More 2’s Complement:

- **Addition (binary)**
  - Add the numbers with all digits, including the sign bits, and throw away the carry on the left end.

- **Example:**

  \[-74_{10} + 51_{10} = -23_{10}\]

  \[74_{10} = 01001010_2 \rightarrow -74_{10} = 10110110_2\]

  \[51_{10} = 00110011_2\]
More 2’s Complement:

- Addition (binary)

\[
\begin{array}{c}
\begin{array}{c}
1 \ 1 \ 1 \ 1 \\
-74_{10} = 10110110_2 \\
+ 51_{10} = 00110011_2 \\
\hline
11101001_2 \\
= -23_{10}
\end{array}
\end{array}
\]

- How?
More 2’s Complement:

- Addition (binary)

\[ 11101001_2 = \text{2’s Complement of } 23_{10} \]
\[ (00010111_2) \]

Example:

\[ 74_{10} + (-51_{10}) = 23_{10} \]

\[ 74_{10} = 01001010_2 \]

\[ 51_{10} = 00110011_2 \rightarrow -51_{10} = 11001101_2 \]
• More 2’s Complement:
  ◦ Addition (binary)

\[
\begin{align*}
1 & 1 & 1 \\
74_{10} & = & 01001010_2 \\
+ & -51_{10} & = 11001101_2 \\
& & 100010111_2 \\
& = & 23_{10}
\end{align*}
\]

• In this case, we throw away the left-most “overflow” bit
More 2’s Complement:

- Addition (binary)

\[ 11101001_2 = \text{2’s Complement of } 23_{10} \]
\[ (00010111_2) \]

- Example:

\[-74_{10} + (-51_{10}) = 125_{10}\]

\[ 74_{10} = 01001010_2 \rightarrow -74_{10} = 10110110_2 \]

\[ 51_{10} = 00110011_2 \rightarrow -51_{10} = 11001101_2 \]
More 2’s Complement:

- Addition (binary)

\[
\begin{align*}
&\quad\quad 1\ 1\ 1\ 1\ 1\ 1\ 1 \\
&-74_{10} = 10110110_2 \\
&+\quad -51_{10} = \underline{11001101}_2 \\
&\quad\quad 100010111_2 = 01111101_2 \\
&\quad\quad = -125_{10}
\end{align*}
\]

- In this case, we throw away the left-most “overflow” bit
More 2’s Complement

- There is only 1 zero:
  
  \[00000000_2 = 11111111_2 + 1_2\]
  
  \[= 100000000_2\]
  
  \[= 00000000_2\] (throw away the left-most 1)

- The new problem is:
  
  \[-128_{10} = 100000000_2 \text{ (}128_{10} = 01111111_2\text{)},\]
  
  it’s 2’s Complement is \[01111111_2 + 1_2 = 10000000_2\], so there is no positive \[128_{10}\]

- Why?
More 2’s Complement

- Because there are an even number of representations, so if there is only 1 zero, then there must be some number that has either no positive or no negative representation

- Note that with 2’s Complement, only numbers 0 to 127 can be represented with 8 bits, whereas numbers 0 to 255 can be represented with unsigned binary numbers

- Why?
More 2’s Complement
  ◦ Because of the left-most bit being the sign bit
  ◦ A shortcoming of 2’s Complement
• More 2’s Complement
  ◦ Hex:
    • to find the 2’s Complement of a hex number:
      ◦ 1. Subtract the number from FFFF to get the 1’s Complement
      ◦ 2. Add 1 to get the 2’s Complement
      ◦ 3. If the first digit is less than or equal to 7, the number is positive. If the first digit is greater than or equal to 8, the number is negative
More 2’s Complement

- Hex:
  - Example: $3\text{DA6}_{16}$

$$
\begin{align*}
\text{FFFF}_{16} & \quad \text{FFFF}_{16} \\
- \quad 3\text{DA6}_{16} & \quad \rightarrow \text{1’s Complement} \\
\text{C259}_{16} & \quad \rightarrow \text{1’s Complement}
\end{align*}
$$

$$
\begin{align*}
\text{C259}_{16} & \quad + \quad \text{1}_{16} \\
\text{C25A}_{16} & \quad \rightarrow \text{2’s Complement}
\end{align*}
$$
“A Given length” (number of bits)

- If the number is too long, throw away the extra bits on the left if they are all the same.

- If not, or if the left-most bit of the new number is different from the thrown away ones, then the given length is too short → overflow.

- If the number is too short, add more bits that are the same as the left-most bit to the left.
“A Given length” (number of bits)

- Example:
  - given length = 8 bits \(\rightarrow\) throw away bits:
    
    \[
    \begin{align*}
    111111111011000 & \quad \text{– good} \\
    1111111101011000 & \quad \text{– bad} \\
    000000001011000 & \quad \text{– good}
    \end{align*}
    \]
    
  - sign extension: 8 bits to 16 bits
    
    \[
    10100101 \rightarrow 1111111110100101
    \]
"A Given length" (number of bits)

- Overflow

  - to add two numbers of a given length:
    - If they have different signs, no overflow
    - If they have the same sign but the resulting sign is different, overflow
    - If the resulting sign remains the same, no overflow
“A Given length” (number of bits)

- Overflow

Example:

\[
\begin{align*}
87_{10} & = 01010111_2 \\
+ 42_{10} & = \underline{00101010}_2 \\
10000001_2 & = -127_{10} \rightarrow \text{overflow} \\
& \quad \text{(different sign)} \\
-87_{10} & = 10101001_2 \\
+ -42_{10} & = \underline{11010110}_2 \\
101111111_2 & = 127_{10} \rightarrow \text{overflow} \\
& \quad \text{(different sign)}
\end{align*}
\]
• “A Given length” (number of bits)
  ◦ Overflow

  • Example:
    
    \[
    \begin{align*}
    \text{-} & 53_{10} = 11001011_2 \\
    + & -42_{10} = 11010110_2 \\
    \text{110100001}_2 &= -95_{10} \rightarrow \textbf{no} \text{ overflow (same sign)}
    \end{align*}
    \]
Shift Operations (bitwise operations)

1. Logical shift:
   - On one end, the bit shifted out is lost
   - On the other end, a 0 is shifted in

Example: Shift $10111011_2$ by 2 bits to the left

$10111011_2 \rightarrow 11101100_2$

add 2 0s
Shift Operations (bitwise operations)

2. Arithmetic shift:
   - Treats data as a signed integer
   - On right shift: sign bit (left-most bit) is replicated
   - On left shift: the sign stays the same, a 0 is shifted in on the right
     - If the bits shifted out are different from the sign bit, then overflow
     - Left shift is the same as Logical shift
   - Left shift by 1 bit corresponds to multiplication by 2
   - Right shift by 1 bit corresponds to division by 2
   - Apply the “given length” knowledge to determine if there is overflow
Shift Operations (bitwise operations)

- Example: Shift 11011011 by 1 bit to the right

\[ 11011011 \rightarrow 11101101 \]
replicate sign bit

- Example: Shift 11011011 by 1 bit to the left

\[ 11011011 \rightarrow 10110110 \rightarrow \text{no overflow (sign bit is the same)} \]
add a 0
• Shift Operations (bitwise operations)
  - Example: Shift 10111011 by 1 bit to the left

  10111011 → 01110110 → overflow (sign bit is different)

  add a 0
Shift Operations (bitwise operations)

3. Rotate (cyclic) shift:
- Preserves all the bits being operated on
- Left shift rotates the bits around to the right end
- Right shift rotates the bits around to the left end
- Used often in cryptography

Example: Shift 10111011 by 1 bit to the left

\[
\begin{array}{c}
 10111011 \\
\end{array} \rightarrow \begin{array}{c}
 01110111 \\
\end{array}
\]
Shift Operations (bitwise operations)

- Why use bitwise operations?
  - Because depending on the processor, bitwise operations, such as multiplication and division, may be faster than the actual mathematical operations.
  - However, with modern CPUs, the pipelining is such that it’s about the same speed.
Floating Point Numbers

We have seen signed and unsigned integers so far, so what about numbers with fractions (real numbers)?

Example: $3.14159265_{10}$
$0.000000001_{10} = 1.0 \times 10^{-9}$
(normalized number)
• Floating Point Numbers
  
  ◦ In binary: $1.xxxxxxxxx_2 \times 2^{yyyy}$
    
    - significand (mantissa)
    - exponent
  
  ◦ Hex is similar, except that you use 16 in place of 2
Floating Point Numbers
- Single-precision numbers (32 bits)

<table>
<thead>
<tr>
<th>sign</th>
<th>exponent (include sign)</th>
<th>significand</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits</td>
<td>31 30</td>
<td>23 22 0</td>
</tr>
</tbody>
</table>

- Also known as a “float” in C/C++, Java, etc.
• **Floating Point Numbers**
  ◦ Double-precision numbers (64 bits)

  ![Binary representation of double-precision numbers](image)

  - Sign (1)
  - Exponent (include sign) (11)
  - Significand (20) (continued)

  - Both single- and double-precision number formats are IEEE 754 standards

  - Also known as a “double” in C/C++, Java, etc.
The ASCII Character Set

- The American Standard Code for Information Interchange
  - First published in 1963, last updated in 1986

- 26 upper-case letters (65 – 90)
- 26 lower-case letters (97 - 122)
- 10 digits (48 – 57)
- 66 other symbols (punctuation, math, etc.)
- Total: 128, requires 7 bits binary
The ASCII Character Set

The American Standard Code for Information Interchange

- 0 – 31, 127 are non-printing characters, of which 1 – 26 are control characters:
  - 13 – carriage return
  - 10 – newline
  - 9 – tab
  - 127 – delete
- 32 – space, or blank character ‘ ’ (printing)
- Digit character = ‘0’ + digit value, i.e. ‘7’ = ‘0’ + 7 = 48 + 7 = 55

Note: a digital character is different than it’s value
The ASCII Character Set

- $n^{th}$ upper-case letter = ‘A’ + $n - 1$
- $n^{th}$ lower-case letter = ‘a’ + $n - 1$
- A lower-case letter = corresponding upper-case letter + (‘a’ – ‘A’ = 32)
## ASCII Character Set

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Char</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Chr</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Chr</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Chr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>000</td>
<td>NUL (null)</td>
<td>32</td>
<td>20</td>
<td>040</td>
<td>Space</td>
<td>@</td>
<td>96</td>
<td>60</td>
<td>140</td>
<td>&lt;</td>
<td>96;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>001</td>
<td>SOH (start of heading)</td>
<td>33</td>
<td>21</td>
<td>041</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>101</td>
<td>&gt;</td>
<td>97;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>002</td>
<td>STX (start of text)</td>
<td>34</td>
<td>22</td>
<td>042</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>102</td>
<td>&lt;</td>
<td>98;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>003</td>
<td>ETX (end of text)</td>
<td>35</td>
<td>23</td>
<td>043</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>103</td>
<td>&lt;</td>
<td>99;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>004</td>
<td>EOT (end of transmission)</td>
<td>36</td>
<td>24</td>
<td>044</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>104</td>
<td>&lt;</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>005</td>
<td>ENQ (enquiry)</td>
<td>37</td>
<td>25</td>
<td>045</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>105</td>
<td>&lt;</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>006</td>
<td>ACK (acknowledge)</td>
<td>38</td>
<td>26</td>
<td>046</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>106</td>
<td>&lt;</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>007</td>
<td>BEL (bell)</td>
<td>39</td>
<td>27</td>
<td>047</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>107</td>
<td>&lt;</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>010</td>
<td>BS (backspace)</td>
<td>40</td>
<td>28</td>
<td>050</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>110</td>
<td>&lt;</td>
<td>e</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>011</td>
<td>TAB (horizontal tab)</td>
<td>41</td>
<td>29</td>
<td>051</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>111</td>
<td>&lt;</td>
<td>f</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>012</td>
<td>LF (NL line feed, new line)</td>
<td>42</td>
<td>2A</td>
<td>052</td>
<td>&lt;</td>
<td>74</td>
<td>4A</td>
<td>112</td>
<td>&lt;</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>013</td>
<td>VT (vertical tab)</td>
<td>43</td>
<td>2B</td>
<td>053</td>
<td>&lt;</td>
<td>75</td>
<td>4B</td>
<td>113</td>
<td>&lt;</td>
<td>h</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>014</td>
<td>FF (NP form feed, new page)</td>
<td>44</td>
<td>2C</td>
<td>054</td>
<td>&lt;</td>
<td>76</td>
<td>4C</td>
<td>114</td>
<td>&lt;</td>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>015</td>
<td>CR (carriage return)</td>
<td>45</td>
<td>2D</td>
<td>055</td>
<td>&lt;</td>
<td>77</td>
<td>4D</td>
<td>115</td>
<td>&lt;</td>
<td>j</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>016</td>
<td>SO (shift out)</td>
<td>46</td>
<td>2E</td>
<td>056</td>
<td>&lt;</td>
<td>78</td>
<td>4E</td>
<td>116</td>
<td>&lt;</td>
<td>k</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>017</td>
<td>SI (shift in)</td>
<td>47</td>
<td>2F</td>
<td>057</td>
<td>&lt;</td>
<td>79</td>
<td>4F</td>
<td>117</td>
<td>&lt;</td>
<td>l</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>020</td>
<td>DLE (data link escape)</td>
<td>48</td>
<td>30</td>
<td>060</td>
<td>&lt;</td>
<td>80</td>
<td>50</td>
<td>120</td>
<td>&lt;</td>
<td>m</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>021</td>
<td>DC1 (device control 1)</td>
<td>49</td>
<td>31</td>
<td>061</td>
<td>&lt;</td>
<td>81</td>
<td>51</td>
<td>121</td>
<td>&lt;</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>022</td>
<td>DC2 (device control 2)</td>
<td>50</td>
<td>32</td>
<td>062</td>
<td>&lt;</td>
<td>82</td>
<td>52</td>
<td>122</td>
<td>&lt;</td>
<td>o</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>4</td>
<td>023</td>
<td>DC3 (device control 3)</td>
<td>51</td>
<td>33</td>
<td>063</td>
<td>&lt;</td>
<td>83</td>
<td>53</td>
<td>123</td>
<td>&lt;</td>
<td>p</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>024</td>
<td>DC4 (device control 4)</td>
<td>52</td>
<td>34</td>
<td>064</td>
<td>&lt;</td>
<td>84</td>
<td>54</td>
<td>124</td>
<td>&lt;</td>
<td>q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>6</td>
<td>025</td>
<td>NAK (negative acknowledge)</td>
<td>53</td>
<td>35</td>
<td>065</td>
<td>&lt;</td>
<td>85</td>
<td>55</td>
<td>125</td>
<td>&lt;</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>7</td>
<td>026</td>
<td>SYN (synchronous idle)</td>
<td>54</td>
<td>36</td>
<td>066</td>
<td>&lt;</td>
<td>86</td>
<td>56</td>
<td>126</td>
<td>&lt;</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>8</td>
<td>027</td>
<td>ETB (end of trans. block)</td>
<td>55</td>
<td>37</td>
<td>067</td>
<td>&lt;</td>
<td>87</td>
<td>57</td>
<td>127</td>
<td>&lt;</td>
<td>t</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>9</td>
<td>030</td>
<td>CAN (cancel)</td>
<td>56</td>
<td>38</td>
<td>070</td>
<td>&lt;</td>
<td>88</td>
<td>58</td>
<td>130</td>
<td>&lt;</td>
<td>u</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>A</td>
<td>031</td>
<td>EM (end of medium)</td>
<td>57</td>
<td>39</td>
<td>071</td>
<td>&lt;</td>
<td>89</td>
<td>59</td>
<td>131</td>
<td>&lt;</td>
<td>v</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>B</td>
<td>032</td>
<td>SUB (substitute)</td>
<td>58</td>
<td>3A</td>
<td>072</td>
<td>&lt;</td>
<td>90</td>
<td>60</td>
<td>132</td>
<td>&lt;</td>
<td>w</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>C</td>
<td>033</td>
<td>ESC (escape)</td>
<td>59</td>
<td>3B</td>
<td>073</td>
<td>&lt;</td>
<td>91</td>
<td>61</td>
<td>133</td>
<td>&lt;</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>D</td>
<td>034</td>
<td>FS (file separator)</td>
<td>60</td>
<td>3C</td>
<td>074</td>
<td>&lt;</td>
<td>92</td>
<td>62</td>
<td>134</td>
<td>&lt;</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>E</td>
<td>035</td>
<td>GS (group separator)</td>
<td>61</td>
<td>3D</td>
<td>075</td>
<td>&lt;</td>
<td>93</td>
<td>63</td>
<td>135</td>
<td>&lt;</td>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>F</td>
<td>036</td>
<td>RS (record separator)</td>
<td>62</td>
<td>3E</td>
<td>076</td>
<td>&lt;</td>
<td>94</td>
<td>64</td>
<td>136</td>
<td>&lt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>1</td>
<td>037</td>
<td>US (unit separator)</td>
<td>63</td>
<td>3F</td>
<td>077</td>
<td>&lt;</td>
<td>95</td>
<td>65</td>
<td>137</td>
<td>&lt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Numbering Systems
• Flowcharts
  ◦ Simple, graphical ways to represent programs using standard symbols, i.e.

  ![Flowchart Diagram]

  Operation, (i.e. addition) or data movement (i.e. load register, load from memory)

  Condition: if true go “Y” branch, if false go “N” branch
Flowcharts
- Example: compute $p = x^n$, $p = 1$, $n = 5$, $x = 2$
  (n and x are user defined, p is hard-coded)

Numbering Systems
Computer Logic

- While low level bit manipulations may seem to be abstract to a modern programmer, they are actually relevant on every level that a computer operates on.

- All computer operations and logical decisions can be reduced to bits and their values, which in turn represent electrical signals and their states (on or off).
Computer Logic

Numbering Systems
- Computer Logic

- Switches are the fundamental building blocks of computers

Increasing complexity:

- Switches
- Gates
- Circuits
- Central Processing Unit
• Questions/comments/dirty limericks?