CS-35101
Thomas Moore
4/16/22

Summary:
For lab 4 I used strings that correspond to each singular digit. These strings are used from lines 51 to 108 as each digit has its one branch that can print the names of the digits. I use several loops within the code, the second one on line 32 uses division to break down the numbers longer than one digit into individual numbers. While the loop on line 45 is used to detect negative numbers and return “Invalid Entry”. If number is longer than a single digit it will store the digits in correct order by using the stack.

Conclusion:

This lab challenged me by turning a basic program in most languages into a 100+ line program that required several forms of register storage and arithmetic. I was having a problem storing some of the data until I realized that I could store it on the stack using $sp. I discovered that $sp works on first in last out principles so I could easily access my data in an organized fashion. The second lesson I learned was the differences between mflo and mfhi. Using div stores either the quotient in the LO register and the remainder in the HI register.

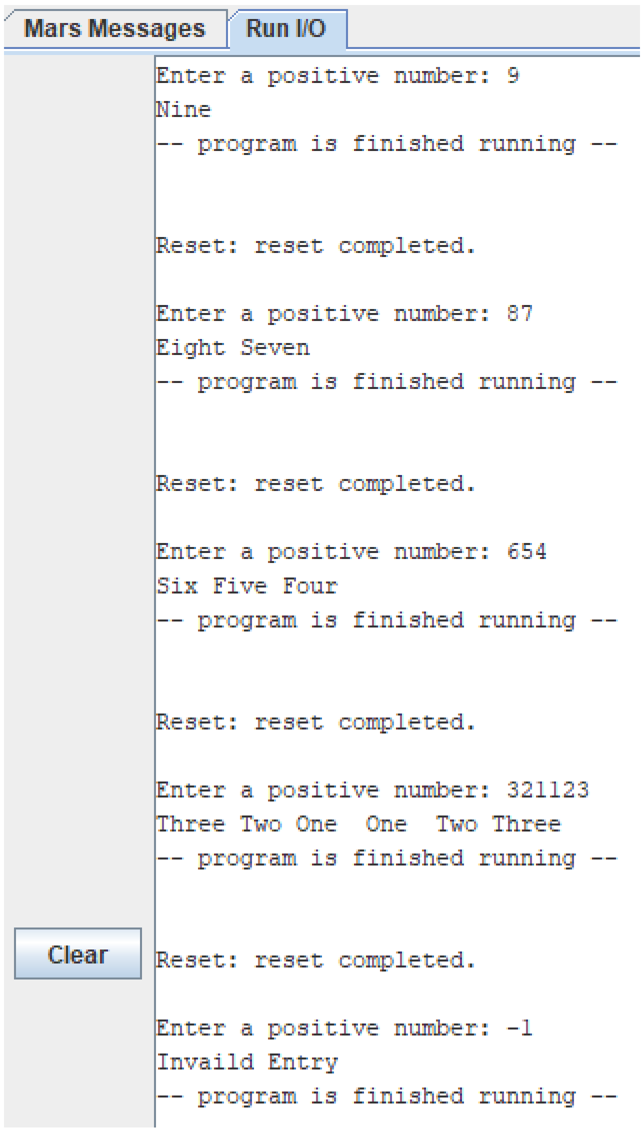
Lab 4 Code:

```
1.  #Thomas Moore
2.  .data
3.          zero: .asciiz "Zero "
4.          one: .asciiz "One  "
5.          two: .asciiz "Two "
6.          three: .asciiz "Three "
7.          four: .asciiz "Four "
8.          five: .asciiz "Five "
9.          six: .asciiz "Six "
10.         seven: .asciiz "Seven "
11.         eight: .asciiz "Eight "
12.         nine: .asciiz "Nine "
13.         intro: .asciiz "Enter a positive number: "
14.         error: .asciiz "Invaild Entry"
15.
16. .text
17.         la $a0, intro
18.         li $v0, 4 #print string
19.         syscall
20.         li $v0, 5 #read in int
21.         syscall
```

```
        move $t0, $v0 #move input to $t0
        blt $t0, $0,invalid #if negative branch to invalid
        li $t1, 10 #storing int 10
        li $v0, 4
        li $t4, -1 #storing int -1

loop1:
        bne $t0, $0, digit
        beq $t2, $t0, check

digit:
        div $t0, $t1
        mflo $t3
        mfhi $t2
        move $t0, $t3
        addi $t4, $t4, 1 #iterations
        addi $sp, $sp, 4
        sw $t2, 0($sp)
        j loop1

check:
        lw $t2, 0($sp)

loop2:
        beq $t4, $0, terminate
        addi $sp, $sp, -4
        lw $t2, 0($sp)
        addi $t4, $t4, -1

pone:
        bne $t2, 1, ptwo
        la $a0, one
        syscall
        j loop2

ptwo:
        bne $t2, 2, pthree
        la $a0, two
        syscall
        j loop2

pthree:
        bne $t2, 3, pfour
        la $a0, three
        syscall
        j loop2

pfour:
        bne $t2, 4, pfive
        la $a0, four
        syscall
        j loop2

pfive:
        bne $t2, 5, psix
        la $a0, five
        syscall
        j loop2

psix:
        bne $t2, 6, pseven
        la $a0, six
        syscall
        j loop2

pseven:
        bne $t2, 7, peight
        la $a0, seven
        syscall
        j loop2

peight:
        bne $t2, 8, pnine
        la $a0, eight
        syscall
        j loop2

pnine:
                bne $t2, 9, pzero
                la $a0, nine
                syscall
                j loop2

        pzero:
                la $a0, zero
                syscall
                j loop2

```

```
invalid:
        la $a0, error
        li $v0, 4
        syscall

terminate:
        li $v0, 10 #standard termination
        syscall
```

Example Output:



```
Enter a positive number: 9
Nine
-- program is finished running --


Reset: reset completed.

Enter a positive number: 87
Eight Seven
-- program is finished running --


Reset: reset completed.

Enter a positive number: 654
Six Five Four
-- program is finished running --


Reset: reset completed.

Enter a positive number: 321123
Three Two One  One  Two Three
-- program is finished running --


Reset: reset completed.

Enter a positive number: -1
Invaild Entry
-- program is finished running --
```