Lab 1 Report

Amber Raymer

Amberraymer0562@floridapoly.edu

EEL4768C: Computer Architecture and Organization

Due Date: 24 January, 2019

1.0 Project Description

The purpose of this project was to take in two integers from the user at runtime and then print the sums. In part a, we only had to calculate the sums of the integers. In part b, it required us to calculate the sums in hexadecimal. This was done using the MIPS instruction set, using MARS 4.5.

2.0 Program Design

This design followed the outline specified in the lab directions. In the main function, we first request an integer from the user. We then add $v0 (which holds our user input) to zero and place it into $t0. Next, we repeat the same process as stated above for placing our second user input into $t1. After our integers are placed into $t0 and $t1, we use the add instruction to find the sum of $t0 and $t1 and then place that value into $t2. To print our sum, we place $t2 into $a0 using another add instruction in order to pass the sum to be printed using “li $v0, 1” for part a, where “1” is the argument that $a0 uses to print an integer. In part b, we use “li $v0, 34” to print the sum as a hexadecimal value. Lastly, we use “li $v0, 10” (“10” being the argument used to terminate) and a “syscall” to exit our program.

Table 0a: MIPS Assembly Code (Part a)

lab1a.asm

```
# Amber Raymer -- 1/21/19
# lab1a.asm -- The purpose of this lab is to take two integers from the user at runtime and then
return
#               the sum.
# Registers used - $t0: First user input
#                   - $t1: Second user input
#                   - $t2: Holds the sum of $t0 and $t1
#                   - $v0: Syscall to read initial user input, return value
#                   - $a0: Syscall parameter
# Resources: http://courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html (Syscall
function parameters)

main:
# Get first number from the user, put into $t0
li $v0, 5
syscall
add $t0, $v0, $0        # Place first value into $t0

# Get second number from the user, put into $t1
li $v0, 5
syscall
add $t1, $v0, $0        # Place second value into $t1

# Compute the sum, put into $t2
add $t2, $t0, $t1       # $t2 = $t0 + $t1

# Print $t2
add $a0, $t2, $0        # $a0 = $t2 (our sum)
li $v0, 1               # Print sum
syscall

li $v0, 10      # Exit
syscall
# end of lab1.asm
```

Table 0a: MIPS Assembly Code (Part b)

lab1b.asm

```
# Amber Raymer -- 1/21/19
# lab1b.asm -- The purpose of this lab is to take two hexadecimal values from the user at runtime and
#               then return the sum.
# Registers used - $t0: First user input
#                  - $t1: Second user input
#                  - $t2: Holds the sum of $t0 and $t1
#                  - $v0: Syscall to read initial user input, return value
#                  - $a0: Syscall parameter
# Resources: http://courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html (Syscall function parameters)

main:
# Get first number from the user, put into $t0
li $v0, 5
syscall
add $t0, $v0, $0        # Place first value into $t0

# Get second number from the user, put into $t1
li $v0, 5
syscall
add $t1, $v0, $0        # Place second value into $t1

# Compute the sum, put into $t2
add $t2, $t0, $t1       # $t2 = $t0 + $t1

# Print $t2
add $a0, $t2, $0        # $a0 = $t2 (our sum)

li $v0, 34              # Print sum as hex value
syscall

li $v0, 10      # Exit
syscall
# end of lab1.asm
```

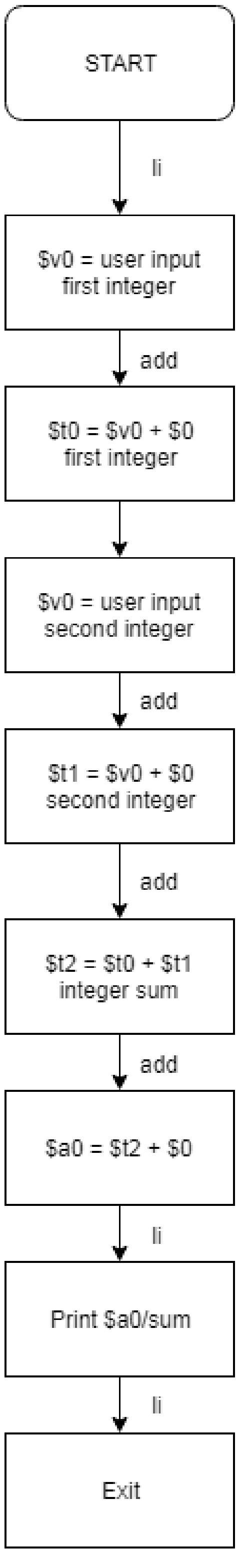

START
li
$v0 = user input first integer
add
$t0 = $v0 + $0 first integer
$v0 = user input second integer
add
$t1 = $v0 + $0 second integer
add
$t2 = $t0 + $t1 integer sum
add
$a0 = $t2 + $0
li
Print $a0/sum
li
Exit

Figure 1a: Flowchart of lab1a.asm

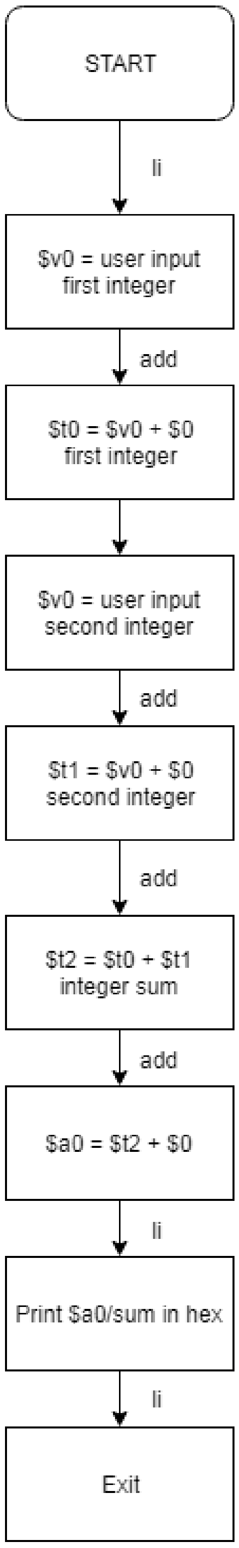

START
li
$v0 = user input first integer
add
$t0 = $v0 + $0 first integer
$v0 = user input second integer
add
$t1 = $v0 + $0 second integer
add
$t2 = $t0 + $t1 integer sum
add
$a0 = $t2 + $0
li
Print $a0/sum in hex
li
Exit

Figure 1b: Flowchart of lab1b.asm

3.0 Symbol Table

The registers used in the MIPS code and their purpose are listed below. The labels used in the MIPS code are also identified and described. The data type is also identified where applicable. Table 1 is a symbol table for the registers used and Table 2 is a symbol table for the labels used.

Table 1: Registers Used

Register	Use
$v0	Tells MARS what to do when a syscall is called and used to return values.
$a0	The argument to be used during some syscalls. Also used to pass arguments depending on the parameter used.
$t0	Holds the first integer from the user input at runtime.
$t1	Holds the second integer from the user input at runtime.
$t2	Holds the sum of the values in $t0 and $t1.

Table 2: Labels Used

Label	Use
main	Executed first at runtime.

4.0 Learning Coverage

The skills implemented during this lab include:

- Taking integers for user input
- Passing variables to different registers
- How to use the “add” instruction
- How to print an integer/hexadecimal value
- How to exit execution

5.0 Prototype in C-Language

A prototype for this MIPS program was written in C that mimics the functionality of the desired assembly program. The source code is shown below and the flowcharts shown above were modeled first in C. The commented, fully operational C-language code is listed below:

Table 3a: C-prototype Code

C-language Code – Part a

```
#include <stdio.h>
int main()
{
   int t0; // first number
   int t1; // second number
   int t2; // sum

   printf("Enter two integers: "); // This string output was left out of the assembly program
   scanf("%d %d", &t0, &t1); // Enter t0 and t1

   t2 = t0 + t1; // Save sum of t0 and t1 into t2

   printf("%d + %d = %d", t0, t1, t2); // Print the sum (The string output was left out of the
assembly program)

   return 0;
}
```

Table 3b: C-prototype Code

C-language Code – Part b

```
#include <stdio.h>
int main()
{
   unsigned char t0; // First hex value
   unsigned char t1; // Second hex value
   unsigned char t2; // Hex value sum

   printf("Enter two hexadecimal values: "); // This string output was left out of the assembly
program
   scanf("%x %x", &t0, &t1); // Store hex values

   t2 = t0 + t1; // Sum of hex values

   printf("%X + %X = 0x%X", t0, t1, t2); // Display sum using capitalized hex values (This
string output was left out of the assembly program)

   return 0;
}
```

6.0 Test Plan

To properly test this code, the user must input two integers (part a) or two hexadecimal values (part b). If all test cases give proper outputs, the program functions correctly. Test cases and expected output verified by the C-prototype are listed in Table 3.

Table 4a: Test Cases Used (Part a)

Test Case	Pixel Data	Expected Output
Standard/random	\$t0 = 3 \$t1 = 2	\$t2 = 5 (sum)
Maximum values	\$t0 = 2,147,483,647 \$t1 = 2,147,483,647	Runtime exception at 0x00400018: arithmetic overflow
Minimum values	\$t0 = 0 \$t1 = 0	\$t2 = 0 (sum)
Most bits different	\$t0 = 131,071 \$t1 = 0 \$t0 = 0 \$t1 = 131,071	\$t2 = 131,071 \$t2 = 131,071

Table 4b: Test Cases Used (Part b)

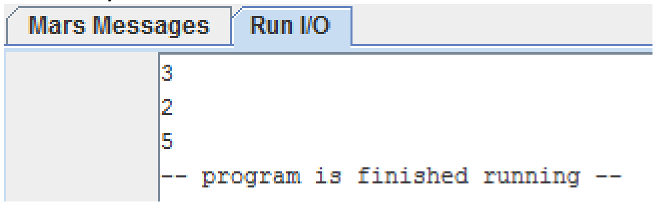
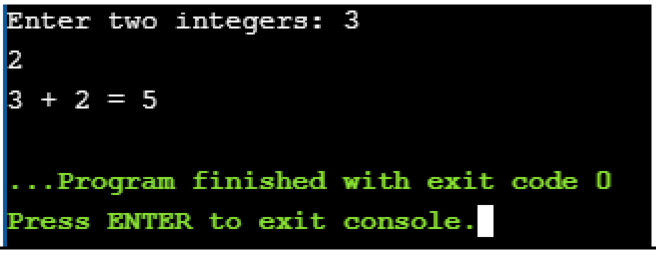
Test Case	Pixel Data	Expected Output
Standard/random	\$t0 = 3 \$t1 = 2 \$t0 = 9 \$t1 = 1	\$t2 = 0x00000005 (sum) \$t2 = 0x0000000A (sum)
Maximum values	\$t0 = 2,147,483,647 \$t1 = 2,147,483,647	Runtime exception at 0x00400018: arithmetic overflow
Minimum values	\$t0 = 0 \$t1 = 0	\$t2 = 0 (sum)
Most bits different	\$t0 = 131,071 \$t1 = 0 \$t0 = 0 \$t1 = 131,071	\$t2 = 131,071 \$t2 = 131,071

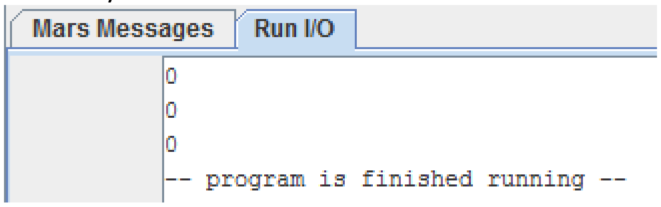
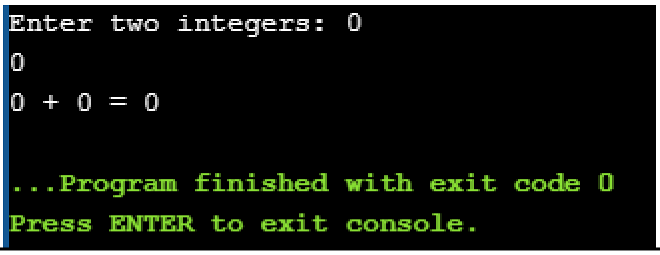
7.0 Test Results

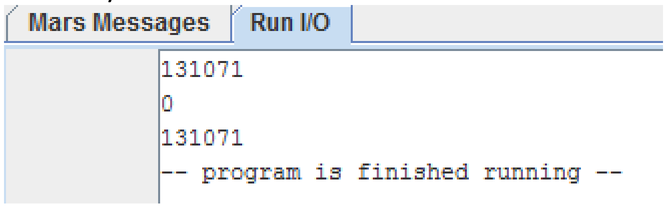
An image depicting three successful test runs of the program is shown below.

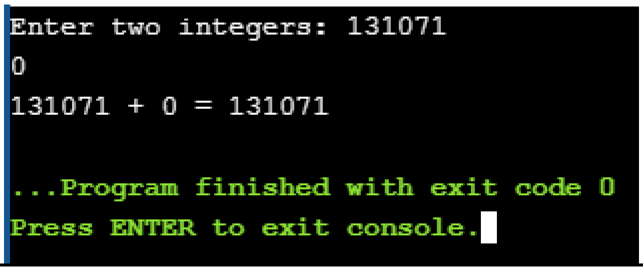
Table 5a: Test Cases – Part a

Test Cases – Part a	

Standard/random	Assembly:
	
	`3`
	`2`
	`5`
	`-- program is finished running --`
	
	C:
	
	`Enter two integers: 3`
	`2`
	`3 + 2 = 5`
	
	`...Program finished with exit code 0`
	`Press ENTER to exit console.`

Minimum values	Assembly:
	
	`0`
	`0`
	`0`
	`-- program is finished running --`
	
	C:
	
	`Enter two integers: 0`
	`0`
	`0 + 0 = 0`
	
	`...Program finished with exit code 0`
	`Press ENTER to exit console.`

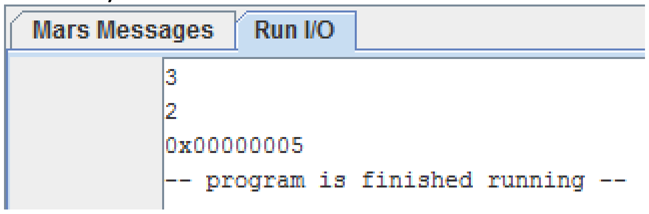
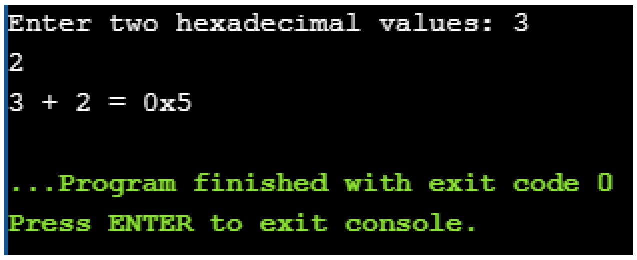
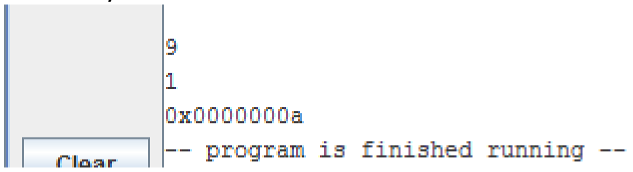
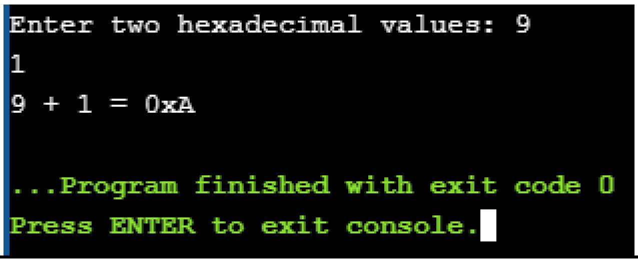
Most bits different	Assembly:
	
	`131071`
	`0`
	`131071`
	`-- program is finished running --`

	C: 

```
Enter two integers: 131071
0
131071 + 0 = 131071

...Program finished with exit code 0
Press ENTER to exit console.
```

Table 5b: Test Cases – Part b

Test Cases – Part a	
Standard/random	Assembly:  C:  Assembly:  C: 

Assembly (first):

```
3
2
0x00000005
-- program is finished running --
```

C (first):

```
Enter two hexadecimal values: 3
2
3 + 2 = 0x5

...Program finished with exit code 0
Press ENTER to exit console.
```

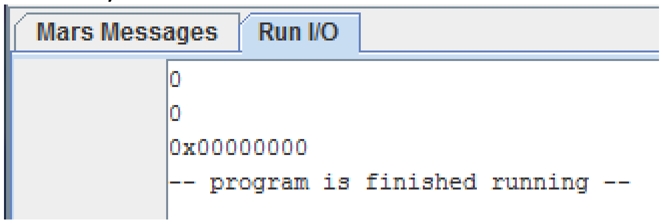
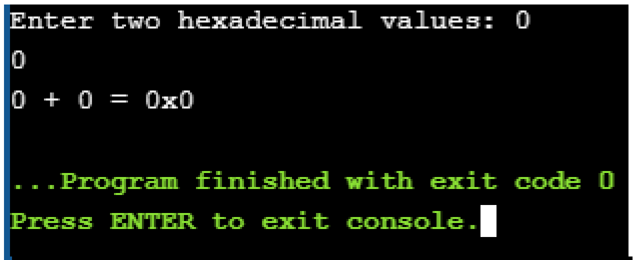
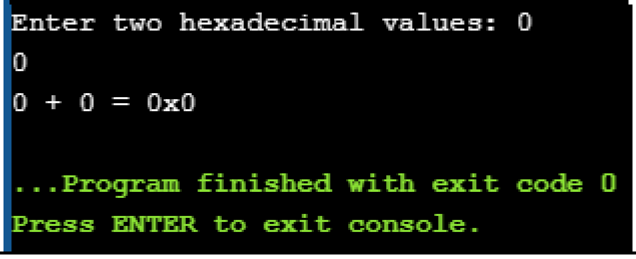
Assembly (second):

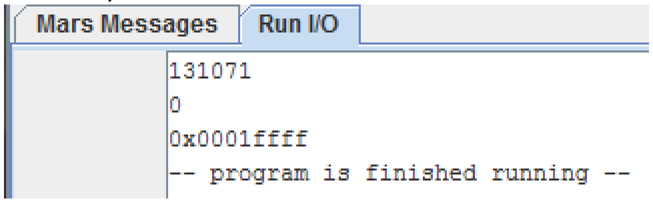
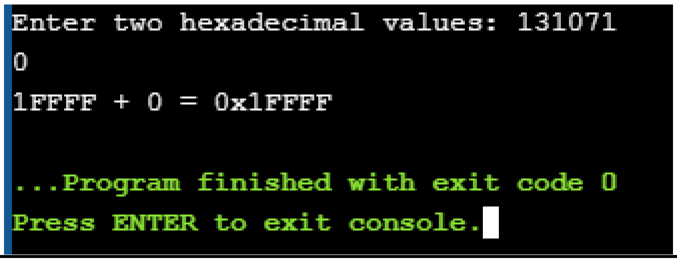
```
9
1
0x0000000a
-- program is finished running --
```

C (second):

```
Enter two hexadecimal values: 9
1
9 + 1 = 0xA

...Program finished with exit code 0
Press ENTER to exit console.
```

Minimum values	Assembly:
	
	Mars Messages Run I/O
	`0`
	`0`
	`0x00000000`
	`-- program is finished running --`
	
	C:
	
	`Enter two hexadecimal values: 0`
	`0`
	`0 + 0 = 0x0`
	
	`...Program finished with exit code 0`
	`Press ENTER to exit console.`
	
	`Enter two hexadecimal values: 0`
	`0`
	`0 + 0 = 0x0`
	
	`...Program finished with exit code 0`
	`Press ENTER to exit console.`

Most bits different	Assembly:
	
	Mars Messages Run I/O
	`131071`
	`0`
	`0x0001ffff`
	`-- program is finished running --`
	
	C:
	
	`Enter two hexadecimal values: 131071`
	`0`
	`1FFFF + 0 = 0x1FFFF`
	
	`...Program finished with exit code 0`
	`Press ENTER to exit console.`

8.0 References

8.1 Lecture Notes

EEL 4768C Module-01 and Module-02: Slides adapted from Dr. DeMara (UCF), Dr. Sahawneh (FPU), Dr. Mutlu (CMU). They are available on Canvas.

8.2 MARS Simulator

The MARS Simulator for MIPS processors, available at:
http://courses.missouristate.edu/kenvollmar/mars/

8.3 MARS Syscall Reference Guide

http://courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html