A *data definition statement* sets aside storage in memory for a variable, and may optionally assign it a name. If multiple initializers are used in the same data definition, its label refers only to the offset of the first initializer. To create a string data definition, enclose a sequence of characters in quotes. The DUP operator generates a repeated storage allocation, using a constant expression as a counter. The current location counter operator ($) is used in address-calculation expressions.

x86 processors store and retrieve data from memory using *little-endian* order: The least significant byte of a variable is stored at its starting (lowest) address value.

A *symbolic constant* (or symbol definition) associates an identifier with an integer or text expression. Three directives create symbolic constants:

- The equal-sign directive (=) associates a symbol name with a constant integer expression.
- The EQU and TEXTEQU directives associate a symbolic name with a constant integer expression or some arbitrary text.

3.8 Key Terms

3.8.1 Terms

assembler
big endian
binary coded decimal (BCD)
calling convention
character literal
code label
code segment
compiler
constant integer expression
data definition statement
data label
data segment
decimal real
directive
encoded real
executable file
floating-point literal
identifier
initializer
instruction
instruction mnemonic
integer constant
integer literal
intrinsic data type
label
linker
link library
listing file
little-endian order
macro
memory model
memory operand
object file
operand
operator precedence
packed binary coded decimal
process return code
program entry point
real number literal
reserved word
source file
stack segment
string literal
symbolic constant
system function

3.8.2 Instructions, Operators, and Directives

+	(add, unary plus)
=	(assign, compare for equality)
/	(divide)
*	(multiply)
( )	(parentheses)
–	(subtract, unary minus)
ADD	
BYTE	
CALL	
.CODE	
COMMENT	
.DATA	
DWORD	
END	
ENDP	
DUP	
EQU	
MOD	
MOV	
NOP	
PROC	
SBYTE	
SDWORD	
.STACK	
TEXTEQU	

3.9 Review Questions and Exercises

3.9.1 Short Answer

1. Provide examples of three different instruction mnemonics.
2. What is a calling convention, and how is it used in assembly language declarations?
3. How do you reserve space for the stack in a program?
4. Explain why the term *assembler language* is not quite correct.
5. Explain the difference between big endian and little endian. Also, look up the origins of this term on the Web.
6. Why might you use a symbolic constant rather than an integer literal in your code?
7. How is a source file different from a listing file?
8. How are data labels and code labels different?
9. *(True/False):* An identifier cannot begin with a numeric digit.
10. *(True/False):* A hexadecimal literal may be written as 0x3A.
11. *(True/False):* Assembly language directives execute at runtime.
12. *(True/False):* Assembly language directives can be written in any combination of uppercase and lowercase letters.
13. Name the four basic parts of an assembly language instruction.
14. *(True/False):* MOV is an example of an instruction mnemonic.
15. *(True/False):* A code label is followed by a colon (:), but a data label does not end with a colon.
16. Show an example of a block comment.
17. Why is it not a good idea to use numeric addresses when writing instructions that access variables?

18. What type of argument must be passed to the ExitProcess procedure?
19. Which directive ends a procedure?
20. In 32-bit mode, what is the purpose of the identifier in the END directive?
21. What is the purpose of the PROTO directive?
22. *(True/False):* An Object file is produced by the Linker.
23. *(True/False):* A Listing file is produced by the Assembler.
24. *(True/False):* A link library is added to a program just before producing an Executable file.
25. Which data directive creates a 32-bit signed integer variable?
26. Which data directive creates a 16-bit signed integer variable?
27. Which data directive creates a 64-bit unsigned integer variable?
28. Which data directive creates an 8-bit signed integer variable?
29. Which data directive creates a 10-byte packed BCD variable?

3.9.2 Algorithm Workbench

1. Define four symbolic constants that represent integer 25 in decimal, binary, octal, and hexadecimal formats.
2. Find out, by trial and error, if a program can have multiple code and data segments.
3. Create a data definition for a doubleword that stored it in memory in big endian format.
4. Find out if you can declare a variable of type DWORD and assign it a negative value. What does this tell you about the assembler's type checking?
5. Write a program that contains two instructions: (1) add the number 5 to the EAX register, and (2) add 5 to the EDX register. Generate a listing file and examine the machine code generated by the assembler. What differences, if any, did you find between the two instructions?
6. Given the number 456789ABh, list out its byte values in little-endian order.
7. Declare an array of 120 uninitialized unsigned doubleword values.
8. Declare an array of byte and initialize it to the first 5 letters of the alphabet.
9. Declare a 32-bit signed integer variable and initialize it with the smallest possible negative decimal value. (Hint: Refer to integer ranges in Chapter 1.)
10. Declare an unsigned 16-bit integer variable named **wArray** that uses three initializers.
11. Declare a string variable containing the name of your favorite color. Initialize it as a nullterminated string.
12. Declare an uninitialized array of 50 signed doublewords named **dArray**.
13. Declare a string variable containing the word "TEST" repeated 500 times.
14. Declare an array of 20 unsigned bytes named **bArray** and initialize all elements to zero.

15. Show the order of individual bytes in memory (lowest to highest) for the following doubleword variable:

```
val1 DWORD 87654321h
```

3.10 Programming Exercises

★ **1. Integer Expression Calculation**
Using the *AddTwo* program from Section 3.2 as a reference, write a program that calculates the following expression, using registers: A = (A + B) – (C + D). Assign integer values to the EAX, EBX, ECX, and EDX registers.

★ **2. Symbolic Integer Constants**
Write a program that defines symbolic constants for all seven days of the week. Create an array variable that uses the symbols as initializers.

★★ **3. Data Definitions**
Write a program that contains a definition of each data type listed in Table 3-2 in Section 3.4. Initialize each variable to a value that is consistent with its data type.

★ **4. Symbolic Text Constants**
Write a program that defines symbolic names for several string literals (characters between quotes). Use each symbolic name in a variable definition.

★★★★ **5. Listing File for AddTwoSum**
Generate a listing file for the AddTwoSum program and write a description of the machine code bytes generated for each instruction. You might have to guess at some of the meanings of the byte values.

★★★ **6. AddVariables Program**
Modify the AddVariables program so it uses 64-bit variables. Describe the syntax errors generated by the assembler and what steps you took to resolve the errors.