Notation

• **Byte** = 8 bits (a sequence of 0’s and 1’s)

• To indicate larger amounts of storage, some prefixes taken from the metric system are used
  - One **kilobyte** (KB) = $2^{10}$ bytes = 1024 bytes $\approx 10^3$ bytes
  - One **megabyte** (MB) = 1024 KB = 1,048,576 bytes $\approx 10^6$ bytes
  - One **gigabyte** (GB) = 1024 MB = 1,073,741,824 bytes $\approx 10^9$ bytes
  - One **terabyte** (TB) = 1024 GB $\approx 10^{12}$ bytes
Binary and Decimal Data Representation

- All information on a computer is stored as sequences of 0’s and 1’s.
- Binary numbers have 2 as their base
  - only two digits: 0 and 1
  - $10111_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2 + 1$
- Compare with decimal numbers where 10 is their base:
  - ten digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - $12345 = 1 \cdot 10^4 + 2 \cdot 10^3 + 3 \cdot 10^2 + 4 \cdot 10 + 5$
Converting Binary/Decimal Numbers

**Example.**
This is how we convert a binary number to a decimal number

\[ 10111_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2 + 1 = 16 + 4 + 2 + 1 = 23 \]

**Example.**
This is how we convert the decimal number 13 to a binary number. We divide the decimal number repeatedly by 2 until a quotient of zero results. The successive remainders are the digits of the binary number from right to left. Notation: \( r \) means remainder.

\[ 13 \div 2 = 6r1, \ 6 \div 2 = 3r0, \ 3 \div 2 = 1r1, \ 1 \div 2 = 0r1 \]

Therefore, the binary number is \( 1101_2 \) (we take the remainders \textit{from right to left}). You can check that \( 1101_2 \) is equal to 13.
Hexadecimal Number System

- The hexadecimal system uses the base of 16 to represent numbers
  - The first ten digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - The next six digits are A, B, C, D, E and F representing 10, 11, 12, 13, 14, 15, respectively

- Programmers use this system as a convenient way of organizing groups of binary numbers. One hexadecimal digit occupies 4 bits. Therefore, **1 byte is represented by 2 hexadecimal digits.**

**Examples**
- The decimal number 16 is represented as $10_{16}$ in hexadecimal notation
- The decimal number 26 is represented as $1A_{16}$
- The decimal number 2748 is represented as $ABC_{16}$
There is a close relation between hexadecimal and binary systems

Bin = binary system, Hex = hexadecimal system

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

Note that 1 byte (8 bits) can be represented using two hexadecimal digits.
The schemes for the internal machine representation of characters are based on the assignment of a numeric code to each symbol to be represented.

- **ASCII** (American Standard Code for Information Interchange) is the standard coding scheme used in programming languages.
- It uses 8 bits to represent characters.
- Unfortunately, it can code only 256 different characters (some of them are called control characters).
In order to increase the number of characters to be represented, a new scheme called **Unicode** is used in some programming languages (including C++).

- It uses 16 bits to represent each character, so it can represent 65,536 different symbols.
- It is possible in Unicode to represent symbols from non-Latin languages: Arabic, Russian, Greek, Hebrew, and Thai.
- **ASCII is a subset of Unicode.**
<table>
<thead>
<tr>
<th>Code_{10}</th>
<th>47</th>
<th>48</th>
<th>49</th>
<th>50</th>
<th>51</th>
<th>52</th>
<th>53</th>
<th>54</th>
<th>55</th>
<th>56</th>
<th>57</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code_{16}</td>
<td>2F</td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
</tr>
<tr>
<td>Character</td>
<td>/</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code_{10}</th>
<th>58</th>
<th>59</th>
<th>60</th>
<th>61</th>
<th>62</th>
<th>63</th>
<th>64</th>
<th>65</th>
<th>66</th>
<th>67</th>
<th>68</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code_{16}</td>
<td>3A</td>
<td>3B</td>
<td>3C</td>
<td>3D</td>
<td>3E</td>
<td>3F</td>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
</tr>
<tr>
<td>Character</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code_{10}</th>
<th>91</th>
<th>92</th>
<th>93</th>
<th>94</th>
<th>95</th>
<th>96</th>
<th>97</th>
<th>98</th>
<th>99</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code_{16}</td>
<td>5B</td>
<td>5C</td>
<td>5D</td>
<td>5E</td>
<td>5F</td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
</tr>
<tr>
<td>Character</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
<td>‘</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>
A **byte** is a memory location that can hold eight binary digits, either 0 or 1.

Byte is most often abbreviated as B, hence MB for megabyte (b is used as an abbreviation for bit).

A computer main memory is divided into numbered bytes (starting from 0). The number of a byte is called its **address**.

Memory addresses may be used as arguments for program instructions.

Several (usually 2, 4, or 8) subsequent bytes are combined together (and are sometimes called word).

Each group of bytes has a unique address which is equal to the address of its first byte.
Program Execution

- Program instructions
  - must be stored in main memory in binary form
  - must be expressed in a form that the machine can understand— they must be written in the machine language
  - consists of two parts:
    - a numeric opcode representing a basic machine instruction (load, multiply, add, store,…)
    - addresses of the operands

- The control unit (part of the CPU)
  - fetches each instruction from memory to the instruction register
  - decodes the instruction to determine the operation and the address(es) of the operand(s)
  - fetches the operand
  - performs the required operation (using the ALU if necessary)
Control Unit

Arithmetic Logic Unit

Registers

Main Memory

Secondary Memory

Input Devices

Output Devices

CPU

Data Representation
Programming Languages and Compilers

- **Assembly language**
  - uses names in place of numeric opcodes, and variable names in place of numeric addresses
  - is translated by an assembler into machine code

- **High-level languages (C, C++, Java)**
  - source programs written in high-level languages must be translated into machine code by a compiler or interpreter
  - a compiler produces from the source program an object program that is specific to that machine; a linker connects the object program with libraries producing an executable program; the executable program can be executed any time later
  - an interpreter translates source program statements into machine language, and after each statement is translated, the resulting code (machine instruction) is immediately executed
A compiler of a programming language translates a source program into object program (that is specific for that computer).

Usually it is not possible to run an executable program on platforms other than that where it was generated.

To port a program to another platform, it is necessary to recompile it using a compiler that will generate a different executable program for that platform.

Another approach is to combine compiler and interpreter. *Such an approach is used for Java.*

- a source program is compiled into machine-independent intermediate code, called bytecode
- the bytecode is executed by a special interpreter called Java Virtual Machine (JVM)
Programming

• Programming is the process of translating a problem’s solution into instructions that a computer can process.

• Software Development Life Cycle (SDLC)
  - Analysis: analyze the problem.
  - Design: break the problem down into simple components (use top-down or bottom-up approach)
  - Development (coding): translate the simple components into computer code (use hardware and/or software)
  - Test: test the working solution and fix the problems (try it with different data)
  - Maintenance: keep it working, make requested changes/fixes.