

1.0 Program Input/Output

1.1 Program 1

In the first program we have the user input two numbers both separated by pressing enter. The program outputs a list of numbers that are some linear combination of the two numbers and an iterator, i . This happens from $i=0$ to $i=5$.

1.2 Program 2

The second program takes no inputs and Gives no outputs. It only changes the values of certain addresses in memory based on an iterator and a predetermined formula to determine the values.

1.3 Program 3

The third program takes two integer inputs each separated by pressing enter. It outputs an integer that is calculated from the first two, but changes based on a branching logic structure.

2.0 Program Design

2.1 Program 1

In the first program, I start by creating some messages that will be printed for the user in the “.data” section. Then I set the loop length and create an iterator starting value. In the “.text” section I load in the loop length to \$t4. Next, I print the first message that prompts the user to enter a number followed by a newline character. That number gets stored in \$s1. Then, I do the exact same thing for the second number. This number gets stored in \$s2. This is where I implement labels and jump functions to make the loop. I load our iterator into \$s0. Using basic math instructions like mul, add, I evaluate the inputs based on the given formula then print that result followed by a newline character. I then add 1 to the iterator and check to see if it's less than the loop length. If it is, then I branch back to the loop label, if not, the program will continue on to the end of the program where it exits.

2.2 Program 2

In the second program, I allocate space for the array. Then I create the ascii messages to print. I also make an index integer, and a length integer. In the “.text” portion I load the index and length values into memory and print the first message. I then create some registers that I can use to multiply the variables with. I also load the address for the array so I can save values into it. I start a loop and do the math that is defined in the program definition. I then calculate the offset for the array value and save the value to the correct place in the array. Then I increment the index, and check if the index is less than the length value. If it is, I branch back to the top of the loop, otherwise, the program will print the last message and then end.

2.3 Program 3

In the last program, I set the messages and newline characters in the “.data” section and also load some integers into memory so I can do math on them easily. This program also asks for two integers from the user separated by the enter key. I then load in those numbers and prompt the user for the numbers. I save those numbers to their respective registers and then use branching logic to determine if both meet the conditions listed in the project description. If they do meet the description, I set a temporary register equal to 1. Then I compare those registers to be equal, if they are I jump to the first case, and if not I jump to the second case. These branches apply the correct calculations, print the value and jump to the end of the program where the program ends.

3.0 Symbol Tables

3.1 Program 1

t0	Holds the multiply step for each loop
t1	Holds the addition step for each loop
t4	Holds the loop length for the comparison
s0	Is the iterator value/ the first
s1	Is the first number input
s2	Is the second number input

3.2 Program 2

t0	Is the iterator value
t1	Is the length of the loop
t2	Holds temporary multiply
t3	Holds temporary subtraction
t4	Is the calculated offset for the array
t5	Stand in for the number 2
t6	Stand in for the number 4
t7	Stand in for the number 8
s0	Array Address
s3	The value that is held in the array for each loop

3.3 Program 3

t0	Stand in for the number 5
t1	Stand in for the number 4
t3	Temporary “truth” value
t4	Temporary “truth” value
t5	Calculated output value
s1	Users first input
s2	Users second input

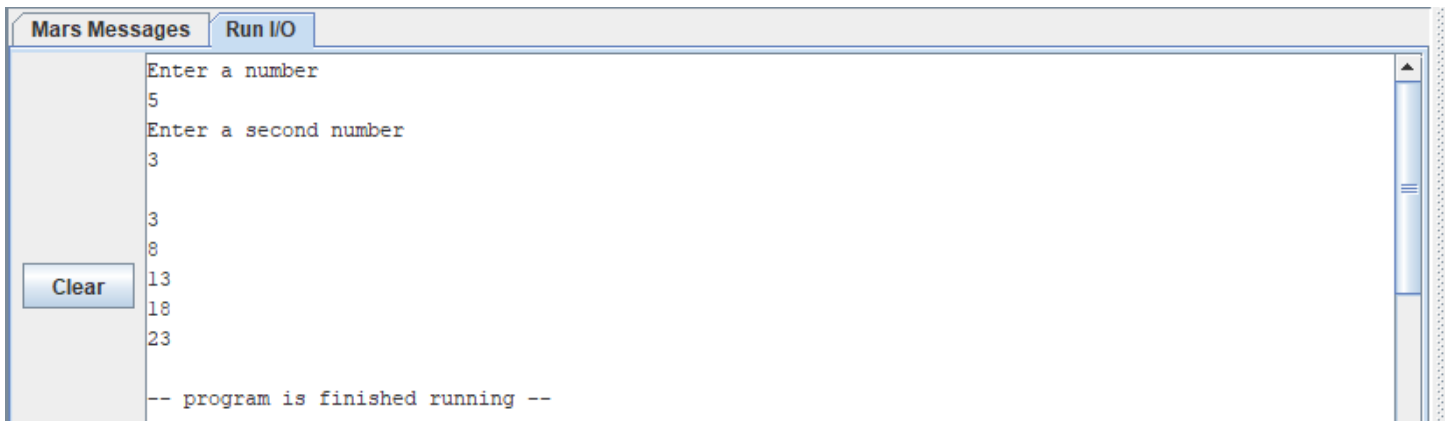
4.0 Learning Coverage

- 4.1: How to address an array in MIPS using offsets and pointers
- 4.2: How to use jump statements in MIPS with labels and j instructions
- 4.3: How to create conditional logic with conditional branching statements
- 4.4: How to properly exit a program
- 4.5: How to combine conditional logic with temporary registers to hold “Truth” values

5 Test Results

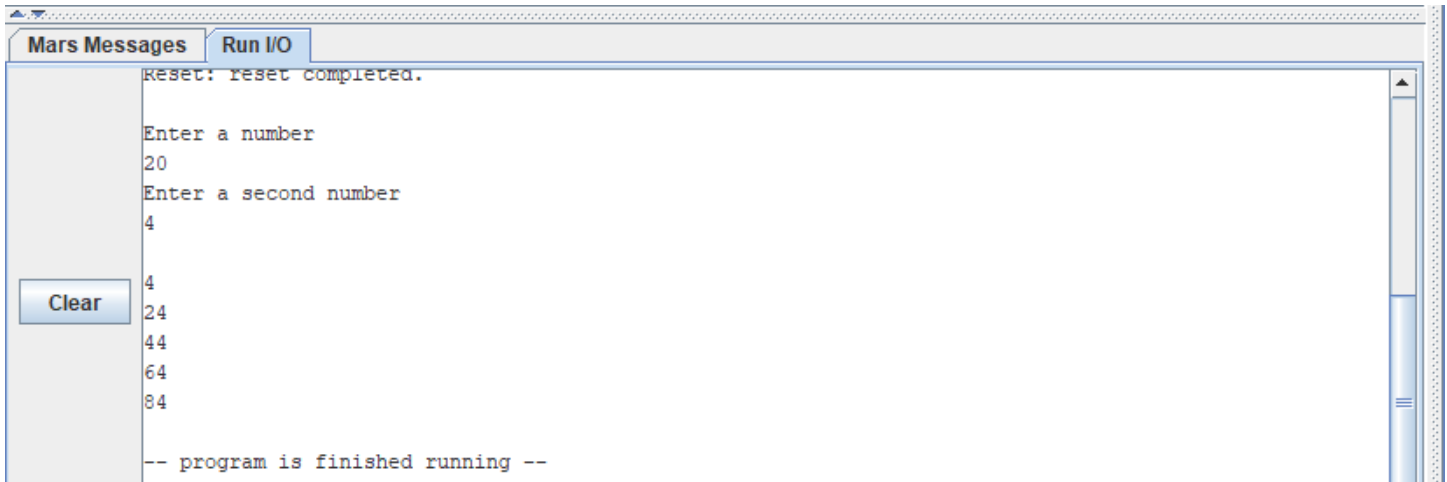
5.1 Program 1 and Program 3

Program 1:



The screenshot shows the Mars Messages window with the 'Run I/O' tab selected. The text area contains the following output: 'Enter a number', '5', 'Enter a second number', '3', '3', '8', '13', '18', '23', and '-- program is finished running --'. A 'Clear' button is visible on the left side of the window.

```
Enter a number
5
Enter a second number
3
3
8
13
18
23
-- program is finished running --
```



The screenshot shows the Mars Messages window with the 'Run I/O' tab selected. The text area contains the following output: 'Reset: reset completed.', 'Enter a number', '20', 'Enter a second number', '4', '4', '24', '44', '64', '84', and '-- program is finished running --'. A 'Clear' button is visible on the left side of the window.

```
Reset: reset completed.
Enter a number
20
Enter a second number
4
4
24
44
64
84
-- program is finished running --
```

Program 3:

Mars MessagesRun I/O

-- program is finished running --

Loop Begins
Loop Ends

-- program is finished running --

ClearEnter a number
5
Enter a number
3
2

-- program is finished running --

Mars MessagesRun I/O

2

-- program is finished running --

Reset: reset completed.

ClearEnter a number
5
Enter a number
10
50

-- program is finished running --

5.2 Program 2

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x706f6f4c	0x67654220	0x0a736e69	0x6f6f4c00	0x6e452070	0x000a7364	0x0000000a
0x10010080	0x00000000	0x00000005	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Mars MessagesRun I/O

4

4
24
44
64
84

Clear-- program is finished running --

Loop Begins
Loop Ends

-- program is finished running --