Course Name: Computer Architecture Lab
Course Number and Section: **14:332:333:02**

Experiment: Lab 3 – Arithmetic Operations (Basic and Float) and Combinatorial Logic

Lab Instructor: Bangtian Liu

Date Performed: 2/21/18

Date Submitted: 3/7/18

Submitted by: Yuqing Feng (yf184)

Purpose

In this lab experiment, the focus was on bit operations and floating point number manipulation. Logical operations using not, and, and or can be done on the bits of numbers. The main focus was on floating point numbers. After learning about the floating point format, we know that floating point numbers are not exact when stored, which is why doubles are more accurate as they have more bytes to get more exact numbers. Floating point numbers use different registers, different reading and printing system calls, and different commands to perform operations on them. Operations with floating point numbers, from mathematical problems to array sorting, were explored in this lab and in the assignments.

Approach

Reading and Printing Floating Pointer Numbers

For reading and printing, “syscall” is used. For syscall to know what to do, $v0 must be loaded with a code. To read a float, $v0 must be set to 6 and the value inputted is located in $f0, which must be moved before another syscall is called. To print a float, the value must be loaded into $f12 and $v0 code should be 2.

$v0 code	What it does	Arguments needed
1	Print int	Value in $a0
2	Print float	Value in $f12
3	Print double	Value in $f12
4	Print string	Value in $a0
5	Read int	-
6	Read float	-
7	Read double	-
8	Read string	Addr. to store $a0 No. of char $a1
9	Memory allocation	Bytes storage $a0
10	Exit program	
11	Print character	Integer in $a0
12	Read character	

Bit Operations

1. A or B – each bit of A is ored with the corresponding B and if there is at least one 1 in the pair, the output bit is 1
2. A and B – both bits must be 1 for the output bit to be 1
3. not A – inverts every bit of A, so if one bit is 0, it will be 1 in the output, and vice versa

Range Checking

1. Integers:
    a. blt A, B– branches if A is less than B → checks ranges of user inputs
2. Floating Point:
    a. c.lt.s $f1, $f4
    b. bc1t inputError

c. nop

c.lt.s compares $f1 with $f4 and sets the conditional bit if $f1 < $f4.
bclt L1 branches to L1 if the condition bit is true. This is used to error check user inputs.
nop is to give bclt time to happen, does nothing

Operations (Float)

The format for operations are the same, but the commands are different, mostly with a .s appended. The ones I used in these assignments are:

1. mul.s
2. div.s
3. mov.s
4. li.s
5. sub.s

Loading and Saving (Array)

Declaring the array of floats in data was done in the format “array: .float #”.

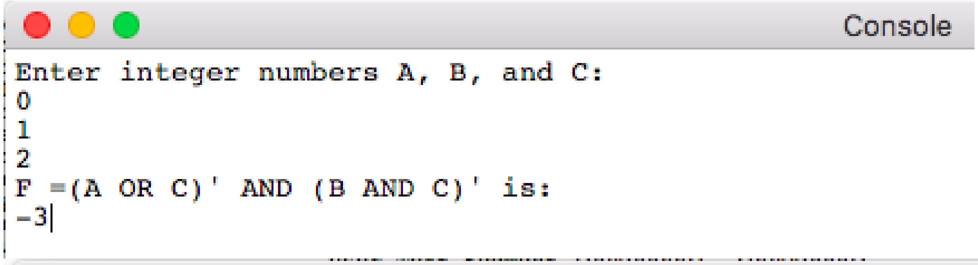
1. Loading
    a. lwc1 $f1, 0($s1) → loads the value at $s1 in the array into $f1.
2. Saving
    a. swc1 $f1, 0($s2) → saves $f1 value into $s2 location in array
    b. used for swapping and putting new value into the array index

Results (and screenshots of program results)

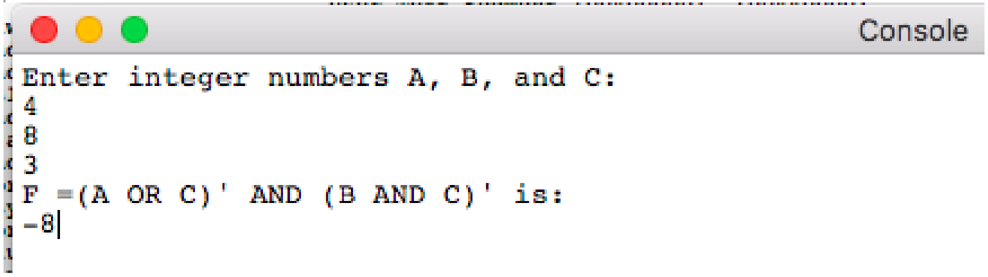
Assignment 1

The program reads three integers and calculates F = (A OR C)’ AND (B AND C)’. My program reads in three integers: A, B, and C. I then OR A and C and store it in A. I invert it with NOT, so $t0 stores (A OR C)’. I then AND B and C and then invert it with NOT. $t1 now stores (B AND C)’. Finally, I AND $t0 and $t1 values together to get the value of F.

The results from the code are below:



```
Enter integer numbers A, B, and C:
0
1
2
F =(A OR C)' AND (B AND C)' is:
-3
```



```
Enter integer numbers A, B, and C:
4
8
3
F =(A OR C)' AND (B AND C)' is:
-8
```

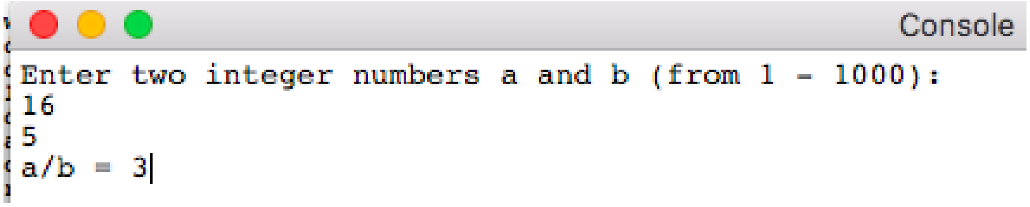
MIPS Code:

```
#Yuqing Feng
#170006296 yf184

.data
  inputStr: .asciiz "Enter integer numbers A, B, and C:\n"
  outputStr: .asciiz "F =(A OR C)' AND (B AND C)' is: \n"

.text
main:

  li $v0, 4
  la $a0, inputStr
  syscall

  li $v0, 5
  syscall
  move $t0, $v0 # a

  li $v0, 5
  syscall
  move $t1, $v0  # b

  li $v0, 5
  syscall
  move $t2, $v0  # c

  or $t0, $t0, $t2  # a = a or c
  not $t0, $t0     # a = a'

  and $t1, $t1, $t2 # b = b and c
  not $t1, $t1     # b = b'

  and $t1, $t0, $t1   # $t1 = a and b

  li $v0, 4
  la $a0, outputStr
  syscall

  li $v0, 1
  move $a0, $t1
  syscall

  li $v0, 10
  syscall
```

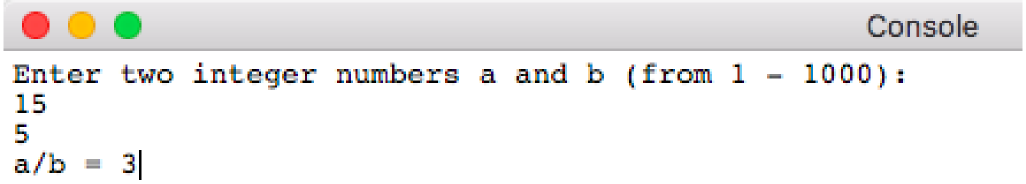
Assignment 2

This program asks for integers a and b between 1 and 1000 and calculates a/b. I first checked to make sure they were within the range and ended the program if the inputs were wrong. I used sub instructions to divide the two numbers.

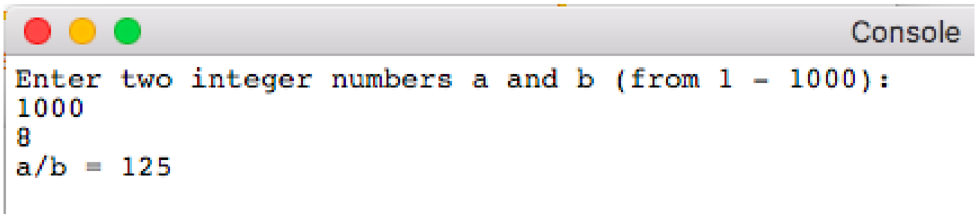
The results from the code is below. As you can see, the integer divisions are done correctly and the code rejects inputs outside the given range (1 – 1000 inclusive).



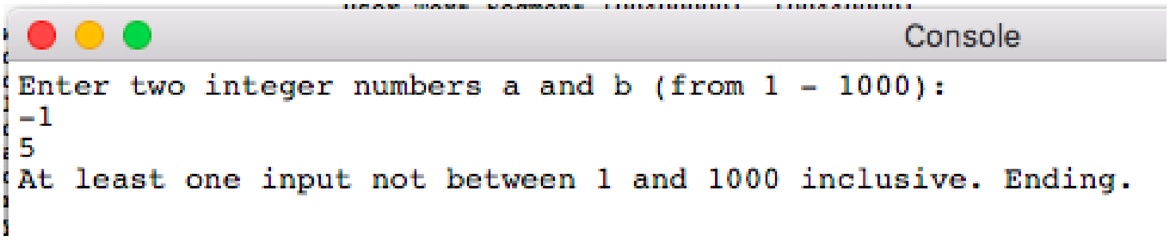
```
Enter two integer numbers a and b (from 1 - 1000):
16
5
a/b = 3
```



```
Enter two integer numbers a and b (from 1 - 1000):
15
5
a/b = 3
```



```
Enter two integer numbers a and b (from 1 - 1000):
1000
8
a/b = 125
```



```
Enter two integer numbers a and b (from 1 - 1000):
-1
5
At least one input not between 1 and 1000 inclusive. Ending.
```

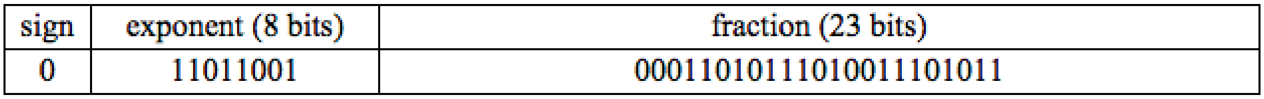
MIPS Code:

```
#Yuqing Feng
#170006296 yf184

.data
  inputStr: .asciiz "Enter two integer numbers a and b (from 1 - 1000):\n"
  outputStr: .asciiz "a/b = "
  badStr: .asciiz "At least one input not between 1 and 1000 inclusive. Ending."

.text
main:

  li $v0, 4
  la $a0, inputStr
  syscall

  li $v0, 5
  syscall
  move $t0, $v0 # a

  li $v0, 5
  syscall
  move $t1, $v0  # b

  #check ranges and make sure within the range
  li $t2, 1
  bltu $t0, $t2, badInput
  bltu $t1, $t2, badInput
  li $t2, 1000
  bltu $t2, $t0, badInput
  bltu $t2, $t1, badInput

  #divide using subtraction
  li $t2, 0 #number of divisions
  j loop

loop:
  blt $t0, $t1, endLoop #if a less than b, endLoop
  sub $t0, $t0, $t1
  addi $t2, $t2, 1
  j loop
endLoop:

  li $v0, 4
  la $a0, outputStr
  syscall

  li $v0, 1
  move $a0, $t2
  syscall

  li $v0, 10
  syscall
```

```
badInput:
  li $v0, 4
  la $a0, badStr
  syscall

  li $v0, 10
  syscall
```

Assignment 3

A. Calculate the decimal number from this:



sign	exponent (8 bits)	fraction (23 bits)
0	11011001	00011010111010011101011

The sign 0 means the number is positive. The exponent is (0d127) 01111111 + power = 11011001. The power number is equal to 90, so the floating point expression is 1.00011010111010011101011 * 2 ^ 90 = **1.2381*10^27**.

B. Turn 13.438 into binary and make the table for floating point below:

Sign	Exponent (8 bits)	Fraction (23 bits)
0	10000010	10101110000001000001100

-13.428 in binary is 1101.01110000001000001100
-in exponential form: 1.10101110000001000001100 * 2^3
-the exponent section is 127+3 = 10000010 in binary.
-The fraction is everything after the decimal point (23 bits)

C. Turn 13.438 into binary and make the table for double point below:

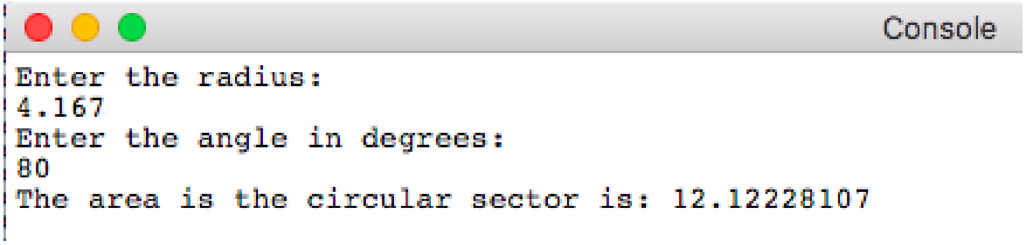
Sign	Exponent (11 bits)	Fraction (52 bits)
0	00010000010	1010111000000100000110001001001101110100101111000110

-13.428 in binary is 1101.0111000000100000110001001001101110100101111000110101
-in exponential form: 1.1010111000000100000110001001001101110100101111000110 * 2^3
-the exponent section is 127+3 = 00010000010 in binary.
-The fraction is everything after the decimal point (52 bits)

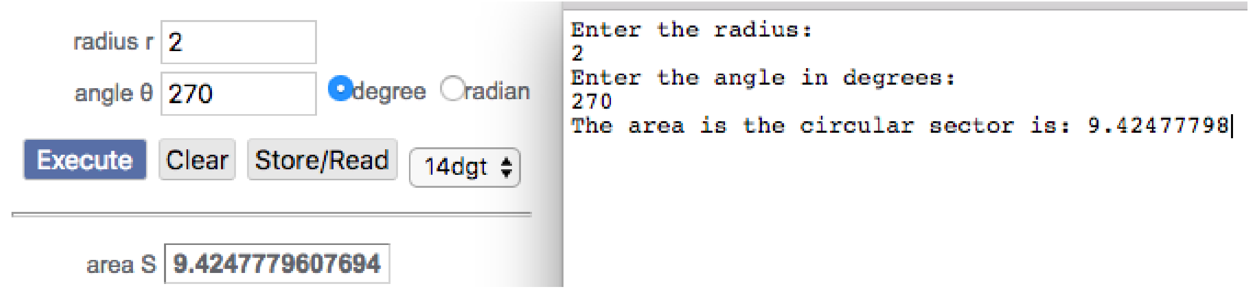
Assignment 4

This program calculates the area of a circular sector using the equation A = pi*r^2*angle/360. The inputs and outputs are floating point numbers. The radius and angle were put into floating point registers and the constants pi and 360.0 were loaded into registers. Using divisions and multiplications but for floating point numbers, the formula was calculated and outputted to the user.

The results from the program are below: (the results were checked for accuracy with an online calculator)



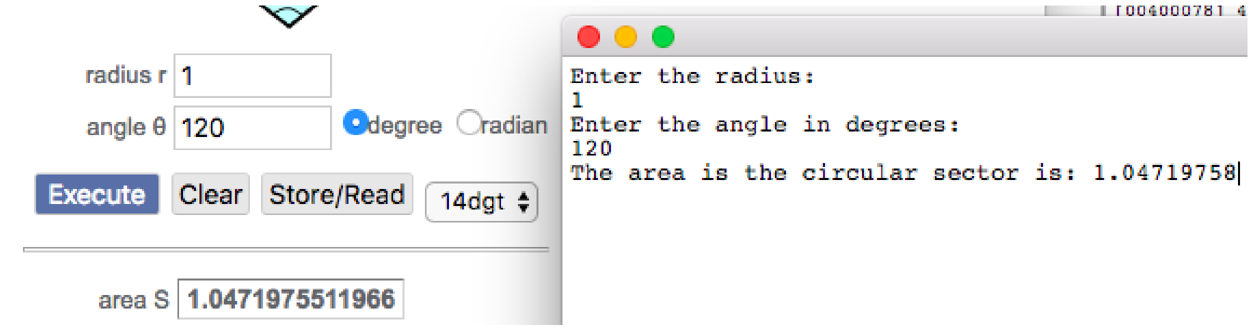
```
Enter the radius:
4.167
Enter the angle in degrees:
80
The area is the circular sector is: 12.12228107
```


radius r: 2

angle θ: 270 ● degree ○ radian

area S: 9.4247779607694

```
Enter the radius:
2
Enter the angle in degrees:
270
The area is the circular sector is: 9.42477798
```


radius r: 1

angle θ: 120 ● degree ○ radian

area S: 1.0471975511966

```
Enter the radius:
1
Enter the angle in degrees:
120
The area is the circular sector is: 1.04719758
```

<u>MIPS Code:</u>

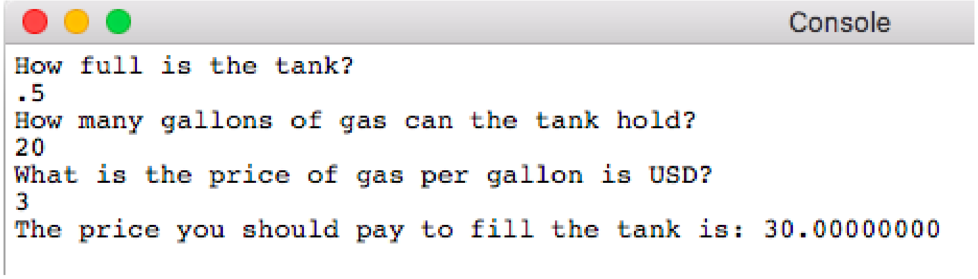
```
#Yuqing Feng
#170006296 yf184

.data
  inputRadStr: .asciiz "Enter the radius:\n"
  inputAngStr: .asciiz "Enter the angle in degrees:\n"
  outputStr: .asciiz "The area is the circular sector is: "

.text
main:

  li $v0, 4
  la $a0, inputRadStr
  syscall

  li $v0, 6
  syscall
  mov.s $f1, $f0  #f1 = radius
```

```
li $v0, 4
la $a0, inputAngStr
syscall

li $v0, 6
syscall
mov.s $f2, $f0  #f2 = angle

li.s $f3, 360.0  #f3 = 360
li.s $f4, 3.14159265359  #f4 = pi

div.s $f2, $f2, $f3   # angle = angle/360
mul.s $f1, $f1, $f1   # r = r^2
mul.s $f1, $f1, $f4   # r = r^2 * pi
mul.s $f12, $f1, $f2   # f12 = pi*r^2 * angle/360

li $v0, 4
la $a0, outputStr
syscall

li $v0, 2
syscall

li $v0, 10
syscall
```

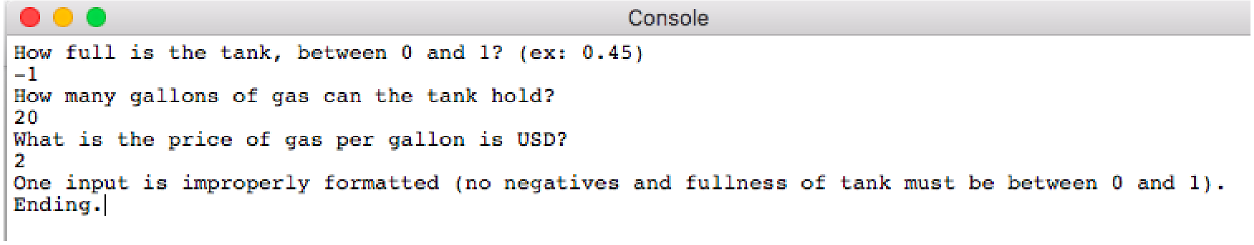
<u>Assignment 5</u>

The program calculates the cost to fill up the gas in a car. To get the amount of money to fill up the tank, I subtracted the fullness of the tank from 1 to get the proportions of an empty tank. I multiplied that with the total gallons to get the gallons needed to fill up the tank. To get the amount of money to fill up, the gallons needed * price per gallon = price to fill up. The program also error checked to make sure the fullness of the tank is a number between 0 and 1 (because fraction of the tank that is full) and is nonnegative. I also made sure the gallons of gas and price of the gas are nonnegative, and if any inputs are in the wrong format, the program ends (shown below).

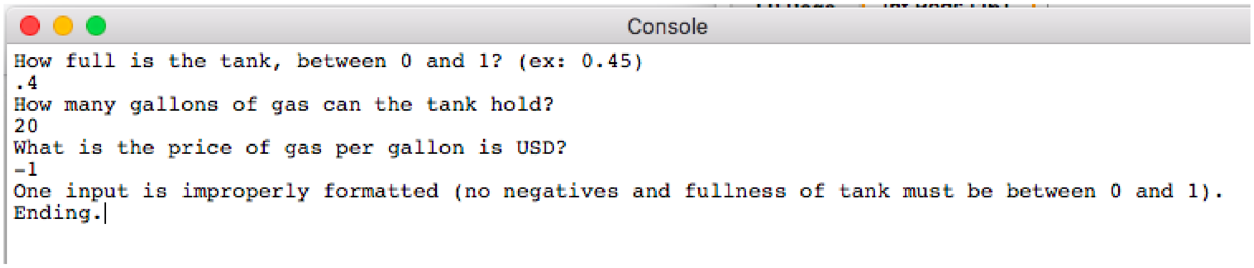
The results from the program is below:



```
How full is the tank?
.5
How many gallons of gas can the tank hold?
20
What is the price of gas per gallon is USD?
3
The price you should pay to fill the tank is: 30.00000000
```



```
How full is the tank, between 0 and 1? (ex: 0.45)
-1
How many gallons of gas can the tank hold?
20
What is the price of gas per gallon is USD?
2
One input is improperly formatted (no negatives and fullness of tank must be between 0 and 1).
Ending.
```



```
How full is the tank, between 0 and 1? (ex: 0.45)
.4
How many gallons of gas can the tank hold?
20
What is the price of gas per gallon is USD?
-1
One input is improperly formatted (no negatives and fullness of tank must be between 0 and 1).
Ending.
```

MIPS Code:

```
#Yuqing Feng
#170006296 yf184

.data
  inputFullStr: .asciiz "How full is the tank, between 0 and 1? (ex: 0.45)\n"
  inputGalStr: .asciiz "How many gallons of gas can the tank hold? \n"
  inputPriceStr: .asciiz "What is the price of gas per gallon is USD? \n"
  outputStr: .asciiz "The price you should pay to fill the tank is: "
  errorStr: .asciiz "At least one input is improperly formatted. Ending."

.text
main:

  li $v0, 4
  la $a0, inputFullStr
  syscall

  li $v0, 6
  syscall
  mov.s $f1, $f0  #f1 = how full tank is

  li $v0, 4
  la $a0, inputGalStr
  syscall

  li $v0, 6
  syscall
```

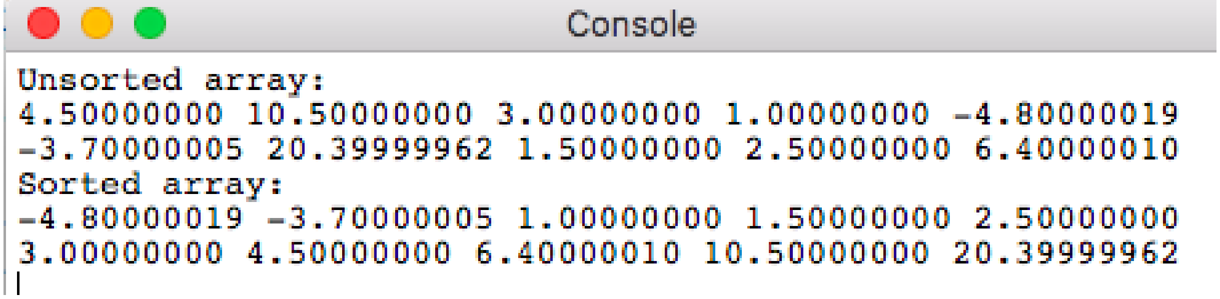
```
 mov.s $f2, $f0  #f2 = total gal tank can hold

 li $v0, 4
 la $a0, inputPriceStr
 syscall

 li $v0, 6
 syscall
 mov.s $f3, $f0  #f3 = price per gallon

 #error checking --> referenced https://chortle.ccsu.edu/AssemblyTutorial/Chapter-32/ass32_3.html
 li.s $f4, 1.0
 c.lt.s $f4, $f1
 bc1t inputError #if fullness of tank greater than 1, bad
 nop

 li.s $f4, 0.0
 c.lt.s $f1, $f4
 bc1t inputError #if fullness of tank less than 0, bad
 nop

 c.lt.s $f2, $f4
 bc1t inputError #is total gas tank is less than 0, bad
 nop

 c.lt.s $f3, $f4
 bc1t inputError #if price is less than or equal to 0, bad
 nop

 #calculations
 li.s $f4, 1.0
 sub.s $f1, $f4, $f1   #f1 = 1-f1 --> how empty tank is

 mul.s $f1, $f1, $f2   #how empty tank is * total gal = gallons needed
 mul.s $f12, $f1, $f3  #gal needed * price per gallon = total price

 li $v0, 4
 la $a0, outputStr
 syscall

 li $v0, 2
 syscall

 li $v0, 10
 syscall
```

```
inputError:
  li $v0, 4
  la $a0, errorStr
  syscall

  li $v0, 10
  syscall
```

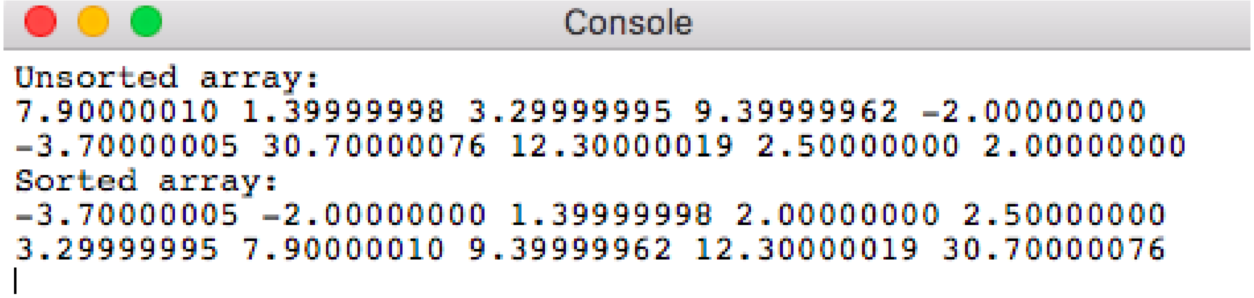
<u>Assignment 6</u>

This program declares an array of 10 floating point elements in the data section and then sorts it in the code. Bubblesort is used, and I used https://www.javatpoint.com/bubble-sort-in-java as a reference for how to bubblesort because I never learned it in a class before. I print the old array and then jump to the sorting loop before printing again. I set up all my registers before going into the sections (sorting and printing) to make sure the values I'm expecting are in there. I bubble sort with two loops and I swap values if they are less than the next array index's values. I keep sorting until no more swaps are made, and I check this with a swap counter in the inner loop.

The results for two arrays I tested this on are below:



```
Unsorted array:
4.50000000 10.50000000 3.00000000 1.00000000 -4.80000019
-3.70000005 20.39999962 1.50000000 2.50000000 6.40000010
Sorted array:
-4.80000019 -3.70000005 1.00000000 1.50000000 2.50000000
3.00000000 4.50000000 6.40000010 10.50000000 20.39999962
```



```
Unsorted array:
7.90000010 1.39999998 3.29999995 9.39999962 -2.00000000
-3.70000005 30.70000076 12.30000019 2.50000000 2.00000000
Sorted array:
-3.70000005 -2.00000000 1.39999998 2.00000000 2.50000000
3.29999995 7.90000010 9.39999962 12.30000019 30.70000076
```

<u>MIPS Code:</u>

```
#Yuqing Feng
#170006296 yf184
#Bubble sort floating point array

.data
  unsortedArrayStr: .asciiz "Unsorted array:\n"
  sortedArrayStr: .asciiz "Sorted array:\n"
  spaceStr: .asciiz " "
  newLineStr: .asciiz "\n"
  swapStr: .asciiz " swapped with "

  #10 elements
  #array: .float 4.5, 10.5, 3.0, 1.0, -4.8, -3.7, 20.4, 1.5, 2.5, 6.4
  array: .float 7.9, 1.4, 3.3, 9.4, -2.0, -3.7, 30.7, 12.3, 2.5, 2.0

.text
main:

  li $v0, 4
  la $a0, unsortedArrayStr
  syscall

  #set up printing array
  li $t0, 0     #counter i
  li.s $f1, 0.0    #stores value of item
  li $t2, 40    #stores only 10 items (8*10)
  la $s0, array   #base address
  la $s1, array   #incrementing address

  jal printArray

  #set up sorting array
  li $t0, 0         #counter i
  li $t1, 0         #counter j
  li.s $f1, 0.0     #float for arr[j]
  li.s $f2, 0.0     #float for arr[j+1]
  li $t2, 1         #number of swaps
  li $t3, 36        #n-1
  li $t4, 0         #n-1-i for j
  la $s0, array     #base address
  la $s1, array     #incrementing address
  la $s2, array     #will be always $s1 + 4 --> arr[j+1]

  jal sortOuter

  li $v0, 4
  la $a0, sortedArrayStr
  syscall

  #set up printing array
  li $t0, 0     #counter i
  li.s $f1, 0.0     #stores value of item
  li $t2, 40    #stores only 10 items (8*10)
  la $s0, array   #base address
  la $s1, array   #incrementing address

  jal printArray

  li $v0, 10
  syscall

sortOuter:

  beqz $t2, endOuter  #end when num swaps = 0

  #reinitialize
  li $t1, 0   #j = 0
  li $t2, 0   #num swaps = 0
  sub $t4, $t3, $t0       #t1 = $t3 - $t0 --> n-1-i
  la $s1, array     #incrementing address for j
  la $s2, array     #will be always $s1 + 4 --> arr[j+1]

  sortInner:
    beq $t1, $t4, endInner  #if j = n-1-i, end inner loop

    lwc1 $f1, 0($s1)        #arr[j]
    addi $s2, $s1, 4        #j+1 of array
    lwc1 $f2, 0($s2)        #arr[j+1]

    c.le.s $f1, $f2
    bc1t continueInner           #if arr[j] <= arr[j+1], skip swap
    nop

    #here, arr[j] > arr[j+1] --> do swap
    swc1 $f2, 0($s1)      #arr[j] = arr[j+1] value
    swc1 $f1, 0($s2)      #arr[j+1] = arr[j] value
    addi $t2, $t2, 1      #increment number of swaps

    continueInner:
    addi $t1, $t1, 4  #j++
    add $s1, $s0, $t1
    j sortInner
  endInner:

  addi $t0, $t0, 4  #i++
  j sortOuter

endOuter:
  jr $ra

printArray:
  #load with floating point -> reference: http://www.ece.lsu.edu/ee4720/2014/lfp.s.html

  lwc1 $f1, 0($s1)
  beq $t0, $t2, endPrint #counter

  mov.s $f12, $f1
  li $v0, 2
  syscall

  li $v0, 4
  la $a0, spaceStr
  syscall

  addi $t0, $t0, 4  #4 because float = 4 bytes
  add $s1, $s0, $t0
  j printArray
endPrint:
  li $v0, 4
  la $a0, newLineStr
  syscall
  jr $ra
```

Conclusion

Assignment 1 introduced me to bit logic and operations by using and, or, and not to get the result of F. Assignment 2 forces me to divide without using div, so I used subtractions for the two integers to divide them. Assignment 3 taught me how to convert from floating point format and to convert to floating and double format. Assignment 4 and 5 are applications of using floating point numbers and operations on them for common math and real-life problems. Finally, Assignment 6 puts my knowledge on sorting, bubble sort, and floating point number arrays together to bubble sort floating point numbers. Overall, the same operations for integers apply for floating point numbers but with a ".s" at the end of commands, different registers for use, and different ways to load and save floating point numbers.