Shift Operations

- shift bitstring to left or right: $<<, >>$
  - simplest: no carry-over, vacant positions filled with 0s
- Shifting a two’s complement number involves carry over
- when a two's-complement number is shifted to the right, the most-significant bit must be maintained;
  - For left shift, it’s filled in with zeros
- very fast machine instructions
Shift Operations

- equivalent to multiplication/division by 2
- programming languages, operators << and >>
- \( a \ll b \equiv a \times 2^b \)
- \( a \gg b \equiv a \div 2^b \) – \textbf{not always true} for negative two’s complement numbers
Data Representation

- interpretation is in the eye of the beholder
- what does this represent?
  - 01110111011010000111100100111111
Byte

- convention: 8 bits = 1 byte
  - power of 2
  - signed range -128 .. 127
- historical
  - useful range to represent characters, control
  - 8-bit circuit width
Word

- A word is a fixed-sized piece of data handled as a unit by the processor - 32bits, 64 bits etc.
- Number of bits in a word: word size or word length
- Ordering: little-endian, big-endian
  - little-endian: least-significant byte first
  - big-endian: most-significant byte first
    - “natural” way of writing numbers
Ordering

• 32 bit value \texttt{0x01234567} (Hexadecimal representation)
  • stored as four bytes 0x01, 0x23, 0x45, 0x67, on Big-endian (eg. TCP/IP) while on Little-Endian (eg. Intel x86), it will be stored in reverse order:
Character

- ASCII – American Standard Code for Information Interchange: 128 (7-bit) characters

- ASCII was created initially for remote teletype machines (before digital computers)
<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>Null</td>
<td>32</td>
<td>20</td>
<td>Space</td>
<td>64</td>
<td>40</td>
<td>$</td>
<td>96</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>Start of heading</td>
<td>33</td>
<td>21</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>A</td>
<td>97</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>Start of text</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>B</td>
<td>98</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>End of text</td>
<td>35</td>
<td>23</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>C</td>
<td>99</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>End of transmit</td>
<td>36</td>
<td>24</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>D</td>
<td>100</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>Enquiry</td>
<td>37</td>
<td>25</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>E</td>
<td>101</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>Acknowledge</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>F</td>
<td>102</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>Audible bell</td>
<td>39</td>
<td>27</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>G</td>
<td>103</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>Backspace</td>
<td>40</td>
<td>28</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>H</td>
<td>104</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>Horizontal tab</td>
<td>41</td>
<td>29</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>I</td>
<td>105</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>Line feed</td>
<td>42</td>
<td>2A</td>
<td>^</td>
<td>74</td>
<td>4A</td>
<td>J</td>
<td>106</td>
<td>6A</td>
<td>j</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>Vertical tab</td>
<td>43</td>
<td>2B</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>K</td>
<td>107</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>Form feed</td>
<td>44</td>
<td>2C</td>
<td>,</td>
<td>76</td>
<td>4C</td>
<td>L</td>
<td>108</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>Carriage return</td>
<td>45</td>
<td>2D</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>M</td>
<td>109</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>Shift out</td>
<td>46</td>
<td>2E</td>
<td>.</td>
<td>78</td>
<td>4E</td>
<td>N</td>
<td>110</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>Shift in</td>
<td>47</td>
<td>2F</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>O</td>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>Data link escape</td>
<td>48</td>
<td>30</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>P</td>
<td>112</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>Device control 1</td>
<td>49</td>
<td>31</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>Q</td>
<td>113</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>Device control 2</td>
<td>50</td>
<td>32</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>R</td>
<td>114</td>
<td>72</td>
<td>r</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>Device control 3</td>
<td>51</td>
<td>33</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>S</td>
<td>115</td>
<td>73</td>
<td>s</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>Device control 4</td>
<td>52</td>
<td>34</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>T</td>
<td>116</td>
<td>74</td>
<td>t</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>Neg. acknowledge</td>
<td>53</td>
<td>35</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>U</td>
<td>117</td>
<td>75</td>
<td>u</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>Synchronous idle</td>
<td>54</td>
<td>36</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>V</td>
<td>118</td>
<td>76</td>
<td>v</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>End trans. block</td>
<td>55</td>
<td>37</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>W</td>
<td>119</td>
<td>77</td>
<td>w</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>Cancel</td>
<td>56</td>
<td>38</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>X</td>
<td>120</td>
<td>78</td>
<td>x</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>End of medium</td>
<td>57</td>
<td>39</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>Y</td>
<td>121</td>
<td>79</td>
<td>y</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>Substitution</td>
<td>58</td>
<td>3A</td>
<td>:</td>
<td>90</td>
<td>5A</td>
<td>Z</td>
<td>122</td>
<td>7A</td>
<td>z</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>Escape</td>
<td>59</td>
<td>3B</td>
<td>:</td>
<td>91</td>
<td>5B</td>
<td>[</td>
<td>123</td>
<td>7B</td>
<td>(</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>File separator</td>
<td>60</td>
<td>3C</td>
<td>&lt;</td>
<td>92</td>
<td>5C</td>
<td>\</td>
<td>124</td>
<td>7C</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>Group separator</td>
<td>61</td>
<td>3D</td>
<td>=</td>
<td>93</td>
<td>5D</td>
<td>)</td>
<td>125</td>
<td>7D</td>
<td>)</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>Record separator</td>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
<td>94</td>
<td>5E</td>
<td>^</td>
<td>126</td>
<td>7E</td>
<td>~</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>Unit separator</td>
<td>63</td>
<td>3F</td>
<td>?</td>
<td>95</td>
<td>5F</td>
<td></td>
<td>127</td>
<td>7F</td>
<td></td>
</tr>
</tbody>
</table>
• Unicode

• Unicode provides over 100,000 characters and symbols
  • for characters, symbols, etc.
• UTF: Unicode Transformation Format

• UTF-8, UTF-16, UTF-32 etc.
• UTF-32 is fixed length encoding
Variable Length Encoding

- **UTF-16**
  - 2-byte encoding: $0000_{hex}$ .... $FFFF_{hex}$ except
  - 4-byte encoding:
    - $D800_{hex}$ ... $DBFF_{hex}$ + $DC00_{hex}$ ... $DFFF_{hex}$

- **UTF-8**
  - similar principle: variable encoding
  - 1-4 bytes
  - 1-byte encoding compatible to ASCII
String

- string = array of characters

- length of string?
  - terminating character, e.g. NUL
  - start with length field

- string manipulations costly
Data Representation

- 01110111011010000111100100111111
- 01110111 01101000 01111001 00111111
- 7 7 6 8 7 9 3 F
- 77 68 79 3F
- w h y ?

- Or: 2,003,335,487_{dec}
Big Integers

- word size currently 32 or 64 bits
- programming libraries offer big integer types
- complex data structures – more costly
  - operations in software, rather than hardware
Floating Point

- how to express fractional numbers?
- how to express very large numbers?

- scientific notation
  - $-2.34 \times 10^{56}$ – normalized
  - $0.0002 \times 10^{-4}$ – not normalized
  - $987.02 \times 10^{9}$ – not normalized

- binary: $1.0_{\text{two}} x 2^{-1}$
Notes

- Radix point
  - “decimal” point for base-10
  - “binary” point for base-2

- “floating” point representation?
  - no fixed number of digits before and after the radix point; that is, the radix point can float

- vs. fixed-point notation
  - fixed number of digits before and after the point
  - less flexible (but faster – details later)
Terminology

- $I.F \times B^E$
  - $I$ – integer
  - $F$ – fraction
  - $B$ – base
  - $E$ – exponent
  - $I.F$ – significand (mantissa)

Notes

- binary & normalized => first digit is always 1 (one digit before the point)
- base usually implicit – standard is 2
Representation

- standard: IEEE 754
  - single precision: 32 bits
  - double precision: 64 bits
  - quadruple precision: 128 bits

- integer (always 1) not represented
  - this is called *implicit bit*

- need to represent fraction, exponent, signs

- in order to interpret a floating point number, you must know the number of bits used for the sign (usually 1), the exponent part and the fraction part, as well as the bias
Format for a 32-bit Number

- bits 0...22 – fraction
- bits 23...30 – exponent
- bit 31 – sign
- bias is 127

<table>
<thead>
<tr>
<th>Exponent</th>
<th>Fraction</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>nonzero</td>
<td>subnormal</td>
</tr>
<tr>
<td>1-254</td>
<td>anything</td>
<td>normal</td>
</tr>
<tr>
<td>255</td>
<td>0</td>
<td>infinity</td>
</tr>
<tr>
<td>255</td>
<td>nonzero</td>
<td>NaN</td>
</tr>
</tbody>
</table>
Example

- $1 1000 0001 011 0000 0000 0000 0000$
- $S = 1$
- $E = 1000 0001 = 129_{\text{dec}}; 129 - 127 = 2$
- $F = 011 0000 0000 0000 0000$
- Denormalize: $1.011 \times 2^{2} = 101.1$
- Convert: $101.1 = 1 \times (2^{2}) + 0 \times (2^{1}) + 1 \times (2^{0}) + 1 \times (2^{1})$
- $= 4 + 1 + 0.5 = 5.5$
- Add sign: $-5.5 \rightarrow$ final decimal value