What is a Computer?

A computer consists of a CPU, memory, hard disk, monitor, printer, and communication devices.
• 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
Memory

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

```
Bus

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

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
How Data is Stored?

- **What’s binary?**
  - a base-2 number system

- **What do humans use?**
  - base-10

- **Why?**

- **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</td>
</tr>
<tr>
<td>2001</td>
<td>01100001</td>
</tr>
<tr>
<td>2002</td>
<td>01110110</td>
</tr>
<tr>
<td>2003</td>
<td>01100001</td>
</tr>
<tr>
<td>2004</td>
<td>00000011</td>
</tr>
</tbody>
</table>

Encoding for character ‘J’
Encoding for character ‘a’
Encoding for character ‘v’
Encoding for character ‘a’
Encoding for number 3
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:

\[
\begin{array}{cccc}
7 & 4 & 2 & 3 \\
10^3 & 10^2 & 10^1 & 10^0 \\
\end{array}
\]

\[ = 7 \times 10^3 + 4 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 = 7000 + 400 + 20 + 3 = 7423 \]

• 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
• The base of the hex number system is 16 since the hex number system has sixteen digits.
Number Systems

• Computers use binary numbers internally because storage devices like memory and disk are made to store 0s and 1s.
  • A number or a text inside a computer is stored as a sequence of 0s and 1s.
  • Each 0 and 1 is called a bit (short for binary digit)

• Binary numbers are not intuitive, 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.
Number Systems

Binary: 0, 1

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

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
Number Systems

- Binary numbers tend to be very long and cumbersome:
  - For example: 1010 1010 1010

- Hexadecimal numbers are often used to abbreviate binary numbers:
  - For example: AAA

- 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.

- The octal number system has 8 digits:
  - 0, 1, 2, 3, 4, 5, 6, and 7
  - Each octal digit corresponds to 3 bits
Given a binary number \( b_n b_{n-1} b_{n-2} \ldots b_2 b_1 b_0 \)
the equivalent decimal value is

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

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

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

10101011 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
Decimals => Binary

• To convert a decimal number $d$ to a binary number is to find the binary digits $b_n, b_{n-1}, b_{n-2}, ..., b_2, b_1, b_0$ such that

$$d = b_n \times 2^n + b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + ... + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

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

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

```
  2 | 123
  ---+----
     1 | 61
     2 | 30
     4 | 15
     8 | 7
    16 | 3
    32 | 1
```

Remainder

```
Quotient
  2 | 123
  ---+----
     1 | 61
     2 | 30
     4 | 15
     8 | 7
    16 | 3
    32 | 1
```

Thus, the binary representation of 123 is 1111011.
# Hexadecimals <=> Binary

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
<th>Decimal</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>A</td>
<td>10</td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
<td>11</td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
<td>12</td>
</tr>
<tr>
<td>1101</td>
<td>D</td>
<td>13</td>
</tr>
<tr>
<td>1110</td>
<td>E</td>
<td>14</td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
<td>15</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.

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,
Hexadecimals => Decimals

- The hexadecimal number system has sixteen 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.

Given a hexadecimal number $h_n h_{n-1} h_{n-2} \ldots h_2 h_1 h_0$,

The equivalent decimal value is

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

$7F$ in hex is $7 \times 16^1 + 15 = 127$ in decimal

$FFFF$ in hex $15 \times 16^3 + 15 \times 16^2 + 15 \times 16 + 15 = 65535$ in decimal

- Octal number system is similar, but base is 8.
To convert a decimal number $d$ to a hexadecimal number is to find the hexadecimal digits $h_n, h_{n-1}, h_{n-2}, \ldots, h_2, h_1, h_0$ 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 $7B$ in hexadecimal. The conversion is conducted as follows:

Octal number system is similar, but base is 8.
## Octal <=> Binary

<table>
<thead>
<tr>
<th>Binary</th>
<th>Octal</th>
<th>Decimal</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.

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,

```
1 1 1 0 0 0 1 1 0 1
```

```
1 6 1 5
```
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.
Memory: What goes in each memory segment?

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

- **Heap Segment**
  - for dynamic data (whenever you use new)
  - data for constructed objects
  - persistent as long as an existing object variable references this region of memory

- **Global Segment**
  - data that can be reserved at compile time
  - global data (like static data)
How objects are stored?

- You must understand that in Java, every object/reference variable stores a memory address
  - 32 bit numbers (4 bytes)
  - OR
  - 64 bit numbers (8 bytes)
- These addresses point to memory locations where the objects’ data is stored
So Hardware stores 0s & 1s

- 0101010101010101010101010101 …
- Data is byte addressable
  - we can access or change any byte (group of 8 bits) independently as needed

**How do we store text?**

- Numerically (using its code)
- Each character is stored in memory as a number
- Standard character sets: ASCII & Unicode
  - ASCII uses 1 byte per character
    - ‘A’ is 65
<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</td>
<td>0</td>
<td>0</td>
<td>040</td>
<td>#32</td>
<td>space</td>
<td>64</td>
<td>40</td>
<td>100</td>
<td>#64</td>
<td>`</td>
<td>96</td>
<td>60</td>
<td>140</td>
<td>#96</td>
<td>`</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>001</td>
<td>SOH</td>
<td>1</td>
<td>1</td>
<td>041</td>
<td>#33</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>101</td>
<td>#65</td>
<td>A</td>
<td>97</td>
<td>61</td>
<td>141</td>
<td>#97</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>002</td>
<td>STX</td>
<td>2</td>
<td>2</td>
<td>042</td>
<td>#34</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>102</td>
<td>#66</td>
<td>B</td>
<td>98</td>
<td>62</td>
<td>142</td>
<td>#98</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>003</td>
<td>ETX</td>
<td>3</td>
<td>3</td>
<td>043</td>
<td>#35</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>103</td>
<td>#67</td>
<td>C</td>
<td>99</td>
<td>63</td>
<td>143</td>
<td>#99</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>004</td>
<td>EOT</td>
<td>4</td>
<td>4</td>
<td>044</td>
<td>#36</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>104</td>
<td>#68</td>
<td>D</td>
<td>100</td>
<td>64</td>
<td>144</td>
<td>#100</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>005</td>
<td>ENQ</td>
<td>5</td>
<td>5</td>
<td>045</td>
<td>#37</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>105</td>
<td>#69</td>
<td>E</td>
<td>101</td>
<td>65</td>
<td>145</td>
<td>#101</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>006</td>
<td>ACK</td>
<td>6</td>
<td>6</td>
<td>046</td>
<td>#38</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>106</td>
<td>#70</td>
<td>F</td>
<td>102</td>
<td>66</td>
<td>146</td>
<td>#102</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>007</td>
<td>BEL</td>
<td>7</td>
<td>7</td>
<td>047</td>
<td>#39</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>107</td>
<td>#71</td>
<td>G</td>
<td>103</td>
<td>67</td>
<td>147</td>
<td>#103</td>
<td>g</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>010</td>
<td>BS</td>
<td>8</td>
<td>8</td>
<td>050</td>
<td>#40</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>110</td>
<td>#72</td>
<td>H</td>
<td>104</td>
<td>68</td>
<td>150</td>
<td>#104</td>
<td>h</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>011</td>
<td>TAB</td>
<td>9</td>
<td>9</td>
<td>051</td>
<td>#41</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>111</td>
<td>#73</td>
<td>I</td>
<td>105</td>
<td>69</td>
<td>151</td>
<td>#105</td>
<td>i</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>012</td>
<td>LF</td>
<td>10</td>
<td>A</td>
<td>052</td>
<td>#42</td>
<td>*</td>
<td>74</td>
<td>4A</td>
<td>112</td>
<td>#74</td>
<td>J</td>
<td>106</td>
<td>6A</td>
<td>152</td>
<td>#106</td>
<td>J</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>013</td>
<td>VT</td>
<td>11</td>
<td>B</td>
<td>053</td>
<td>#43</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>113</td>
<td>#75</td>
<td>K</td>
<td>107</td>
<td>6B</td>
<td>153</td>
<td>#107</td>
<td>k</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>014</td>
<td>FF</td>
<td>12</td>
<td>C</td>
<td>054</td>
<td>#44</td>
<td>,</td>
<td>76</td>
<td>4C</td>
<td>114</td>
<td>#76</td>
<td>L</td>
<td>108</td>
<td>6C</td>
<td>154</td>
<td>#108</td>
<td>l</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>015</td>
<td>CR</td>
<td>13</td>
<td>D</td>
<td>055</td>
<td>#45</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>115</td>
<td>#77</td>
<td>M</td>
<td>109</td>
<td>6D</td>
<td>155</td>
<td>#109</td>
<td>m</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>016</td>
<td>SO</td>
<td>14</td>
<td>E</td>
<td>056</td>
<td>#46</td>
<td>.</td>
<td>78</td>
<td>4E</td>
<td>116</td>
<td>#78</td>
<td>N</td>
<td>110</td>
<td>6E</td>
<td>156</td>
<td>#110</td>
<td>n</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>017</td>
<td>SI</td>
<td>15</td>
<td>F</td>
<td>057</td>
<td>#47</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>117</td>
<td>#79</td>
<td>O</td>
<td>111</td>
<td>6F</td>
<td>157</td>
<td>#111</td>
<td>o</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>020</td>
<td>DLE</td>
<td>16</td>
<td>10</td>
<td>060</td>
<td>#48</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>120</td>
<td>#80</td>
<td>P</td>
<td>112</td>
<td>70</td>
<td>160</td>
<td>#112</td>
<td>p</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>021</td>
<td>DC1</td>
<td>17</td>
<td>11</td>
<td>061</td>
<td>#49</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>121</td>
<td>#81</td>
<td>Q</td>
<td>113</td>
<td>71</td>
<td>161</td>
<td>#113</td>
<td>q</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>022</td>
<td>DC2</td>
<td>18</td>
<td>12</td>
<td>062</td>
<td>#50</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>122</td>
<td>#82</td>
<td>R</td>
<td>114</td>
<td>72</td>
<td>162</td>
<td>#114</td>
<td>r</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>023</td>
<td>DC3</td>
<td>19</td>
<td>13</td>
<td>063</td>
<td>#51</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>123</td>
<td>#83</td>
<td>S</td>
<td>115</td>
<td>73</td>
<td>163</td>
<td>#115</td>
<td>s</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>024</td>
<td>DC4</td>
<td>20</td>
<td>14</td>
<td>064</td>
<td>#52</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>124</td>
<td>#84</td>
<td>T</td>
<td>116</td>
<td>74</td>
<td>164</td>
<td>#116</td>
<td>t</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>025</td>
<td>NAK</td>
<td>21</td>
<td>15</td>
<td>065</td>
<td>#53</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>125</td>
<td>#85</td>
<td>U</td>
<td>117</td>
<td>75</td>
<td>165</td>
<td>#117</td>
<td>u</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>026</td>
<td>SYN</td>
<td>22</td>
<td>16</td>
<td>066</td>
<td>#54</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>126</td>
<td>#86</td>
<td>V</td>
<td>118</td>
<td>76</td>
<td>166</td>
<td>#118</td>
<td>v</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>027</td>
<td>ETB</td>
<td>23</td>
<td>17</td>
<td>067</td>
<td>#55</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>127</td>
<td>#87</td>
<td>U</td>
<td>119</td>
<td>77</td>
<td>167</td>
<td>#119</td>
<td>w</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>030</td>
<td>CAM</td>
<td>24</td>
<td>18</td>
<td>070</td>
<td>#56</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>130</td>
<td>#88</td>
<td>X</td>
<td>120</td>
<td>78</td>
<td>170</td>
<td>#120</td>
<td>x</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>031</td>
<td>EM</td>
<td>25</td>
<td>19</td>
<td>071</td>
<td>#57</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>131</td>
<td>#89</td>
<td>Y</td>
<td>121</td>
<td>79</td>
<td>171</td>
<td>#121</td>
<td>y</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>032</td>
<td>SUB</td>
<td>26</td>
<td>1A</td>
<td>072</td>
<td>#58</td>
<td>0</td>
<td>90</td>
<td>5A</td>
<td>132</td>
<td>#90</td>
<td>Z</td>
<td>122</td>
<td>7A</td>
<td>172</td>
<td>#122</td>
<td>z</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>033</td>
<td>ESC</td>
<td>27</td>
<td>1B</td>
<td>073</td>
<td>#59</td>
<td>;</td>
<td>91</td>
<td>5B</td>
<td>133</td>
<td>#91</td>
<td>[</td>
<td>123</td>
<td>7B</td>
<td>173</td>
<td>#123</td>
<td>{</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>034</td>
<td>FS</td>
<td>28</td>
<td>1C</td>
<td>074</td>
<td>#60</td>
<td>&lt;</td>
<td>92</td>
<td>5C</td>
<td>134</td>
<td>#92</td>
<td>\</td>
<td>124</td>
<td>7C</td>
<td>174</td>
<td>#124</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>035</td>
<td>GS</td>
<td>29</td>
<td>1D</td>
<td>075</td>
<td>#61</td>
<td>=</td>
<td>93</td>
<td>5D</td>
<td>135</td>
<td>#93</td>
<td>]</td>
<td>125</td>
<td>7D</td>
<td>175</td>
<td>#125</td>
<td>}</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>036</td>
<td>RS</td>
<td>30</td>
<td>1E</td>
<td>076</td>
<td>#62</td>
<td>&gt;</td>
<td>94</td>
<td>5E</td>
<td>136</td>
<td>#94</td>
<td>^</td>
<td>126</td>
<td>7E</td>
<td>176</td>
<td>#126</td>
<td>~</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>037</td>
<td>US</td>
<td>31</td>
<td>1F</td>
<td>077</td>
<td>#63</td>
<td>?</td>
<td>95</td>
<td>5F</td>
<td>137</td>
<td>#95</td>
<td>_</td>
<td>127</td>
<td>7F</td>
<td>177</td>
<td>#127</td>
<td>DEL</td>
</tr>
</tbody>
</table>
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
```
Programming Languages

Machine Language  Assembly Language  High-Level Language

• Assembly languages were invented to allow operations to be expressed with mnemonic abbreviations
• A program called assembler is used to convert assembly language programs into machine code

For example, to add two numbers, you might write an instruction in assembly code like this:

```
ADDF3 R1, R2, R3
```

![Diagram](image-url)
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:   cmp  %eax, %ebx
jle   D
subl  %eax, %ebx
B:   cmp  %eax, %ebx
jne   A
C:   movl  %ebx, (%esp)
call  putint
movl  $-4(%ebp), %ebx
leave
ret
D:   subl  %ebx, %eax
jmp   B
```
Programming Languages

Machine Language  Assembly Language  High-Level Language

Assembly: Far easier to use than binary machine learning
BUT: not very user friendly, very low-level operations, programming is time consuming

High Level programming Languages (HLL):
– more user friendly, easy to use
– more flexible
– platform independent
Popular High-Level Languages

- FORTRAN (FORmula TRANslation)
- LISP
- COBOL (CCommon 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++ (an object-oriented language, based on C)
- Java
- C# (a Java-like language developed by Microsoft)
- python
What’s a compiler?

- A software program
  - Input: High Level Language source code
  - Output: Assembly Code
- It is typically integrated with an assembly
  - together they can make an executable or binary program
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 independence for compiled Java code

How?

The Java Virtual Machine
Java programs are compiled into Java bytecode
Bytecode is then executed by the
Java Virtual Machine (JVM)
Java Virtual Machine
- A program that runs Java programs and manages memory for Java programs.

Why?
- Each platform is different (Mac/PC/Linux/etc.)
- Java can be used to develop Web applications.
- Java Applets
- Java Web Applications
- Java can also be used to develop applications for handheld devices such as Palm and cell phones
JDK Versions

- JDK 1.02 (1995)
- JDK 1.1 (1996)
- J2SE 1.2 (1998)
- J2SE 1.3 (2000)
- J2SE 5.0 (2004)
- Java SE 6 (2006)
- Java SE 7 (2011)
- 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 (expected March 2020)
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 Server Pages.

- **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)
```plaintext
... 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
```

Steps:
1. **Create/Modify Source Code**
2. **Compile Source Code** i.e., `javac Welcome.java`
3. **Run Bytecode** i.e., `java Welcome`
4. **Result**

- If compilation errors
- If runtime errors or incorrect result

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
Running Programs from command line

```
pfodor@sparky ~$ emacs Welcome.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!
```
Compiling and Running Java from the Command Window

• Set path to JDK bin directory
  set PATH=c:\Java\jdk1.8.0\bin
• Set classpath to include the current directory
  set CLASSPATH=. 
• Compile your source code:
  javac Welcome.java
• Run your bytecode:
  java Welcome
Running Programs in Eclipse
//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!");
    }
}

Execute statement
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
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.
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
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.

  System.out.println("Welcome to Java!");

  is a statement to display the greeting "Welcome to Java!"

• Every statement in Java ends with a semicolon (;)
A pair of braces in a program forms a block that groups components of a program.

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

Blocks

- **Class block**
- **Method block**
We use end-of-line style for braces:

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

End-of-line style
Variable, class, and method names

• What’s an API?
  • Application Programming Interface
  • a library of code to use

• Names
  • For Variables, Classes, and Methods
  • From 2 sources:
    • your own classes, variables, and methods
    • the Oracle/Sun (or someone else’s) API
  • Your Identifiers (Names) – Why name them?
    • they are your data and commands
    • 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, & '_'
  - '$' is allowed, but only for special use
- Cannot begin with a digit!
- Uppercase and lowercase letters are considered to be different characters
- Examples:
  - Legal: myVariable, my_class, my4Var
  - Illegal: 4myVariable, my class, my!Var, @$myClass
Common Java Naming Conventions

- Variables & Methods start with lower case letters: \texttt{x, toString}
- Classes start with upper case letters: \texttt{Person}
- Variables and Class identifiers should generally be nouns
- Method identifiers should be verbs
- Use Camel notation: \texttt{myVariable, MyClass}
- Although it is legal, do not begin with ‘\_’ (underscore).
- Use descriptive names: \texttt{LinkedList, compareTo}

\texttt{area = PI * radius * radius;}

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

- Syntax Errors
  - Detected by the compiler
- Runtime Errors
  - Causes the program to abort
- Logic Errors
  - Produces incorrect result
public class ShowSyntaxError {
    public static void main(String[] args) {
        i = 30; // Detected by the compiler
        System.out.println(i + 4);
    }
}

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
public class ShowRuntimeError {
    public static void main(String[] args) {
        int i = 1 / 0;  // Division with 0
    }
}
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);
    }
}
Logic Errors Debugging

- Logic errors are also called **bugs**
- The process of finding and correcting errors is called **debugging**
- Methods:
  - 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.
- Execute a single statement at a time.
- Trace into or stepping over a method.
- Display variables.
- Display call stack.
- Modify variables.
public void refreshDisplay(String option) {
    System.out.println("Option: "+option);
    if (option.equals("b41") || option.equals("b42")
    11.setText("Round "+rounds++
        "+"Action: "+option);
    if (option.equals("b41") ){
        bet = 1;
    } else if( option.equals("b42") ){
        bet = 2;
    } else if( option.equals("b43") ){
        bet = 3;
    } else if( option.equals("b44") ){
        bet = 4;
    } else bet = 5;
    142.setText("Bet: $"+bet);
    b41.setEnabled(false);
    b42.setEnabled(false);
    b51.setEnabled(true);
    b6.setEnabled(false);
    } else if(option.equals("b52")){
        // implement second step of baccarat
System.out.println("Option: "+option);
if (option.equals("b41") || option.equals("b42")) {
    bet = 1;
} else if (option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals("b44") || option.equals("b45") || option.equals(....)