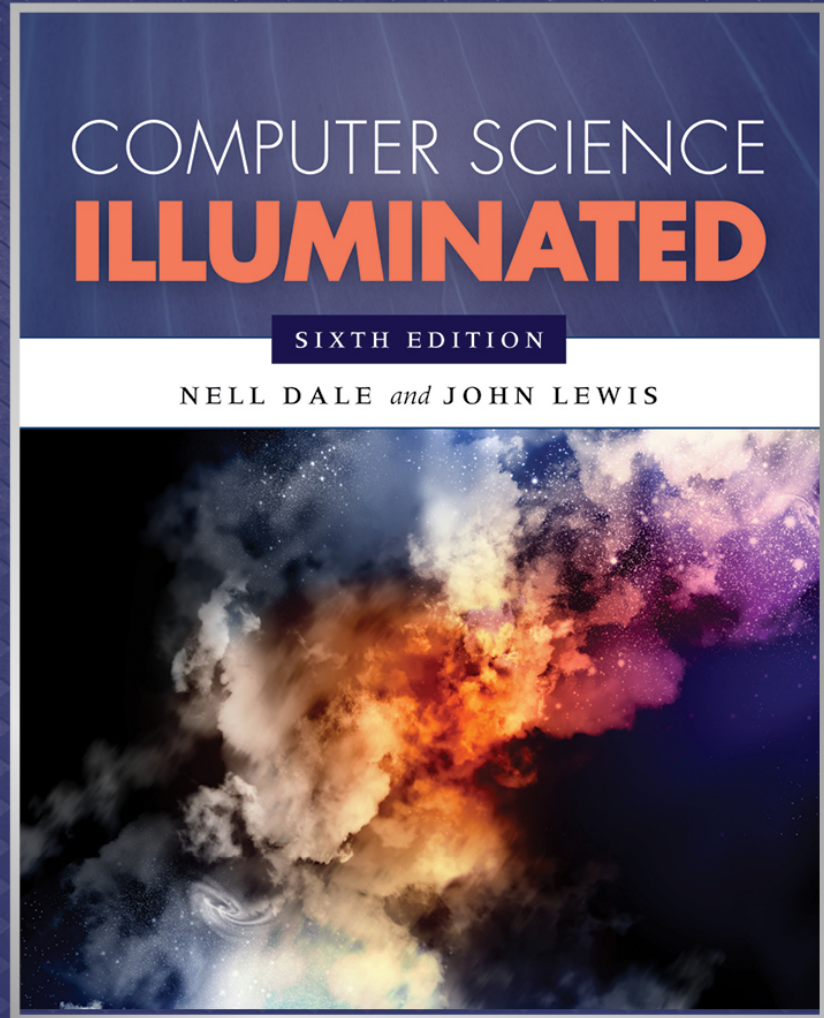


Chapter 6

Low-Level Programming Languages and Pseudocode



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**

Chapter Goals

- Describe the **steps** in **creating** and **running** an assembly-language program
- **Write** a simple program in assembly language
- 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

Chapter Goals

- 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

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?

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 specific set of machine instructions
- The digital logic of the CPU recognizes the binary representations of the instructions
- Each machine-language instruction does only one (typically) 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

No; 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”) (used to hold data and results of operations)

The main memory unit is made up of 64KB
(**65,636 bytes**) of storage

Architecture of Pep/8

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



Pep/8's Memory

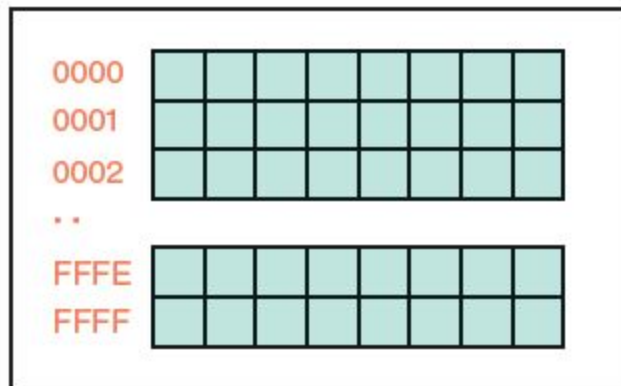


FIGURE 6.1 Pep/8's architecture

Instruction Format

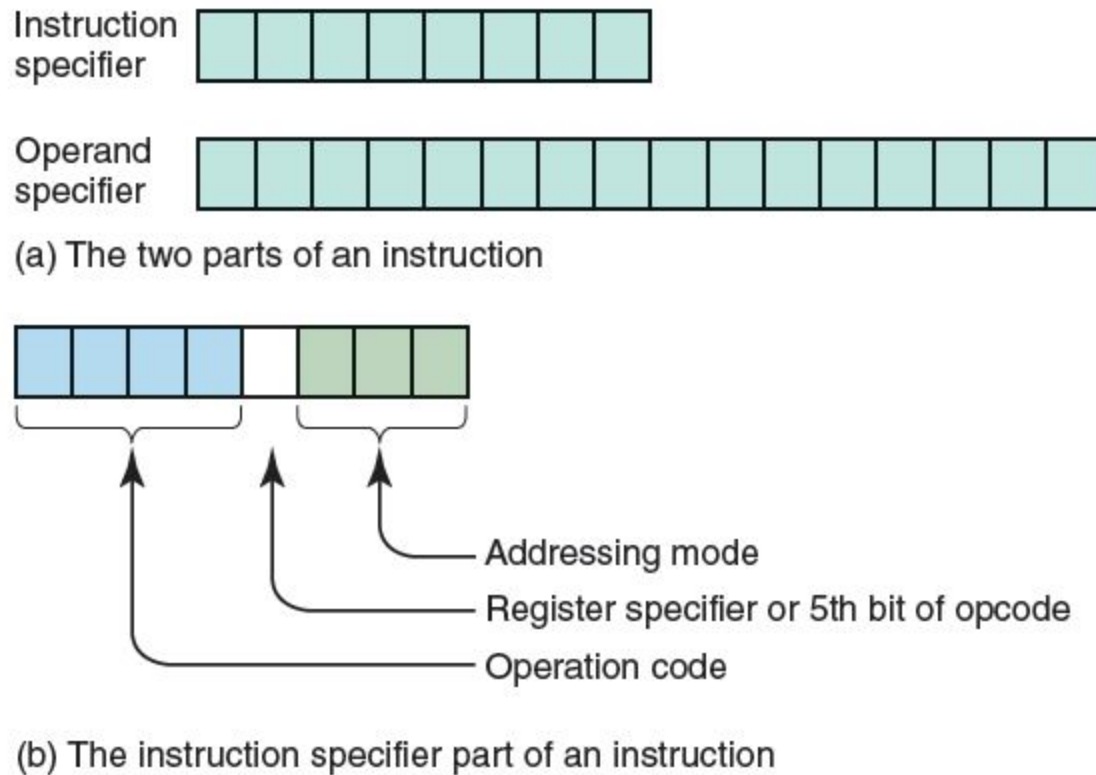


FIGURE 6.2 Pep/8 instruction format

Instruction Format

Operation code

Specifies which instruction is to be carried out

Register specifier

Specifies which register is to be used (for our purposes it always specifies the accumulator)

Addressing-mode specifier

Says how to interpret the operand part of the instruction

Instruction Format

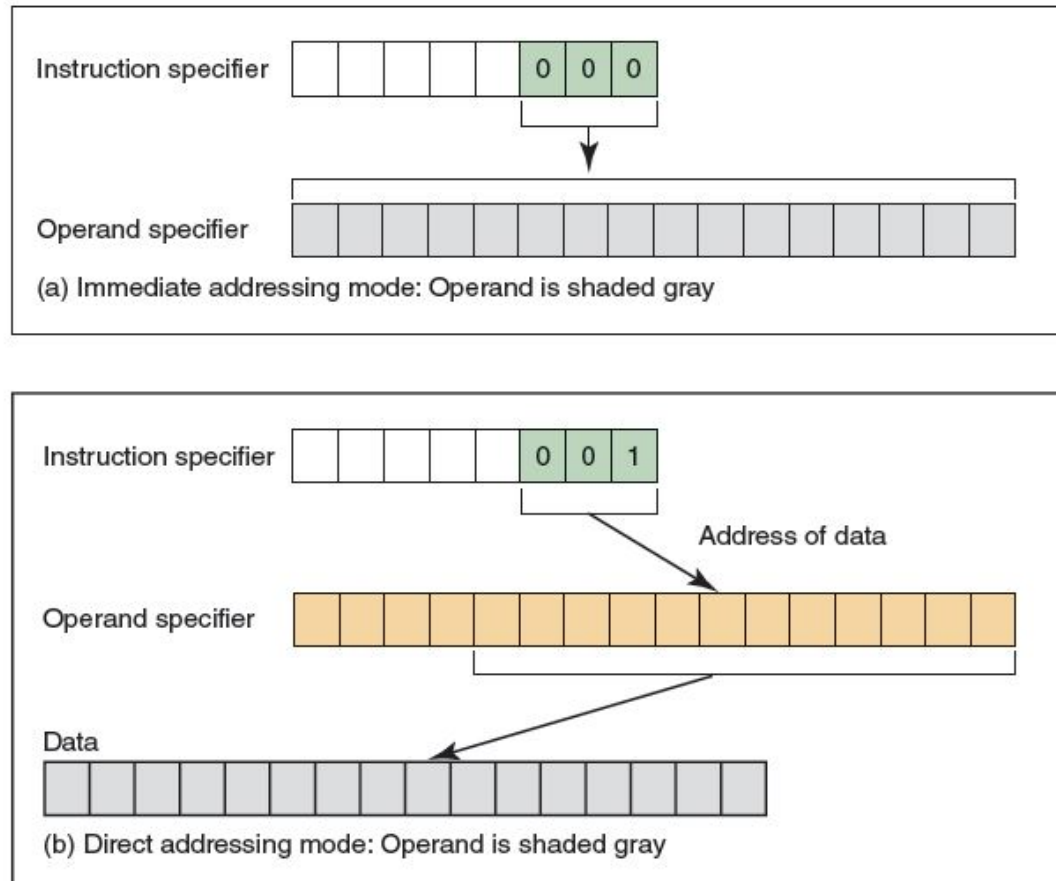


FIGURE 6.3 The difference between immediate addressing mode and direct addressing mode

Instruction Format

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

Some Sample Instructions

Opcode	Meaning of Instruction
0000	Stop execution
1100	Load the operand into the A register
1110	Store the contents of the A register into the operand
0111	Add the operand to the A register
1000	Subtract the operand from the A register
01001	Character input to the operand
01010	Character output from the operand

FIGURE 6.4 Subset of Pep/8 instructions

Sample Instructions

What do these instructions mean?

Instruction specifier

1	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---

Operand specifier

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Instruction specifier

1	1	0	0	0	0	0	1
---	---	---	---	---	---	---	---

Operand specifier

0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Sample Instructions

What do these instructions mean?

Instruction specifier

1	1	1	0	0	0	0	0
---	---	---	---	---	---	---	---

Operand specifier

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Instruction specifier

1	1	1	0	0	0	0	1
---	---	---	---	---	---	---	---

Operand specifier

0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Sample Instructions

What do these instructions mean?

Instruction specifier

0	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---

Operand specifier

0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Instruction specifier

0	1	1	1	0	0	0	1
---	---	---	---	---	---	---	---

Operand specifier

0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Sample Instructions

What do these instructions mean?

Instruction specifier

0	1	0	0	1	0	0	1
---	---	---	---	---	---	---	---

Operand specifier

0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Why is there only one on this page?

Sample Instructions

What do these instructions mean?

Instruction specifier

0	1	0	1	0	0	0	0
---	---	---	---	---	---	---	---

Operand specifier

0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Instruction specifier

0	1	0	1	0	0	0	1
---	---	---	---	---	---	---	---

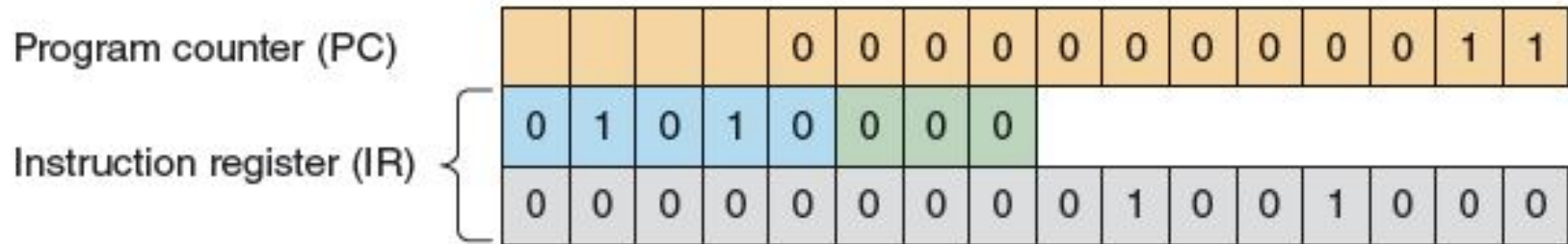
Operand specifier

0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Written Algorithm of Hello

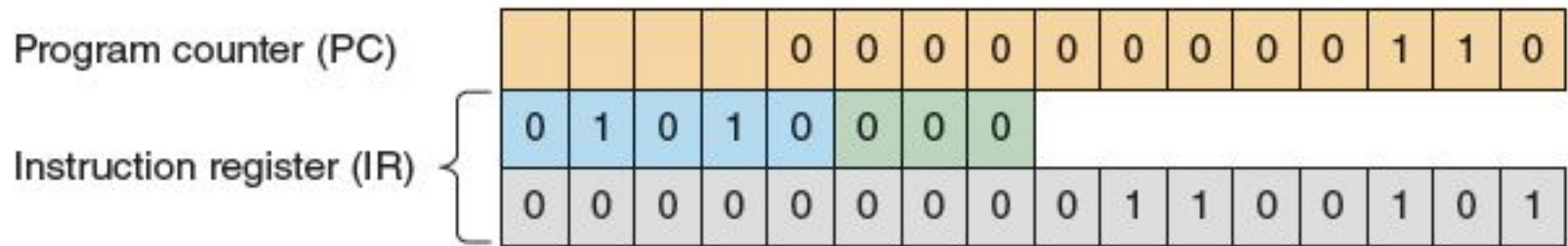
Action	Binary Instruction	Hex Instruction
Write 'H'	01010000 0000000001001000	50 0048
Write 'e'	01010000 0000000001100101	50 0065
Write 'l'	01010000 0000000001101100	50 006C
Write 'l'	01010000 0000000001101100	50 006C
Write 'o'	01010000 0000000001101111	50 006F
Stop	00000000	00

Hand Simulation



*Where in the fetch/execute cycle is this?
How much is the PC incremented?*

Hand Simulation



Where in the fetch/execute cycle is this?

Pep/8 Simulator

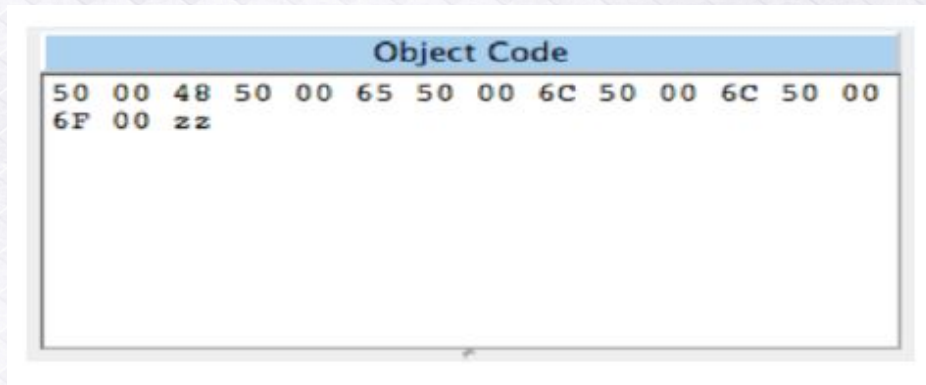
Pep8/Simulator

A program that behaves just like the Pep/8 virtual machine behaves

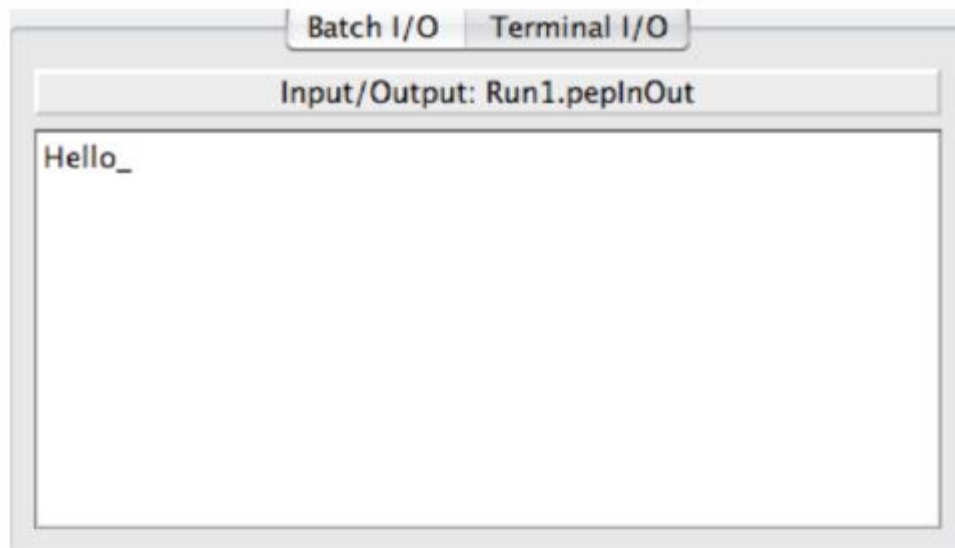
To run a program

Enter the hexadecimal code, byte by byte with blanks between each

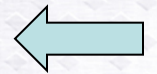
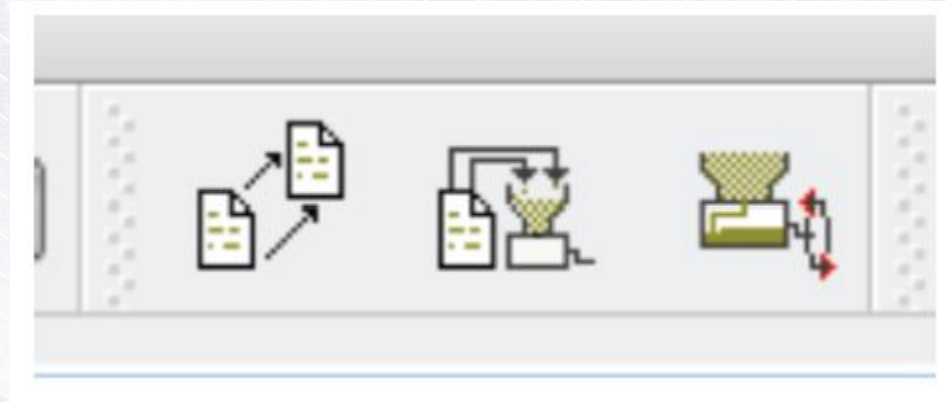
Pep/8 Simulator



*What are the
"zz"s for?*



Pep/8 Simulator



execute



load

Your version may look just a bit different.

Pep/8 Simulator

CPU

☒ Trace Program☐ Trace Traps☐ Trace Load

N0

Z0

V0

C0

Accumulator

0000 (hex)

0 (dec)

(char)

Index Register

0000 (hex)

0 (dec)

Stack Pointer

FBCF (hex)

64463 (dec)

Program Counter

0000 (hex)

Instruction Register

50 (hex)

OpCode

0101 0aaa (bin)

CHARO (mnemonic)

Non-Unary

Addressing Mode Specifier

Operand Specifier

0048

(bin)

000

(mode)

i

Operand

0048

Single Step

Resume

Pep/8 Simulator

Action	Binary Instruction	Hex Instruction
Input a letter into location F	01001001 00000000000001111	49 000F
Input a letter into F + 1	01001001 00000000000010000	49 0010
Write out second letter	01010001 00000000000010000	51 0010
Write out first letter	01010001 00000000000001111	51 000F
Stop	00000000	00

What does this program do?

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

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

Remember the difference between immediate and direct addressing?

*i : immediate
d: direct*

Pep/8 Assembly Language

Pseudo-op	Argument	Meaning
.ASCII	"Str\x00"	Represents a string of ASCII bytes
.BLOCK	Number of bytes	Creates a block of bytes
.WORD	Value	Creates a word and stores a value in it
.END		Signals the end of the assembly-language program

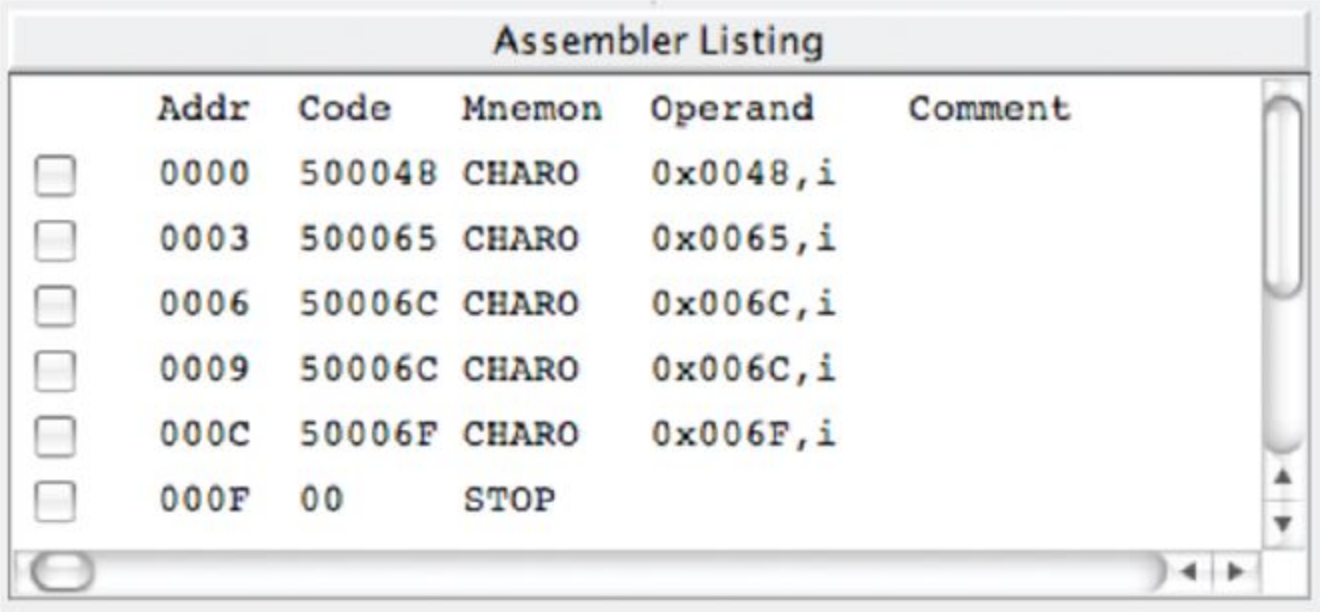
What is the difference between operations and pseudo operations?

Pep/8 Assembly Language

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



The image shows a window titled "Assembler Listing" containing a table of assembly code. Each row in the table has a checkbox on the left. The table has six columns: Addr, Code, Mnemon, Operand, and Comment. The data is as follows:

	Addr	Code	Mnemon	Operand	Comment
<input type="checkbox"/>	0000	500048	CHARO	0x0048,i	
<input type="checkbox"/>	0003	500065	CHARO	0x0065,i	
<input type="checkbox"/>	0006	50006C	CHARO	0x006C,i	
<input type="checkbox"/>	0009	50006C	CHARO	0x006C,i	
<input type="checkbox"/>	000C	50006F	CHARO	0x006F,i	
<input type="checkbox"/>	000F	00	STOP		

Assembly Process

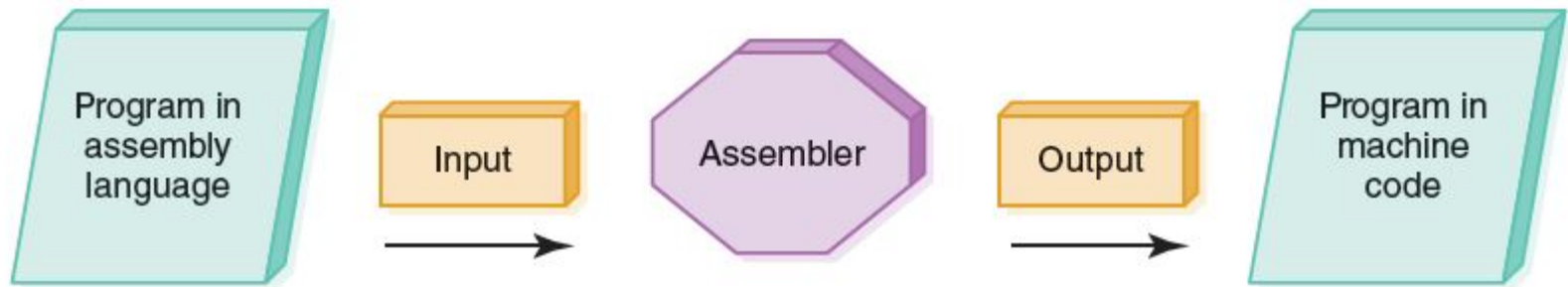


FIGURE 6.5 Assembly process

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
sum:    .WORD                  0x0000
num1:   .BLOCK 2
num2:   .BLOCK 2
num3:   .BLOCK 2
main:   LDA                    sum,d
        DECI                   num1,d
        ADDA                   num1,d
        DECI                   num2,d
        ADDA                   num2,d
        DECI                   num3,d
        ADDA                   num3,d
        STA                    sum,d
        DECO                   sum,d
        STOP
        .END
```


Selection

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

How many ways to finish?

Iteration

Problem: Read and sum limit values.

How many values?

Where does this value come from?

*Will require repeating the reading and
summing*

How do we know when we are done?

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

Following an Algorithm

Never-Fail Blender Hollandaise

1 cup butter
4 egg yolks
1/4 teaspoon salt
1/4 teaspoon sugar

1/4 teaspoon Tabasco
1/4 teaspoon dry mustard
2 tablespoons lemon juice

Heat butter until bubbling. Combine all other ingredients in blender. With blender turned on, pour butter into yolk mixture in slow stream until all is added. Turn blender off. Keeps well in refrigerator for several days. When reheating, heat over hot—not boiling—water in double boiler. Makes about 1-1/4 cups sauce.

Following an Algorithm

Algorithm for preparing a Hollandaise sauce

IF concerned about cholesterol

Put butter substitute in a pot

ELSE

Put butter in a pot

Turn on burner

Put pot on the burner

WHILE (NOT bubbling)

Leave pot on the burner

Put other ingredients in the blender

Turn on blender

WHILE (more in pot)

Pour contents into lender in slow steam

Turn off blender

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, but it's important to be consistent and unambiguous

Following Pseudocode

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

What is 93 in base 8?

93/8 gives 11 remainder 5

11/8 gives 1 remainder 3

1/8 gives 0 remainder 1

answer

1 3 5



Following Pseudocode

(a) Initial values				
<i>decimalNumber</i>	<i>newBase</i>	<i>quotient</i>	<i>remainder</i>	<i>answer</i>
93	8	?	?	?
(b) After first time through loop (93/8)				
<i>decimalNumber</i>	<i>newBase</i>	<i>quotient</i>	<i>remainder</i>	<i>answer</i>
11	8	11	5	5
(c) After second time through loop (11/8)				
<i>decimalNumber</i>	<i>newBase</i>	<i>quotient</i>	<i>remainder</i>	<i>answer</i>
1	8	1	3	35
(d) After third time through loop (1/8)				
<i>decimalNumber</i>	<i>newBase</i>	<i>quotient</i>	<i>remainder</i>	<i>answer</i>
0	8	0	1	135

FIGURE 6.6 Walk-through of a conversion algorithm

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 world and storing them into variables

Get, 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."

Pseudocode Functionality

Selection

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

IF (age < 12)

Write "Pay children's rate"

Write "You get a free box of popcorn"

ELSE IF (age < 65)

Write "Pay regular rate"

ELSE

Write "Pay senior citizens rate"

Pseudocode Functionality

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 Example

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

Otherwise

print second, then first

Pseudocode Example

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

Walk Through

Data

Fill in values during each iteration

3

55 70

2 1

33 33

numberOfPairs

numberRead

number1

number2

What is the output?

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

What does “thoroughly” mean?

Testing

Test plan implementation

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

Important Threads

Operations of a Computer

Computer can store, retrieve, and process data

Computer's Machine Language

A set of instructions the machine's hardware is built to recognize and execute

Machine-language Programs

Written by entering a series of these instructions in binary form

Important Threads

Pep/8: A Virtual Computer with One Register (A) and two-part instructions:

One part tells which action the instruction performs; the other part details where the data to be used can be found

Pep/8 Assembly Language

A language that permits the user to enter mnemonic codes for each instruction rather than binary numbers

Pseudocode

Shorthand-type language people use to express algorithms

Important Threads

Testing Programs

All programs must be tested; code coverage testing and data coverage (black-box testing) are two common approaches

Ethical Issues

Software Piracy and Copyrighting

Have you ever "borrowed" software from a friend?

Have you ever "lent" software to a friend?

According to IDC, lowering software piracy by 10% over the next four years would create 500,000 jobs

Who am I?



© Karl Staedele/dpa/Corbis

Turing, Atanasoff, Eckert, and Mauchly were my contemporaries. Why were we unaware of each other's work?

Do you know?



How is a computer data base helping endangered species?

What would chess grandmaster Jan Helm Donner do if he had a hammer?

What are Nigerian check scams?

Why does an anthropologist work for Intel?

What is the Music Genome Project?

What music streaming website uses the Music Genome Project?

What is the difference between certification and licensing?