Matthew Michael Sherlin
Dr. Augustine Samba
Computer Organization
November 15, 2020

Assembly Code File:

```
# Matthew Michael Sherlin
# Dr. Augustine Samba
# Computer Organization
# November 15, 2020

.data
        prompt1: .asciiz "Spelling out a positive integer program.\n"
        prompt2: .asciiz "Enter an integer: "
        zero: .asciiz "Zero "
        one: .asciiz "One "
        two: .asciiz "Two "
        three: .asciiz "Three "
        four: .asciiz "Four "
        five: .asciiz "Five "
        six: .asciiz "Six "
        seven: .asciiz "Seven "
        eight: .asciiz "Eight "
        nine: .asciiz "Nine "
        messageError: .asciiz "Invalid entry"
.text

li $v0, 4           #reading the first prompt
la $a0, prompt1
syscall

li $v0, 4           #reading the second prompt
la $a0, prompt2
syscall

li $v0, 5           #entering user input
syscall
move $t0, $v0           #moving user input into saved register

li $t9, 10              #setting the divisor to 10
li $t8, -1              #getting offset for digits

blt $t0, $zero, error   #case for where input is less than zero
beq $t0, $zero, one0    #case for when input is equal to only one zero

while:                  #create while loop while the input isn't zero
bne $t0,$0, loop1
```

```
beq $t2,$0,end          #when the quotient is equal to zero, branch to the end of loop1

loop1:                  #loop in order to find the digits to print out
div $t0,$t9
mflo $t3                #quotient stored into t3
mfhi $t2                #remainder stored into t2

add $t0,$t3,$zero
addi $t8,$t8,1          #add one to digit count
addi $sp,$sp,4          #add word onto stack pointer for each of the digits
sw $t2,0($sp)           #store remainder onto current word in the stack
j while

end:
lw $t2,0($sp)

loop2:
beq $t8,$0,exit         #if there are zero digits left, exit
addi $sp,$sp,-4         #get rid of last word on stack
lw $t2,0($sp)           #load next word to find the number inside it
addi $t8,$t8,-1         #subtract from the digit count

labe1:
bne $t2,1,label2        #branch to next number if it is not equal (same for all)
li $v0, 4
la $a0,one              #read the number prompt out
syscall
j loop2

label2:
bne $t2,2,label3
li $v0, 4
la $a0,two
syscall
j loop2

label3:
bne $t2,3,label4
li $v0, 4
la $a0, three
syscall
j loop2

label4:
bne $t2,4,label5
li $v0, 4
la $a0,four
syscall
```

```
j loop2

label5:
bne $t2,5,label6
li $v0, 4
la $a0,five
syscall
j loop2

label6:
bne $t2,6,label7
li $v0, 4
la $a0,six
syscall
j loop2

label7:
bne $t2,7,label8
li $v0, 4
la $a0,seven
syscall
j loop2

label8:
bne $t2,8,label9
li $v0, 4
la $a0,eight
syscall
j loop2

label9:
bne $t2,9,label0
li $v0, 4
la $a0,nine
syscall
j loop2

label0:
li $v0, 4
la $a0,zero
syscall
j loop2

one0:                   #case if the value is a single 0 or equal to zero (000 for example)
li $v0, 4
la $a0, zero
syscall
j exit
```

```
error:                  #error case if number is negative
li $v0, 4
la $a0, messageError
syscall

exit:
li $v0, 10          #terminate program run and exit
syscall
```

Project Implementation:

In order to get this program to work, I first began by reading the initial prompts out and retrieved the integer used in the program. I used simple la and li instructions and move to achieve this basic part. Next, I had to create an algorithm to achieve the result desired. I began initializing two registers to divide by ten and to count my digits used in the number. Next, I created two special cases for the integer (if it is less than or equal to one zero). Next, I created a while loop to iterate through until the integer was not zero. I used a bne instruction for this. Inside the loop, I divided the number and stored the quotient and remainder into registers. This is done by the div instruction itself. Next, I added the quotient back to the register to be divided by ten again, and then I increased the digit counter, added a word to the stack, then finally stored the remainder onto the stack. Basically, this just added the numbers onto the stack so that they can be popped out in reverse order. Next, outside of the loop, I created another loop that will run until the number of digits is not equal to zero using another bne. I looped through each number to write it out, and after each number, I deleted a word off the stack and stored the next one to write it also. After each time, I decreased the digit count. Each of the label cases goes through and compares the number until it finds the correct label. It loops through until the digit case it met, then it jumps to the exit.

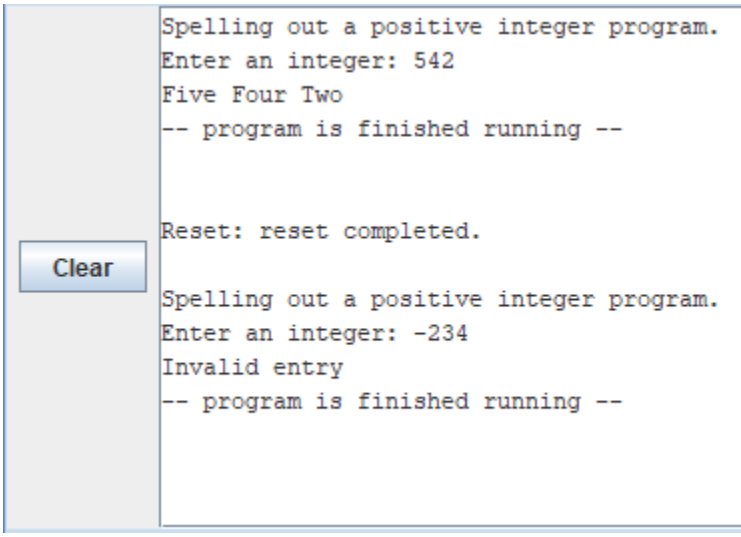
Working Code Screen Print:

<u>Transcription:</u>

Spelling out a positive integer program.
Enter an integer: 542
Five Four Two
-- program is finished running --

Reset: reset completed.

Spelling out a positive integer program.
Enter an integer: -234
Invalid entry
-- program is finished running --



Conclusion:

To conclude, I learned quiet a few things during this lab. I struggled initially with the algorithm a lot, but after doing some research I learned how to properly use the stack and loading and storing words in MIPS. We had talked about it previously in class; however, this was the first real usage of it. I knew that the way to do this would be to divide by 10 and each remainder reversed would be the way to do it, however the actual implementation was difficult. I did not learn any new instructions this time, however. I was having trouble getting it to work with an entry of 0 or multiple 0s, so I just created cases for that before the loops to take care of these situations. Overall, this lab was tough, a much larger jump in difficulty than any of the other labs thus far. I did enjoy doing this and coding in MIPS is still enjoyable for me.