Integer Numbers

The Number Bases of Integers

Textbook Chapter 2 +
Number Systems

- **Unary**, or marks:
  
  - \(/\)/ = 7
  - \(/\)/ + \(/\)/ = \(/\)/\(/\)/

- Grouping lead to Roman Numerals:
  
  - VII + V = VVII = 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

\[1 \cdot 10^3 + 8 \cdot 10^2 + 3 \cdot 10 + 2 \cdot 10^0\]
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 \times 60^0
  300 + 45 = 345_{10}
# 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></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
- 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>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>
<td>٩</td>
</tr>
<tr>
<td>Eastern Arabic-Indic (Persian and Urdu)</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>
<td>٩</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>ஐ</td>
<td>ஐ</td>
<td>ஐ</td>
<td>ஐ</td>
<td>ஐ</td>
<td>ஐ</td>
<td>ஐ</td>
<td>ஐ</td>
<td>ஐ</td>
<td>ஐ</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:
\[
1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0
\]
\[
- 1001_2 = 8 + 1 = 9_{10}
\]
\[
1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0
\]
\[
- 11000_2 = 16 + 8 = 24_{10}
\]
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>
Examples of Converting Hex to Decimal

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

  $= 160 + 3$

  $= 163_{10}$

- $210_{16} = 2 \times 16^2 + 1 \times 16^1 + 0 \times 16^0$

  $= 2 \times 256 + 1 \times 16 + 0 \times 1$

  $= 512 + 16 + 0$

  $= 528_{10}$

- $3E8_{16} = 3 \times 16^2 + E \times 16^1 + 8 \times 16^0$

  $= 3 \times 256 + 14 \times 16 + 8 \times 1$

  $= 768 + 224 + 8$

  $= 1000_{10}$
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): $0 \ldots (b - 1)$
- $S_{n-1}S_{n-2} \ldots S_2S_1S_0$

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

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: $2010_{10} = ?_5$

$2010 \div 5 = 402 \ R 0$
$402 \div 5 = 80 \ R 2$
$80 \div 5 = 16 \ R 0$
$16 \div 5 = 3 \ R 1$
$3 \div 5 = 0 \ R 3$
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
  9 = 0
  3 - 3 = 0

11010010
2
4
8
16
27
81
243
729
128
256
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

$111_2 = 7_8$
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

- 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 (hex)
  3
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>

0011 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

\[
\begin{array}{c}
\text{11} \\
\text{113}_5 \\
+ \\
\text{42}_5 \\
\text{-----} \\
\text{210}_5 \\
\end{array}
\]
Addition

Just like other addition

Examples:

\[
\begin{align*}
100001 \quad (33) & \quad + 0001010 \quad (10) \\
011101 \quad (29) & \quad + 0001110 \quad (14) \\
\hline
1011110 \quad (62) & \quad \text{1000} \quad (24)
\end{align*}
\]
A Little Bit on Adding

More generally, it's just like decimal!!

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

\[
\begin{array}{c}
x \\
+ y \\
\hline
\text{sum}
\end{array}
\begin{array}{c}
0011 \\
+ 0001 \\
\hline
0100
\end{array}
\]
010101 + 1011101 = 1110010
Largest Number

- What is the largest number that we can represent in \( n \) digits...
  - In base 10?
  - In base 2?
  - In octal?
  - In hex?
  - In base 7?
  - In base \( b \)?

- How many different numbers can we represent with \( n \) digits in base \( b \)?
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

<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>
<tr>
<td>False</td>
<td>0</td>
</tr>
<tr>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>1e-4</td>
<td>0</td>
</tr>
<tr>
<td>5</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>$1.8446\ldots \times 10^{19}$</td>
<td>20 billion billion</td>
</tr>
</tbody>
</table>
bit + 1

nibble 4

byte 8
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 effect 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
   - 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

-32 16 8 4 2 1

• 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
Two’s Complement

1 \cdot -8 + 0 + 1 \cdot 2 + 1 \cdot 1 = -8 + 2 + 1 = -5

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
Two’s Complement

Number of integers representable is $-2^{n-1}$ to $2^{n-1}-1$

So if 4 bits:

$$[-8, ... , -1, 0, +1, ..., +7] = 8 + 1 + 7 = 16 = 2^4 \text{ numbers}$$

With 32 bits:

$$[-2^{31}, ... , -1, 0, +1, ..., (2^{31}-1)] = 2^{31} + 1 + (2^{31}-1) = 2^{32} \text{ numbers}$$

$$[-2147483648, ..., -1, 0, +1, ..., 2147483647] \sim \pm 2B$$
Two’s Complement Conversion

12 \quad 8 + 4 = 1100

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

\[ \begin{array}{c}
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
+1 & & & & & & & \\
\hline
1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
\end{array} \]
Two’s Complement Conversion

12 \text{ 8} + 4 = 1100

What is -12 in 8-bit 2’s complement form:

\begin{align*}
\text{11111010} & \quad \text{(12 in 8-bit 2’s complement)} \\
\text{+ 1} & \\
\hline
\text{11110100} & \quad \text{(-12 in 8-bit 2’s complement)}
\end{align*}
10010110_{25C} \rightarrow \ ?_{10}

\begin{align*}
011 & 01001 \\
+1 & \\
\hline
011 & 01010
\end{align*}

64 + 32 + 8 + 2
\[ \begin{align*} \frac{1}{128} & \quad 25c \geq 10 \\
64 & + 32 + 16 + 8 + 4 + 2 + 1 = 127 \end{align*} \]
Addition: 2’s complement

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

  \[
  \begin{array}{c}
  000011 \quad (3) \\
  +111100 \quad (-4) \\
  \hline
  \overline{111111} \quad (-1)
  \end{array}
  \]

  \[
  \begin{array}{c}
  101000 \quad (-24) \\
  +010000 \quad (16) \\
  +001000 \quad (8) \\
  \hline
  \overline{111000} \quad (-8) \\
  \end{array}
  \]

- 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

More examples: Convert to 2SC and do the addition

-20 + 15

5 + 12

-12 + -25
Addition: 2’s complement

More examples: Convert to 2SC and do the addition

\[ -20 + 15 \]

\[ 2^6 - 1 \]
\[ 2^5 \]
\[ 32 \]

\[ 001100 \rightarrow \]
\[ 110011 \]
\[ +1 \]
\[ 1 \]
\[ 110100 \]
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 = 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:

\[
\begin{align*}
10110 \quad (-10) \\
-00011 \quad (3)
\end{align*}
\]

\[
\begin{align*}
10110 \quad (-10) \\
+11101 \quad (-3) \\
\hline
10011 \quad (-13)
\end{align*}
\]
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 inverter.
Subtraction: unsigned

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

\[
\begin{align*}
11100 \ (+28) \\
- \ 10110 \ (+22)
\end{align*}
\]

Becomes

\[
\begin{align*}
\text{1} \\
11100 \\
+01001 \\
00110
\end{align*}
\]

Only take 5 bits of result