Where are we?

Chapter 1: The Big Picture
Chapter 2: Binary Values and Number Systems
Chapter 3: Date Representation
Chapter 4. Gates and Circuits
Chapter 5. Computing Components
Chapter 6. Low-Level Programming Languages and Pseudocode
Chapter 7. Problem Solving and Algorithms
Chapter 8. Abstract Data Types and Subproblems
Chapter 9. Object-Oriented Design and High-Level Programming languages
Chapter 10. Operating Systems
Chapter 11. File Systems and Directories
Chapter 12. Information Systems
Chapter 13. Artificial Intelligence
Chapter 14. Simulation, Graphics, Gaming, and Other Applications
Chapter 15. Networks
Chapter 16. The World Wide Web
Chapter 17. Computer Security
Chapter 18. Limitations and Computing
Chapter Goals

- List the operations that a computer can perform
- Describe the important features of the Pep/8 virtual machine
- Distinguish between immediate addressing mode and direct addressing mode
- Write a simple machine-language program
- Distinguish between machine language and assembly language
- Describe the steps in creating and running an assembly-language program
Chapter Goals

- Write a simple program using the Pep/8 virtual machine
- Distinguish between instructions to the assembler and instructions to be translated
- Distinguish between following an algorithm and developing one
- Describe the pseudocode constructs used in expressing an algorithm
- Use pseudocode to express an algorithm
- Describe two approaches to testing
- Design and implement a test plan for a simple assembly-language program
Computer Operations

- **Computer**
  - A programmable electronic device that can store, retrieve, and process data

What operations can a computer execute?
Machine Language

- Machine language
  - The language made up of binary coded instructions built into the hardware of a particular computer and used directly by the computer

- Why would anyone choose to use machine language?
  - (Hint: they had no choice. Why?)
Machine Language

- Characteristics of machine language:
  - Every processor type has its own set of specific machine instructions
  - The relationship between the processor and the instructions it can carry out is completely integrated
  - Each machine-language instruction does only one very low-level task
Pep/8 Virtual Computer

- **Virtual computer**
  - A hypothetical machine designed to contain the important features of a real computer that we want to illustrate

- **Pep/8**
  - A virtual computer designed by Stanley Warford that has 39 machine-language instructions

- We are not going to cover all of them!
Features in Pep/8

- Pep/8 Registers/Status Bits Covered
  - The program counter (PC) (contains the address of the next instruction to be executed)
  - The instruction register (IR) (contains a copy of the instruction being executed)
  - The accumulator (A register)

- The memory unit is made up of 65,636 bytes of storage
Architecture of Pep/8

Figure 6.1 Pep/8’s Architecture

Pep/8’s CPU (as discussed in this chapter)

- A register (accumulator)
- Program counter (PC)
- Instruction register (IR)

Pep/8’s Memory

```
0000
0001
0002
...
FFFF
FFFF
```
Instruction Format

Figure 6.2 Pep/8 Instruction Format
Instruction Format

- **Operation code**
  - Specifies which instruction is to be carried out

- **Register specifier**
  - Specifies which register is to be used (only use A in this chapter)

- **Addressing-mode specifier**
  - Says how to interpret the operand part of the instruction
Instruction Format

Figure 6.3 Difference between immediate addressing mode and direct addressing mode
### Some Sample Instructions

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Meaning of Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Stop execution</td>
</tr>
<tr>
<td>1100</td>
<td>Load the operand into the A register</td>
</tr>
<tr>
<td>1110</td>
<td>Store the contents of the A register into the operand</td>
</tr>
<tr>
<td>0111</td>
<td>Add the operand to the A register</td>
</tr>
<tr>
<td>1000</td>
<td>Subtract the operand to the A register</td>
</tr>
<tr>
<td>01001</td>
<td>Character input to the operand</td>
</tr>
<tr>
<td>01010</td>
<td>Character output from the operand</td>
</tr>
</tbody>
</table>

*Figure 6.4 Subset of Pep/8 instructions*
Sample Instructions

What do these instructions mean?

Load the operand into the register A
Sample Instructions

What do these instructions mean?

Store the contents of the register A into the operand
Sample Instructions

What do these instructions mean?

Instruction specifier

Operand specifier

Instruction specifier

Operand specifier
Sample Instructions

What do these instructions mean?

- Instruction specifier: 01001001
- Operand specifier: 0000000000000000101010
Sample Instructions

What do these instructions mean?

Instruction specifier: 010100000

Operand specifier: 000000000000110000001

Instruction specifier: 01010001

Operand specifier: 000000000000000101010
<table>
<thead>
<tr>
<th>Left Digit(s)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</td>
<td>BEL</td>
<td>BS</td>
<td>HT</td>
</tr>
<tr>
<td>1</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
<td>CR</td>
<td>SO</td>
<td>SI</td>
<td>DLE</td>
<td>DC1</td>
<td>DC2</td>
<td>DC3</td>
</tr>
<tr>
<td>2</td>
<td>DC4</td>
<td>NAK</td>
<td>SYN</td>
<td>ETB</td>
<td>CAN</td>
<td>EM</td>
<td>SUB</td>
<td>ESC</td>
<td>FS</td>
<td>GS</td>
</tr>
<tr>
<td>3</td>
<td>RS</td>
<td>US</td>
<td></td>
<td>!</td>
<td>“</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
</tr>
<tr>
<td>4</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</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>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>7</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>8</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
<td>\</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>9</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
</tr>
<tr>
<td>10</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
</tr>
<tr>
<td>11</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
<td></td>
<td>~</td>
<td>DEL</td>
<td></td>
</tr>
</tbody>
</table>
## Written Algorithm of Hello

<table>
<thead>
<tr>
<th>Action</th>
<th>Binary Instruction</th>
<th>Hex Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write “H”</td>
<td>01010000 0000000000100100</td>
<td>50 0048</td>
</tr>
<tr>
<td>Write “e”</td>
<td>01010000 00000000001100101</td>
<td>50 0065</td>
</tr>
<tr>
<td>Write “l”</td>
<td>01010000 00000000001101100</td>
<td>50 006C</td>
</tr>
<tr>
<td>Write “l”</td>
<td>01010000 00000000001101100</td>
<td>50 006C</td>
</tr>
<tr>
<td>Write “o”</td>
<td>01010000 00000000001101111</td>
<td>50 006F</td>
</tr>
<tr>
<td>Stop</td>
<td>00000000</td>
<td>00</td>
</tr>
</tbody>
</table>
Hand Simulation

What is the fetch/execute cycle?
How much is the PC incremented?
Hand Simulation

What is the fetch/execute cycle here?
Pep/8 Simulator

- **Pep8/Simulator**
  - A program that behaves just like the Pep/8 virtual machine behaves

- **To run a program**
  - Enter the hexadecimal code, byte by byte with blanks between each
What are the "zz"s for?

Pep/8 Simulator

<table>
<thead>
<tr>
<th>Action</th>
<th>Binary Instruction</th>
<th>Hex Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write &quot;H&quot;</td>
<td>01010000&lt;br&gt;00000000001001000</td>
<td>50&lt;br&gt;0048</td>
</tr>
<tr>
<td>Write &quot;e&quot;</td>
<td>01010000&lt;br&gt;00000000001100101</td>
<td>50&lt;br&gt;0065</td>
</tr>
<tr>
<td>Write &quot;I&quot;</td>
<td>01010000&lt;br&gt;00000000001101100</td>
<td>50&lt;br&gt;006C</td>
</tr>
<tr>
<td>Write &quot;I&quot;</td>
<td>01010000&lt;br&gt;</td>
<td>50</td>
</tr>
</tbody>
</table>

Object Code

Hello_
What is a loader? What does it do?

<table>
<thead>
<tr>
<th>Address</th>
<th>E0</th>
<th>00</th>
<th>48</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>E0</td>
<td>00</td>
<td>48</td>
</tr>
<tr>
<td>03</td>
<td>E0</td>
<td>00</td>
<td>65</td>
</tr>
<tr>
<td>06</td>
<td>E0</td>
<td>00</td>
<td>6C</td>
</tr>
<tr>
<td>09</td>
<td>E0</td>
<td>00</td>
<td>6C</td>
</tr>
<tr>
<td>0C</td>
<td>E0</td>
<td>00</td>
<td>6F</td>
</tr>
<tr>
<td>0F</td>
<td>00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Where does execution begin?
Pep/8 Simulator
What does this program do?

<table>
<thead>
<tr>
<th>Action</th>
<th>Binary Instruction</th>
<th>Hex Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input a letter into location F</td>
<td>01001001</td>
<td>49 000F</td>
</tr>
<tr>
<td></td>
<td>0000000000001111</td>
<td></td>
</tr>
<tr>
<td>Input a letter into F + 1</td>
<td>01001001</td>
<td>49 0010</td>
</tr>
<tr>
<td></td>
<td>0000000000010000</td>
<td></td>
</tr>
<tr>
<td>Write out second letter</td>
<td>01010001</td>
<td>51 0010</td>
</tr>
<tr>
<td></td>
<td>0000000000010000</td>
<td></td>
</tr>
<tr>
<td>Write out first letter</td>
<td>01010001</td>
<td>51 000F</td>
</tr>
<tr>
<td></td>
<td>000000000001111</td>
<td></td>
</tr>
<tr>
<td>Stop</td>
<td>00000000</td>
<td>00</td>
</tr>
</tbody>
</table>
Assembly Language

- Assembly language
  - A language that uses mnemonic codes to represent machine-language instructions

- Assembler
  - A program that reads each of the instructions in mnemonic form and translates it into the machine-language equivalent
## Pep/8 Assembly Language

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand, Mode Specifier</th>
<th>Meaning of Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>STOP</td>
<td></td>
<td>Stop execution</td>
</tr>
<tr>
<td>LDA</td>
<td>0x008B,i</td>
<td>Load 008B into register A</td>
</tr>
<tr>
<td>LDA</td>
<td>0x008B,d</td>
<td>Load the contents of location 8B into register A</td>
</tr>
<tr>
<td>STA</td>
<td>0x008B,d</td>
<td>Store the contents of register A into location 8B</td>
</tr>
<tr>
<td>ADDA</td>
<td>0x008B,i</td>
<td>Add 008B to register A</td>
</tr>
<tr>
<td>ADDA</td>
<td>0x008B,d</td>
<td>Add the contents of location 8B to register A</td>
</tr>
<tr>
<td>SUBA</td>
<td>0x008B,i</td>
<td>Subtract 008B for register A</td>
</tr>
<tr>
<td>SUBA</td>
<td>0x008B,d</td>
<td>Subtract the contents of location 8B from register A</td>
</tr>
<tr>
<td>BR</td>
<td></td>
<td>Branch to the location specified in the operand specifier</td>
</tr>
<tr>
<td>CHARI</td>
<td>0x008B,d</td>
<td>Read a character and store it into location 8B</td>
</tr>
<tr>
<td>CHARO</td>
<td>0x008B,i</td>
<td>Write the character 8B</td>
</tr>
<tr>
<td>CHARO</td>
<td>0x000B,d</td>
<td>Write the character stored in location 0B</td>
</tr>
<tr>
<td>DECI</td>
<td>0x008B,d</td>
<td>Read a decimal number and store it into location 8B</td>
</tr>
<tr>
<td>DECO</td>
<td>0x008B,i</td>
<td>Write the decimal number 139 (8B in hex)</td>
</tr>
<tr>
<td>DECO</td>
<td>0x008B,d</td>
<td>Write the decimal number stored in location 8B</td>
</tr>
</tbody>
</table>

Remember the difference between immediate and direct addressing?

i: immediate  
d: direct
## Pep/8 Assembly Language

<table>
<thead>
<tr>
<th>Pseudo-op</th>
<th>Argument</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>.ASCII</td>
<td>“Str\x00”</td>
<td>Represents a string of ASCII bytes</td>
</tr>
<tr>
<td>.BLOCK</td>
<td>Number of bytes</td>
<td>Creates a block of bytes</td>
</tr>
<tr>
<td>.WORD</td>
<td>Value</td>
<td>Creates a word and stores a value in it</td>
</tr>
<tr>
<td>.END</td>
<td></td>
<td>Signals the end of the assembly-language program</td>
</tr>
</tbody>
</table>
Program "Hello"

| CHARO   | 0x0048,I           | ; Output an 'H' |
| CHARO   | 0x0065,I           | ; Output an 'e' |
| CHARO   | 0x006C,I           | ; Output an 'l' |
| CHARO   | 0x006C,I           | ; Output an 'l' |
| CHARO   | 0x006F,I           | ; Output an 'o' |

STOP
.END
### Pep/8 Assembly Language

#### Assembler Listing

<table>
<thead>
<tr>
<th>Addr</th>
<th>Code</th>
<th>Mnemon</th>
<th>Operand</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>500048</td>
<td>CHARO</td>
<td>0x0048,i</td>
<td></td>
</tr>
<tr>
<td>0003</td>
<td>500065</td>
<td>CHARO</td>
<td>0x0065,i</td>
<td></td>
</tr>
<tr>
<td>0006</td>
<td>50006C</td>
<td>CHARO</td>
<td>0x006C,i</td>
<td></td>
</tr>
<tr>
<td>0009</td>
<td>50006C</td>
<td>CHARO</td>
<td>0x006C,i</td>
<td></td>
</tr>
<tr>
<td>000C</td>
<td>50006F</td>
<td>CHARO</td>
<td>0x006F,i</td>
<td></td>
</tr>
<tr>
<td>000F</td>
<td>00</td>
<td>STOP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A New Program

Problem: Read and sum three values and print the sum

How would you do it by hand?
Our Completed Program

BR       main ; Branch around data
sum:    .WORD  0x0000 ; Set up word with zero
num1:   .BLOCK 2 ; Set up a two byte block for num1
num2:   .BLOCK 2 ; Set up a two byte block for num2
num3:   .BLOCK 2 ; Set up a two byte block for num3
main:   LDA    sum,d ; Load zero into the accumulator
        DECI   num1,d ; Read and store num1
        ADDA   num1,d ; Add num1 to accumulator
        DECI   num2,d ; Read and store num2
        ADDA   num2,d ; Add num2 to accumulator
        DECI   num3,d ; Read and store num3
        ADDA   num3,d ; Add num3 to accumulator
        STA    sum,d ; Store accumulator into sum
        DECO   sum,d ; Output sum
        STOP   ; Stop the processing
.END    ; End of the program
Decision Making

BRLT i  Set PC to operand if A < 0
BREQ I  Set PC to operand if A = 0

negMsg:  CHARO  0x0045,i
        BR  finish
main:  LDA  sum,d
        ...
        BRLT  negMsg
        STA  sum,d
        DECO  sum,d
finish:  STOP
Pseudocode

- Pseudocode
  - A mixture of English and formatting to make the steps in an algorithm explicit
- Algorithm to Convert base-10 number to other bases

While (the quotient is not zero)
  - Divide the decimal number by the new base
  - Make the remainder the next digit to the left in the answer
  - Replace the original decimal number with the quotient
Developing an Algorithm

- Two methodologies used to develop computer solutions to a problem
  - Top-down design focuses on the tasks to be done
  - Object-oriented design focuses on the data involved in the solution
- But first, let's look at a way to express algorithms: pseudocode
Pseudocode

- Pseudocode
  - A way of expressing algorithms that uses a mixture of English phrases and indentation to make the steps in the solution explicit

- There are no grammar rules in pseudocode

- Pseudocode is not case sensitive
Following Pseudocode

While (the quotient is not zero)
    Divide the decimal number by the new base
    Set the next digit to the left in the answer to the remainder
    Set the decimal number to the quotient

What is 93 in base 8?

93/8 gives 11 remainder 5
11/6 gives 1 remainder 3
1/ 8 gives 0 remainder 1

answer 1 3 5
Following Pseudocode

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>a. Initial values</strong></td>
<td><strong>b. After first time through loop (93/8)</strong></td>
<td><strong>c. After second time through loop (11/8)</strong></td>
<td><strong>d. After third time through loop (1/8)</strong></td>
<td></td>
</tr>
<tr>
<td>decimalNumber</td>
<td>newBase</td>
<td>quotient</td>
<td>remainder</td>
<td>answer</td>
</tr>
<tr>
<td>93</td>
<td>8</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>11</td>
<td>8</td>
<td>11</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>1</td>
<td>3</td>
<td>35</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>135</td>
</tr>
</tbody>
</table>

_Easier way to organize solution_
Pseudocode for Complete Computer Solution

Write "Enter the new base"
Read newBase
Write "Enter the number to be converted"
Read decimalNumber
Set quotient to 1
WHILE (quotient is not zero)
    Set quotient to decimalNumber DIV newBase
    Set remainder to decimalNumber REM newBase
    Make the remainder the next digit to the left in the answer
    Set decimalNumber to quotient
Write "The answer is "
Write answer
Pseudocode Functionality

- **Variables**
  - Names of places to store values
    - `quotient`, `decimalNumber`, `newBase`

- **Assignment**
  - Storing the value of an expression into a variable
    - Set `quotient` to 64
    - `quotient <- 64`
    - `quotient <- 6 * 10 + 4`
Pseudocode Functionality

- **Output**
  - Printing a value on an output device
  - *Write, Print*

- **Input**
  - Getting values from the outside word and storing them into variables
  - *Get, Read*
Repetition

Repeating a series of statements

Set count to 1

WHILE ( count < 10)

Write "Enter an integer number"

Read aNumber

Write "You entered " + aNumber

Set count to count + 1

How many values were read?
Pseudocode Functionality

Selection

Making a choice to execute or skip a statement (or group of statements)

Read number

IF (number < 0)

Write number + " is less than zero."

or

Write "Enter a positive number."
Read number

IF (number < 0)

Write number + " is less than zero."

Write "You didn't follow instructions."
Selection

Choose to execute one statement (or group of statements) or another statement (or group of statements)

\[ \text{IF ( age < 12 )} \]
\[ \quad \text{Write "Pay children's rate"} \]
\[ \quad \text{Write "You get a free box of popcorn"} \]
\[ \text{ELSE IF ( age < 65 )} \]
\[ \quad \text{Write "Pay regular rate"} \]
\[ \text{ELSE} \]
\[ \quad \text{Write "Pay senior citizens rate"} \]
Problem: Read in pairs of positive numbers and print each pair in order.

WHILE (not done)
    Write "Enter two values separated by blanks"
    Read number1
    Read number2
    Print them in order
Pseudocode Example

How do we know when to stop?
   Let the user tell us how many

Print them in order?
   If first number is smaller
       print first, then second
   If first number if larger
       print second, then first
Write "How many pairs of values are to be entered?"
Read numberOfPairs
Set numberRead to 0
WHILE (numberRead < numberOfPairs)
    Write "Enter two values separated by a blank; press return"
    Read number1
    Read number2
    IF(number1 < number2)
        Print number1 + " " + number2
    ELSE
        Print number2 + " " number1
    Increment numberRead
Translating Pseudocode

To What?

Assembly language
  Very detailed and time consuming
High-level language
  Easy as you'll see in Chapter 9
Testing

- Test plan
  - A document that specifies how many times and with what data the program must be run in order to thoroughly test it

- Code coverage
  - An approach that designs test cases by looking at the code

- Data coverage
  - An approach that designs test cases by looking at the allowable data values
Testing

- Test plan implementation
  - Using the test cases outlined in the test plan to verify that the program outputs the predicted results
Conclusion

- Machine language
- Assembly language
- Algorithm and Pseudocode
- Testing
Picture sources of today’s slides

- Jones & Barlett Learning’s slides
- Prof. Jaehoon Jeong’s slides