

Lab #5

Programming Problems

PURPOSE

This lab is an introduction to 68000 assembly language programming. Students will write, assemble, download, execute, debug, and demonstrate assembly language problems that solve specific yet simple programming problems. Concepts from the previous labs are used extensively.

Upon completing this lab, students will be able to do the following:

- Write, test, and debug 68000 assembly language programs to solve defined problems in data computation, manipulation, or conversion.

PREPARATION

Prior to the scheduled lab session, read the following chapter from your textbook:

- Chapter 5 (Programming Examples)

MATERIALS

Hardware:

- 68KMB 68000-based computer
- PC host computer
- RS232C serial interface cable

MS-DOS Software:

- A68K 68000 cross assembler
- XLINK 68000 linker, locator, conversion utility
- PC-VT VT100 terminal emulator
- EDIT MS-DOS text editor (or equivalent)

68000 Programs:

- EXAMPLE provided (to be entered)
- SHIFT32 to be written
- LENSTR to be written
- SAME to be written
- HEXCHAR to be written
- LOOKUP to be written

INTRODUCTION

The following programming example demonstrates how problems are stated in this lab.

Problem: Write a 68000 program called EXAMPLE to compute the sum of three 16-bit words of data. The data are stored in memory starting at address \$9000, identified by the label NUMBERS. Store the result immediately after the data, in SUM at memory location \$9006.

Sample Conditions:

Before:

Address	Contents
009000	1234
009002	5678
009004	0ABC
009006	0000

After:

Address	Contents
009000	1234
009002	5678
009004	0ABC
009006	7368

Note: \$1234 + \$5678 + \$0ABC
= \$7368

The sample conditions show the source data and the memory location where the result is stored. The result of the addition is shown in the "after" contents of memory location \$9006. This allows the programmer to work through the problem by hand to verify the objective.

Below is one possible solution to this problem.

Solution:

```

1
*****
2                                     *   EXAMPLE.SRC
*
3
*****
4 00008000          CODE      EQU      $8000
5 00009000          DATA     EQU      $9000
6
7 00008000          ORG       CODE
8 00008000 207C0000  EXAMPLE  MOVEA.L  #NUMBERS,A0 ;use A0 as pointer
   00008004 9000
9 00008006 323C0003          MOVE.W  #COUNT,D1  ;use D1 as counter
10 0000800A 4240          CLR.W   D0          ;init D0 = 0
11 0000800C D058          LOOP    ADD.W   (A0)+,D0
12 0000800E 5341          SUBQ    #1,D1
13 00008010 66FA          BNE     LOOP
14 00008012 3080          MOVE.W  D0,(A0)
15 00008014 4E4E          TRAP    #14
16
17 00009000          ORG       DATA
18 00009000 1234          NUMBERS DC.W   $1234
19 00009002 5678          DC.W   $5678
20 00009004 0ABC          DC.W   $0ABC
21 00000003          COUNT    EQU     (*-NUMBERS)/2
22 00009006 0000          SUM    DC.W   0
23 00009008          END      EXAMPLE

```


2. Enter the example problem in a file called EXAMPLE.SRC. Assemble the program and convert the object output file to S-records. Begin execution of PC-VT and transfer the program to the 68KMB.

Use MON68K's single-step facility (described in lab #3) to verify your work above. If you completed the table without any errors, congratulations: You are ready to undertake problems in assembly language programming. Ask your lab instructor for assistance if you have trouble determining the correct conditions after any instruction.



Part II: Programming Problems

Solve each of the following 68000 assembly language programming problems. Make appropriate use of comments, labels, and assembler directives in your source code. Enter your name, the date, and the problem name in comment lines at the top of each source file.

Prior to demonstrating your programs to your lab instructor, obtain printouts of the listing files. Be prepared to demonstrate your program with sample data specified by your lab instructor.

Problem 1: Write a program called SHIFT32 to shift a 32-bit binary number until the most-significant bit of the number is 1. The address of the number is defined by the longword variable NUM at location \$9000. Store the normalized (shifted) number in the variable NORM at location \$9004. Store the number of left shifts required in the byte variable SHIFTS at location \$9008. If the number is zero, clear NORM and SHIFTS.

Sample Conditions:

Before:

Address	Contents
009000	0000
009002	9100
009004	FFFF
009006	FFFF
009008	FFFF
009100	1234
009102	5678

After:

Address	Contents
009004	91A2
009006	B3C0
009008	0003

Note: Test your program with values of 0, \$FFFFFFFF, etc.

5.2



Problem 2: Write a program called LENSTR to determine the length of a string of characters. The starting address of the string is contained in the 32-bit variable START at location \$9000. The end of the string is marked by an ASCII null character. Place the length of the string (excluding the null character) in the variable LENGTH at location \$9004.

Sample Conditions:

Before:

Address	Contents
009000	0000
009002	9040
009004	5555
009040	4142
009042	4344
009044	4546
009046	4748
009048	00FF

After:

Address	Contents
009004	0008

Note: Place the ASCII string in your program by enclosing the characters within single quotes after a DC.B directive.



Problem 3: Write a program called SAME to compare two strings of ASCII characters to see if they are the same. The starting addresses are contained in the longword variables START1 at location \$9000 and START2 at location \$9004. The first byte of each string contains the string length (in bytes) and is followed by the string. If the two strings match, clear the variable MATCH at location \$9008; otherwise set its value to -1.

Sample Conditions:

Before:

Address	Contents
009000	0000
009002	9040
009004	0000
009006	9050
009008	5555
009040	0441
009042	4243
009044	44FF
009050	0441
009052	4244
009054	5FDD

After:

009008	FFFF	= -1 (Strings are different!)
--------	------	-------------------------------

Note: Test your program with several different string conditions. Place strings in your program in the appropriate way.

5.4 

Problem 4: Write a program called HEXCHAR to convert the contents of the variable HEX at location \$9000 to an ASCII character representing the hexadecimal value of the variable. HEX contains a single hexadecimal digit (the four most significant bits are zero). Store the ASCII character in the variable CHAR at location \$9001.

Sample Conditions:

Before:

Address	Contents
009000	0F00

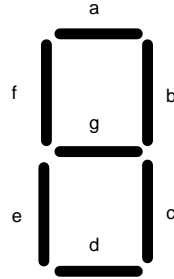
After:

009000	0F46
--------	------

Note: Verify that your program works for any hexadecimal value.

5.5 

Problem 5: Write a program called LOOKUP to convert the contents of the BCD variable DIGIT at location \$9000 to a seven-segment code and store it in the variable CODE at location \$9001. If DIGIT does not contain a single BCD digit, clear CODE. Assume a standard segment arrangement (e.g., 74LS47) with segment *a* as bit 0 and segment *g* as bit 6 (bit 7 = 0, always). This is shown below. Assume a segment is ON for a 1 and OFF for a 0. Hint: begin by constructing a table of BCD-to-CODE mappings.



Sample Conditions:

Before:

Address	Contents
009000	0400

After:

009000	0466
--------	------

Note: Verify that your program works for any value from 0 to F.

5.6 

CONCLUSION

Having completed this lab, students are capable of writing small 68000 programs in assembly language.