Low-Level Languages and Pseudocode

CST111—Introduction to Information Technology

Computer Operations

- A computer is a programmable electronic device that stores, retrieves, and processes data.
- Data and the instructions to manipulate the data are logically the same and can be stored in the same place.

Machine Language

- Language made up of binary coded instructions built into the hardware of a particular computer and used directly by the computer.
- 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.

Features in Pep/8

- A register is a small storage area in the ALU for holding special data and intermediate results.
- The Pep/8 registers/status bits to be 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 can store up to 65,636 bytes of storage.

Instruction Format

- Pep/8 instructions are 24-bit words.
- There are two parts to an instruction:
  - The 8-bit instruction specifier—specifies information about the operation to be carried out.
  - The 16-bit operand specifier—holds either the operand itself (data to be processed) or the address where the operand is located (not all instructions use the operand).

Instruction Format

- The instruction specifier part of an instruction has three parts:
  - Operation code—the first four bits which specify which instruction is to be carried out.
  - Register specifier—fifth bit that specifies which register is to be used (only use A (the accumulator) in this chapter).
  - Addressing-mode specifier—the last three bits which says how to interpret the operand.
9 **Instruction Format** *(Page 3)*

- Addressing-mode specifier values:
  - 000—the operand is in the operand specified; known as immediate (i) mode
  - 001—the operand is the memory address where the operand is stored; known as direct (d) mode
  - There are additional addressing modes that are not covered here

12 **Some Pep/8 Instructions** *(Page 1)*

- 0000 Stop execution—the program halts
  - Instruction is 4 bits so it only occupies one byte in which the three right-most bits are ignored
- 1100 Load the operand into the A register—loads a word (two bytes) into the accumulator
  - The addressing-mode specifier determines where the word is located

14 **Some Pep/8 Instructions** *(Page 2)*

- 1110 Store the A register to the operand—stores the contents of the accumulator into the location specified in the operand
  - Again the addressing-mode specifier determines where the word will be stored

16 **Some Pep/8 Instructions** *(Page 3)*

- 0111 Add the operand to the A register—adds the operand to value currently stored in the accumulator
  - The addressing-mode specifier determines where the word is located

18 **Some Pep/8 Instructions** *(Page 4)*

- 1000 Subtract the operand from the A register—subtracts the operand from value currently stored in the accumulator
  - Again addressing-mode specifier determines where the word is located

19 **Some Pep/8 Instructions** *(Page 5)*

- 01001 Character input to the operand—allow program to enter an ASCII character from the input device while the program is running
  - This 5-bit instruction uses the register specifier bit
  - Only direct addressing (001) is allowed so the character is stored in the address shown in the operand

21 **Some Pep/8 Instructions** *(Page 6)*

- 01010 Character output from the operand—sends an ASCII character to the output device while the program is running
  - This 5-bit instruction uses the register specifier bit
  - The addressing-mode specifier determines where the ASCII character is located

24 **Hand Simulation**

- Simulate the program’s execution by *hand-tracing* the steps of the fetch-execute cycle:
  1. Fetch the next instruction from memory address named in the program counter
  2. Decode the instruction and update the program counter
  3. Get data (operand) if necessary
  4. Execute the instruction
Running Pep/8 Machine Language Programs  
- Three different views of the Pep/8 Simulator windows can be selected from the View menu:
  - Code Only—view the Source Code, Object Code, and Assembly Listing windows only
  - Code/CPU—same as Code Only but also view the CPU registers
  - Code/CPU/Memory—same as Code/CPU but also view memory

Running Pep/8 Machine Language Programs  
- To run a machine language program:
  - Enter the machine code in *hexadecimal*, byte by byte with blanks between each, into the Object Code window
  - Load the program into memory (from the Build menu select the Load command)
  - Execute the program (from the Build menu select the Execute command)
  - (The steps can be combined by selecting from the Run Object command from the Build menu)

Running Pep/8 Machine Language Programs  
- To trace through a program step-by-step:
  - From the Build menu select the Start Debugging Object command
  - Click the <Single Step> button in the CPU window

Assembly Language  
- Assembly language—a language that uses mnemonic codes to represent machine language instructions
  - I.e. ADDA in Pep/8 assembly language is the equivalent of the machine language instruction 0111 (Add the operand to the A register)
  - Assembler—a program that reads each of the instructions in mnemonic form and translates it into the machine language equivalent

Some Pep/8 Assembly Language Instructions  
- STOP—stop execution
  - LDA—load operand into register A (accumulator)
  - STA—store contents of register A (accumulator) to the operand
  - ADDA—add the operand to register A (accumulator)
  - SUBA—subtract the operand from register A (accumulator)
  - BR—branch to the memory location specified in the operand specifier

Some Pep/8 Assembly Language Instructions  
- CHARI—read an ASCII character from input device and store it into memory address specified in the operand
- CHARO—write the ASCII character stored in the operand to the output device
- DECI—read decimal number from input device and store it into memory address specified in the operand
- DECO—write decimal number stored in the operand to the output device

Assembler Directives  
- Additionally Pep/8 assembly language includes special instructions that can be sent directly to the assembler:
  - .ASCII—represents a string of ASCII bytes
A computer is a programmable electronic device that stores, retrieves, and processes data. The instruction set of a computer determines where the word is located.

Three different views of the Pep/8 Simulator windows can be selected from the View menu: Pep/8 Simulator, Pep/8 Simulator with register display, and Pep/8 Simulator with the execute screen. The Pep/8 Simulator is a way of expressing algorithms using a mixture of pseudocode, a language instruction, a programming language, and a machine designed to contain the important features of the target computer.

There are no grammar rules in assembly language. If the program is assembled successfully, the assembler converts the symbolic representation of the program to a machine code representation that the computer can execute.

The Pep/8 registers/status bits to be covered:

- CP: Carry Parity
- C: Carry flag
- Z: Zero flag
- N: Negative flag
- V: Overflow flag
- I: Interrupt flag
- D: Debug flag
- X: Debug Extension flag
- T: Time flag
- B: Break flag
- $: Breakflag
- S: Stack flag
- F: Carry flag
- G: Overflow flag
- H: Int Overflow flag
- I: Int Overflow flag
- J: Int Overflow flag
- K: Int Overflow flag
- L: Int Overflow flag
- M: Int Overflow flag
- N: Int Overflow flag
- O: Int Overflow flag
- P: Int Overflow flag
- Q: Int Overflow flag
- R: Int Overflow flag
- S: Int Overflow flag
- T: Int Overflow flag
- U: Int Overflow flag
- V: Int Overflow flag
- W: Int Overflow flag
- X: Int Overflow flag
- Y: Int Overflow flag
- Z: Int Overflow flag

To trace through a program step-by-step, use the <Single Step> button in the CPU window. The program can be executed either in the default mode (the program is executed line by line) or in the Code Only mode (the program is executed line by line without displaying the contents of the CPU registers).

Some Pep/8 Assembly Language Instructions:
- LDA: Load the operand into the A register
- STA: Store A register to the operand
- ADDA: Add the operand to register A (accumulator)
- DECI: Subtract the operand from the accumulator and store the result in the accumulator
- DECO: Subtract the operand from the accumulator and store the result in the accumulator

There are two common methodologies used to develop computer solutions to a problem: Pseudocode and Assembly Language. Pseudocode is a high-level language statement that describes the algorithm, while Assembly Language is a low-level language that directly specifies the computer instructions.

Example:

```plaintext
1100 Load the operand into the A register
0x006C,i ; Output an 'l'
0x0048,i ; Output an 'H'
01010 Character output from the operand
STOP
.END
```

Running Pep/8 Assembly Language Programs

- To run an assembly language program:
  - Enter the mnemonic code into the Source Code window
  - Assemble (translate into machine language) the program (from the Build menu select the Assemble command)
  - Load the program into memory (from the Build menu select the Load command)
  - Execute the program (from the Build menu select the Execute command)
  - The steps can be combined by selecting from the Run Source command from the Build menu)

- To trace through a program step-by-step:
  - From the Build menu select the Start Debugging Source command
  - Click the <Single Step> button in the CPU window

A New Program

- Problem—read and sum three values and print sum

Example:

```plaintext
num1 = scan.nextDouble();
num2 = scan.nextDouble();
num3 = scan.nextDouble();
sum = num1 + num2 + num3;
System.out.println(sum);
```

A New Program

- Problem—read and sum three values and print sum

Solution:

- Load zero into accumulator
- Read and store num1 (a memory location)
- Add num1 to accumulator
- Read and store num2 (a memory location)
- Add num2 to accumulator
- Read and store num3 (a memory location)
- Add num3 to accumulator

A New Program

- Problem—read and sum three values and print sum

Solution:

- Load zero into accumulator
- Read and store num1 (a memory location)
- Add num1 to accumulator
- Read and store num2 (a memory location)
- Add num2 to accumulator
- Read and store num3 (a memory location)
- Add num3 to accumulator
• Problem—read and sum three values and print sum
• Solution (con.):
  ● Store accumulator into sum (a memory location)
  ● Output sum
  ● Stop the processing
  ● End of program

48 Pep/8 Simulator—Sum.pep

```assembly
BR main
sum: .WORD 0x0000
num1: .BLOCK 2
num2: .BLOCK 2
num3: .BLOCK 2
main: LDA sum,d ; 'd' = direct
       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
```

49 Pseudocode

• Pseudocode—a mixture of English and formatting to make the steps in an algorithm explicit
• For example: an algorithm to convert base-10 numbers 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

51 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 lenter in slow steam
Turn off blender

52 Developing an Algorithm
- There are two common 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
- A way to express algorithms is called pseudocode

53 Pseudocode
- Pseudocode is a way of expressing algorithms using 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

56 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

57 Pseudocode Functionality (Page 1)
- Variables—the names of places to store values
  - quotient, decimalNumber, newBase
- Assignment—storing value of an expression into a variable
  - Set quotient to 64
  - quotient ← 64
  - quotient ← 6 * 10 + 4

58 Pseudocode Functionality (Page 2)
- Output—printing a value on an output device
  - Write, Print
- Input—getting values from the outside world and storing them into variables
  - Get, Read

59 Pseudocode Functionality (Page 3)
- 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

60 Pseudocode Functionality  (Page 4)
- 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."

61 Pseudocode Functionality  (Page 5)
- 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"

62 Pseudocode Example  (Page 1)
- 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
- But ...

63 Pseudocode Example  (Page 2)
- 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

64 Pseudocode Example  (Page 3)
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

65 Translating Pseudocode
   • To What? ...
     ● Assembly language
       • Very detailed and time consuming
     ● High-level language
       • Easy as you will see in Chapter 9

66 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
     value

67 Testing
   • Test plan implementation—using the test cases outlined in the test plan to verify that
     the program outputs the predicted result