Mitchell Jonker
CSCE-212-001
2 February, 2022

Project 1 Report

1.0 | Program inputs and outputs utilized in each of the three programs are an important component of the program, because they are critical to the function of the programs.

1.1 | Program 1 uses system output to the command terminal, and takes in information through the command terminal as well.

1.2 | Program 2 uses system output via the command terminal to prompt to the user to enter values for variables a, b, c, and d through accepting input from the terminal.

1.3 | Program 3 runs without any input from the user. It simply runs when triggered to do so in MARS. The output of the program is directed to the command terminal.Program 1 greets the user in the terminal, asking for their name.

2.0 | The design of each program is as follows:

2.1 | Program 1 greets the user and asks their name with a string, printed by loading a predefined string into the $a0 address, storing the immediate value 4 in register $v0, and initiating a syscall. This sequence of actions causes the program to print the string to the command terminal. The program then loads the address of the name string with a limit of 50 characters total for the name string. Immediate value 4 was loaded into register $v0, where syscall would then read that value and act according to that number. To prepare the system to accept a string, the number 8 is loaded into $v0, and syscall is executed. At this point the program waits for the user to enter their name. Once the user enters the name, the program loads address register $a0 with the welcome string, initiates a print string using the same method as the greeting at the beginning of the program, followed by printing the name (which was stored upon user entering their name). While a very simple program, these small elements of MIPS code are essential to mastering the programming language.

2.2 | Program 2 prompts the user to enter the values for four different variables, stored in $s0-$s-4. These variables are initially defined in the beginning of the program to 0. While this may not be necessary, I have added these, with the addition of comments, to aid in the development of the program. Each variable is saved in the same way. The system prompts the user using a predefined string with syscall (while in print string mode), then moves to “accept/read integer” mode where the program then waits for the user to enter the value for that specific variable. After this, the entered value is saved in its respective $s(x) register. Immediately after the variable is saved, it is checked to see if it is less than zero (if it is negative). If the variable is greater than or equal to zero (not negative), then the program will follow an identical process, beginning by prompting the user for the proceeding variable. In any case that the integer is negative, the program notifies the user that a negative value was entered, and that they must restart. After

all 4 variables have been entered (and none of them were negative), the equation F is calculated. This is calculated by using three add's, one add immediate, a subtraction function. The small parts of the function F that need to be calculated individually are stored in temporary registers. After the final result of F has been stored (in $t4), the system presents the user with the result by printing the string "F = " followed by the resulting value. Upon outputting the value, the program sets $v0 to 10 and syscall is executed, ending the program.

2.3 | Program 3 first notifies the user that the program started, using a string. Like Program 2, the values I, J, and K are initiated with their beginning values early on, so that they can be easily identified and traced through the program. At the beginning of the program, I = 0, J = 3, and K = 5. The program then begins a loop. Since we only want the program to run while I is less than 5, this is the first thing the program checks. If the I is greater than or equal to 5, the program jumps to the exit section, where the user is notified that the program is ending, then the program stops. Meanwhile, if that condition is not met, the program will continue to operate without exiting. This means that the program continues through the loop endlessly until the condition is met (and therefore causing the loop to stop and the program to quit). The loop contains a small mathematical calculation. The calculation of F is as follows: F = I+J-K. Since this is addition and subtraction, it does not matter which order the segments are calculated. This program adds I and J together into a temporary register ($t0), then subtracts K from register $t0, and stores that in $t1. The program then prints the string "f = " and then prints what f equals at that moment. A new line is then printed so that the output into the command terminal is cleaner. This is followed by incrementing the variable I. This is done by using the add immediate function as follows: "addi $s0, $s0, 1" this is basically the C equivalent to "X = X+1". This is followed by the jump command directing the program to cycle back to the top of the loop segment. This program will cycle until the condition is met, where the user will be notified and the program will end. This is a simple loop, but efficiently allows a programmer to cycle through an incremental process easily.

3.0 | Symbol Table:

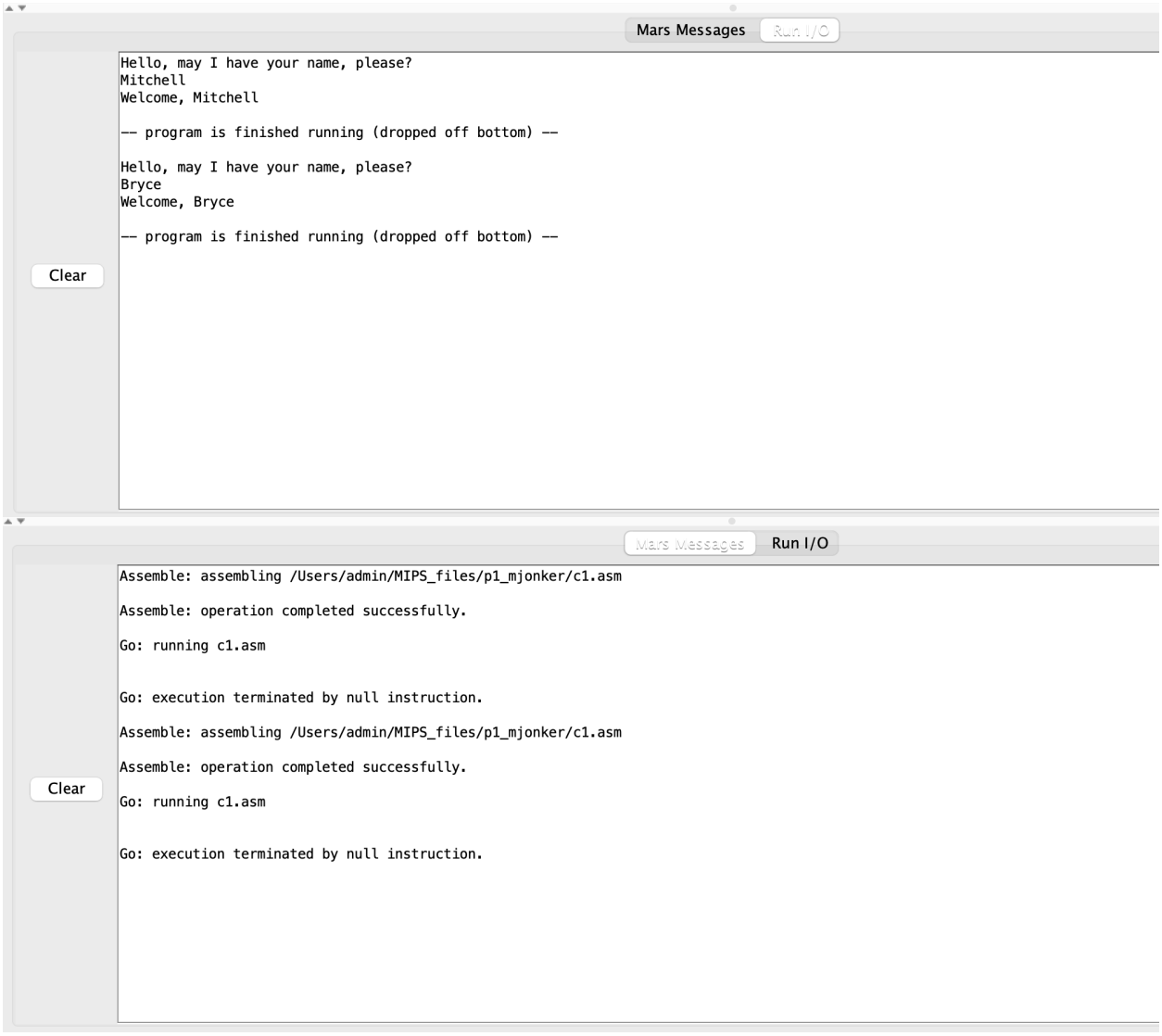
Register	Purpose
$a0	Syscall argument. This register holds or accepts the data of a syscall action.
$v0	Syscall behavior register. Syscall acts differently based on what value is stored here.
$s0 - $s4	These saved registers are used to store variables I used throughout a program, such as I, J, & K.
$t0 - $t4	These temporary data registers are used to store variables as I calculate larger functions such as F in Program 2.

4.0 | Learning coverage:
I learned various functions and technical applications through this MARS programming experience with the MIPS programming language:

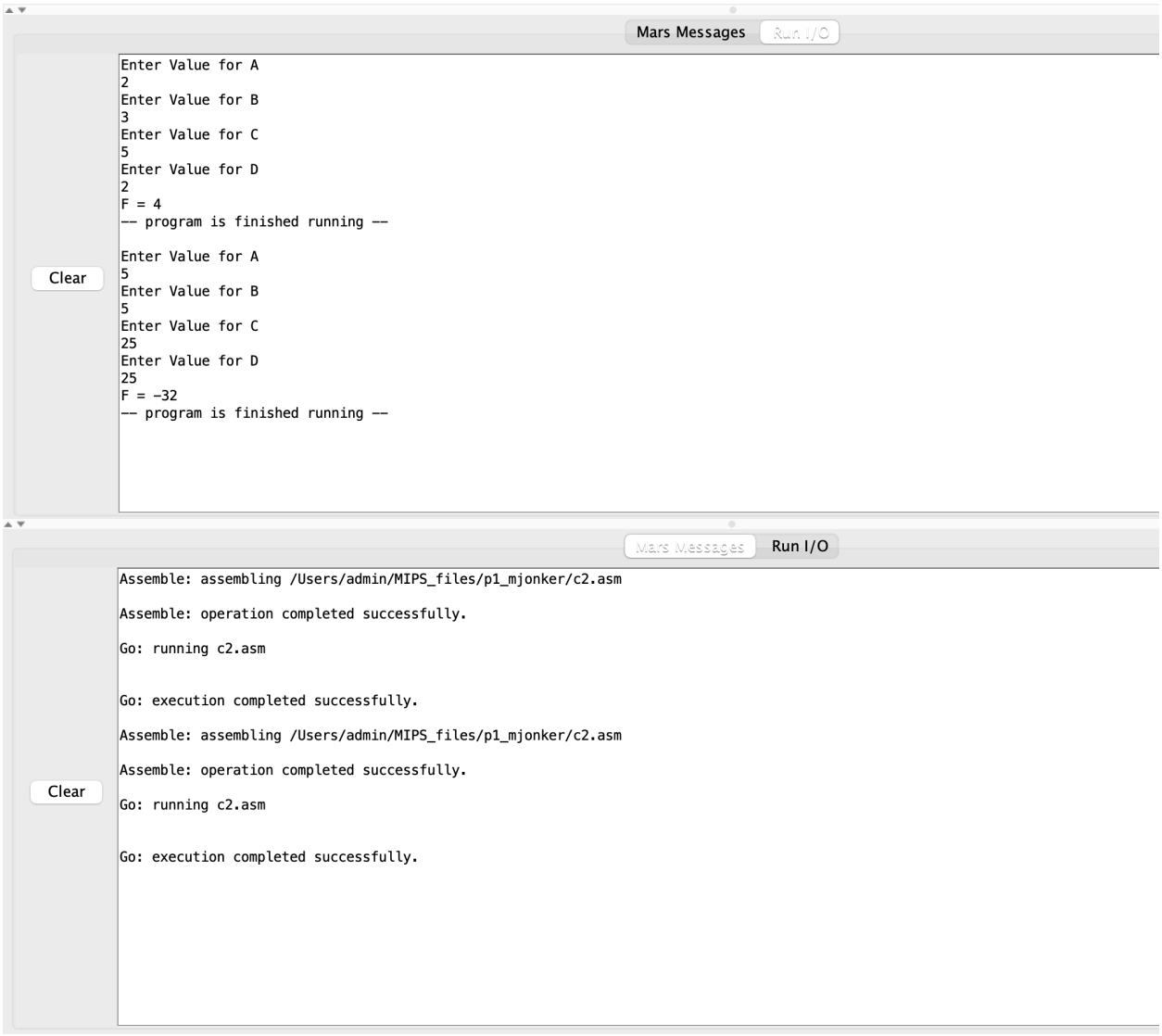
4.1 | I learned how to use syscall, and how the integer value that is given to it changes its behavior.
4.2 | I learned the use of registers and how there are different types of registers for different purposes, such as a temporary register ($t(x)) for temporary operations, and return value registers ($v(x)) for using different functions like syscall.
4.3 | I learned how to operate different basic arithmetic instructions such as add, addi, and sub. These allow MIPS programming functions to calculate various equations.
4.4 | I learned how to operate jump instructions such as bltz, bne, and j. These allow for the use of looping through code to repeat an action, as well as other operations.
4.5 | I learned the use of some data transfer functions like move, allowing the transfer of data in MIPS.

5.0 | Test Results:
The following are the results from each program, listed in increasing order (Project 1, followed by Project 2, followed by Project 3) - there are two screenshots per program. The first is of the MARS Messages panel, which indicates the successful compilation and execution of a program, while the second screenshot is of the Run I/O panel, which includes the dialog between the program and the user. Note that Programs 1 and 2 are executed twice, to show different outputs based on different inputs, but are included within one screenshot (the screenshots include both runs, in a continuous terminal window). Meanwhile, Program 3 was only executed once, and still consists of two screenshots.

5.1 | Program 1 Results (2 runs total, both panels listed):



5.2 | Program 2 Results (2 runs total, both panels listed):



5.3 | Program 3 Results (both panels listed):