Vietnam National University HCMC
INTERNATIONAL UNIVERSITY

Lecture 4: Programming layer

IT064IU - INTRODUCTION TO COMPUTING

DEC 2022

Contents

Contents

1. **Low-Level Programming Languages and Pseudocode**
   - Pep/8 virtual machine
   - Immediate and direct addressing modes
   - machine-language program.
   - assembly-language program.
   - algorithm
   - pseudocode

Computer Operations

A computer is a **programmable** electronic device that can **store**, **retrieve**, and **process** data.

Programmable: The instructions that manipulate data are stored within the machine along with the data. To change what the computer does to the data, we change the instructions.

The instructions that the CU executes can **store** data into the memory, **retrieve** data from the memory, and **process** the data in some way in the ALU.

Machine Language

Machine Language is the language made up of **binary-coded instructions** that is used directly by the computer.

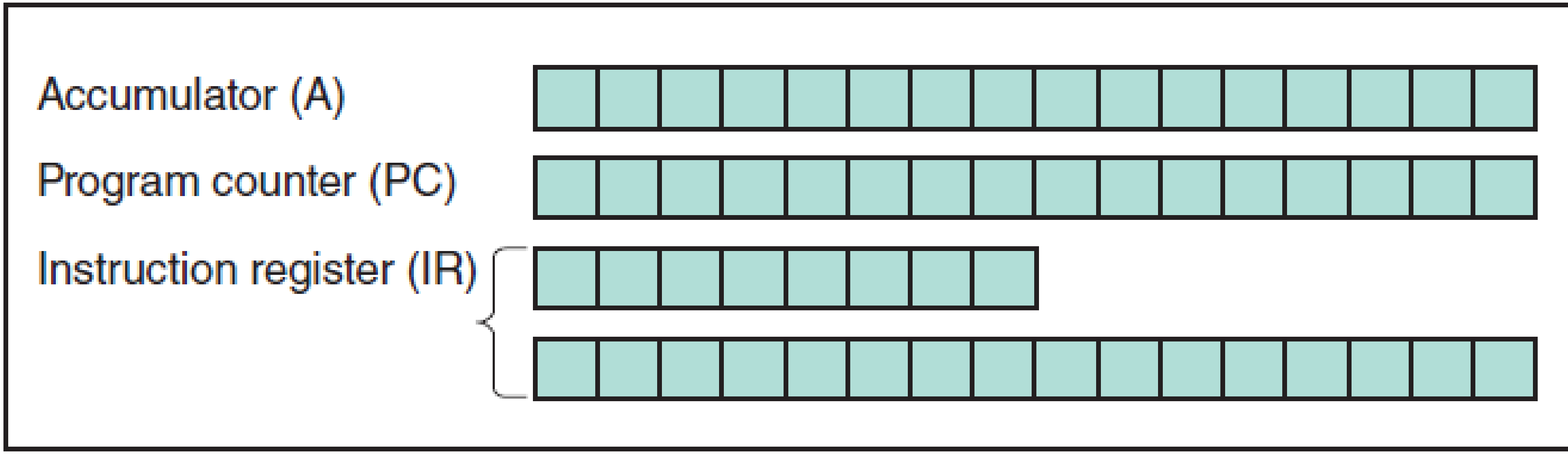
Each machine-language instruction performs **only one very low-level task**.

Machine code **differs from machine to machine**.

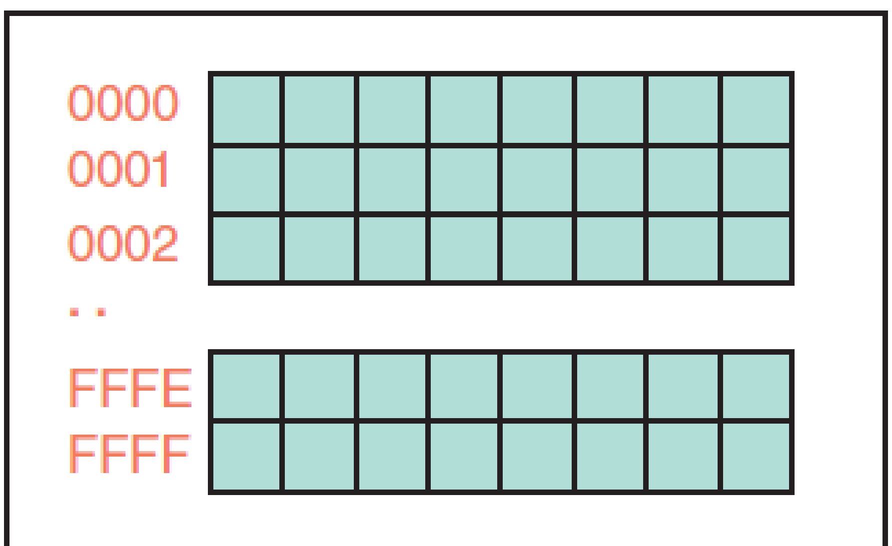
Pep/9: A Virtual Computer

- The memory unit of the Pep/8 is made up of 65,536 bytes of storage, numbered from 0 through 65,535
- The word length in Pep/9 is 2 bytes, or 16 bits.
- Pep/9 has seven registers, three of them are:
  - The program counter (PC), which contains the address of the next instruction to be executed
  - The instruction register (IR), which contains a copy of the instruction being executed
  - The accumulator (A register): holds data and the results of operations; it is the special storage register in the ALU

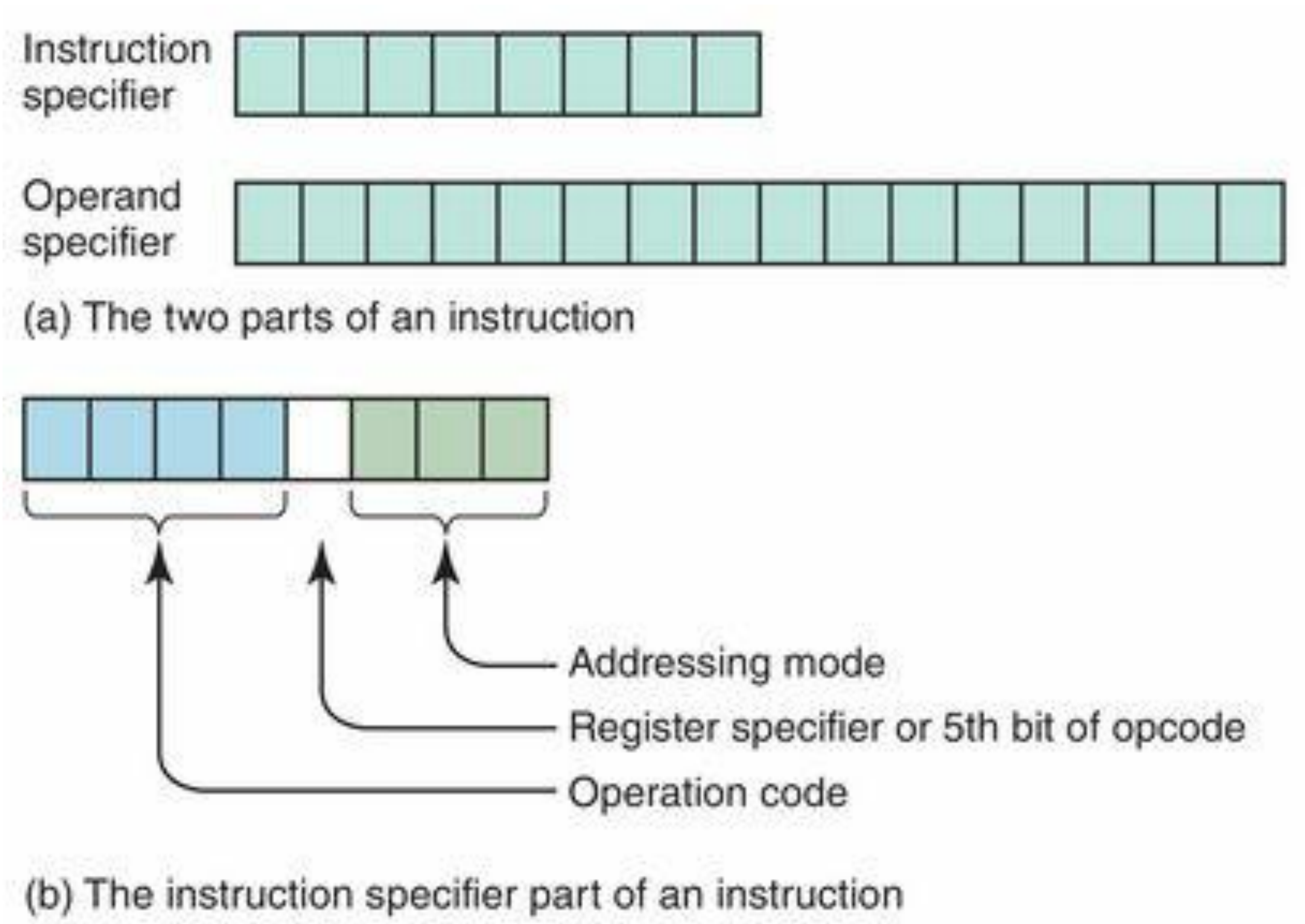
Pep/9's CPU (as discussed in this chapter)

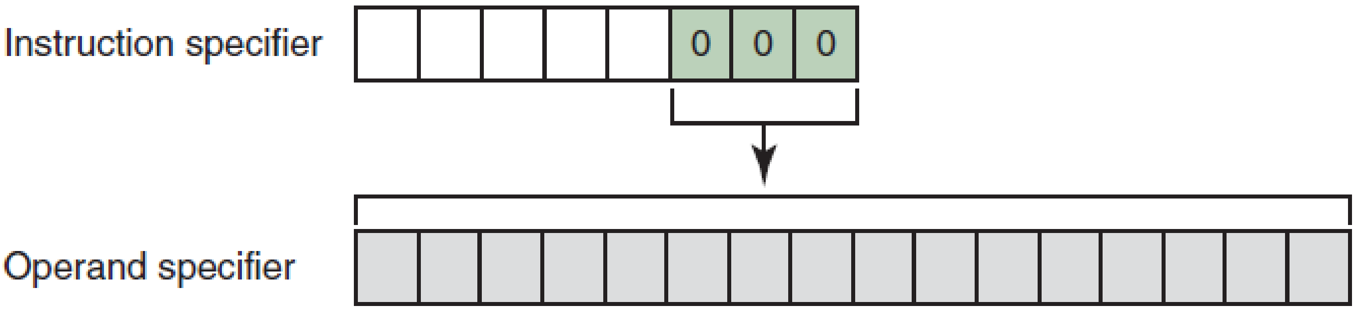


Pep/9's Memory



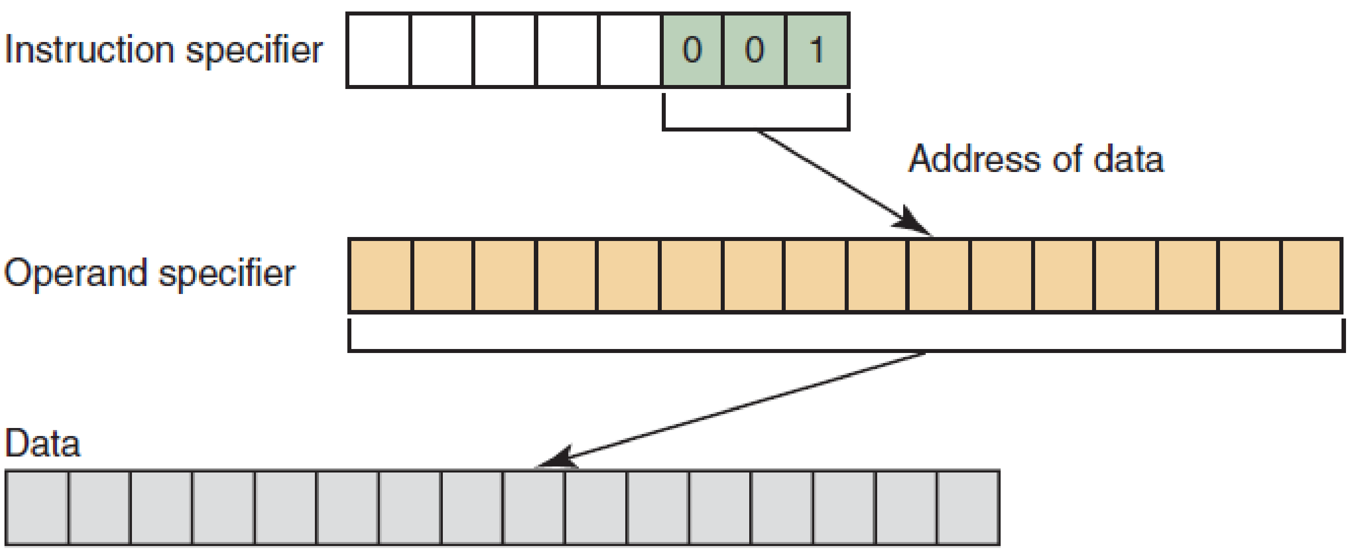
Pep/9 instruction format





(a) Immediate addressing mode: the operand specifier contains the data to be processed.

Immediate addressing mode



(b) Direct addressing mode: the operand specifier contains the address of the data to be processed.

Directly addressing mode

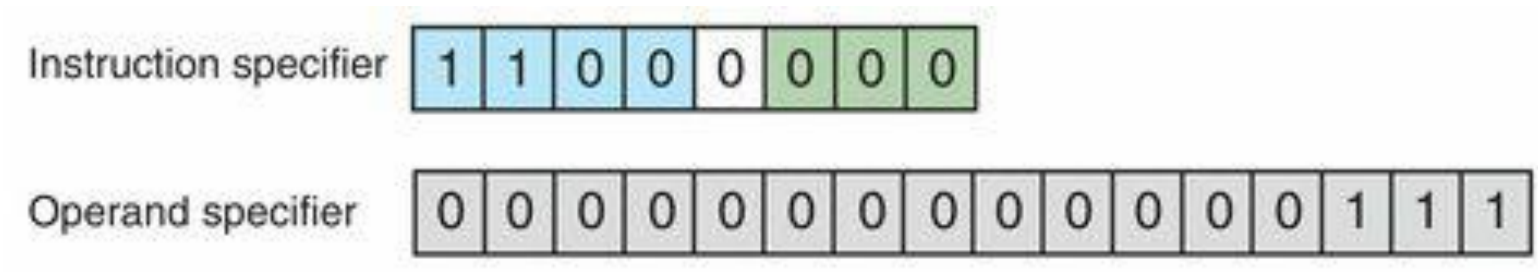
Subset of Pep/9 instructions

Opcode	Meaning of Instruction
0000	Stop execution
1100	Load word into the A register
1101	Load byte into the A register
1110	Store word from the A register
1111	Store byte from the A register
0110	Add the operand to the A register
0111	Subtract the operand from the A register

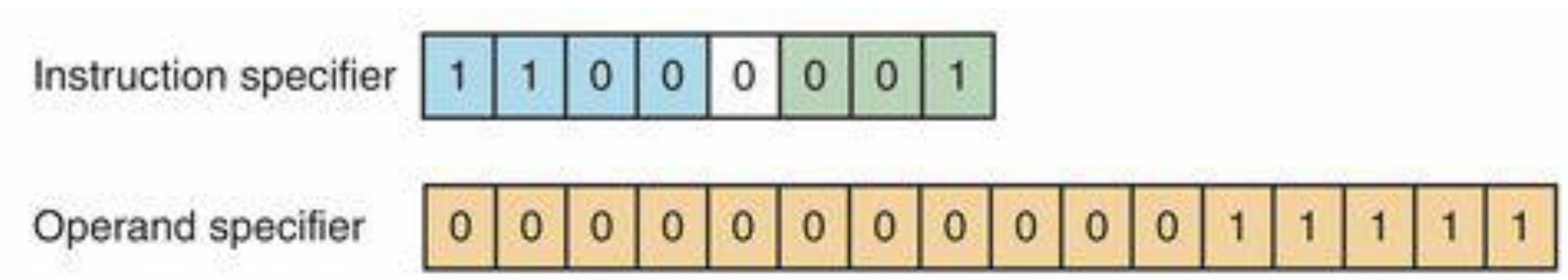
Code samples

1100 Load the word into the A register.

- **Immediate mode**: load 0007 to A



- **Direct mode**: load the content (**1 word**) located from leftmost byte 001F to A



Code samples

1101 Load 1 **byte** into the A register

- Immediate mode: the first byte of the operand specifier is ignored, and only the second byte of the operand specifier is loaded
- Direct mode: only 1 byte is loaded from the memory location specified instead of 2 bytes

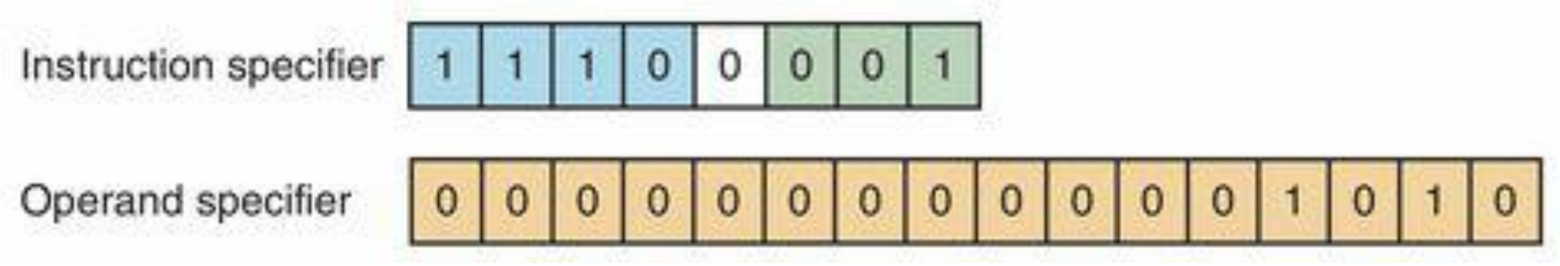
Instruction specifier	1	1	1	0	0	0	0	1								
Operand specifier	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

stores the contents of the A register into the word beginning at location 000A

Code samples

1110 Store word from A register.

- **Direct mode**: stores the contents of the A into **the word** beginning at location 000A.



Code samples

1111 Store byte from the A register

- Similar to previous instruction, but it stores only 1 byte instead of 2 bytes (one word)
- only the second byte of the A register (the accumulator) is stored in the address given in the operand specifier. The first 8 bits of the accumulator are ignored.

Code samples

0110: Add the operand to the A register.

- **Immediate mode**: The contents of the second and third bytes of the instruction (the operand specifier) are added to the contents of the A register (0x20A).

Instruction specifier	0	1	1	0	0	0	0	0

Operand specifier	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0

- **Direct mode**: the contents of the operand located at 0x020A are added into the A register.

Instruction specifier	0	1	1	0	0	0	0	1

Operand specifier	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0

Code samples

0111: Subtract the operand from the A register

- Similar to add operand
- Both immediate and direct addressing modes

Code samples

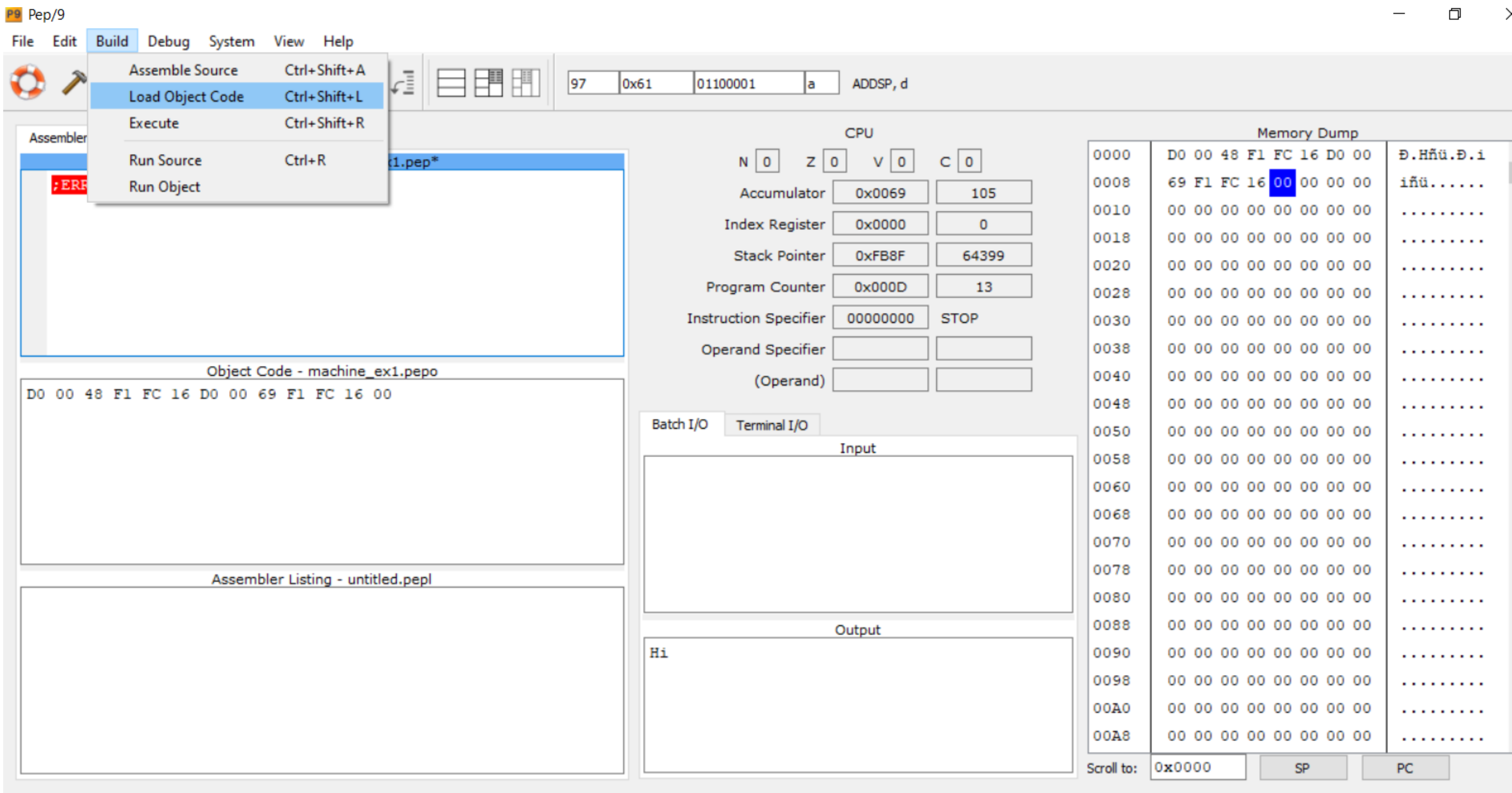
Input and output

- the input device is at address 0xFC15 and the output device is at address 0xFC16
- To read a character from the input device: load the value from the input device address into the accumulator (A register)
- To write a character to the output device: load the character value into the accumulator and then store the value of the accumulator to the output device address.

A Program Example

Action	Binary Instruction	Hex Instruction
Load 'H' into accumulator	1101 0000 0000 0000 0100 1000	D0 00 48
Store byte from accumulator to output device	1111 0001 1111 1100 0001 0110	F1 FC 16
Load 'i' into accumulator	1101 0000 0000 0000 0110 1001	D0 00 69
Store byte from accumulator to output device	1111 0001 1111 1100 0001 0110	F1 FC 16
Stop	0000 0000	00

In Pep/9



Observe the fetch-execute circle using debug -> start debuging object (choose single step)

Another example

Action	Binary Instruction	Hex Instruction
Read first character from input device into accumulator	1101 0001 1111 1100 0001 0101	D1 FC 15
Store character from accumulator to memory	1111 0001 0000 0000 0001 0011	F1 00 13
Read second character from input device into accumulator	1101 0001 1111 1100 0001 0101	D1 FC 15
Print second character to output device	1111 0001 1111 1100 0001 0110	F1 FC 16
Load first character from memory	1101 0001 0000 0000 0001 0011	D1 00 13
Print first character to output device	1111 0001 1111 1100 0001 0110	F1 FC 16
Stop	0000 0000	00

Exercises

Ex. 16-20, p.187

a. A2 11 , b. A2 12

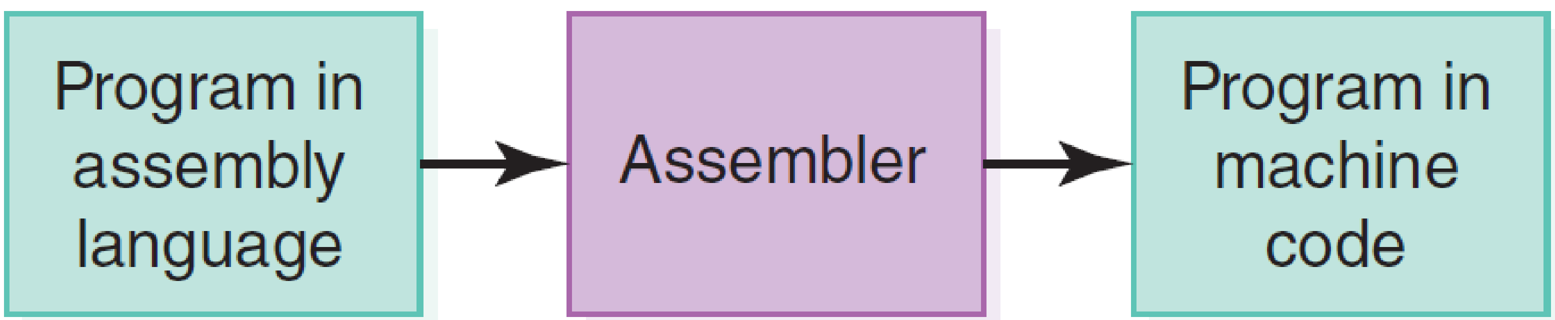
c. 00 02 , d. 11 00

e. 00 FF

0001	A2
0002	11
0003	00
0004	FF

Assembly Language

- A low-level programming language in which a **mnemonic represents** each of the machine-language instructions **for a particular computer**
- **Assembler** A program that translates an assembly-language program in machine code



Pep/9 Assembly Language

Mnemonic	Operand, Mode	Meaning
STOP		Stop execution
LDWA	0x008B,i	Load word 008B into accumulator
LDWA	0x008B,d	Load word located at 008B into accumulator
LDBA	0x008B,i	Load byte 008B into accumulator
LDBA	0x008B,d	Load byte located at 008B into accumulator
STWA	0x008B,d	Store word from accumulator to location 008B
STBA	0x008B,d	Store byte from accumulator to location 008B
ADDA	0x008B,i	Add 008B into accumulator
ADDA	0x008B,d	Add word located at 008B into accumulator
SUBA	0x008B,i	Subtract 008B from accumulator
SUBA	0x008B,d	Subtract word located at 008B from accumulator

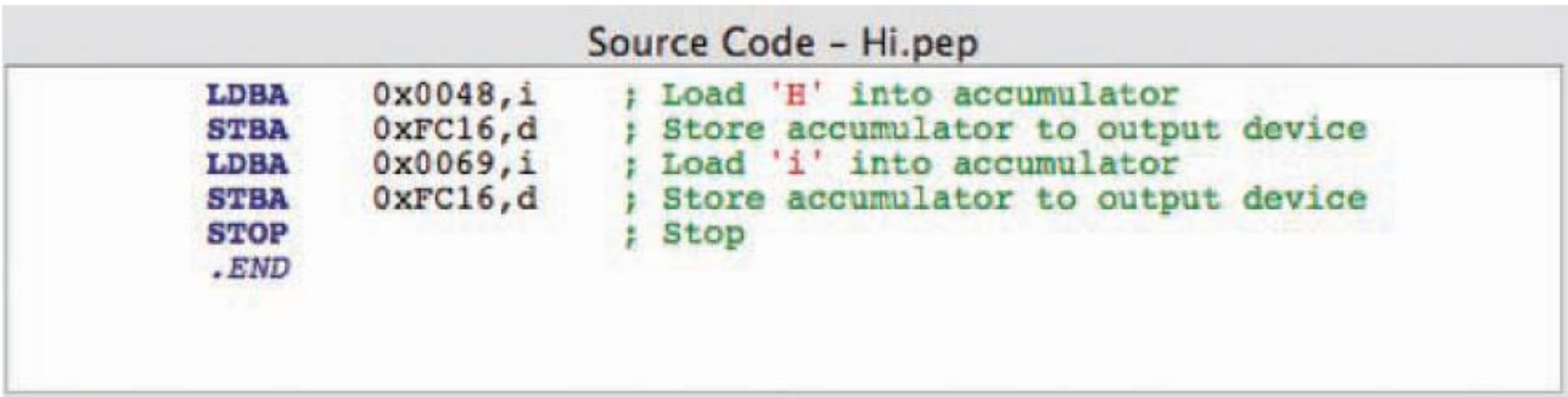
Assembler Directives

Assembler directives Instructions to the translating program

Pseudo-op	Operand	Meaning
`.END`		Signals the end of the assembly-language program
`.ASCII`	"banana\x00"	Represents a string of ASCII characters
`.WORD`	0x008B	Reserves a word in memory and stores a value in it
`.BLOCK`	number of bytes	Reserves a particular number of bytes in memory

Excercise

Assembly program writing “Hi”




```
LDBA    0x0048,i    ; Load 'H' into accumulator
STBA    0xFC16,d    ; Store accumulator to output device
LDBA    0x0069,i    ; Load 'i' into accumulator
STBA    0xFC16,d    ; Store accumulator to output device
STOP                ; Stop
.END
```

Comment is an explanatory text for the human reader

Numeric Data, Branches, and Labels

Mnemonic	Operand, Mode	Meaning
DECI	0x008B,d	Read a decimal number and store it into location 008B
DECO	0x008B,i	Write the decimal number 139 (8B in hex)
DECO	0x008B,d	Write the decimal number stored at location 008B
STRO	0x008B,d	Write the character string stored at location 008B
BR	0x001A	Branch to location 001A
BRLT	0x001A	Branch to location 001A if the accumulator is less than zero
BREQ	0x001A	Branch to location 001A if the accumulator is equal to zero
CPWA	0x008B	Compare the word stored at 008B with the accumulator

A program to read in two numbers and output their sum.

Source Code – AddNums.pep

```
         BR      main           ; Branch around data
sum:     .WORD   0x0000         ; Set up sum and initialize to zero
num1:    .BLOCK  2              ; Set up two byte block for num1
num2:    .BLOCK  2              ; Set up two byte block for num2

main:    LDWA    sum,d          ; Load zero into 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
         STWA    sum,d          ; Store accumulator into sum
         DECO    sum,d          ; Output sum
         STOP                   ; Stop
         .END
```

A Program with Branching

Source Code – AddNums2.pep

```
         BR      main          ; Branch to main program
sum:     .WORD   0x0000        ; Set up sum and initialize to zero
num1:    .BLOCK  2             ; Set up a two byte block for num1
num2:    .BLOCK  2             ; Set up a two byte block for num2
negMsg:  .ASCII  "Error\x00"   ; Error message in case sum is negative

error:   STRO    negMsg,d      ; Print the error message
         BR      finish

main:    LDWA    sum,d         ; Load zero into 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
         BRLT    error         ; Branch to error if A < 0
         STWA    sum,d         ; Store accumulator into sum
         DECO    sum,d         ; Output sum
finish:  STOP                  ; Stop
         .END
```

Loop (reading p.170)

What is the output?
Please explain

Expressing Algorithms

Algorithm: A plan or outline of a solution; a logical sequence of steps that solve a problem

Pseudocode: A language designed to express algorithms

Variables: places in memory where values are stored

Assignment: *Set sum to 0* or *sum <—0*

Input/Output: Read, get, input / Write, display, print

Selection: IF … ELSE

Repetition: WHILE

(See Table 6.1, p.286)

Examples of pseudocode algorithms

```
Set sum to 0
Read num1
Set sum to sum + num1
Read num2
Set sum to sum + num2
Read num3
Set sum to sum + num3
If (sum < 0)
  Write 'E'
ELSE
  Write sum
```

```
Set counter to 0
Set sum to 0
Read limit
While (counter < limit)
    Read num
    Set sum to sum + num
    Set counter to counter + 1
Print sum
```

TABLE 6.1 Pseudocode Statements

Construct	What It Means	Words Used or Example
Variables	Represent named places into which values are stored and from which values are retrieved.	Names that represent the role of a value in a problem are just written in pseudocode
Assignment	Storing a value into a variable.	Set number to 1 number <—1
Input/output	Input: reading in a value, probably from the keyboard. Output: displaying the contents of a variable or a string, probably on the screen.	Read number Get number Write number Display number Write "Have a good day"
Repetition (iteration, looping)	Repeat one or more statements as long as a condition is true.	While (condition) //Execute indented statement(s)

Selection: *if-then*	If a condition is true, execute the indented statements; if a condition is not true, skip the indented statements.	IF (newBase = 10)
		  Write "You are converting"
		  Write "to the same base."
		//Rest of code

Selection: *if-then-else*	If a condition is true, execute the indented statements; if a condition is not true, execute the indented statements below ELSE.	IF (newBase = 10)
		  Write "You are converting"
		  Write "to the same base."
		ELSE
		  Write "This base is not the "
		   Write "same."
		//Rest of code

Quiz

Write algorithm and assembly program to find the smallest value in 03 input values?

1. Describe the algorithm
2. Write the code

High level language - Python

1. Print ‘Hello’
2. Add three numbers which are input from the keyboard
3. Input n numbers from the keyboard, calculate the sum of only positive numbers (using while loop, if condition)

Pseudocode - example

```
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
```

More concrete pseudocode

```
Write "Enter the new base"
Read newBase
Write "Enter the number to be converted"
Read decimalNumber
Set answer to 0
Set quotient to decimal number
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 ", answer
```

Contents

The computer problem-solving process

Analysis and specification phase

Analyze	Understand (define) the problem.
Specification	Specify the problem that the program is to solve.

Algorithm development phase

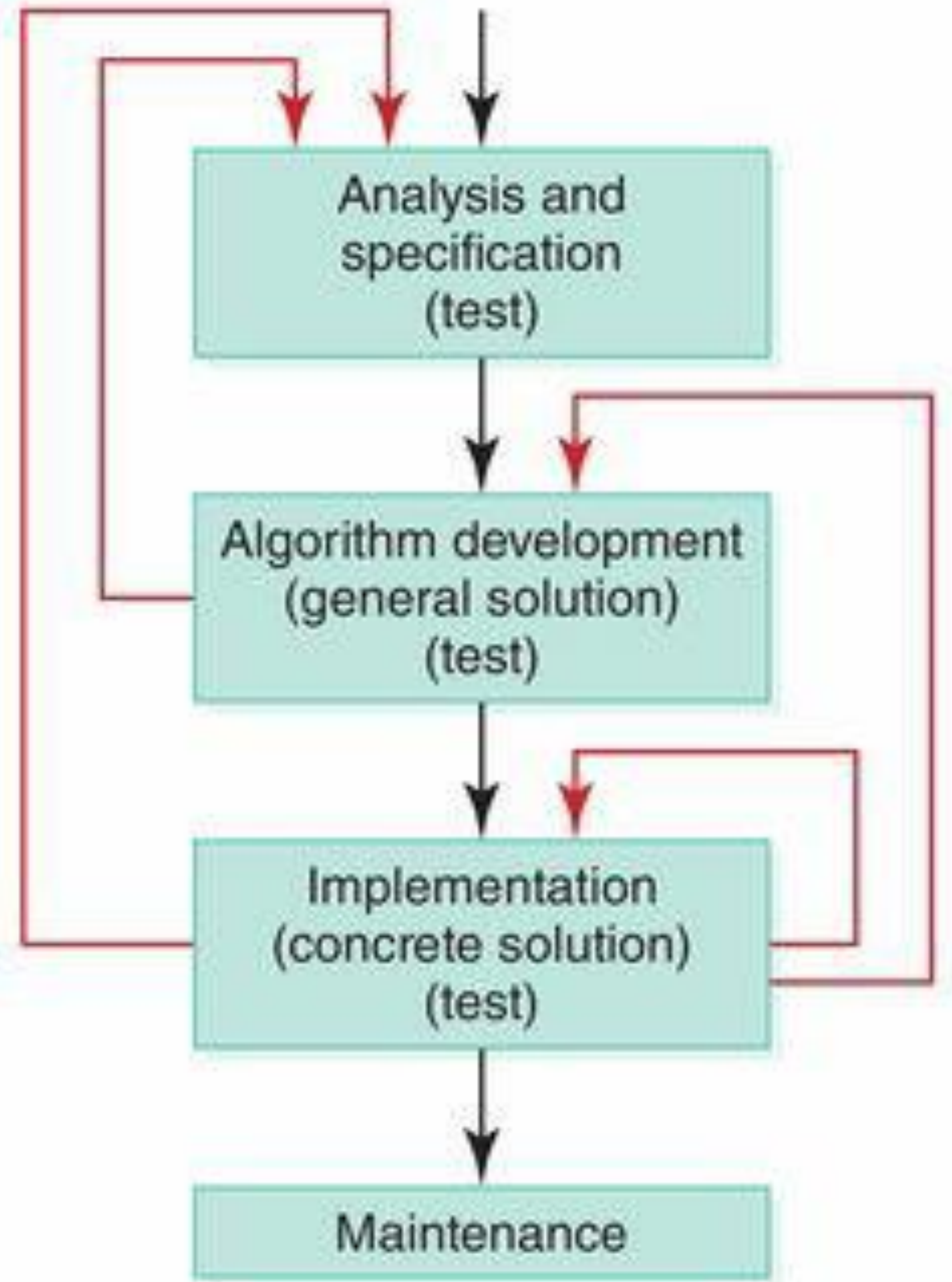
Develop algorithm	Develop a logical sequence of steps to be used to solve the problem.
Test algorithm	Follow the steps as outlined to see if the solution truly solves the problem.

Implementation phase

Code	Translate the algorithm (the general solution) into a programming language.
Test	Have the computer follow the instructions. Check the results and make corrections until the answers are correct.

Maintenance phase

Use	Use the program.
Maintain	Modify the program to meet changing requirements or to correct any errors.


Analysis and specification (test)
Algorithm development (general solution) (test)
Implementation (concrete solution) (test)
Maintenance

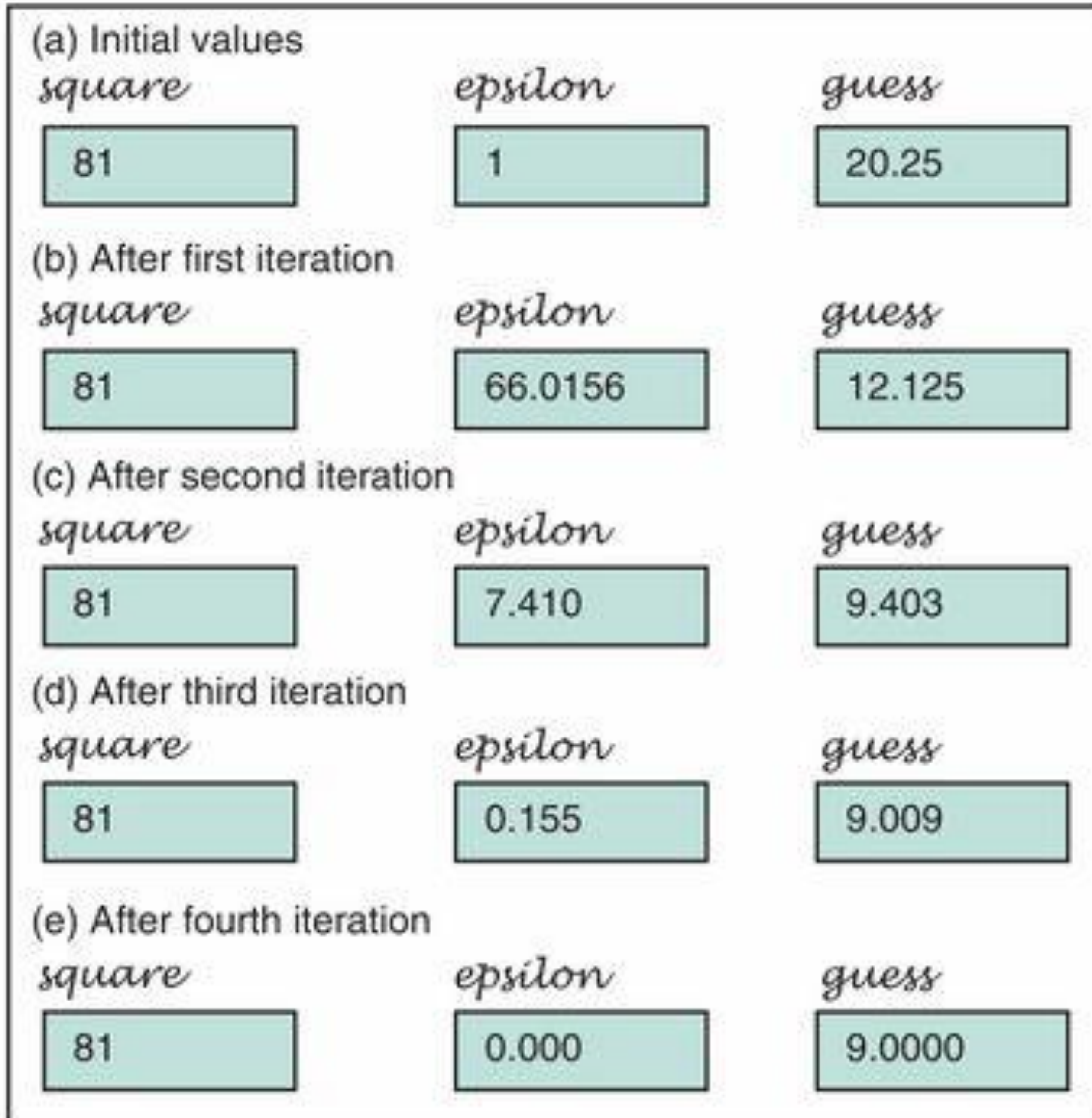
Calculating square root

```
Read in square
Set guess to square/4
Set epsilon to 1
WHILE (epsilon > 0.001)
    Calculate new guess
    Set epsilon to abs(square - guess * guess)
    Write out square and the guess
```

Calculate new guess

```
Set newGuess to (guess + (square/guess)) / 2.0
```

Test algorithm - example

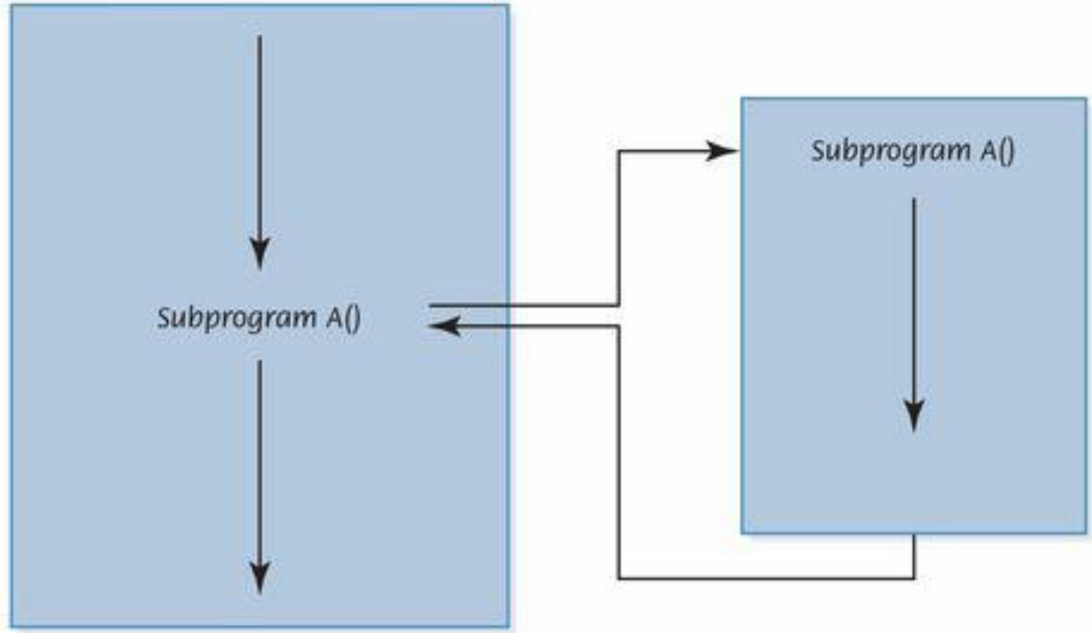


Contents

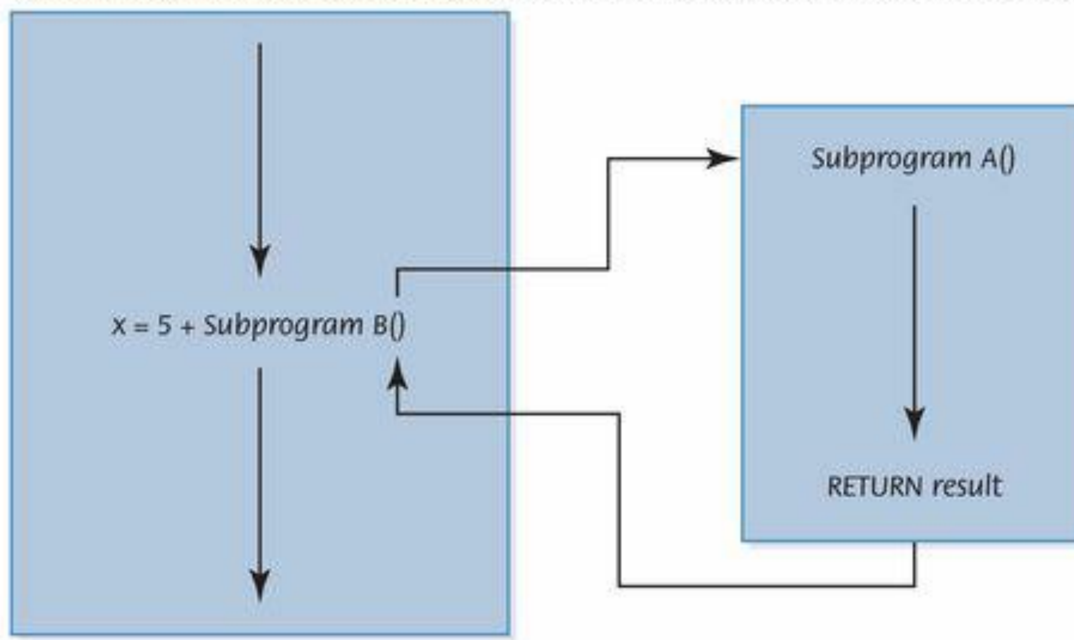
1. Low-Level Programming Languages and Pseudocode
2. Computer problem-solving process
3. **Abstract Data Types and Subprograms**
4. Object-Oriented Design and High-Level Programming Languages

Subprogram

(a) Subprogram A does its task and calling unit continues with next statement



(b) Subprogram B does its task and returns a value that is added to 5 and stored in x



Recursive Algorithms

Recursion The ability of an algorithm to call itself

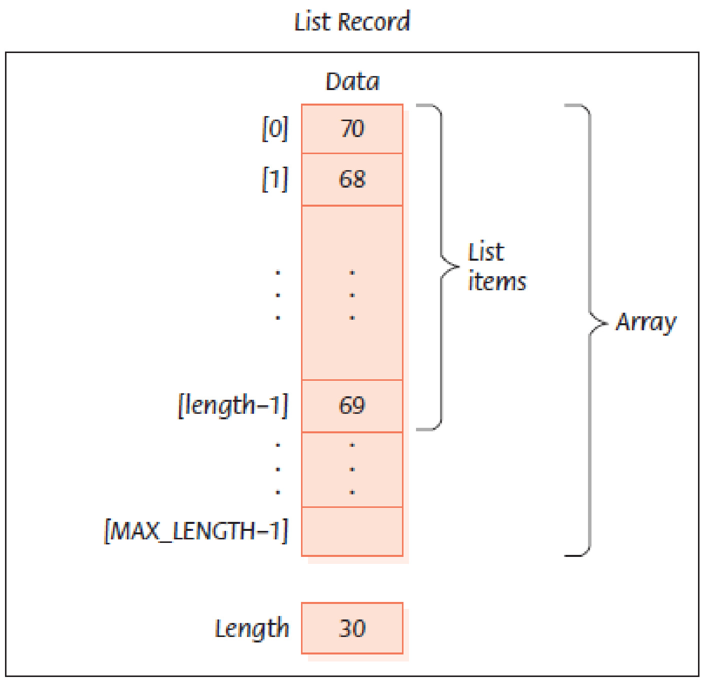
```
Write "Enter N"
Read N
Set result to Factorial(N)
Write result + " is the factorial of " + N
```

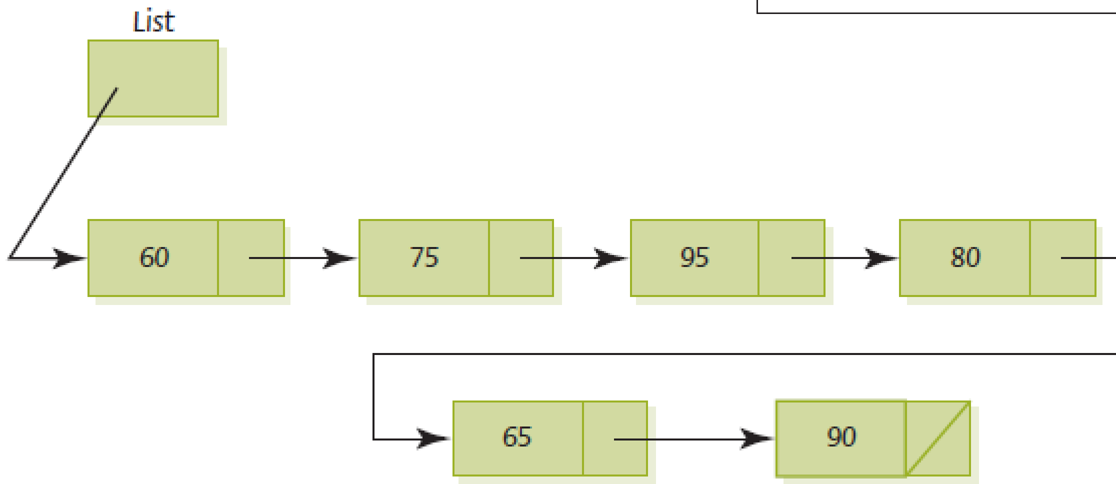
Factorial(N)

```
IF (N equals 0)
    RETURN 1
ELSE
    RETURN N * Factorial(N – 1)
```

Data structures

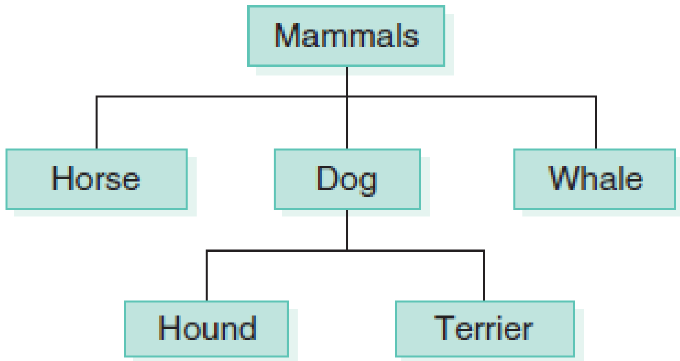
1. Stacks
   - Last in – first out
2. Queues
   - First in – First out
3. Lists
   - Arrays, linked list

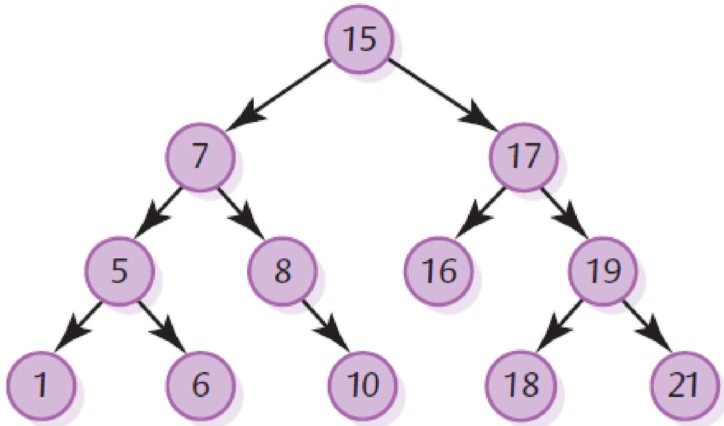




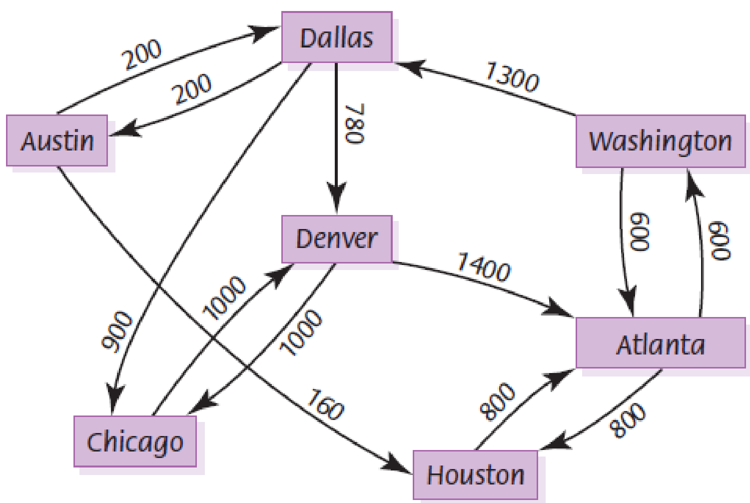
Data structures

4. Trees





5. Graphs



Contents

Object-Oriented Methodology

Top-down design:

- closely mirrors the way humans solve problems
- produces a hierarchy of tasks

Object-oriented design:

- produces a solution to a problem in terms of self-contained entities called ***objects***, which are composed of both *data* and *operations* that manipulate the data.
- focuses on the objects and their interactions within a problem

Objects

Object An entity or thing that is relevant in the context of a problem.

Class A description of a group of objects with similar properties and behaviors

Fields Named items in a class; can be data or subprograms

Method A named algorithm that defines one aspect of the behavior of a class

- Student object: student ID, DOB, address, GPA,…

Translation Process

Compiler A program that translates a high-level language program into machine code

Interpreter A program that inputs a program in a high-level language and directs the computer to perform the actions specified in each statement

Functionality in High-Level Languages

Boolean expression A sequence of identifiers, separated by compatible operators, that evaluates to either true or false

Symbol	Meaning	Example	Evaluation
<	Less than	Number1 < Number2	True if Number1 is less than Number2; false otherwise
<=	Less than or equal	Number1 <= Number2	True if Number1 is less than or equal to Number2; false otherwise
>	Greater than	Number1 > Number2	True if Number1 is greater than Number2; false otherwise
>=	Greater than or equal	Number1 >= Number2	True if Number1 is greater than or equal to Number2; false otherwise
`!=` or `<>` or `/=`	Not equal	Number1 != Number2	True if Number1 is not equal to Number2; false otherwise
`=` or `==`	Equal	Number1 == Number2	True if Number1 is equal to Number2; false otherwise

```
...
WHILE (numberRead < numberOfPairs)
    ...
    IF (number1 < number2)
        Print number1, " ", number2
    ELSE
        Print number2, " ", number1
```

Control Structures

Language	*if* Statement

Python	if temperature > 75:
	    print “No jacket is necessary”
	else:
	    print “A light jacket is appropriate”
	# Idention marks grouping

VB .NET	If (Temperature > 75) Then
	    MsgBox(“No jacket is necessary”)
	Else
	    MsgBox(“A light jacket is appropriate”)
	End If

C++	if (temperature > 75)
	    cout << “No jacket is necessary”;
	else
	    cout << “A light jacket is appropriate”;

Java	if (temperature > 75)
	    System.out.print (“No jacket is necessary”);
	else
	    System.out.print (“A light jacket is appropriate”);

Control Structures

Language	Count-Controlled Loop with a *while* Statement

Python	`count = 0`
	`while count < limit:`
	`    ...`
	`    count = count + 1`
	`# Indention marks loop body`

VB .NET	`Count = 1`
	`While (Count <= Limit)`
	`    ...`
	`    Count = Count + 1`
	`End While`

C++/Java	`count = 1;`
	`while (count <= limit)`
	`{`
	`    ...`
	`    count = count + 1;`
	`}`

Python programming

Chapters 1-3, book How to Think Like a Computer Scientist

Thank you ☺