What is a Computer?

• A computer consists of a CPU, memory, hard disk, monitor, printer, input and communication devices.
Central Processing Unit (CPU)

- Central Processing Unit (CPU)
  - retrieves instructions from memory and executes them
  - the CPU speed is measured in hertz = cycles per second (Hz, MHz = MegaHertz, GHz = Gigahertz)
    - 1 megahertz = 1 million pulses per second
    - 1 gigahertz = 1 billion pulses per second

Bus

- Storage Devices
  - e.g., Disk, CD, and Tape
- Memory
- CPU
- Communication Devices
  - e.g., Modem, and NIC
- Input Devices
  - e.g., Keyboard, Mouse
- Output Devices
  - e.g., Monitor, Printer
(Main) Memory

- Stores data and program instructions for CPU to execute
  - ordered sequence of bytes (8 bits – binary base unit)
How Data is Stored?

- In binary
- What’s binary?
  - a base-2 number system
- What do humans use?
  - base-10
  - Why? 10 fingers.
- Why do computers like binary?
  - electronics
    - easier to make hardware that stores and processes binary numbers than decimal numbers
  - more efficient: space & cost

<table>
<thead>
<tr>
<th>Memory address</th>
<th>Memory content</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>01001010 Encoding for character ‘J’</td>
</tr>
<tr>
<td>2001</td>
<td>01100001 Encoding for character ‘a’</td>
</tr>
<tr>
<td>2002</td>
<td>01110110 Encoding for character ‘v’</td>
</tr>
<tr>
<td>2003</td>
<td>01100001 Encoding for character ‘a’</td>
</tr>
<tr>
<td>2004</td>
<td>00000011 Encoding for number 3</td>
</tr>
</tbody>
</table>
Number Systems

• The digits in the **decimal number system** are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.
  • A decimal number is represented using a sequence of one or more of these digits.
    • The value that each digit in the sequence represents depends on its position.
    • A position in a sequence has a value that is an integral power of 10.
      • e.g., the digits 7, 4, 2, and 3 in decimal number 7423 represent 7000, 400, 20, and 3, respectively:

\[
\begin{array}{cccc}
7 & 4 & 2 & 3 \\
10^3 & 10^2 & 10^1 & 10^0 \\
= 7 \times 10^3 + 4 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 & = 7000 + 400 + 20 + 3 = 7423
\end{array}
\]

• We say that 10 is the **base** or **radix** of the decimal number system.
  • The base of the binary number system is 2 since the binary number system has two digits: 0 and 1.
  • The base of the hex number system is 16 since the hex number system has sixteen digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.
  • The base of the octal number system is 8 with digits: 0,1,2,3,4,5,6,7.
Number Systems

Decimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Binary: 0, 1

Hexadecimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Octal: 0, 1, 2, 3, 4, 5, 6, 7
Binary

- Computers use binary numbers internally because storage devices like memory and disk are made to store 0s and 1s.
- Each 0 and 1 is called a bit (short for binary digit)
- Binary numbers are not intuitive to us, since we use decimal numbers in our daily life.
  - When you write a number like 20 in a program, it is assumed to be a decimal number.
  - Internally, computer software is used to convert decimal numbers into binary numbers, and vice versa.
- A number or a text inside a computer is stored as a sequence of 0s and 1s.
Binary Numbers => Decimals

Given a binary number \((b_nb_{n-1}b_{n-2}...b_2b_1b_0)_2\)
the equivalent decimal value is

\[ b_n \times 2^n + b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + ... + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0 \]

\((10)_2\) in binary is \(1 \times 2^1 + 0 = 2\) in decimal

\((1010)_2\) in binary is \(1 \times 2^3 + 0 \times 2^2 + 1 \times 2 + 0 = 10\) in decimal

\((10101011)_2\) in binary

\[1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2 + 1 = 171\] in decimal
Binary Numbers => Decimals

\[2^0 = 1\]
\[2^1 = 2\]
\[2^2 = 4\]
\[2^3 = 8\]
\[2^4 = 16\]
\[2^5 = 32\]
\[2^6 = 64\]
\[2^7 = 128\]
\[2^8 = 256\]
\[2^9 = 512\]
\[2^{10} = 1024\]
Decimals => Binary

• To convert a decimal number \( d \) to a binary number is to find the binary digits \( (b_n, b_{n-1}, b_{n-2}, \ldots, b_2, b_1, b_0)_2 \) such that

\[
d = b_n \times 2^n + b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \ldots + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0
\]

• These numbers can be found by successively dividing \( d \) by 2 until the quotient is 0. The remainders are \( b_0, b_1, b_2, \ldots, b_{n-2}, b_{n-1}, b_n \)

For example, the decimal number 123 is \((1111011)_2\) in binary. The conversion is conducted as follows:

\[
\begin{align*}
2 & \overline{123} \\
2 & \overline{61} \\
2 & \overline{30} \\
2 & \overline{15} \\
2 & \overline{7} \\
2 & \overline{3} \\
2 & \overline{1} \\
\end{align*}
\]

\[
\begin{align*}
123 &= 61 \times 2 + 1 \\
61 &= 30 \times 2 + 1 \\
30 &= 15 \times 2 + 0 \\
15 &= 7 \times 2 + 1 \\
7 &= 3 \times 2 + 1 \\
3 &= 1 \times 2 + 1 \\
1 &= 0 \times 2 + 1 \\
\end{align*}
\]

\[
\begin{align*}
b_6 & = 1 \\
b_5 & = 1 \\
b_4 & = 1 \\
b_3 & = 0 \\
b_2 & = 0 \\
b_1 & = 1 \\
b_0 & = 1 \\
\end{align*}
\]

Quotient: 2, Remainder: 1
• Binary numbers tend to be very long and cumbersome:
  • For example: \((1010 1010 1010)_2\)
• Hexadecimal and octal numbers are often used to abbreviate binary numbers:
  • For example: \((1010 1010 1010)_2 = (AAA)_H\)
    and \((101 010 101 010)_2 = (5252)_8\)
• The hexadecimal number system has 16 digits:
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.
  • The letters A, B, C, D, E, and F correspond to the decimal numbers 10, 11, 12, 13, 14, and 15.
  • Each hex digit corresponds to 4 bits
• The octal number system has 8 digits:
  • 0, 1, 2, 3, 4, 5, 6, and 7
  • Each octal digit corresponds to 3 bits
### Hexadecimals <=> Binary

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

To convert a hexadecimal number to a binary number, simply convert each digit in the hexadecimal number into a **four-digit** binary number. For example, 

\[(38D)_H = (1110001101)_2\]

To convert a binary number to a hexadecimal, convert every **four binary digits from right to left** in the binary number into a hexadecimal number. For example, 

\[(1110001101)_2 \rightarrow (38D)_H\]
Octal <=> Binary

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

To convert an octal number to a binary number, simply convert each digit in the octal number into a three-digit binary number. For example, 

\[(1615)_8 = (1110001101)_2\]

To convert a binary number to an octal number, convert every three binary digits from right to left in the binary number into an octal digit. For example,

\[
(\begin{array}{cccc}
1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 \\
1 & 1 & 0 & 1
\end{array})_2
\]

\[
(\begin{array}{cccc}
1 & 6 & 1 & 5
\end{array})_8
\]
Hexadecimals => Decimals

• Given a hexadecimal number \((h_nh_{n-1}h_{n-2}...h_2h_1h_0)_H\)
  The equivalent decimal value is
  \[ h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \ldots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0 \]

  \((7F)_H\) in hex is \(7 \times 16^1 + 15 = 127\) in decimal

  \((FFFF)_H\) in hex \(15 \times 16^3 + 15 \times 16^2 + 15 \times 16 + 15 = 65535\) in decimal

• Octal number system is similar, but \textit{base} is 8.
  \((12)_8\) in octal is \(1 \times 8^1 + 2 = 10\) in decimal
Decimals => Hexadecimals

To convert a decimal number $d$ to a hexadecimal number is to find the hexadecimal digits $(h_n, h_{n-1}, h_{n-2}, \ldots, h_2, h_1, h_0)_H$ such that

$$d = h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \ldots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

These numbers can be found by successively dividing $d$ by 16 until the quotient is 0. The remainders are $h_0, h_1, h_2, \ldots, h_{n-2}, h_{n-1}, h_n$

For example, the decimal number 123 is $(7B)_H$ in hexadecimal. The conversion is conducted as follows:

- Octal number system is similar, but base is 8.
Windows Calculator

The Windows Calculator is a useful tool for performing number conversions. To run it, choose *Programs, Accessories, and Calculator* from the Start button, and switch to Programmer View:
Memory: What goes in each memory segment?

• **Stack Segment**
  - variables declared inside methods
    - removed from memory when a method returns

• **Heap Segment**
  - for dynamic data (whenever you use the operator `new`)
    - i.e., the data for constructed objects
      - persistent as long as an existing a variable references this region of memory

• **Global Segment**
  - data that can be reserved at compile time
    - the programs
    - global data (like static data)
    - constants (e.g., interned Strings)
So Hardware stores 0s & 1s

- How do we store text?
  - Numerically (i.e., using numeric codes)
    - Each character is stored in memory as a number
  - Standard character encoding sets: ASCII, Unicode
    - ASCII uses 1 byte per character (128 chars)
      - For example: ‘A’ is 65
    - Unicode: ~65K different characters
## ASCII Table


<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> </td>
<td>Space</td>
<td>64</td>
<td>40</td>
<td>100</td>
<td> </td>
<td>0</td>
<td>96</td>
<td>60</td>
<td>140</td>
<td> </td>
<td>96</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>!</td>
<td>65</td>
<td>41</td>
<td>101</td>
<td> </td>
<td>A</td>
<td>97</td>
<td>61</td>
<td>141</td>
<td> </td>
<td>97</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> </td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>102</td>
<td> </td>
<td>B</td>
<td>98</td>
<td>62</td>
<td>142</td>
<td> </td>
<td>98</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>#</td>
<td>67</td>
<td>43</td>
<td>103</td>
<td> </td>
<td>C</td>
<td>99</td>
<td>63</td>
<td>143</td>
<td> </td>
<td>99</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>$</td>
<td>68</td>
<td>44</td>
<td>104</td>
<td> </td>
<td>D</td>
<td>100</td>
<td>64</td>
<td>144</td>
<td> </td>
<td>100</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>%</td>
<td>69</td>
<td>45</td>
<td>105</td>
<td> </td>
<td>E</td>
<td>101</td>
<td>65</td>
<td>145</td>
<td> </td>
<td>101</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> </td>
<td> </td>
<td>70</td>
<td>46</td>
<td>106</td>
<td> </td>
<td>F</td>
<td>102</td>
<td>66</td>
<td>146</td>
<td> </td>
<td>102</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></td>
<td>71</td>
<td>47</td>
<td>107</td>
<td> </td>
<td>G</td>
<td>103</td>
<td>67</td>
<td>147</td>
<td> </td>
<td>103</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>(</td>
<td>72</td>
<td>48</td>
<td>110</td>
<td> </td>
<td>H</td>
<td>104</td>
<td>68</td>
<td>150</td>
<td> </td>
<td>104</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>)</td>
<td>73</td>
<td>49</td>
<td>111</td>
<td> </td>
<td>I</td>
<td>105</td>
<td>69</td>
<td>151</td>
<td> </td>
<td>105</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> </td>
<td>*</td>
<td>74</td>
<td>4A</td>
<td>112</td>
<td> </td>
<td>J</td>
<td>106</td>
<td>6A</td>
<td>152</td>
<td> </td>
<td>106</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> </td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>113</td>
<td> </td>
<td>K</td>
<td>107</td>
<td>6B</td>
<td>153</td>
<td> </td>
<td>107</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> </td>
<td></td>
<td>76</td>
<td>4C</td>
<td>114</td>
<td> </td>
<td>L</td>
<td>108</td>
<td>6C</td>
<td>154</td>
<td> </td>
<td>108</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> </td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>115</td>
<td> </td>
<td>M</td>
<td>109</td>
<td>6D</td>
<td>155</td>
<td> </td>
<td>109</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> </td>
<td></td>
<td>78</td>
<td>4E</td>
<td>116</td>
<td> </td>
<td>N</td>
<td>110</td>
<td>6E</td>
<td>156</td>
<td> </td>
<td>110</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> </td>
<td></td>
<td>79</td>
<td>4F</td>
<td>117</td>
<td> </td>
<td>O</td>
<td>111</td>
<td>6F</td>
<td>157</td>
<td> </td>
<td>111</td>
</tr>
<tr>
<td>16</td>
<td>G</td>
<td>020</td>
<td>DLE (data link escape)</td>
<td>48</td>
<td>30</td>
<td>060</td>
<td> </td>
<td></td>
<td>80</td>
<td>50</td>
<td>120</td>
<td> </td>
<td>P</td>
<td>112</td>
<td>70</td>
<td>160</td>
<td> </td>
<td>112</td>
</tr>
<tr>
<td>17</td>
<td>H</td>
<td>021</td>
<td>DC1 (device control 1)</td>
<td>49</td>
<td>31</td>
<td>061</td>
<td> </td>
<td></td>
<td>81</td>
<td>51</td>
<td>121</td>
<td> </td>
<td>Q</td>
<td>113</td>
<td>71</td>
<td>161</td>
<td> </td>
<td>113</td>
</tr>
<tr>
<td>18</td>
<td>I</td>
<td>022</td>
<td>DC2 (device control 2)</td>
<td>50</td>
<td>32</td>
<td>062</td>
<td> </td>
<td></td>
<td>82</td>
<td>52</td>
<td>122</td>
<td> </td>
<td>R</td>
<td>114</td>
<td>72</td>
<td>162</td>
<td> </td>
<td>114</td>
</tr>
<tr>
<td>19</td>
<td>J</td>
<td>023</td>
<td>DC3 (device control 3)</td>
<td>51</td>
<td>33</td>
<td>063</td>
<td> </td>
<td></td>
<td>83</td>
<td>53</td>
<td>123</td>
<td> </td>
<td>S</td>
<td>115</td>
<td>73</td>
<td>163</td>
<td> </td>
<td>115</td>
</tr>
<tr>
<td>20</td>
<td>K</td>
<td>024</td>
<td>DC4 (device control 4)</td>
<td>52</td>
<td>34</td>
<td>064</td>
<td> </td>
<td></td>
<td>84</td>
<td>54</td>
<td>124</td>
<td> </td>
<td>T</td>
<td>116</td>
<td>74</td>
<td>164</td>
<td> </td>
<td>116</td>
</tr>
<tr>
<td>21</td>
<td>L</td>
<td>025</td>
<td>NAK (negative acknowledge)</td>
<td>53</td>
<td>35</td>
<td>065</td>
<td> </td>
<td></td>
<td>85</td>
<td>55</td>
<td>125</td>
<td> </td>
<td>U</td>
<td>117</td>
<td>75</td>
<td>165</td>
<td> </td>
<td>117</td>
</tr>
<tr>
<td>22</td>
<td>M</td>
<td>026</td>
<td>SYN (synchronous idle)</td>
<td>54</td>
<td>36</td>
<td>066</td>
<td> </td>
<td></td>
<td>86</td>
<td>56</td>
<td>126</td>
<td> </td>
<td>V</td>
<td>118</td>
<td>76</td>
<td>166</td>
<td> </td>
<td>118</td>
</tr>
<tr>
<td>23</td>
<td>N</td>
<td>027</td>
<td>ETB (end of trans. block)</td>
<td>55</td>
<td>37</td>
<td>067</td>
<td> </td>
<td></td>
<td>87</td>
<td>57</td>
<td>127</td>
<td> </td>
<td>W</td>
<td>119</td>
<td>77</td>
<td>167</td>
<td> </td>
<td>119</td>
</tr>
<tr>
<td>24</td>
<td>O</td>
<td>030</td>
<td>CAN (cancel)</td>
<td>56</td>
<td>38</td>
<td>070</td>
<td> </td>
<td></td>
<td>88</td>
<td>58</td>
<td>130</td>
<td> </td>
<td>X</td>
<td>120</td>
<td>78</td>
<td>170</td>
<td> </td>
<td>120</td>
</tr>
<tr>
<td>25</td>
<td>P</td>
<td>031</td>
<td>EM (end of medium)</td>
<td>57</td>
<td>39</td>
<td>071</td>
<td> </td>
<td></td>
<td>89</td>
<td>59</td>
<td>131</td>
<td> </td>
<td>Y</td>
<td>121</td>
<td>79</td>
<td>171</td>
<td> </td>
<td>121</td>
</tr>
<tr>
<td>26</td>
<td>Q</td>
<td>032</td>
<td>SUB (substitute)</td>
<td>58</td>
<td>3A</td>
<td>072</td>
<td> </td>
<td></td>
<td>90</td>
<td>5A</td>
<td>132</td>
<td> </td>
<td>Z</td>
<td>122</td>
<td>7A</td>
<td>172</td>
<td> </td>
<td>122</td>
</tr>
<tr>
<td>27</td>
<td>R</td>
<td>033</td>
<td>ESC (escape)</td>
<td>59</td>
<td>3B</td>
<td>073</td>
<td> </td>
<td></td>
<td>91</td>
<td>5B</td>
<td>133</td>
<td> </td>
<td>[</td>
<td>123</td>
<td>7B</td>
<td>173</td>
<td> </td>
<td>123</td>
</tr>
<tr>
<td>28</td>
<td>S</td>
<td>034</td>
<td>FS (file separator)</td>
<td>60</td>
<td>3C</td>
<td>074</td>
<td> </td>
<td></td>
<td>92</td>
<td>5C</td>
<td>134</td>
<td> </td>
<td>\</td>
<td>124</td>
<td>7C</td>
<td>174</td>
<td> </td>
<td>124</td>
</tr>
<tr>
<td>29</td>
<td>T</td>
<td>035</td>
<td>GS (group separator)</td>
<td>61</td>
<td>3D</td>
<td>075</td>
<td> </td>
<td></td>
<td>93</td>
<td>5D</td>
<td>135</td>
<td> </td>
<td>]</td>
<td>125</td>
<td>7D</td>
<td>175</td>
<td> </td>
<td>125</td>
</tr>
<tr>
<td>30</td>
<td>U</td>
<td>036</td>
<td>RS (record separator)</td>
<td>62</td>
<td>3E</td>
<td>076</td>
<td> </td>
<td></td>
<td>94</td>
<td>5E</td>
<td>136</td>
<td> </td>
<td>^</td>
<td>126</td>
<td>7E</td>
<td>176</td>
<td> </td>
<td>126</td>
</tr>
<tr>
<td>31</td>
<td>V</td>
<td>037</td>
<td>US (unit separator)</td>
<td>63</td>
<td>3F</td>
<td>077</td>
<td> </td>
<td></td>
<td>95</td>
<td>5F</td>
<td>137</td>
<td> </td>
<td>_</td>
<td>127</td>
<td>7F</td>
<td>177</td>
<td> </td>
<td>127</td>
</tr>
</tbody>
</table>
A machine language is a set of instructions executed directly by a computer's central processing unit (CPU).

At the beginning there was only machine language: a sequence of bits that directly controls a processor, causing it to add, compare, move data from one place to another.

Example: GCD program in x86 machine language:

```
55  89 e5  53  83  ec  04  83  e4  f0  e8  31  00  00  00  89  c3  e8  2a  00
00  00  39  c3  74  10  8d  b6  00  00  00  00  39  c3  7e  13  29  c3  39  c3
75  f6  89  1c  24  e8  6e  00  00  00  8b  5d  fc  c9  c3  29  d8  eb  eb  90
```
Assembly languages were invented to allow operations to be expressed with mnemonic abbreviations. For example, to add two numbers, you might write an instruction in assembly code like this:

`ADDF3 R1, R2, R3`

A program called assembler is used to convert assembly language programs into machine language.
Machine Language  Assembly Language  High-Level Language

- Example: GCD program in x86 assembly:

```
pushl %ebp
movl %esp, %ebp
pushl %ebx
subl $4, %esp
andl $-16, %esp
call getint
movl %eax, %ebx
call getint
cmpl %eax, %ebx
je C
A: cmpl %eax, %ebx
cmp %eax, %ebx
je D
subl %eax, %ebx
B: cmpl %eax, %ebx
jne A
C: movl %ebx, (%esp)
call putint
movl -4(%ebp), %ebx
leave
ret
D: subl %ebx, %eax
jmp B
```
Assembly was easier to use than binary machine language, **BUT:** not very user friendly, very low-level operations, machine language dependent, programming is very time consuming.

**High Level programming Languages:** languages with strong abstraction from the details of the computer: methods, classes, etc.

– platform independent
– can use previously-developed libraries
– more user friendly, easy to use
Popular High-Level Languages

- FORTRAN (FORmula TRAnslating)
- LISP
- COBOL (COmmom Business Oriented Language)
- BASIC (Beginner All-purpose Symbolic Instructional Code)
- Pascal (named for Blaise Pascal)
- Ada (named for Ada Lovelace)
- C (whose developer designed B first)
- Visual Basic (Basic-like visual language developed by Microsoft)
- Delphi (Pascal-like visual language developed by Borland)
- C++ (a object-oriented language, based on C)
- Java
- C# (a Java-like language developed by Microsoft)
- python
Compiling Source Code

How does a program run?

- Compilers and interpreters.

What’s a compiler?

- A software program that translates the high-level source program into an equivalent target program (typically in machine language), and then goes away:
Interpretation

- Pure Interpretation
  - Interpreter stays around for the execution of the program
  - Interpreter is the locus of control during execution

![Diagram of the interpretation process]
Compilation vs. Interpretation

- Most modern language implementations include a mixture of both compilation and interpretation.
- Compilation followed by interpretation:

![Diagram showing compilation and interpretation process](c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
The **operating system** (OS) is a program that manages and controls a computer’s activities.

Windows
Mac OsX
Android
Linux
Why Java?

Java is somewhat different from older languages. Java started a principle, "write once, run anywhere." What does that mean? Platform (and operator system) independence for compiled Java code.

How?
The Java Virtual Machine (JVM)
Java programs are compiled into Java bytecode. Bytecode is then executed by the JVM on any OS and any platform.
Java Virtual Machine
A program that runs Java programs and manages memory for Java programs.

Why?
Each platform is different (Mac / PC / Linux / Android / etc.)
Java Development Kit (JDK)

- JDK 1.02 (1995)
- J2SE 5.0 (2004)
- Java SE 8 (2014) Long Term Support (LTS)
- Java SE 9 (2017),
- Java SE 10, 11 (LTS) (2018)
- Java SE 12, 13 (2019), Java SE 14 (March 2020)
  ...
- Java SE 17 (2021)
JDK Editions

- **Java Standard Edition (J2SE)**
  - J2SE can be used to develop client-side standalone applications or applets.

- **Java Enterprise Edition (J2EE)**
  - J2EE can be used to develop server-side applications such as Java servlets and Java ServerPages.

- **Java Micro Edition (J2ME)**
  - J2ME was used to develop applications for mobile devices such as cell phones.

Our textbook uses J2SE to introduce Java programming.
A Simple Java Program

// Welcome.java
// This program prints Welcome to Java!
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
Creating, Compiling, and Running Programs

Source code (developed by the programmer)

```java
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```

Byte code (generated by the compiler for JVM to read and interpret, not for you to understand)

```
... Method Welcome()
  0 aload_0
  ...
Method void main(java.lang.String[])
  0 getstatic #2 ...
  3 ldc #3 <String "Welcome to Java!">
  5 invokevirtual #4 ...
  8 return
```

Create/Modify Source Code

Saved on the disk

Compile Source Code

i.e., javac Welcome.java

stored on the disk

If compilation errors

Run Byteode

i.e., java Welcome

Result

If runtime errors or incorrect result
Running Programs from command line

pfodor@sparky ~$ emacs Welcome.java

```java
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```

pfodor@sparky ~$ javac Welcome.java

pfodor@sparky ~$ java Welcome
Welcome to Java!
Optional: Compiling and Running Java from the Command Window

- Set the path to JDK bin directory, so that the console knows where javac and java are:
  ```
  set PATH=c:\Java\jdk1.8.0\bin
  ```
- Set classpath to include the current directory, so that javac and java know where used libraries are:
  ```
  set CLASSPATH=...
  ```
Running Programs in Eclipse
Trace a Program Execution

//This program prints Welcome to Java!
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}

Enter main method
Trace a Program Execution

//This program prints Welcome to Java!
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
Trace a Program Execution

//This program prints Welcome to Java!
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}

print a message to the console
Anatomy of a Java Program

- Comments
- Reserved words
- Modifiers
- Statements
- Blocks
- Classes
- Methods
- The main method
Comments

• Three types of comments in Java:
  • *Line comment*: A line comment is preceded by two slashes (//) in a line.
  • *Paragraph comment*: A paragraph comment is enclosed between /* and */ in one or multiple lines.
  • *javadoc comment*: javadoc comments begin with /*** and end with */.
    • They are used for documenting classes, data, and methods.
    • They can be extracted into an HTML file using JDK's javadoc command.
The code that explains itself let it be (no need to comment).

Good programmers can always figure out what something is done from the code. But it is much more difficult to figure out why or how it was done.

Just use good meaningful names for your identifiers (variables, methods).

```java
public static int baseX2decimal(int base, String s){
    int dec = 0;
    for(int i=0;i<s.length();i++) {
        char c = s.charAt(i);
        // extract the decimal digit from the character 0..9 or A..Z for 10,11,...
        int e = ('0'<=c && c<='9')  
            ? c-'0' 
            : ('a'<=c && c<='z')  
                ? c-'a'+10  
                : c-'A'+10;
        dec = dec*base + e;
    }
    return dec;
}
```

No other comments are needed
Reserved Words (Keywords)

- *Reserved words* or *keywords* are words that have a *specific meaning to the compiler*
- Cannot be used for other purposes in the program
- Example: *class*
  - the word after *class* is the name for the class
Java Keywords

abstract, assert, boolean, break, byte, case, catch, char, class, const, continue, default, do, double, else, enum, extends, false, final, finally, float, for, goto, if, implements, import, instanceof, int, interface, long, native, new, null, package, private, protected, public, return, short, static, strictfp, super, switch, synchronized, this, throw, throws, transient, true, try, void, volatile, while

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
Modifiers

- Java uses certain reserved words called *modifiers* that specify the *properties* of the data, methods, and classes and how they can be used

- Examples: *public, static, private, final, abstract, protected*
  - A *public* datum, method, or class can be accessed by other programs
  - A *private* datum or method cannot be accessed by other programs
Statements

- A statement represents an action or a sequence of actions.
  
  ```java
  System.out.println("Welcome to Java!");
  ```
  
is a statement to display the greeting "Welcome to Java!"

- Every statement in Java ends with a semicolon (;)
Blocks

A pair of braces in a program forms a block that groups components of a program.

public class Test {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}

Class block
Method block
**Block Styles**

- We use **end-of-line style** for braces:

```java
public class Test {
    public static void main(String[] args) {
        System.out.println("Block Styles");
    }
}
```

```java
public class Test {
    public static void main(String[] args) {
        System.out.println("Block Styles");
    }
}
```
Variable, class, and method Names

- What’s an API?
  - Application Programming Interface
  - a library of code / names to use

- What are Names / Identifiers used for?
  - For Variables, Classes, and Methods

- From 2 sources:
  - the Oracle/Sun (or someone else’s) API
  - your own classes, variables, and methods

- Your Identifiers (Names) – Why name them?
  - they are your data and commands, and you’ll need to reference them elsewhere in your program

```java
int myVariable = 5; // Declaration
myVariable = myVariable + 1; // Using the variable
```
Rules for Identifiers

- Should contain only letters, numbers, & '_'
  - '$' is allowed, but only for special use
- Cannot begin with a digit!
- Although it is legal, do not begin with ‘_’ (underscore)
- Uppercase and lowercase letters are considered to be different characters (Java is case-sensitive)

Examples:
- Legal: `myVariable`, `my_class`, `my4Var`
- Illegal: `4myVariable`, `my_class`, `my!Var`, `@#$myClass`
Common Java Naming Conventions

- Variables & Methods start with lower case letters: `radius, getRadius`
- Classes start with upper case letters: `Circle`
- Variables and Class identifiers should generally be nouns: `radius, Circle`
- Method identifiers should be verbs: `getRadius`
- Use Camel notation: `GeometricObject, getRadius`
- Use descriptive names: `Circle, radius, area`

\[
\text{area} = \pi \times \text{radius} \times \text{radius};
\]
Programming Errors

- Syntax Errors
  - Detected by the compiler
- Runtime Errors
  - Causes the program to abort
- Logic Errors
  - Produces incorrect result (may or may not run into a runtime error)
public class ShowSyntaxError {
    public static void main(String[] args) {
        i = 30; // Detected by the compiler
        System.out.println(i + 4);
    }
}

The program does not compile.
public class ShowRuntimeError {
    public static void main(String[] args) {
        int i = 1 / 0;
        // Runtime error: Division with 0
    }
}

The program compiles (because it is syntactically correct), but it crashes at runtime.
public class ShowLogicError {
    // Determine if a number is between 1 and 100 inclusively
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int number = input.nextInt();
        // Display the result
        System.out.println(
            "The number is between 1 and 100, **inclusively**: " +
            ((1 < number) && (number < 100)) );
        // Wrong result if the entered number is 1 or 100
        System.exit(0);
    }
}

The program compiles and may run without a crash, but the results are incorrect.
Logic Errors Debugging

- Logic errors are also called **bugs**
- The process of finding and correcting errors is called **debugging**

Methods of debugging:
- hand-trace the program (i.e., catch errors by reading the program),
- insert print statements in order to show the values of the variables
- for a large, complex program, the most effective approach for debugging is to use a **debugger utility**
Debugger

Debugger is a program that facilitates debugging. You can use a debugger to:

- Set breakpoints where the execution pauses when we are debugging.
- Execute a single statement at a time.
- Trace into or stepping over a method.
- Display variables.
public void refreshDisplay(String option) {
    System.out.println("Option: "+option);
    if (option.equals("b41") || option.equals("b42") || option.equals("b43") ||
        option.equals("b44") || option.equals("b51") || option.equals("b61") ||
        option.equals("b52")) {
        bet = 1;
        l11.setText("Round "+rounds+" +"+Acc11);
        if (option.equals("b41")) {
            bet = 1;
            b41.enable();
        } else if (option.equals("b42")) {
            bet = 2;
            b42.enable();
        } else if (option.equals("b43")) {
            bet = 3;
            b43.enable();
        } else if (option.equals("b44")) {
            bet = 4;
            b44.enable();
        } else if (option.equals("b51")) {
            bet = 5;
            l42.setText("Bet: "+bet);
            b51.enable();
        } else if (option.equals("b52")) {
            bet = 6;
            b61.enable();
        } else if (option.equals("b43")) {
            bet = 6;
            b61.enable();
        } else if (option.equals("b44")) {
            bet = 6;
            b61.enable();
        } else if (option.equals("b52")) {
            bet = 6;
            b61.enable();
        } else {
            System.out.println("Invalid option");
        }
    } else if (option.equals("b42") || option.equals("b43") ||
        option.equals("b44") || option.equals("b51") || option.equals("b61") ||
        option.equals("b52")) {
        bet = 2;
        l42.setText("Bet: "+bet);
        b42.enable();
    } else if (option.equals("b51") ||
        option.equals("b61") || option.equals("b52")) {
        bet = 1;
        l11.setText("Round "+rounds+" +"+Acc11);
        if (option.equals("b51")) {
            bet = 1;
            b51.enable();
        } else if (option.equals("b61")) {
            bet = 2;
            b61.enable();
        } else if (option.equals("b52")) {
            bet = 3;
            b52.enable();
        } else if (option.equals("b43") ||
            option.equals("b44") || option.equals("b51") || option.equals("b61") ||
            option.equals("b52")) {
            bet = 4;
            l42.setText("Bet: "+bet);
            b43.enable();
        } else if (option.equals("b44") ||
            option.equals("b51") || option.equals("b61") || option.equals("b52")) {
            bet = 5;
            l52.setText("Bet: "+bet);
            b44.enable();
        } else if (option.equals("b52")) {
            bet = 6;
            l62.setText("Bet: "+bet);
            b52.enable();
        } else {
            System.out.println("Invalid option");
        }
    } else if (option.equals("b43") ||
        option.equals("b44") || option.equals("b51") || option.equals("b61") ||
        option.equals("b52")) {
        bet = 3;
        l32.setText("Bet: "+bet);
        b43.enable();
    } else if (option.equals("b44") ||
        option.equals("b51") || option.equals("b61") || option.equals("b52")) {
        bet = 4;
        l42.setText("Bet: "+bet);
        b44.enable();
    } else if (option.equals("b52")) {
        bet = 5;
        l52.setText("Bet: "+bet);
        b52.enable();
    } else {
        System.out.println("Invalid option");
    }
    if (bet > 0) {
        if (option.equals("b41") || option.equals("b42") || option.equals("b43") ||
            option.equals("b44") || option.equals("b51") || option.equals("b61") ||
            option.equals("b52")) {
            bet = 1;
            l11.setText("Round "+rounds+" +"+Acc11);
            if (option.equals("b41")) {
                bet = 1;
                b41.enable();
            } else if (option.equals("b42")) {
                bet = 2;
                b42.enable();
            } else if (option.equals("b43")) {
                bet = 3;
                b43.enable();
            } else if (option.equals("b44")) {
                bet = 4;
                b44.enable();
            } else if (option.equals("b51")) {
                bet = 5;
                l52.setText("Bet: "+bet);
                b51.enable();
            } else if (option.equals("b61")) {
                bet = 6;
                l62.setText("Bet: "+bet);
                b61.enable();
            } else if (option.equals("b52")) {
                bet = 6;
                l62.setText("Bet: "+bet);
                b61.enable();
            } else {
                System.out.println("Invalid option");
            }
        } else if (option.equals("b42") || option.equals("b43") ||
            option.equals("b44") || option.equals("b51") || option.equals("b61") ||
            option.equals("b52")) {
            bet = 2;
            l42.setText("Bet: "+bet);
            b42.enable();
        } else if (option.equals("b51") ||
            option.equals("b61") || option.equals("b52")) {
            bet = 1;
            l11.setText("Round "+rounds+" +"+Acc11);
            if (option.equals("b51")) {
                bet = 1;
                b51.enable();
            } else if (option.equals("b61")) {
                bet = 2;
                b61.enable();
            } else if (option.equals("b52")) {
                bet = 3;
                b52.enable();
            } else if (option.equals("b43") ||
                option.equals("b44") || option.equals("b51") || option.equals("b61") ||
                option.equals("b52")) {
                bet = 4;
                l42.setText("Bet: "+bet);
                b43.enable();
            } else if (option.equals("b44") ||
                option.equals("b51") || option.equals("b61") || option.equals("b52")) {
                bet = 5;
                l52.setText("Bet: "+bet);
                b44.enable();
            } else if (option.equals("b52")) {
                bet = 6;
                l62.setText("Bet: "+bet);
                b52.enable();
            } else {
                System.out.println("Invalid option");
            }
        } else if (option.equals("b43") ||
            option.equals("b44") || option.equals("b51") || option.equals("b61") ||
            option.equals("b52")) {
            bet = 3;
            l32.setText("Bet: "+bet);
            b43.enable();
        } else if (option.equals("b44") ||
            option.equals("b51") || option.equals("b61") || option.equals("b52")) {
            bet = 4;
            l42.setText("Bet: "+bet);
            b44.enable();
        } else if (option.equals("b52")) {
            bet = 5;
            l52.setText("Bet: "+bet);
            b52.enable();
        } else {
            System.out.println("Invalid option");
        }
    } else {
        System.out.println("Invalid option");
    }
}

// implement second step of baccarat
```java
System.out.println("Option: "+option);

if (option.equals("b41") || option.equals("b42")) {
    bet = 1;
} else if (option.equals("b43")) {

```