Chapter Five

Questions, Projects, and Lab Exercises

5.1 Questions

1) List the legal forms of a *boolean* expression in an HLA IF statement.
2) What data type do you use to declare a
   a) 32-bit signed integer?
   b) 16-bit signed integer?
   c) 8-bit signed integer?
3) List all of the 80x86:
   a) 8-bit general purpose registers.
   b) 16-bit general purpose registers.
   c) 32-bit general purpose registers.
4) Which registers overlap with
   a) ax?
   b) bx?
   c) cx?
   d) dx?
   e) si?
   f) di?
   g) bp?
   h) sp?
5) In what register does the condition codes appear?
6) What is the generic syntax of the HLA MOV instruction?
7) What are the legal operand formats for the MOV instruction?
8) What do the following symbols denote in an HLA boolean expression?
   a) @c
   b) @nc
   c) @z
   d) @nz
   e) @o
   f) @no
   g) @s
   h) @ns
9) Collectively, what do we call the carry, overflow, zero, and sign flags?
10) What high level language control structures does HLA provide?

11) What does the *nl* symbol represent?

12) What routine would you call, that doesn't require any parameters, to print a new line on the screen?

13) If you wanted to print a nicely-formatted column of 32-bit integer values, what standard library routines could you call to achieve this?

14) The stdin.getc() routine does not allow a parameter. Where does it return the character it reads from the user?

15) When reading an integer value from the user via the stdin.geti*X* routines, the program will stop with an exception if the user enters a value that is out of range or enters a value that contains illegal characters. How can you trap this error

16) What is the difference between the stdin.ReadLn() and stdin.FlushInput() procedures?

17) Convert the following decimal values to binary:

a) 128	b) 4096	c) 256	d) 65536	e) 254
f) 9	g) 1024	h) 15	i) 344	j) 998
k) 255	l) 512	m) 1023	n) 2048	o) 4095
p) 8192	q) 16,384	r) 32,768	s) 6,334	t) 12,334
u) 23,465	v) 5,643	w) 464	x) 67	y) 888

18) Convert the following binary values to decimal:

a) 1001 1001	b) 1001 1101	c) 1100 0011	d) 0000 1001	e)1111 1111
f) 0000 1111	g) 0111 1111	h) 1010 0101	i) 0100 0101	j) 0101 1010
k) 1111 0000	l) 1011 1101	m) 1100 0010	n) 0111 1110	o) 1110 1111
p) 0001 1000	q) 1001 111 1	r) 0100 0010	s) 1101 1100	t) 1111 0001
u) 0110 1001	v) 0101 1011	w) 1011 1001	x) 1110 0110	y) 1001 0111

19) Convert the binary values in problem 2 to hexadecimal.

20) Convert the following hexadecimal values to binary:

a) 0ABCD	b) 1024	c) 0DEAD	d) 0ADD	e) 0BEEF
f) 8	g) 05AAF	h) 0FFFF	i) 0ACDB	j) 0CDBA
k) 0FEBA	l) 35	m) 0BA	n) 0ABA	o) 0BAD
p) 0DAB	q) 4321	r) 334	s) 45	t) 0E65
u) 0BEAD	v) 0ABE	w) 0DEAF	x) 0DAD	y) 9876

Perform the following hex computations (leave the result in hex):

21) 1234 +9876

22) 0FFF - 0F34

23) 100 - 1

24) 0FFE - 1

25) What is the importance of a nibble?

26) How many hexadecimal digits in:

a) a byte b) a word c) a double word

27) How many bits in a:

a) nibble b) byte c) word d) double word

28) Which bit (number) is the H.O. bit in a:

a) nibble b) byte c) word d) double word

29) What character do we use as a suffix for hexadecimal numbers? Binary numbers? Decimal numbers?

30) Assuming a 16-bit two's complement format, determine which of the values in question 4 are positive and which are negative.

31) Sign extend all of the values in question two to sixteen bits. Provide your answer in hex.

32) Perform the bitwise AND operation on the following pairs of hexadecimal values. Present your answer in hex. (Hint: convert hex values to binary, do the operation, then convert back to hex).

a) 0FF00, 0FF0b) 0F00F, 1234c) 4321, 1234 d) 2341, 3241 e) 0FFFF, 0EDCB

f) 1111, 5789g) 0FABA, 4322h) 5523, 0F572i) 2355, 7466 j) 4765, 6543

k) 0ABCD, 0EFDCl) 0DDDD, 1234m) 0CCCC, 0ABCDn) 0BBBB, 1234o) 0AAAA, 1234

p) 0EEEE, 1248q) 8888, 1248r) 8086, 124F s) 8086, 0CFA7 t) 8765, 3456

u) 7089, 0FEDCv) 2435, 0BCDEw) 6355, 0EFDCx) 0CBA, 6884y) 0AC7, 365

33) Perform the logical OR operation on the above pairs of numbers.

34) Perform the logical XOR operation on the above pairs of numbers.

35) Perform the logical NOT operation on all the values in question four. Assume all values are 16 bits.

36) Perform the two's complement operation on all the values in question four. Assume 16 bit values.

37) Sign extend the following hexadecimal values from eight to sixteen bits. Present your answer in hex.

a) FF	b) 82	c) 12	d) 56	e) 98
f) BF	g) 0F	h) 78	i) 7F	j) F7
k) 0E	l) AE	m) 45	n) 93	o) C0
p) 8F	q) DA	r) 1D	s) 0D	t) DE
u) 54	v) 45	w) F0	x) AD	y) DD

38) Sign contract the following values from sixteen bits to eight bits. If you cannot perform the operation, explain why.

a) FF00	b) FF12	c) FFF0	d) 12	e) 80
f) FFFF	g) FF88	h) FF7F	i) 7F	j) 2
k) 8080	l) 80FF	m) FF80	n) FF	o) 8
p) F	q) 1	r) 834	s) 34	t) 23
u) 67	v) 89	w) 98	x) FF98	y) F98

39) Sign extend the 16-bit values in question 22 to 32 bits.

40) Assuming the values in question 22 are 16-bit values, perform the left shift operation on them.

41) Assuming the values in question 22 are 16-bit values, perform the logical right shift operation on them.

42) Assuming the values in question 22 are 16-bit values, perform the arithmetic right shift operation on them.

43) Assuming the values in question 22 are 16-bit values, perform the rotate left operation on them.

44) Assuming the values in question 22 are 16-bit values, perform the rotate right operation on them.

45) Convert the following dates to the short packed format described in this chapter (see "Bit Fields and Packed Data" on page 81). Present your values as a 16-bit hex number.

a) 1/1/92b) 2/4/56 c) 6/19/60 d) 6/16/86 e) 1/1/99

46) Convert the above dates to the long packed data format described in this chapter.

47) Describe how to use the shift and logical operations to *extract* the day field from the packed date record in question 29. That is, wind up with a 16-bit integer value in the range 0..31.

48) Assume you've loaded a long packed date (See "Bit Fields and Packed Data" on page 81.) into the EAX register. Explain how you can easily access the day and month fields directly, without any shifting or rotating of the EAX register.

49) Suppose you have a value in the range 0..9. Explain how you could convert it to an ASCII character using the basic logical operations.

50) The following C++ function locates the first set bit in the *BitMap* parameter starting at bit position *start* and working up to the H.O. bit . If no such bit exists, it returns -1. Explain, in detail, how this function works.

```
int FindFirstSet(unsigned BitMap, unsigned start)
{
        unsigned Mask = (1 << start);

        while (Mask)
        {
            if (BitMap & Mask) return start;
            ++start;
            Mask <<= 1;
        }
        return -1;
}
```

51) The C++ programming language does not specify how many bits there are in an unsigned integer. Explain why the code above will work regardless of the number of bits in an unsigned integer.

52) The following C++ function is the complement to the function in the questions above. It locates the first zero bit in the *BitMap* parameter. Explain, in detail, how it accomplishes this.

```
int FindFirstClr(unsigned BitMap, unsigned start)
{
        return FindFirstSet(~BitMap, start);
}
```

53) The following two functions set or clear (respectively) a particular bit and return the new result. Explain, in detail, how these functions operate.

```
unsigned SetBit(unsigned BitMap, unsigned position)
{
        return BitMap | (1 << position);
}

unsigned ClrBit(unsigned BitMap, unsigned position)
{
        return BitMap & ~(1 << position);
}
```

54) In code appearing in the questions above, explain what happens if the start and position parameters contain a value greater than or equal to the number of bits in an unsigned integer.

55) Provide an example of HLA variable declarations for the following data types:

a) Eight-bit byte

b) 16-bit word

c) 32-bit dword

d) Boolean

e) 32-bit floating point

f) 64-bit floating point

g) 80-bit floating point

h) Character

56) The long packed date format offers two advantages over the short date format. What are these advantages?

57) Convert the following real values to 32-bit single precision floating point format. Provide your answers in hexadecimal, explain your answers.

a) 1.0	b) 2.0	c) 1.5	d) 10.0
e) 0.5	f) 0.25	g) 0.1	h) -1.0
i) 1.75	j) 128	k) 1e-2	l) 1.024e+3

58) Which of the values in question 41 do not have exact representations?

59) Show how to declare a character variable that is initialized with the character "*".

60) Show how to declare a character variable that is initialized with the control-A character (See "The ASCII Character Set" on page 104 for the ASCII code for control-A).

61) How many characters are present in the standard ASCII character set?

62) What is the basic structure of an HLA program?

63) Which HLA looping control structure(s) test(s) for loop termination at the beginning of the loop?

64) Which HLA looping control structure(s) test(s) for loop termination at the end of the loop?

65) Which HLA looping construct lets you create an infinite loop?

66) What set of flags are known as the "condition codes?"

67) What HLA statement would you use to trap exceptions?

68) Explain how the IN operator works in a boolean expression.

69) What is the *stdio.bs* constant?

70) How do you redirect the standard output of your programs so that the data is written to a text file?

5.2 Programming Projects for Chapter Two

1) Write a program to produce an "addition table." This table should input two small *int8* values from the user. It should verify that the input is correct (i.e., handle the *ex.ConversionError* and *ex.ValueOutOfRange* exceptions) and is positive. The second value must be greater than the first input value. The program will display a row of values between the lower and upper input values. It will also print a column of values between the two values specified. Finally, it will print a matrix of sums. The following is the expected output for the user inputs 15 & 18

```
add     15  16  17  18
15      30  31  32  33
16      31  32  33  34
17      32  33  34  35
18      33  34  35  36
```

2) Modify program (1), above, to draw lines between the columns and rows. Use the hyphen ('-'), vertical bar ('|'), and plus sign ('+') characters to draw the lines. E.g.,

```
add  | 15 | 16 | 17 | 18 |
-----+----+----+----+----+
15   | 30 | 31 | 32 | 33 |
-----+----+----+----+----+
16   | 31 | 32 | 33 | 34 |
-----+----+----+----+----+
17   | 32 | 33 | 34 | 35 |
-----+----+----+----+----+
18   | 33 | 34 | 35 | 36 |
-----+----+----+----+----+
```

For extra credit, use the line drawing character graphics symbols listed in Appendix B to draw the lines. Note: to print a character constant as an ASCII code, use "#nnn" where "nnn" represents the ASCII code of the character you wish to print. For example, "stdout.put(#179);" prints the line drawing vertical bar character. (This option is available only on machines that support the IBM-PC extended character set.)

3) Write a program that generates a "Powers of Four" table. Note that you can create the powers of four by loading a register with one and then successively add that register to itself twice for each power of two.

4) Write a program that reads a list of positive numbers from a user until that user enters a negative or zero value. Display the sum of those positive integers.

5) Write a program that computes (n)(n+1)/2. It should read the value "n" from the user. Hint: you can compute this formula by adding up all the numbers between one and n.

5.3 Programming Projects for Chapter Three

Write each of the following programs in HLA. Be sure to fully comment your source code. See Appendix C for style guidelines and rules when writing HLA programs (and follow these rules to the letter!). Include sample output and a short descriptive write up with your program submission(s).

1) Write a program that reads a line of characters from the user and displays that entire line after converting any upper case characters to lower case. All non-alphabetic and existing lower case characters should pass through unchanged; you should convert all upper case characters to lower case before printing them.

2) Write a program that reads a line of characters from the user and displays that entire line after swapping upper case characters with lower case; that is, convert the incoming lower case characters to upper case and convert the incoming upper case characters to lower case. All non-alphabetic characters should pass through unchanged.

3) Write a program that reads three values from the user: a month, a day, and a year. Pack the date into the long date format appearing in this chapter and display the result in hexadecimal. If the date is between 2000 and 2099, also pack the date into the short packed date format and display that 16-bit value in hexadecimal form. If the date is not in the range 2000..2099, then display a short message suggesting that the translation is not possible.

4) Write a date validation program that reads a month, day, and year from the user, verifies that the date is correct (ignore leap years for the time being), and then packs the date into the long date format appearing in this chapter.

5) Write a "CntBits" program that counts the number of one bits in a 16-bit integer value input from the user. *Do not use any built-in functions in HLA's library to count these bits for you.* Use the shift or rotate instructions to extract each bit in the value.

6) Write a "TestBit" program. This program requires two integer inputs. The first value is a 32-bit integer to test; the second value is an unsigned integer in the range 0..31 describing which bit to test. The program should display true if the corresponding bit (in the test value) contains a one, the program should display false if that bit position contains a zero. The program should always display false if the second value holds a value outside the range 0..31.

7) Write a program that reads an eight-bit signed integer and a 32-bit signed integer from the user that computes and displays the sum and difference of these two numbers.

8) Write a program that reads an eight-bit unsigned integer and a 16-bit unsigned integer from the user that computes and displays the sum and the absolute value of the difference of these two numbers.

9) Write a program that reads a 32-bit unsigned integer from the user and displays this value in binary. Use the SHL instruction to perform the integer to binary conversion.

10) Write a program that uses *stdin.getc* to read a sequence of binary digits from the user (that is, a sequence of '1' and '0' characters). Convert this string to an integer using the AND, SHL, and OR instructions. Display the integer result in hexadecimal and decimal. You may assume the user will not enter more than 32 digits.

11) Using the LAFH instruction, write a program that will display the current values of the carry, sign, and zero flags as boolean values. Read two integer values from the user, add them together, and them immediately capture the flags' values using the LAHF instruction and display the result of these three flags as boolean values. Hint: use the SHL or SHR instructions to extract the specific flag bits.

5.4 Programming Projects for Chapter Four

1) Write an HLA program that reads a single precision floating point number from the user and prints the internal representation of that value using hexadecimal notation.

2) Write a program that reads a single precision floating point value from the user, takes the absolute value of that number, and then displays the result. Hint: this program does not use any arithmetic instructions or comparisons. Take a look at the binary representation for floating point numbers in order to solve this problem.

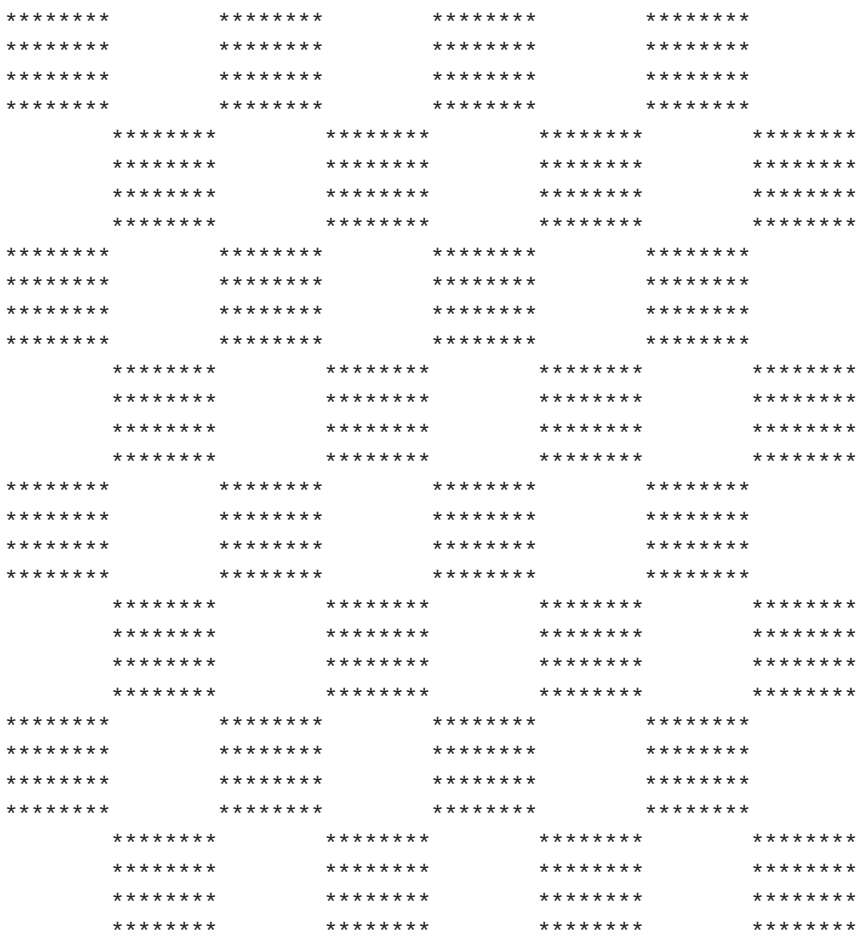
3) Write a program that generates an ASCII character set chart using the following output format:

```
  | 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
--+-------------------------------------------------
20|     !  "  #  ...
30|  0  1  2  3  ...
40|  @  A  B  C  ...
50|  P  Q  R  S  ...
60|  `  a  b  c  ...
70|  p  q  r  s  ...
```

Note that the columns in the table represent the L.O. four bits of the ASCII code, the rows in the table represent the H.O. four bits of the ASCII code. Note: for extra consideration, use the line-drawing graphic char-

acters (see Appendix B) to draw the lines in the table (this option is available only on systems that support the IBM-PC extended chracter set).

4) Using only five FOR loops, four calls to *stdout.putcSize*, and two calls to *stdout.newln*, write a program that draws a checkerboard pattern. Your checkerboard should look like the following:



5.5 Laboratory Exercises for Chapter Two

Before you can write, compile, and run a single HLA program, you will need to install the HLA language system on your computer. If you are using HLA at school or some other institution, the system administrator has probably set up HLA for you. If you are working at home or on some computer on which HLA is not installed, you will need to obtain the HLA distribution package and set it up on your computer. The first section of this set of laboratory exercises deals with obtaining and installing HLA.

Once HLA is installed, the next step is to take stock of what is present in the HLA system. The second part of this laboratory deals with the files that are present in the HLA distribution.

Finally, and probably most important, this set of laboratory exercises discusses how to write, compile, and run some simple HLA programs.

5.5.1 A Short Note on Laboratory Exercises and Lab Reports

Whenever you work on laboratory exercises in this textbook you should always prepare a lab report associated with each exercise. Your instructor may have some specific guidelines concerning the content of

the lab report (if your instructor requires that you submit the report). Be sure to check with your instructor concerning the lab report requirements.

At a bare minimum, a lab report should contain the following:

- A title page with the lab title (chapter #), your name and other identification, the current date, and the due date. If you have a course-related computer account, you should also include your login name.
- If you modify or create a program in a lab exercise, the source code for that program should appear in the laboratory report (do not simply reprint source code appearing in this text in order to pad your lab report).
- Output from all programs should also appear in the lab report.
- For each exercise, you should provide a write-up describing the purpose of the exercise, what you learned from the exercise, and any comments about improvements or other work you've done with the exercise.
- If you make any mistakes that require correction, you should include the source code for the incorrect program with your lab report. Hand write on the listing where the error occurs and describe (in handwriting, on the listing) what you did to correct the problem. **Note**: no one is perfect. If you turn in a lab report that has no listings with errors in it, this is a clear indication that you didn't bother to perform this part of the exercise.
- Appropriate diagrams.

The lab report should be prepared with a word processing program. Hand-written reports are unacceptable (although hand-drawn diagrams are acceptable if a suitable drawing package isn't available). The report should be proofread and of finished quality before submission. Only the listings with errors (and hand written annotations) should be in a less than finished quality. See the "HLA Programming Style Guidelines" appendix for important information concerning programming style. Adhere to these guidelines in the HLA programs you submit.

5.5.2 Compiling Your First Program

Once HLA is operational, you can compile and run actual working programs. The HLA distribution contains lots of example HLA programs, including the HLA programs appearing in this text. Since these examples are already written, tested, and ready to compile and run, it makes sense to work with one of these example files when compiling your first program.

A good first program is the "Hello World" program appearing earlier in this volume (repeated below):

```
program helloWorld;
#include( "stdlib.hhf" );

begin helloWorld;

    stdout.put( "Hello, World of Assembly Language", nl );

end helloWorld;
```

The source code for this program appears in the "HelloWorld.hla" file in the "aoa\volume1\ch01" directory. Create a new subdirectory (e.g., in your "home" directory) and name this new directory "lab1". From the command window prompt, you can create the new subdirectory using the following command:

```
mkdir lab1
```

The first command above switches you to the root directory (assuming you're not there already). The second command (mkdir = "make directory") creates the *lab1* directory[1].

Copy the "Hello World" program (HelloWorld.hla) to this *lab1* directory using the following command window statement[2]:


```
copy c:\AoA\Volume1\CH02\HelloWorld.hla   c:\lab1
```

From the command prompt window, switch to this new directory using the command:

cd lab1

To compile this program, type the following at the command prompt:

hla HelloWorld

After a few moments, the command prompt should reappear. At this point, your program has been successfully compiled. To run the executable (HelloWorld.exe) that HLA has produced, you would use the following command:

HelloWorld

The program should run, display "Hello World", and then terminate. At that time the command window should be waiting for another command.

If you have not successfully completed the previous steps, return to the previous section and repeat the steps to verify that HLA is operational on your system. Remember, each time you start a new command window under Microsoft Windows, you must execute the "ihla.bat" file (or otherwise set up the environment) in order to make HLA accessible in that command window.

In your lab report, describe the output of the HLA compiler. For additional compilation information, use the following command to compile this program:

hla -v HelloWorld

The "-v" option stands for *verbose* compile. This presents more information during the compilation process. Describe the output of this verbose compilation in your lab report. If possible, capture the output and include the captured output with your lab report. To capture the output to a file, use a command like the following:

hla -test -v HelloWorld >capture.txt

This command sends most of the output normally destined to the screen to the "capture.txt" output file. You can then load this text file into an editor for further processing. Of course, you may choose a different filename than "capture.txt" if you so desire.

5.5.3 Compiling Other Programs Appearing in this Chapter

The *volume1/ch02* directory contains all the sample programs appearing in Chapter Two. They include

- HelloWorld.hla
- CharInput.hla
- CheckerBoard.hla
- DemoMOVaddSUB.hla
- DemoVars.hla
- fib.hla
- intInput.hla

1. Some school's labs may not allow you to place information on the C: drive. If you want or need to place your personal working directory on a different drive, just substitute the appropriate drive letter for "C:" in these examples.
2. You may need to modify this statement if the AoA directory does not appear in the root directory of the C: drive or in "/usr"

- NumsInColums.hla
- NumsInColums2.hla
- PowersOfTwo.hla

Copy each of these files to your lab1 subdirectory. Compile and run each of these programs. Describe the output of each in your lab report.

5.5.4 Creating and Modifying HLA Programs

In order to create or modify HLA programs you will need to use a *text* editor to manipulate HLA source code. Windows provides two low-end text editors: notepad.exe and edit.exe. Notepad.exe is a windows-based application while edit.exe is a console (command prompt) application. Neither editor is particularly good for editing program source code; if you have an option to use a different text editor (e.g., the Microsoft Visual Studio system that comes with VC++ and other Microsoft languages), you should certainly do so. Since the choice of text editor is very personal, this text will not make any assumptions about your choice; this means that this lab exercise cannot explain the steps you need to follow in order to edit an HLA program. However, this text does assume that you've written, compiled, and run high level language programs; you can generally use the same tools for editing HLA programs that you've been using for high level language programs.

Warning: do not use Microsoft Word, Wordpad, or any other word processing programs to create or modify HLA programs. Word processing programs insert extra characters into the document that are incompatible with HLA. If you accidentally save a source file from one of these word processors, you will not be able to compile the program[3].

For the time being, edit the "HelloWorld.hla" program and modify the statement:

stdout.put("Hello, World of Assembly Language", nl);

Change the text 'World of Assembly Language' to your name, e.g.,

stdout.put("Hello Randall Hyde", nl);

After you've done this, save the file to disk and recompile and run the program. Assuming you haven't introduced any typographical errors into the program, it should compile and run without incident. After making the modifications to the program, capture the output and include the captured output in your lab report. You can capture the output from this program by using the I/O redirection operator as follows:

Windows: HelloWorld >out.txt

This sends the output ("Hello Randall Hyde") to the "out.txt" text file rather than to the display. Include the sample output and the modified program in your lab report. Note: don't forget to include any erroneous source code in your lab report to demonstrate the changes you've made during development of the code.

5.5.5 Writing a New Program

To create a brand-new program is relatively east. Just create a new file with your text editor and enter the HLA program into that file. As an exercise, enter the following program into your editor (note: this program is not available in the AoA directory, you must enter this file yourself):

```
program onePlusOne;
#include( "stdlib.hhf" );
```

3. Note that many word processing programs provide a "save as text" option. If you accidentally destroy a source file by saving it from a word processor, simply reenter the word processor and save the file as text.


```
static
    One: int32;

begin onePlusOne;

    mov( 1, One );
    mov( One, eax );
    add( One, eax );
    mov( eax, One );
    stdout.put( “One + One = “, One, nl );

end onePlusOne;
```

Program 5.1 OnePlusOne Program

Remember, HLA is very particular about the way you spell names. So be sure that the alphabetic case is correct on all identifiers in this program. Before attempting to compile your program, proof read it to check for any typographical errors.

After entering and saving the program above, exit the editor and compile this program from the command prompt. If there are any errors in the program, reenter the editor, correct the errors, and then compile the program again. Repeat until the program compiles correctly.

Note: If you encounter any errors during compilation, make a printout of the program (with errors) and hand write on the printout where the errors occur and what was necessary to correct the error(s). Include this printout with your lab report.

After the program compiles successfully, run it and verify that it runs correctly. Include a printout of the program and the captured output in your lab report.

5.5.6 Correcting Errors in an HLA Program

The following program (HasAnError.hla in the appropriate AoA directory) contains a minor syntax error (a missing semicolon). Compile this program:

```
// This program has a syntactical error to
// demonstrate compilation errors in an HLA
// program.

program hasAnError;
#include( "stdlib.hhf" );
begin hasAnError;

    stdout.puts( "This program doesn't compile!" )  // missing ";"

end hasAnError;
```

Program 5.2 Sample Program With a Syntax Error

When you compile this program, you will notice that it doesn't report the error on line nine (the line actually containing the error). Instead, it reports the error on line 11 (the "end" statement) since this is the first point at which the compiler can determine that an error has occurred.

Capture the error output of this program into a text file using the following command:

Windows: hla -test HasAnError >err1.txt

Include this output in your laboratory report.

Correct the syntax error in this program and compile and run the program. Include the source code of the corrected program as well as its output in your lab report.

5.5.7 Write Your Own Sample Program

Conclude this laboratory exercise by writing a simple little program of your own. Include the source code and sample output in your lab report. If you have any syntax errors in your code, be sure to include a printout of the incorrect code with hand-written annotations describing how you fixed the problem(s) in your program.

5.6 Laboratory Exercises for Chapter Three and Chapter Four

Accompanying this text is a significant amount of software. The software can be found in the AoA/volume1 directory. Inside this directory is a set of directories with names like *ch03* and *ch04*, with the names obviously corresponding to chapters in this textbook. All the source code to the example programs in this chapter can be found in the *ch03* and *ch04* subdirectories. The *ch04* subdirectory also contains some executable programs for this chapter's laboratory exercises as well as the (Borland Delphi) source code for the lab exercises. Please see this directory for more details.

5.6.1 Data Conversion Exercises

In this exercise you will be using the "convert.exe" program found in the *ch04* subdirectory. This program displays and converts 16-bit integers using signed decimal, unsigned decimal, hexadecimal, and binary notation.

When you run this program it opens a window with four *edit boxes*. (one for each data type). Changing a value in one of the edit boxes immediately updates the values in the other boxes so they all display their corresponding representations for the new value. If you make a mistake on data entry, the program beeps and turns the edit box red until you correct the mistake. Note that you can use the mouse, cursor control keys, and the editing keys (e.g., DEL and Backspace) to change individual values in the edit boxes.

For this exercise and your laboratory report, you should explore the relationship between various binary, hexadecimal, unsigned decimal, and signed decimal values. For example, you should enter the unsigned decimal values 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, and 32768 and comment on the values that appear in the other text boxes.

The primary purpose of this exercise is to familiarize yourself with the decimal equivalents of some common binary and hexadecimal values. In your lab report, for example, you should explain what is special about the binary (and hexadecimal) equivalents of the decimal numbers above.

Another set of experiments to try is to choose various binary numbers that have exactly two bits set, e.g., 11, 110, 1100, 1 1000, 11 0000, etc. Be sure to comment on the decimal and hexadecimal results these inputs produce.

Try entering several binary numbers where the L.O. eight bits are all zero. Comment on the results in your lab report. Try the same experiment with hexadecimal numbers using zeros for the L.O. digit or the two L.O. digits.

You should also experiment with negative numbers in the signed decimal text entry box; try using values like -1, -2, -3, -256, -1024, etc. Explain the results you obtain using your knowledge of the two's complement numbering system.

Try entering even and odd numbers in unsigned decimal. Discover and describe the difference between even and odd numbers in their binary representation. Try entering multiples of other values (e.g., for three: 3, 6, 9, 12, 15, 18, 21, ...) and see if you can detect a pattern in the binary results.

Verify the hexadecimal <-> binary conversion this chapter describes. In particular, enter the same hexadecimal digit in each of the four positions of a 16-bit value and comment on the position of the corresponding bits in the binary representation. Try entering several binary values like 1111, 11110, 111100, 1111000, and 11110000. Explain the results you get and describe why you should always extend binary values so their length is an even multiple of four before converting them.

In your lab report, list the experiments above plus several you devise yourself. Explain the results you expect and include the actual results that the convert.exe program produces. Explain any insights you have while using the convert.exe program.

5.6.2 Logical Operations Exercises

The "logical.exe" program is a simple calculator that computes various logical functions. It allows you to enter binary or hexadecimal values and then it computes the result of some logical operation on the inputs. The calculator supports the dyadic logical AND, OR, and XOR. It also supports the monadic NOT, NEG (two's complement), SHL (shift left), SHR (shift right), ROL (rotate left), and ROR (rotate right).

When you run the logical.exe program it displays a set of buttons on the left hand side of the window. These buttons let you select the calculation. For example, pressing the AND button instructs the calculator to compute the logical AND operation between the two input values. If you select a monadic (unary) operation like NOT, SHL, etc., then you may only enter a single value; for the dyadic operations, both sets of text entry boxes will be active.

The logical.exe program lets you enter values in binary or hexadecimal. Note that this program automatically converts any changes in the binary text entry window to hexadecimal and updates the value in the hex entry edit box. Likewise, any changes in the hexadecimal text entry box are immediately reflected in the binary text box. If you enter an illegal value in a text entry box, the logical.exe program will turn the box red until you correct the problem.

For this laboratory exercise, you should explore each of the bitwise logical operations. Create several experiments by carefully choosing some values, manually compute the result you expect, and then run the experiment using the logical.exe program to verify your results. You should especially experiment with the masking capabilities of the logical AND, OR, and XOR operations. Try logically ANDing, ORing, and XORing different values with values like 000F, 00FF, 00F0, 0FFF, FF00, etc. Report the results and comment on them in your laboratory report.

Some experiments you might want to try, in addition to those you devise yourself, include the following:

- Devise a mask to convert ASCII values '0'..'9' to their binary integer counterparts using the logical AND operation. Try entering the ASCII codes of each of these digits when using this mask. Describe your results. What happens if you enter non-digit ASCII codes?
- Devise a mask to convert integer values in the range 0..9 to their corresponding ASCII codes using the logical OR operation. Enter each of the binary values in the range 0..9 and describe your results. What happens if you enter values outside the range 0..9? In particular, what happens if you enter values outside the range 0h..0fh?
- Devise a mask to determine whether a 16-bit integer value is positive or negative using the logical AND operation. The result should be zero if the number is positive (or zero) and it should be non-zero if the number is negative. Enter several positive and negative values to test your mask. Explain how you could use the AND operation to test *any* single bit to determine if it is zero or one.
- Devise a mask to use with the logical XOR operation that will produce the same result on the second operand as applying the logical NOT operator to that second operand.
- Verify that the SHL and SHR operators correspond to an integer multiplication by two and an integer division by two, respectively. What happens if you shift data out of the H.O. or L.O. bits? What does this correspond to in terms of integer multiplication and division?
- Apply the ROL operation to a set of positive and negative numbers. Based on your observations in Section 5.6.2, what can you say about the result when you rotate left a negative number or a positive number?
- Apply the NEG and NOT operators to a value. Discuss the similarity and the difference in their results. Describe this difference based on your knowledge of the two's complement numbering system.

5.6.3 Sign and Zero Extension Exercises

The "signext.exe" program accepts eight-bit binary or hexadecimal values then sign and zero extends them to 16 bits. Like the logical.exe program, this program lets you enter a value in either binary or hexadecimal and the program immediately zero and sign extends that value.

For your laboratory report, provide several eight-bit input values and describe the results you expect. Run these values through the signext.exe program and verify the results. For each experiment you run, be sure to list all the results in your lab report. Be sure to try values like $0, $7f, $80, and $ff.

While running these experiments, discover which hexadecimal digits appearing in the H.O. nibble produce negative 16-bit numbers and which produce positive 16-bit values. Document this set in your lab report.

Enter sets of values like (1,10), (2,20), (3,30), ..., (7,70), (8,80), (9,90), (A,A0), ..., (F,F0). Explain the results you get in your lab report. Why does “F” sign extend with zeros while “F0” sign extends with ones?

Explain in your lab report how one would sign or zero extend 16 bit values to 32 bit values. Explain why zero extension or sign extension is useful.

5.6.4 Packed Data Exercises

The packdata.exe program uses the 16-bit Date data type appearing in Chapter Three (see “Bit Fields and Packed Data” on page 81). It lets you input a date value in binary or decimal and it packs that date into a single 16-bit value.

When you run this program, it will give you a window with six data entry boxes: three to enter the date in decimal form (month, day, year) and three text entry boxes that let you enter the date in binary form. The month value should be in the range 1..12, the day value should be in the range 1..31, and the year value should be in the range 0..99. If you enter a value outside this range (or some other illegal value), then the packdata.exe program will turn the data entry box red until you correct the problem.

Choose several dates for your experiments and convert these dates to the 16-bit packed binary form by hand (if you have trouble with the decimal to binary conversion, use the conversion program from the first set of exercises in this laboratory). Then run these dates through the packdata.exe program to verify your answer. Be sure to include all program output in your lab report.

At a bare minimum, you should include the following dates in your experiments:

2/4/68, 1/1/80, 8/16/64, 7/20/60, 11/2/72, 12/25/99, Today’s Date, a birthday (not necessarily yours), the due date on your lab report.

5.6.5 Running this Chapter’s Sample Programs

The *ch03* and *ch04* subdirectories also contain the source code to each of the sample programs appearing in Chapters Three and Four. Compile and run each of these programs. Capture the output and include a printout of the source code and the output of each program in your laboratory report. Comment on the results produced by each program in your laboratory report.

5.6.6 Write Your Own Sample Program

To conclude your laboratory exercise, design and write a program on your own that demonstrates the use of each of the data types presented in this chapter. Your sample program should also show how you can interpret data values differently, depending on the instructions or HLA Standard Library routines you use to operate on that data. Your sample program should also demonstrate conversions, logical operations, sign and zero extension, and packing or unpacking a packed data type (in other words, your program should demonstrate your understanding of the other components of this laboratory exercise). Include the source code, sample output, and a description of your sample program in your lab report.

