starting at least significant bit
  - Why 4?
  - Add leading 0 if needed
- Write one hex digit for each group
Binary to Hex Conversion: Examples

- 1001 1110 0111 0000 (binary)
  9 E 7 0 (hex)

- 0001 1111 1010 0011 (binary)
  1 F A 3 (hex)
### Hex to Binary Conversion

- Write down the 4-bit binary code for each hex digit
- Example:
  - 0x 3 9 c 8

<table>
<thead>
<tr>
<th>Hex</th>
<th>Bin</th>
<th>Hex</th>
<th>Bin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>a</td>
<td>1010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>b</td>
<td>1011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>c</td>
<td>1100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>d</td>
<td>1101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>e</td>
<td>1110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>f</td>
<td>1111</td>
</tr>
</tbody>
</table>

0x11 1001 1100 1000
## Conversion Table

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Octal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>10</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>11</td>
<td>1001</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>12</td>
<td>1010</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>13</td>
<td>1011</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>14</td>
<td>1100</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>15</td>
<td>1101</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>16</td>
<td>1110</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>17</td>
<td>1111</td>
</tr>
</tbody>
</table>
More Conversions

- **Hex → Octal**
  - Do it in 2 steps
  - Hex → binary → octal

- **Decimal → Hex**
  - Do it in 2 steps
  - Decimal → binary → hex

- So why use hex and octal and not just binary and decimal?
Positional Addition

\[ 25 + 5 + 3 = 33 \]
\[ 42 + 2 = 22 \]

\[
\begin{array}{c}
113_5 \\
42_5 \\
3 + 2 \end{array}
\]

\[
\begin{array}{c}
5_{10} = 10_5 \\
5 + 1 \end{array}
\]

\[
\begin{array}{c}
210_5 \\
2.25 + 1.5 + 0.1 \\
55_5 
\end{array}
\]
Addition

Just like other addition

Examples:

\[
\begin{array}{c}
100001 \quad (33) \\
+011101 \quad (29) \\
\hline
111110 \quad (62)
\end{array}
\]

\[
\begin{array}{c}
00001010 \quad (10) \\
+00001110 \quad (14) \\
\hline
00010000 \quad (24)
\end{array}
\]
A Little Bit on Adding

More generally, it's just like decimal!!

\[ 0 + 0 = 0 \]
\[ 1 + 0 = 1 \]
\[ 1 + 1 = 2, \text{ which is } 10 \text{ in binary, sum is } 0, \text{ carry is } 1. \]
\[ 1 + 1 + 1 = 3, \text{ sum is } 1, \text{ carry is } 1. \]

\[
\begin{array}{c}
x \quad 0011 \\
+ y \quad +0001 \\
\hline
\text{sum} \quad 0100 \\
\end{array}
\]
Largest Number

• What is the largest number that we can represent in $n$ digits...
  – In base 10? $10^n - 1$
  – In base 2? $2^n - 1$
  – In octal?
  – In hex?
  – In base 7?
  – In base $b$?

• How many different numbers can we represent with $n$ digits in base $b$?

\[ 6^n - 1 \]
Data Representation

Using binary numbers to represent information
Data Representation

- Goal: Store numbers, characters, sets, database records in the computer.
Bit

Definition: A unit of information. It is the amount of information needed to specify one of two equally likely choices.

- Example: Flipping a coin has 2 possible outcomes, heads or tails. The amount of info needed to specify the outcome is 1 bit.
## Storing Information

A **4-bit nibble** is a useful concept in computer science.

<table>
<thead>
<tr>
<th>Value</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
</tr>
</tbody>
</table>

- Use more bits for more items
- Three bits can represent 8 things: 000, 001, ..., 111
- N bits can represent $2^N$ things

<table>
<thead>
<tr>
<th>N bits</th>
<th>Can represent</th>
<th>Which is approximately</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>16</td>
<td>65,536</td>
<td>65 thousand (64K where K=1024)</td>
</tr>
<tr>
<td>32</td>
<td>4,294,967,296</td>
<td>4 billion</td>
</tr>
<tr>
<td>64</td>
<td>$1.8446 \times 10^{19}$</td>
<td>20 billion billion</td>
</tr>
</tbody>
</table>
Integer Representation

Assume our representation has a fixed number of bits $n$ (e.g., 32 bits).

- Which 4 billion integers do we want?
  - There are an infinite number of integers less than zero and an infinite number greater than zero.

- What bit patterns should we select to represent each integer AND where the representation:
  - Does not affect the result of calculation
  - Does dramatically affect the ease of calculation

- Convert to/from human-readable representation as needed.
Integer Representation

Usual answers:

1. Represent 0 and consecutive positive integers
   - Unsigned integers
2. Represent positive and negative integers
   - Two’s complement
   - To be covered Later
   - Signed magnitude
   - Biased

Unsigned and two’s complement the most common
Unsigned Integers

• Integer represented is binary value of bits:

  0000 -> 0, 0001 -> 1, 0010 -> 2, ...

• Encodes only positive values and zero
• Range: 0 to $2^n - 1$, for n bits
Unsigned Integers

If we have 4 bit numbers:

To find range make \( n = 4 \). Thus \( 2^4 - 1 \) is 15.
Thus the values possible are 0 to 15.

7 would be 0111
17 not represent able
-3 not represent able

For 32 bits:

Range is 0 to \( 2^{32} - 1 = [0: 4,294,967,295] \)
Which is \( 4,294,967,296 \) different numbers.
Two’s Complement

- Two’ Complement sets the top bit negative.
- This makes the hardware that does arithmetic simpler and faster than the other representations as we do not have multiple representations of values.
- How to get 2’s complement representation:
  - Positive: just as if unsigned binary
  - Negative:
    - Take the positive value
    - Take the 1’s complement of it
    - Add 1 (invert)

4 bit

-8 4 2 1
Two’s Complement

Example, what is -5 in 4-bit 2SC?

1. What is 5? 0101
2. Invert all the bits: 1010 (basically find the 1SC)
3. Add one: 1010 + 1 = 1011 which is -5 in 2SC

To get the additive inverse of a 2’s complement integer

1. Take the 1’s complement
2. Add 1

-8 + 2 + 1 = -5

-8421
Two’s Complement

\[ 2^n - 1 \]

0 \Rightarrow \[
\text{Number of integers represent able is } -2^{n-1} \text{ to } 2^{n-1}-1
\]

So if 4 bits:
\[-8, \ldots, -1, 0, +1, \ldots, +7] = 8 + 1 + 7 = 16 = 2^4 \text{ numbers}\]

With 32 bits:
\[-2^{31}, \ldots, -1, 0, +1, \ldots, (2^{31}-1)] = 2^{31} + 1 + (2^{31}-1) = 2^{32} \text{ numbers}
\[-2^{31}47483648, \ldots, -1, 0, +1, \ldots, 2^{31}47483647] \sim \pm2^{31}
Robots

1ms

16-bit 65k

9 minute

32-bit ~ month
Two’s Complement Conversion

What is \(-12\) in 8-bit 2’s complement form?

\[
\begin{align*}
00001100 & \quad 11110100 \\
11110011 & \quad 00001011 \\
+1 & \quad +1 \\
\hline
11110100 & \quad 00001100
\end{align*}
\]
Addition: 2’s complement

- Just like unsigned addition
- Assume 6-bit and observe:

\[
\begin{align*}
000011 \ (3) & \quad 101000 \ (-24) \quad 111111 \ (-1) \\
+111100 \ (-4) & \quad +010000 \ (16) \quad +001000 \ (8) \\
\hline
-111111 \ (-1) & \quad 111000 \ (-8) \quad \times000111 \ (7)
\end{align*}
\]

- Ignore carry-outs (overflow)
- Sign bit is in the \(2^{n-1}\) bit position
- What does this mean for adding different signs?
Addition: 2's complement

6-bits

More examples: Convert to 2SC and do the addition

-20 + 15

001111

-20

010100

101011

+ 1

101100

-12 + -25

101100

+ 001111

111011

4 + 8

12

4 + 8

16

32

64

111011

000100

+ 1

101
Addition: 2’s complement

-20 + 15

5 + 12

-12 + -25

More examples: Convert to 2SC and do the addition
Addition: 2’s complement

More examples: Convert to 2SC and do the addition

-20 + 15

5 + 12

-12 + -25
Subtraction

General rules:

\[ 1 - 1 = 0 \]
\[ 0 - 0 = 0 \]
\[ 1 - 0 = 1 \]
\[ 10 - 1 = 1 \]
\[ 0 - 1 = \text{need to borrow!} \]

- Or replace \((x - y)\) with \(x + (-y)\)
- Can replace subtraction with additive inverse and addition
Subtraction: 2’s complement

Don’t. Just use addition:

\[ x - y \rightarrow x + (-y) \]

Example:

\[
\begin{array}{c}
10110 \quad (-10) \\
- \quad 00011 \quad (3)
\end{array}
\]

\[
\begin{array}{c}
10110 \quad (-10) \\
\downarrow \\
+ \quad 11101 \quad (-3)
\end{array}
\]

\[
\begin{array}{c}
10011 \quad (-13)
\end{array}
\]
Subtraction: 2’s complement

Can also flip bits of bottom # and add an LSB carry in, so for -10 - 3 we get:

```
1
10110
+ 11100
   10011
```

“add 1”

“flip bits of bottom number”

(throw away carry out)

Addition and subtraction are simple in 2’s complement, just need an adder and inverters.
Subtraction: unsigned

For n-bits use the 2’s complement method and overflow if negative

\[
\begin{array}{c}
11100 (+28) \\
- 10110 (+22) \\
\hline
\end{array}
\]

Becomes

\[
\begin{array}{c}
1 \\
11100 \\
+01001 \\
\hline
00110
\end{array}
\]

Only take 5 bits of result
0 - 1
F - T
L - H

Digital Logic
Truth Table

- The most basic representation of a logic function
- It is a perfect induction proof - Lists the output for all possible input combinations
- How many rows of the truth table needed?

\[ 2^n \]
Truth Table: Inverter

- Inverted signals are denoted with an overbar $\bar{A}$
- Or with a prime symbol $A'$
- Or with a bubble in a circuit diagram

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$Y = A'$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Truth Table: AND Gate

- The result of an AND operation is 1 if and only if all inputs are 1
- Depict AND by the multiplication symbol
  - $A \cdot B$
- Or by lumping the signals together
  - $AB$

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A  B</td>
<td>Y = A \cdot B</td>
</tr>
<tr>
<td>0  0</td>
<td>0</td>
</tr>
<tr>
<td>0  1</td>
<td>0</td>
</tr>
<tr>
<td>1  0</td>
<td>0</td>
</tr>
<tr>
<td>1  1</td>
<td>1</td>
</tr>
</tbody>
</table>
Truth Table: NAND Gate

- The result of an NAND operation is 0 if and only if all inputs are 1.
- Depicted by adding a Bar to the $\land$
- Or adding a dot to the gate

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B</td>
<td>Y = A · B</td>
</tr>
<tr>
<td>0 0</td>
<td>1</td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>0</td>
</tr>
</tbody>
</table>
Truth Table: OR Gate

- The result of an OR operation is 1 if and only if any inputs are 1
- Depict OR by the addition symbol
  - $A + B$

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A  B</td>
<td>Y = A + B</td>
</tr>
<tr>
<td>0  0</td>
<td>0</td>
</tr>
<tr>
<td>0  1</td>
<td>1</td>
</tr>
<tr>
<td>1  0</td>
<td>1</td>
</tr>
<tr>
<td>1  1</td>
<td>1</td>
</tr>
</tbody>
</table>
Truth Table: NOR Gate

- The result of an OR operation is 1 if and only if all inputs are 0
- Depict NOR by the addition symbol with bar
- Or add a dot to the gate

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>( Y = A + B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Truth Table: XOR Gate

- The result of an XOR operation is 1 if and only if its inputs differ.

- Depicted by the addition symbol: $A \oplus B$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>$Y = A + B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Truth table to Gates

So giving some arbitrary truth table, how do you go about creating a transistor-based circuit for it?
TT to Gates Procedure

1. Read each row of the truth table independently.
2. For each row that is 1, draw an and gate that is 1 if and only if the inputs match that line of the truth table.
1. This will require many inverters
3. Once all and gates have been created, OR their outputs together.
4. This solution will work but is not always optimal

Boolean algebra
Karnaugh map
Simple example

- XOR Gate – one or the other, but not both

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Synthesis of an Arbitrary gate

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Binary Adding Review

\[ 0 + 0 = 0 \]
\[ 1 + 0 = 1 \]
\[ 1 + 1 = 2, \text{ which is } 10 \text{ in binary, sum is } 0, \text{ carry is } 1. \]
\[ 1 + 1 + 1 = 3, \text{ sum is } 1, \text{ carry is } 1. \]

Can we write that as a truth table
## Full Adder Truth Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C\textsubscript{in}</th>
<th>C\textsubscript{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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
## Full Adder Truth Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C&lt;sub&gt;in&lt;/sub&gt;</th>
<th>C&lt;sub&gt;out&lt;/sub&gt;</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>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>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Boolean Algebra

- $0 \cdot 0 = 0$
- $1 + 1 = 1$
- $1 \cdot 1 = 1$
- $0 + 0 = 0$
- $0 \cdot 1 = 1 \cdot 0 = 0$
- $1 + 0 = 0 + 1 = 1$
- If $x = 0$ then $x' = 1$
- If $x = 1$ then $x' = 0$
Single-Variable Boolean Algebra

- $x \cdot 0 = 0$
- $x + 1 = 1$
- $x \cdot 1 = x$
- $x + 0 = x$
- $x \cdot x = x$
- $x + x = x$
- $x \cdot x' = 0$
- $x + x' = 1$
- $(x')' = x$
Properties of Boolean Algebra

- **Commutative**
  - $x \cdot y = y \cdot x$
  - $x + y = y + x$

- **Associative**
  - $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
  - $x + (y + z) = (x + y) + z$

- **Distributive**
  - $x \cdot (y + z) = x \cdot y + x \cdot z$
  - $x + y \cdot z = (x + y) \cdot (x + z)$
De Morgan’s Laws

• “Break the line, change the sign”

• Two laws:
  – \( A' + B' = (AB)' \)
    • This is the NAND gate
  – \( A' \cdot B' = (A+B)' \)
    • This is the NOR gate
De Morgan’s Laws

\[(A + B)' = A'B' \quad \text{conversely} \quad (AB)' = A' + B'\]

“Break the line, change the sign”

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>A+B</th>
<th>A+B</th>
<th>A</th>
<th>B</th>
<th>A  \cdot B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</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>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
De Morgan’s Laws

\[(A + B)' = A'B' \quad \text{conversely} \quad (AB)' = A' + B'\]

“Break the line, change the sign”

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>AB</th>
<th>(\overline{AB})</th>
<th>(\overline{A})</th>
<th>(\overline{B})</th>
<th>(\overline{A} + \overline{B})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Digital Logic Structures
Basic Logic Gates

- **NOT**
- **OR**
- **NOR**
- **AND**
- **NAND**
- **XOR**
More Than Two Inputs?

- AND and OR gates can take any number of inputs
  - AND gives 1 if all inputs are 1
  - OR gives 1 if any input is 1
- NAND?? NOR??
  - Not associative!

[Diagram showing logic gates and connections]
# One-Bit Full Adder

\[ A + B + C = S, C \]

<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>

Diagram of One-Bit Full Adder.
Four-Bit Full Adder

Ripple-carry adder

Look-ahead
More Logic Structure

- As we start to build more complex structures we need ways to control parts of them
  - To select signals
  - To activate certain outputs
Signal Selection

A
-
+

B
-
-

A
-
×

B
-
÷
Two-Way Multiplexer
Two-Way Multiplexer

2-way multiplexer: the output is equal to one of the two inputs, based on a selector.

<table>
<thead>
<tr>
<th>S</th>
<th>A</th>
<th>B</th>
<th>Y</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>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Four-Way Multiplexer

- $n$-bit selector and $2^n$ inputs, one output
  - output equals one of the inputs, depending on selector
- “Four-to-one mux”
Two-to-Four Decoder

- \( n \) inputs, \( 2^n \) outputs
  - exactly one output is 1 for each possible input pattern
- Generates a walking-ones pattern
- Uses:
  - Convert memory or register address to a control line
  - Convert an opcode to one of \( n \) control lines
  - We will get to this in the LC-3 material
Building functions from logic gates

- Combinational Logic Circuit
  - Output depends only on the current inputs
  - Stateless (memoryless)

- Sequential Logic Circuit
  - Output depends on the sequence of inputs (past and present)
  - Stores information (state) from past inputs
Combinational vs. Sequential
Two types of “combination” locks

Combinational
Success depends only on the values, not the order in which they are set.

Sequential
Success depends on the sequence of values (e.g., R-13, L-22, R-3).
Combinational vs. Sequential

- **Combinational circuit**
  - Always gives the same output for a given set of inputs
  - Example: Adder always generates sum and carry, regardless of previous inputs

- **Sequential circuit**
  - Remembers previous input
  - Output depends on state and input
Synchronization of Sequential Circuits

- These are real devices and require time to compute.
- If we want proper results we need a way to ensure consistent timing.
- One way of doing this is a clock:
  - Repeating signal at certain frequency
  - When you buy a computer this is the number in gigahertz
- All our actions take place in relation to this clock
D-Flip-Flop (the one for Lab)

- Basic Memory Device
- Stores the value of D when conditions are met and outputs it on Q
- Otherwise Q holds the last value of D

- D-flip-flop is edge-triggered (changes only on the edge of the clock)
- This can be both edges or a single type (up or down)
D-Flip Flop: Timing Diagram

Ck 1 0
0 1

D 1 0
0 1

Q 1
0

time

CMPE-012/L
D-Flip-Flop with Write Enable

- Same idea as a Flip-Flop but adds another input
- Instead of changing on clock edges. You can only change on a clock edge when WE is high
D-Flip Flop: Timing Diagram (up)

- **D**
- **Q**
- **Ck**
- **WE**
- **E**
- **Q**
Register

- A register stores a multi-bit value
- Common WE which latches the n-bit value
Memory

Now that we know how to store bits, we can build a memory – a logical $k \times m$ array of stored bits.

**Address Space:**
number of locations
(usually a power of 2)

**Addressability:**
number of bits per location
(e.g., byte-addressable)

\[ k = 2^n \]
locations

\[ m \text{ bits} \]
Memory
Memory
State Machine

The basic type of sequential circuit

- Combines combinational logic with storage
- “Remembers” state, and changes output (and state) based on inputs and current state
Representing Multi-bit Values

- Number bits from right (0) to left (n-1)
  - just a convention -- could be left to right, but must be **consistent**
- Use brackets to denote range:
  \( D[l:r] \) denotes bit \( l \) to bit \( r \), from **left** to **right**

\[
A = \overline{0101001101010101}\n\]

\[

May also see \( A<14:9> \), especially in hardware block diagrams.