Chapter 7
Low-Level Programming Languages

Chapter Goals
- List the operations that a computer can perform
- Discuss the relationship between levels of abstraction and the determination of concrete algorithm steps
- Describe the important features of the Pep/7 virtual machine
- Distinguish between immediate mode addressing and direct addressing

Chapter Goals
- Convert a simple algorithm into a machine-language program
- Distinguish between machine language and assembly language
- Convert a simple algorithm into an assembly-language program
- Distinguish between instructions to the assembler and instruction to be translated
- Design and implement a test plan for a simple assemble-language program

Computer Operations

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

Data and instructions to manipulate the data are logically the same and can be stored in the same place

What operations can a computer execute?
Levels of Abstraction

What do we mean by the expression "levels of abstraction"? Give several examples.

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/7: A Virtual Computer

Virtual computer

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

Pep/7

A virtual computer designed by Stanley Warford that has 32 machine-language instructions

No, we are not going to cover all of them!
Features in Pep/7

Pep/7 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)
- Status bit N (1 if A register is negative; 0 otherwise)
- Status bit Z (1 if the A register is 0; and 0 otherwise)

The memory unit is made up of 4,096 bytes

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 7.3 Difference between immediate-mode and direct-mode addressing

Some Sample Instructions

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Meaning of Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>Stop execution</td>
</tr>
<tr>
<td>00001</td>
<td>Load the operand into the A register</td>
</tr>
<tr>
<td>00010</td>
<td>Store the contents of the A register into operand</td>
</tr>
<tr>
<td>00111</td>
<td>Add the operand to the A register</td>
</tr>
<tr>
<td>00100</td>
<td>Subtract the operand from the A register</td>
</tr>
<tr>
<td>11011</td>
<td>Character input to the operand</td>
</tr>
<tr>
<td>11100</td>
<td>Character output from the operand</td>
</tr>
</tbody>
</table>

Sample Instructions

What do these instructions mean?

<table>
<thead>
<tr>
<th>Instruction specifier</th>
<th>00001000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operand specifier</td>
<td>0000000000000011</td>
</tr>
<tr>
<td>Instruction specifier</td>
<td>00001001</td>
</tr>
<tr>
<td>Operand specifier</td>
<td>00000000011111</td>
</tr>
</tbody>
</table>

Is there something we are not telling you about the addressing mode specifier? How can you tell?
Sample Instructions

What do these instructions mean?

Instruction specifier: 00010001
Operand specifier: 00000000010101

Instruction specifier: 00010000
Operand specifier: 00000000010101

Instruction specifier: 11100010
Operand specifier: 00000000010101

Activity

Write "Hello"
Are we concrete yet?

Write "Hello"
Write "H"
Write "e"
Write "l"
Write "l"
Write "o"
Are we concrete yet?
Algorithms

Write "H"
  Write 48 (hex)
Write "e"
  Write 65 (hex)
  ...
Write "o"
  Write 6F (hex)

Are we concrete yet?

A Program Example

<table>
<thead>
<tr>
<th>Module</th>
<th>Binary Instruction</th>
<th>Hex Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write &quot;H&quot;</td>
<td>11100000 0000000000100100</td>
<td>C068</td>
</tr>
<tr>
<td>Write &quot;e&quot;</td>
<td>11100000 0000000001100101</td>
<td>C06E</td>
</tr>
<tr>
<td>Write &quot;l&quot;</td>
<td>11100000 0000000001101110</td>
<td>C06C</td>
</tr>
<tr>
<td>Write &quot;o&quot;</td>
<td>11100000 000000001101111</td>
<td>C06F</td>
</tr>
<tr>
<td>Stop</td>
<td>00000000</td>
<td>00</td>
</tr>
</tbody>
</table>

Hand Simulation

Program counter (PC)
Instruction Register (IR)

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

Hand Simulation

Program counter (PC)
Instruction Register (IR)

What is the fetch/execute cycle here?
**Hand Simulation**

What is the fetch/execute cycle here?

**Pep/7 Simulator**

A program that behaves just like the Pep/7 virtual machine behaves

To run a program
Enter the hexadecimal code, byte by byte with blanks between each

**Pep/7 Simulator**

What are the "zz"s for?

**Pep/7 Simulator**

What is a loader? What does it do?

Where does execution begin?
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/7 Assembly Language

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operands</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIROT</td>
<td></td>
<td>Step execution</td>
</tr>
<tr>
<td>SLOADA</td>
<td>b1.b0</td>
<td>Load 16-bit from register A</td>
</tr>
<tr>
<td>SLOADC</td>
<td>b1.b0</td>
<td>Load the contents of location b into register A</td>
</tr>
<tr>
<td>CLOAD</td>
<td>b1.b0.b1</td>
<td>Load the contents of registers A to b location A</td>
</tr>
<tr>
<td>ADDA</td>
<td>b1.b0.b0</td>
<td>Add O0000 to register A</td>
</tr>
<tr>
<td>ADDP</td>
<td>b1.b0.b0</td>
<td>Add the contents of location b to register A</td>
</tr>
<tr>
<td>SUBA</td>
<td>b1.b1.b1</td>
<td>Subtract O0000 from register A</td>
</tr>
<tr>
<td>SUBP</td>
<td>b1.b1.b1</td>
<td>Subtract the contents of location b from register A</td>
</tr>
<tr>
<td>CMPAY</td>
<td>b1.b1.b1</td>
<td>Compare A and store if they are equal</td>
</tr>
<tr>
<td>MOV</td>
<td>b1.b1.b1</td>
<td>Move the character from location b to location A</td>
</tr>
<tr>
<td>MOVX</td>
<td>b1.b1.b1</td>
<td>Move the character from register A to location b</td>
</tr>
<tr>
<td>MOVN</td>
<td>b1.b1.b1</td>
<td>Move the character from register A to location b</td>
</tr>
<tr>
<td>MOVY</td>
<td>b1.b1.b1</td>
<td>Move the character from location b to register A</td>
</tr>
<tr>
<td>MOVXN</td>
<td>b1.b1.b1</td>
<td>Move the character from register A to location b</td>
</tr>
<tr>
<td>MOVYN</td>
<td>b1.b1.b1</td>
<td>Move the character from location b to register A</td>
</tr>
<tr>
<td>MOVYX</td>
<td>b1.b1.b1</td>
<td>Move the character from register A to location b</td>
</tr>
<tr>
<td>MOVYNX</td>
<td>b1.b1.b1</td>
<td>Move the character from register A to location b</td>
</tr>
<tr>
<td>MOVYNX</td>
<td>b1.b1.b1</td>
<td>Move the character from register A to location b</td>
</tr>
</tbody>
</table>

What is the difference between operations and pseudo operations?
A New Program

Reading and adding three numbers
Set sum to 0
Read num1
Add num1 to sum
Read num2
Add num2 to sum
Read num3
Add num3 to sum
Write sum

Are we concrete yet?

Our Completed Program

AR Main :branch to location Main
sum: .WORD 0 :set up word with zero as the contents
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: LOADA sum,d :load a copy of sum into accumulator
DEC1 num1,d :read and store a decimal number in num1
ADDA num1,d :add the contents of num1 to accumulator
DEC1 num2,d :read and store a decimal number in num2
ADDA num2,d :add the contents of num2 to accumulator
DEC1 num3,d :read and store a decimal number in num3
ADDA num3,d :add the contents of num3 to accumulator
STORR sum,d :store contents of the accumulator into sum
DECO sum,d :output the contents of sum
STOP :stop the processing
.BRD :end of the program

Decision Making

Mnemonic    Opcode, Mode
             Hypothetical
AREQ       60 :pick up or branch to the entry (P in A is 1)
AREQ       61 :pick up or branch to the entry (P in A is 0)
COMP       28,29 :compare contents of accumulator with contents of
                   address 0080 and set status bits
CMVA       94,95 :compare contents of accumulator with contents of
                   address 0080 and set status bits
           
Remember the status bits A and Z?

Decision Making

Write "Error" if sum is negative.

Add num3 to sum
If sum is negative
Write "Error"
Else
Write sum

Are we concrete yet?
**Decision Making**

Add num3 to sum
If status bit N is 1
Go to NegMsg
Write sum
Quit: STOP
NegMsg: Write the message and go to Quit

**Decision Making**

Read limit
Set sum to 0
While (limit is not zero)
Read number
Set sum to sum + number
Set limit to limit - 1

**Decision Making**

Set the accumulator to limit
Subtract one from the accumulator
Compare accumulator to zero
If status bit Z is 1
    go to Quit
Else
    go to Read
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

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

Important Threads

Programming language
A set of grammar rules, symbols, and special words used to construct a program

Program
A sequence of instructions written to perform a specified task

Syntax
The formal grammar rules governing the construction of valid instructions

Semantics
The rules that give meaning to the instructions

Ethical Issues

Software Piracy and Copyrighting

Have you ever "borrowed" software from a friend?

Have you ever "lent" software to a friend?

Did you know that 107,000 jobs were lost in the US one year due to such "borrowing" and "lending"?
Who am I?
The architecture of the Pep/7 is named after me. Where was I born and from where did I escape? I consulted on which high-level language?

Do you know?
What can Platinum Blue’s software predict?
How much money do software errors (bugs) cost the US economy annually?
To what did the Rosetta Stone provide the key to the translation?