Computer tasks follow a “layered” model

User instructions and interactions

Software
(High-level language)

Assembly language

Machine code

Electrical signals

Definitions

- **High-level programming language**: a language that people use to write computer programs
  - has an English-like syntax
  - examples: Java, C, C++, Perl, VB, C#, Fortran, ...
- **Machine code**: The statements that a computer carries out to perform a certain task
  - binary numbers
- **Assembly language**: A low-level programming language
  - close to machine code, but uses English-like instructions
  - requires more steps to accomplish a task than the same statement in a high-level programming language

Programming procedure

(1) Write the computer program in a high-level programming language

(2) Compile the program (convert it from the programming language into machine code)

(3) Run the program
Assembly language

- Program instructions and registers are accessed by lettered codes
  - like machine code, it takes many instructions to accomplish a single task (3 steps for addition, for example)
- An assembler converts the assembly-language code into machine code, so that the computer can run it
  - similar to a compiler

Instructions

- Each instruction is composed of an opcode and an operand
- Opcode: first 4 bits
- Operand: next 12 bits
  - typically, an address or a number

Note

- Because assemblers and compilers convert programming languages to machine code, they are specific to a particular type of computer
  - e.g., Mac compilers will not run on Windows machines
  - each type of processor has its own assembler
  - machine code is different for each type of computer!

Assembly language example: Super simple CPU (workbook)

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>STOP</td>
<td>Stop the computer</td>
</tr>
<tr>
<td>0001</td>
<td>ADD</td>
<td>Add accumulator to operand</td>
</tr>
<tr>
<td>0010</td>
<td>SUB</td>
<td>Subtract operand from accumulator</td>
</tr>
<tr>
<td>0011</td>
<td>LOD</td>
<td>Load memory cell into accumulator</td>
</tr>
<tr>
<td>0100</td>
<td>LDI</td>
<td>Load immediate into accumulator</td>
</tr>
<tr>
<td>0101</td>
<td>STO</td>
<td>Store accumulator</td>
</tr>
<tr>
<td>0110</td>
<td>INP</td>
<td>Input value and store accumulator</td>
</tr>
<tr>
<td>0111</td>
<td>OUT</td>
<td>Output value from accumulator</td>
</tr>
<tr>
<td>1000</td>
<td>JMP</td>
<td>Jump to instruction</td>
</tr>
<tr>
<td>1001</td>
<td>JNG</td>
<td>Jump to instruction if accumulator &gt; 0</td>
</tr>
<tr>
<td>1010</td>
<td>JZR</td>
<td>Jump to instruction if accumulator = 0</td>
</tr>
<tr>
<td>DAT</td>
<td>Not an instruction, just an indicator that means data is stored here</td>
<td></td>
</tr>
</tbody>
</table>
Example instructions

LOD 56  Load the value that's stored at address 56
STO 56  Store the current accumulator value at address 56
JMP 512  Jump to the instruction stored at address 512
STP 0  Stop the program (the rest of this instruction is ignored)

Example program: Compute 2+3

LDI 2 ; load the number 2 into the accumulator
STO 15 ; store 2 at memory address 15
LDI 3 ; load the number 3 into the accumulator
ADD 15 ; add the number stored at address 15 to
; the value in the accumulator, and store
; the result in the accumulator
OUT 0 ; output whatever's in the accumulator
STO 16 ; store the result at address 16
STP ; we're done!