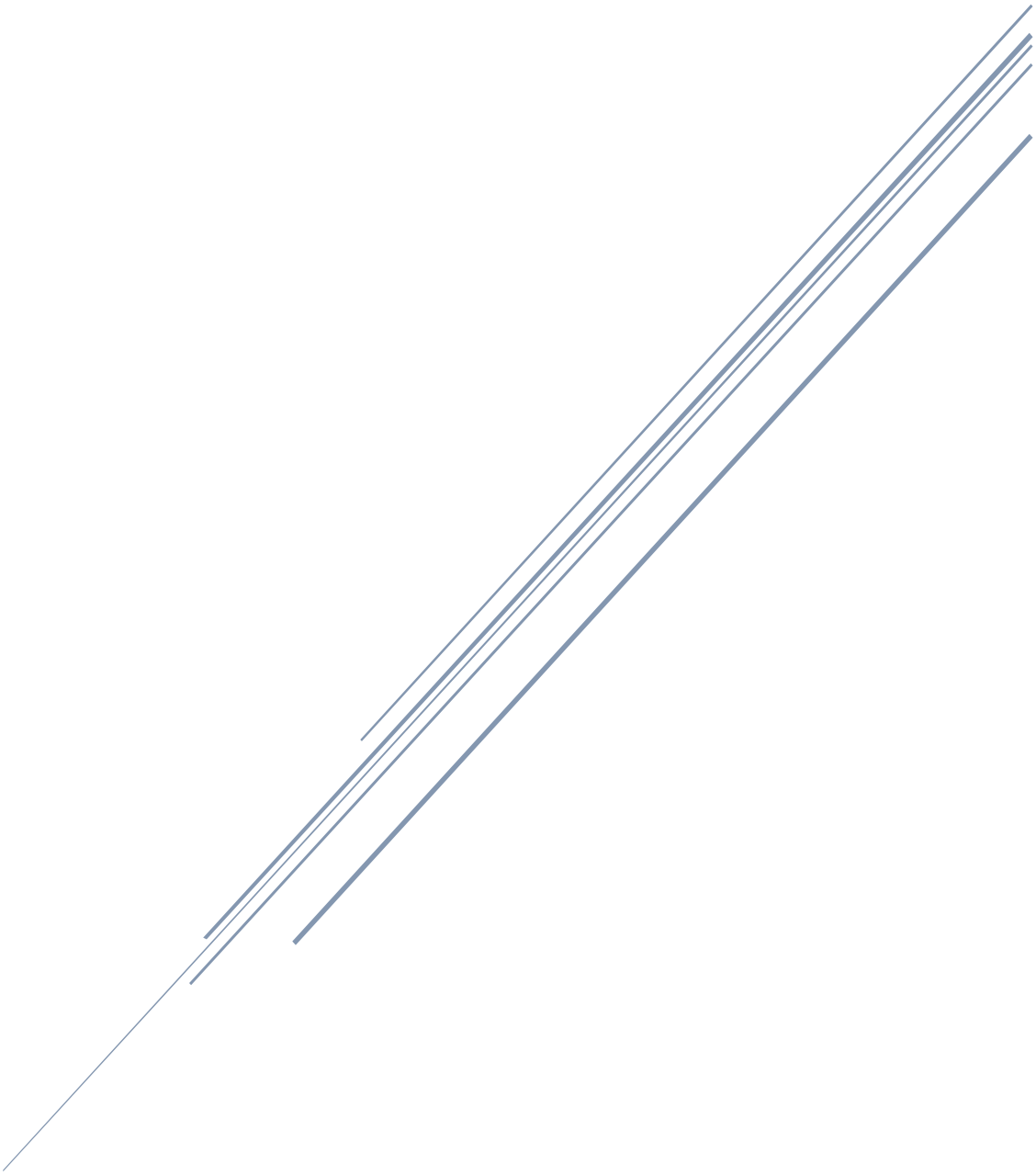
CSF LOGIC ASSIGNMENT

Mars MIPS Assembly



Pritam Sangani
16039231

Task A

Code

```
#TASK A
.data    #tells assembler that we're in data segment
        #variable declarations
        enterName: .asciiz "Enter your name: "
        printName: .asciiz "\nYour name is "
        userInput: .space 20   #clear 20 bytes in memory for userInput
        enterID: .asciiz "\nEnter the last four digits of your student ID number: "
        printID: .asciiz "\nThe last four digits of your student ID number are "
.text    #tells assembler that we're in text segment
#displayName
        #print out enterName string
        la $a0 enterName      #load enterName into $a0 register
        addi $v0 $zero 4      #load service number 4 (for printing a string) into $v0 register
        syscall               #call to system
        #allow user to input their name
        la $a0 userInput
        addi $a1 $zero 20     #reserve 20 bytes of memory in $a1 register
        addi $v0 $zero 8      #load service number 8 (for reading a string) into $v0 register
        syscall
#displayID
        #print out enterID string
        la $a0 enterID
        addi $v0 $zero 4
        syscall
        #read integer that user enters
        addi $v0 $zero 5     #load service number 5 (for reading an integer) into $v0 register
        syscall
```

```
#move integer stored in $v0 register to $t0 register
addu $t0 $zero $v0  #move into $t0 so data doesn't get overwritten
#print out printName string
la $a0 printName
addi $v0 $zero 4
syscall
#display name that user has entered
la $a0 userInput
addi $v0 $zero 4
syscall
#print out printID string
la $a0 printID
addi $v0 $zero 4
syscall
#print out integer that was entered by user
addi $v0 $zero 1        #load service number 1 (to print an integer) into $v0 register
addu $a0 $zero $t0      #move integer stored in $t0 register to $a0 register
syscall
```

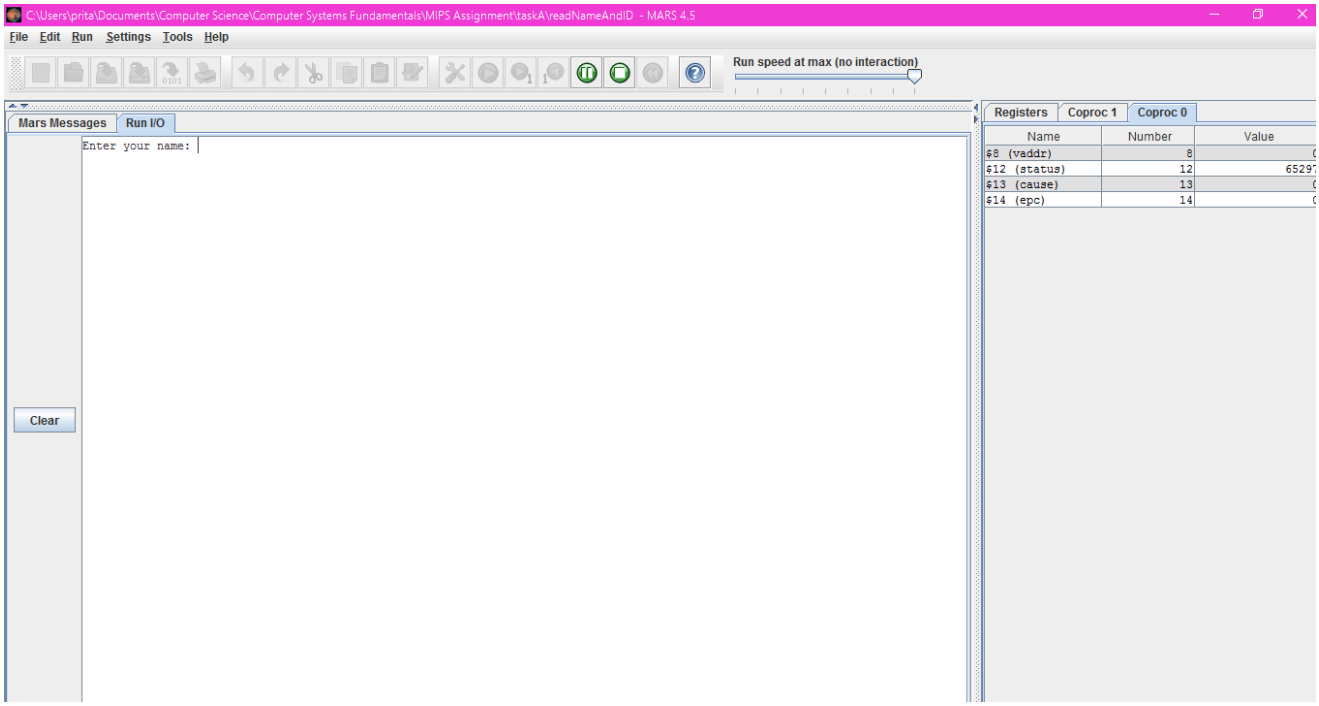
How the Program Works

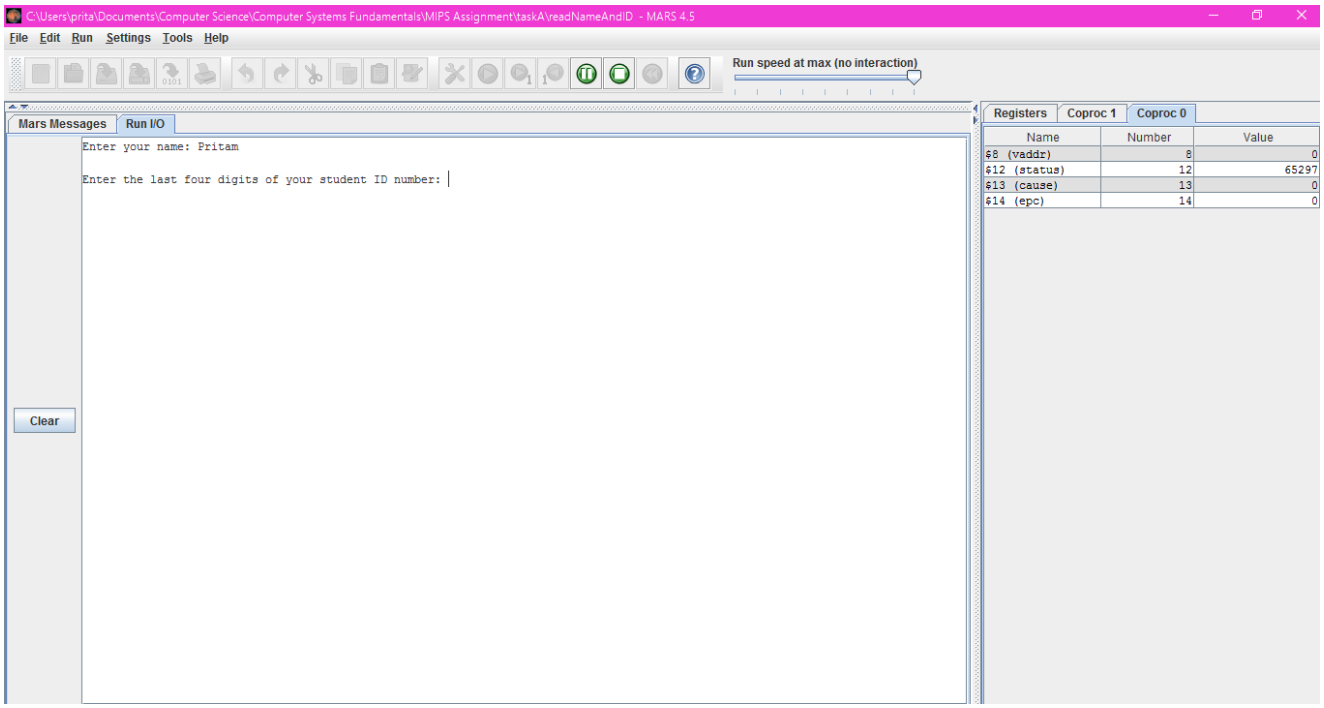
I declared variables for the prompts I was going to use in the .data segment. The instructions went in the .text segment.

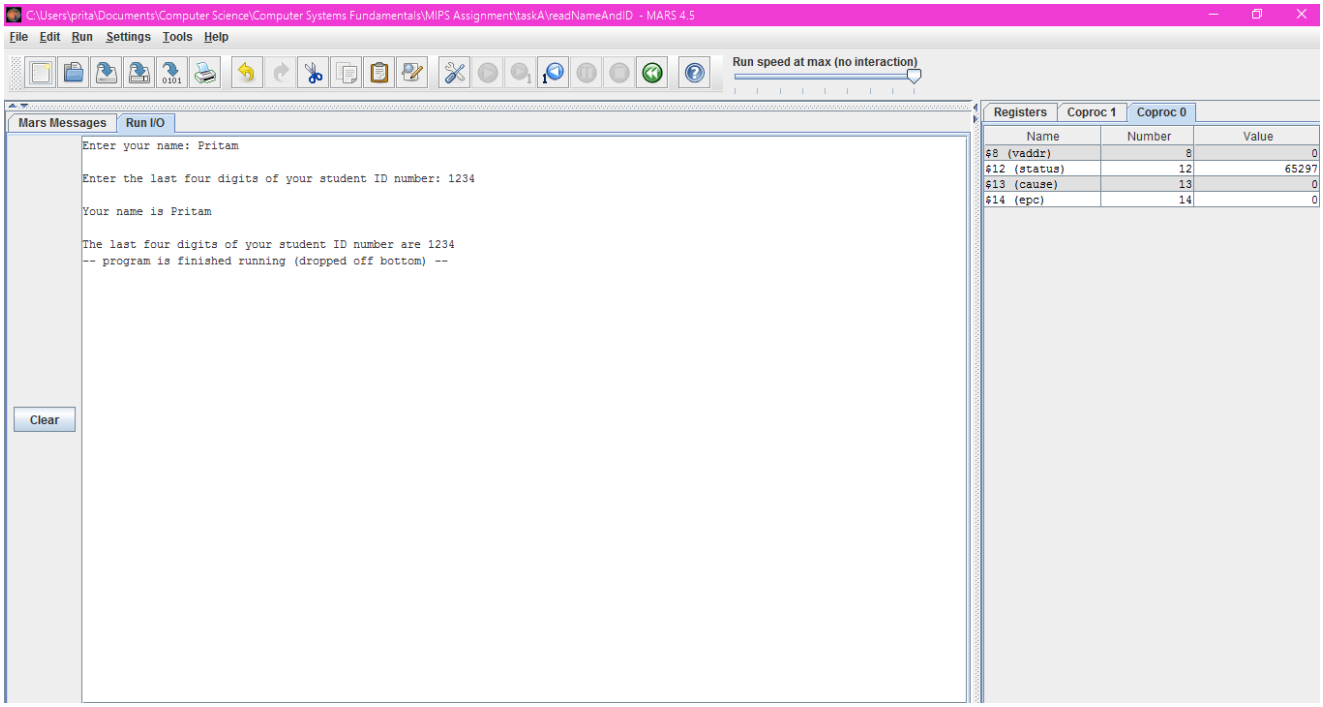
To display the name, I firstly loaded the enterName variable into $a0 and loaded service number 4 into $v0 to tell the system to print a string. I then reserved 20 bytes in $a1 to allow you to enter your name. I used service number 8 to read a string.

To display the ID, I used service number 5 to read an integer. To make sure the data was not overwritten I moved it from $v0 to $t0. I used service number 1 to print the integer and moved the data in $t0 to $a0.

Screenshots of input and output stage







Task B

Code

```
#TASK B
.data	#tells assembler that we're in data segment
	#variable declarations
	enterID: .asciiz "Enter the last three digits of your student ID number: "
	printID: .asciiz "\nThe number you entered multiplied by 2 equals "
.text	#tells assembler that we're in text segment
	#print out enterID string
	la $a0 enterID		#load enterID variable into $a0 register
	addi $v0 $zero 4	#load service number 4 (for printing out a string) into $v0 register
	syscall			#call to system
	#read integer that user enters
	addi $v0 $zero 5	#load service number 5 (for reading an integer) into $v0 register
	syscall
```

```
#move integer stored in $v0 register to $t0 register
addu $t0 $zero $v0
mul $t1 $t0 2          #multiply number in $t0 by 2 and store in $t1
addi $t2 $zero 3       #add 3 to $t2
loop:                  #start of loop
#print out printID string
la $a0 printID
addi $v0 $zero 4
syscall
#print out integer that was entered by user
addi $v0 $zero 1       #load service number 1 (to print an integer) in $v0 register
#move integer stored in $t1 register to $a0 register
addu $a0 $zero $t1
syscall
addi $t2 $t2 -1        #take 1 away from number in $t2 each time it reaches this line
bgez $t2 loop     #if number in $t2 is greater than or equal to zero go back to loop
                  label and rerun code within the loop
```

How the Program Works

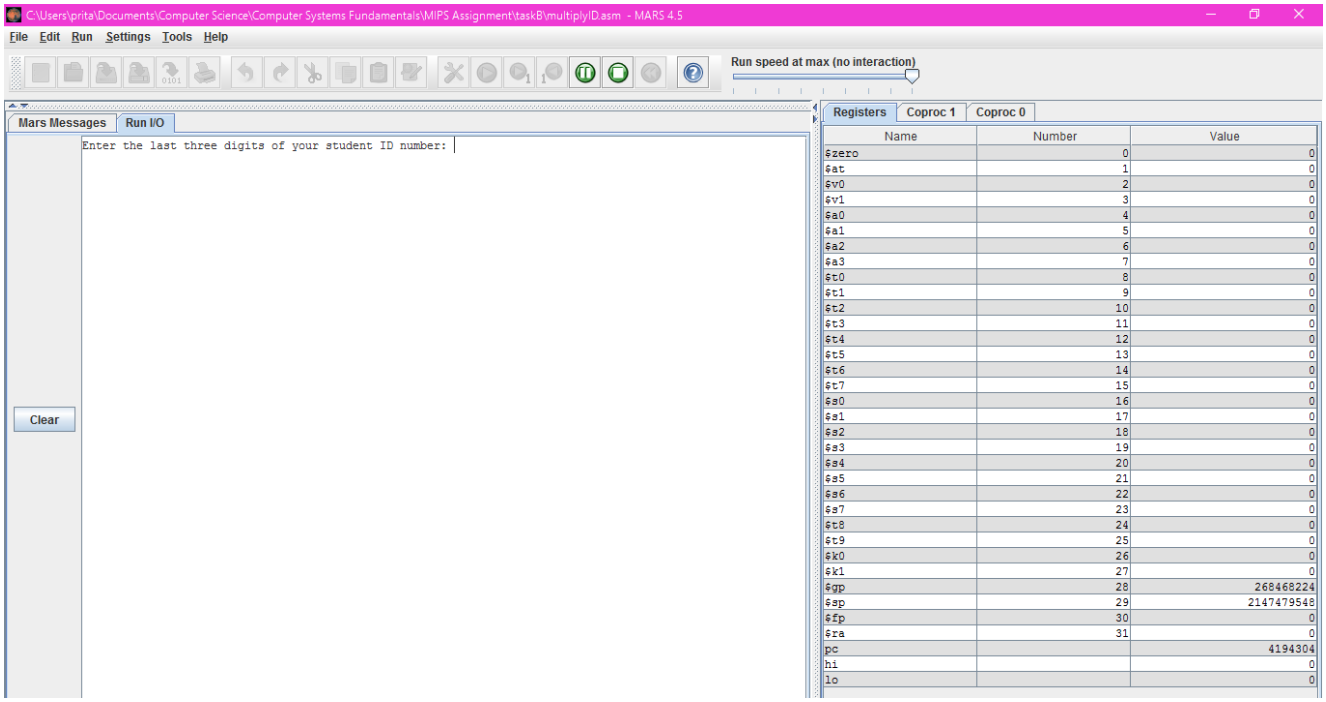
To multiply the number by 2, I used the ‘mul’ instruction and stored the result in $t1.

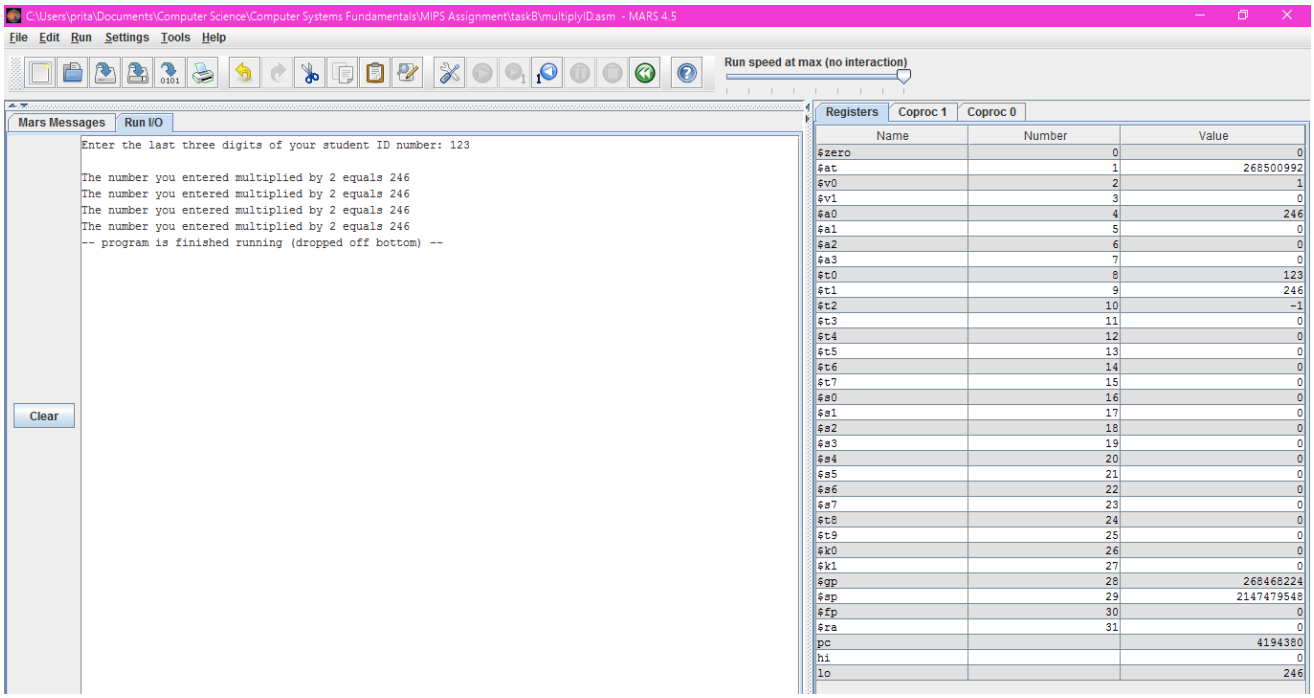
For the loop, I first added 3 to $t2. I put all the instructions for the calculation happening in the loop inside a label which I called ‘loop’.

For the counter, I took one away from the number in $t2 each time the instructions in the loop had finished.

I used the ‘bgez’ instruction to test whether the loop had to occur again. This instruction checks whether the value of $t2 is greater than or equal to zero. If it is, it will branch the program back to the loop label and another loop will occur. Once the counter goes below zero the bgez instruction will tell the system to stop the loop structure.

Screenshots of input and output stage





Task C

Code

```
#TASK C
.data   #tells assembler that we're in data segment
        #variable declarations
        enterNo: .asciiz "\nEnter a three digit number: "
        result: .asciiz "\nThe result of your calculation is: "
        time: .asciiz "\nTotal time of calculation is: "
        ms: .asciiz " ms"
.text   #tells assembler that we're in text segment
        #print out enterNo string
        la $a0 enterNo                  #load enterNo into $a0 register
        addi $v0 $zero 4        #load service number 4 (for printing a string) into $v0
        syscall                         #call for system
        #read integer that user enters
        addi $v0 $zero 5        #load service number 5 (for reading an integer) into $v0
register
        syscall
        move $t0 $v0            #move integer stored in $v0 register to $t0 register
        #get system time for just before loop structure begins
        li $v0 30               #load service number 30 (for system time) into $v0
register
        syscall
        move $t5 $a0            #move time stamp data from $a0 into $t5 register
        #instructions for loop structure
        addi $t1 $zero 50       #add 50 to $t1 (counter for loopOuter)
        loopOuter:
                addi $t3 $zero 100              #add 100 to $t3 (counter for loopInner1)
                loopInner1:
                        addi $t4 $zero 500      #add 50 to $t4 (counter for loopInner2)
```

```
                loopInner2:
                        addi $t4 $t4 -1          #take 1 away from $t4 each time loopInner2 occurs
                        div $t2 $t0 7            #divide number stored in $t0 (entered by user) by 7 and store answer in $t2
                        bgez $t4 loopInner2      #if number in $t4 is greater than or equal to zero go back to loopInner2 label and rerun code within the loop
                addi $t3 $t3 -1
                bgez $t3 loopInner1
        addi $t1 $t1 -1
        bgez $t1 loopOuter
        #get system time for just after loop structure has ended
        li $v0 30               #record timestamp for the end of the loop structure
        syscall
        move $t6 $a0
        #get total time for loop structure
        sub $t7 $t6 $t5             #subtract $t5 from $t6 to get total time for the loop structure
        #print out result string
        la $a0 result
        addi $v0 $zero 4
        syscall
        #print out result of calculation (integer)
        addi $v0 $zero 1
        move $a0 $t2
        syscall
        #print out time string
        la $a0 time
        addi $v0 $zero 4
        syscall
        #print out the total time for loop structure
        addi $v0 $zero 1
```

```
move $a0 $t7
syscall
#print out ms string (displayed after total time)
la $a0 ms
addi $v0 $zero 4
syscall
```

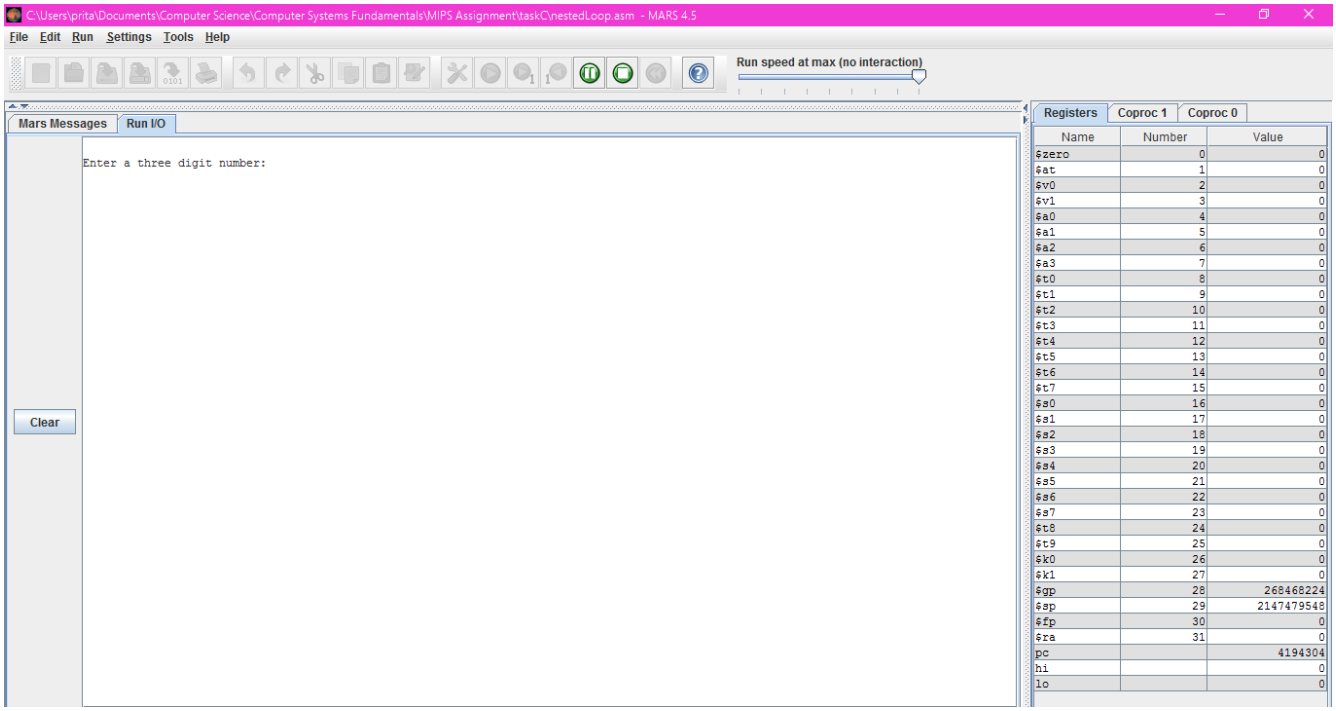
How the Program Works

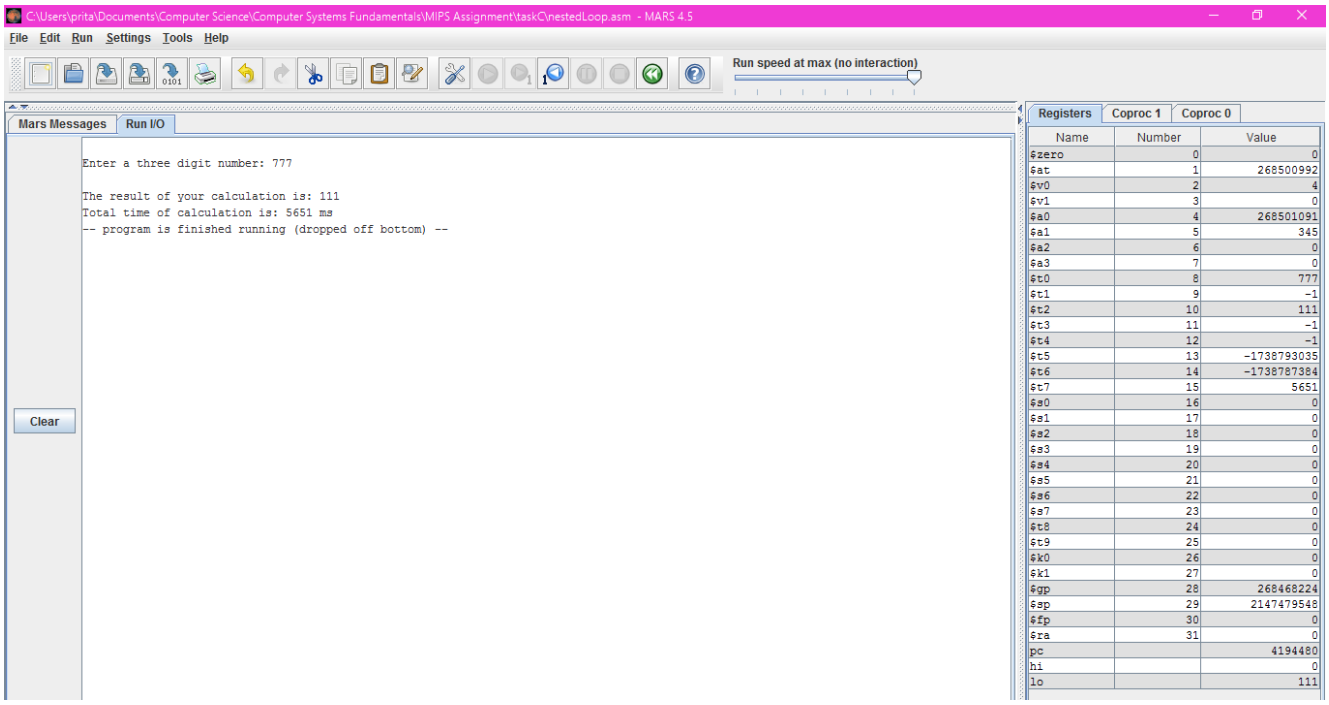
To get the system time, I loaded service number 30 into $v0 just before the loop structure began and after it finished. To get the total time for the loop structure, I took the time recorded before the loop structure had started away from the time recorded after the loop structure had ended. This was done using the ‘sub’ instruction.

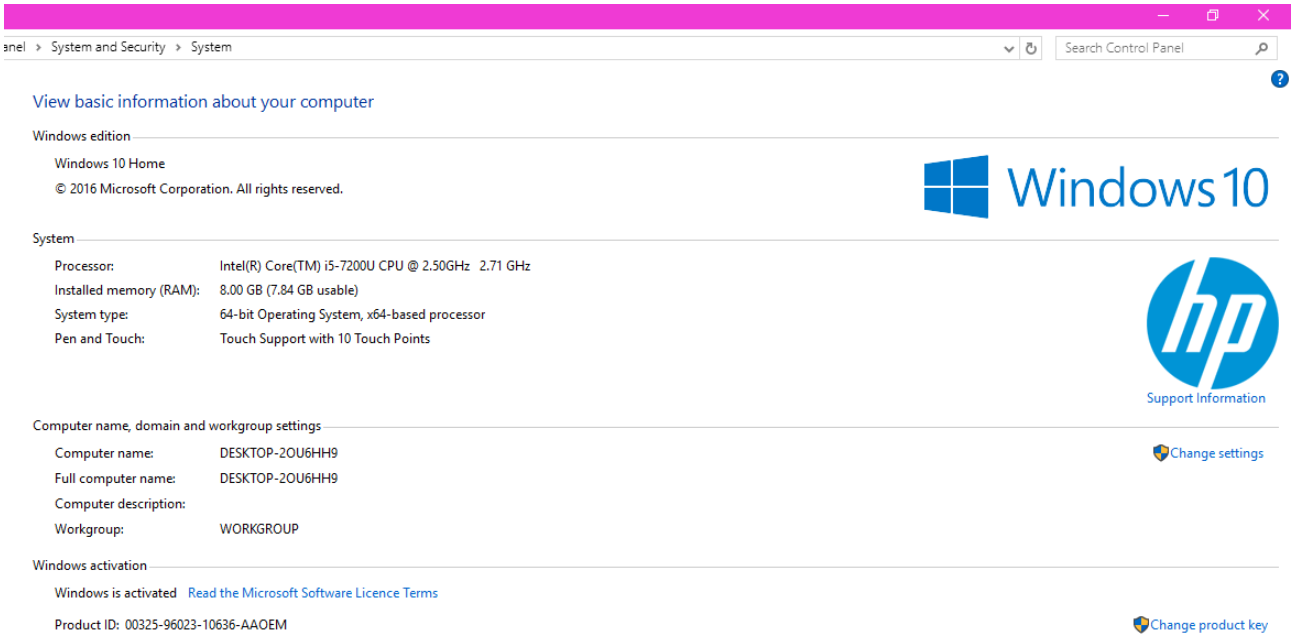
As the system time is outputted in milliseconds, I displayed ‘ms’ after the time to tell the user that the time was in milliseconds.

For the loop structure, I had two inner loops nested inside an outer loop. For the outer loop labelled ‘loopOuter’. I added 50 to $t1, which acted as the counter for the outer loop. Within that loop, I had the first inner loop labelled as ‘loopInner1’. For the counter for this loop, I added 100 to $t3. Within this was the inner-most loop labelled as ‘loopInner2’. For the counter for this loop, I added 500 to $t4. I put the instruction for the calculation inside this loop. I divided the data that was stored in $t0 by 7 and then stored the result in $t2. I then used the ‘bgez’ instruction to check whether the loop counter is greater than or equal to zero. This was done at the end of each loop.

Screenshots of input and output stage






anel › System and Security › System
Search Control Panel
View basic information about your computer
Windows edition
Windows 10 Home
© 2016 Microsoft Corporation. All rights reserved.
Windows 10
System
Processor: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.71 GHz
Installed memory (RAM): 8.00 GB (7.84 GB usable)
System type: 64-bit Operating System, x64-based processor
Pen and Touch: Touch Support with 10 Touch Points
Support Information
Computer name, domain and workgroup settings
Computer name: DESKTOP-2OU6HH9
Full computer name: DESKTOP-2OU6HH9
Computer description:
Workgroup: WORKGROUP
Change settings
Windows activation
Windows is activated Read the Microsoft Software Licence Terms
Product ID: 00325-96023-10636-AAOEM
Change product key

Flow chart for Part C