1. ***This homework must be finished individually.***
2. ***This assignment has been designated by the Department of Computer Science for assessment of certain expected outcomes for its degree programs, as required by our accreditation agencies, the University, and the State of Florida. Departmental policy does not permit a final grade of "C-" or better to be assigned unless the student has at earned a grade of "C-" or better on this assignment, regardless of performance on other work in the course.***

Implementing a Car Simulator in MIPS

Your assignment is to write a MIPS program that will implement a car simulator. This simulator will maintain and display its current state every 10 seconds, update any state that constantly needs to be updated every 50ms, and respond to the user's input after each user command.

The simulator must support the following commands:

- Help
  - Key 'h'
  - Prints out a message showing the available commands
- Accelerate
  - Key 'a'
  - Increases speed by 2 meters per second
- Decelerate
  - Key 'd'
  - Decrease speed by 2 meters per second
- Add Passenger
  - Key 'p'
  - Increase the number of passengers in the car by one
  - Can only do this when fully stopped
- Remove Passenger
  - Key 'r'
  - Decrease the number of passengers in the car by one
  - Can only do this when fully stopped
- Toggle Wipers
  - Key 'w'
  - Toggles wipers between on state and off state
- Increase Brightness
  - Key 'l'
  - Brighten the headlights on the car
- Decrease Brightness
  - Key 'd'
  - Dim the headlights on the car
- Turn Car
  - Key 't'
  - Prepare car for turning
  - Disallows all inputs other than the ones specified below
- Turn Right
  - Key 'r'
  - Turns the car 90 degrees to the right, e.g. north->east
  - Can only do this after using the turn car command

- Turn Left
    - Key ‘l’
    - Turns the car 90 degrees to the left, e.g. north->west
    - Can only do this after using the turn car command
- Cancel Turn
    - Key ‘q’
    - Aborts turn
    - Can only do this after using the turn car command

The simulator needs the following states:

- Speed in meters per second
    - Floating Point
    - Default Value: 0.0
    - Min Value: 0.0
    - Max Value: 100.0
- Total distance traveled in meters
    - Floating Point
    - Default Value: 0.0
    - Min Value: 0.0
- Current direction facing
    - String Value
    - Default Value: north
    - Possible Values: north, south, east, west
- Number of passengers
    - Integer Value
    - Default Value: 0
    - Min Value: 0
    - Max Value: 3
    - The driver isn’t a passenger and doesn’t count towards this
- Wiper State
    - Integer Value
    - Default Value: 0
    - Min Value: 0
    - Max Value: 1
    - 0 = Off, 1 = On
- Headlight Brightness
    - Integer Value
    - Default Value: 0
    - Min Value: 0
    - Max Value: 2
    - 0 = Off, 1 = Dim, 2 = Bright

Error checking is required, including:

- An invalid character is entered.
    - Drop the character and prepare for a new one.
- Making sure values stored are valid:
    - For most values, just make sure that when it goes above the max value or below the min value, it is set to the max value or min value.

    - For string values, check to make sure the string matches one of the possible values. If it doesn't just don't do anything

In such cases, an error message should NOT be displayed and the error should be silently handled.

Other notes:

- Use mapped IO to read the inputs. (Given in the interrupt handler)
- Use interrupt timer instead of busy waiting (Given in the interrupt handler)
- When displaying the current state (every 10 seconds), print each item of the global state
- After typing in a command, display the current state of the proper item
    - E.g. after doing the accelerate command, display the new speed
- When displaying speed (and distance), show the values in the stored meters per second (meters) as well as miles per hour (miles).
    - That is you have to have one or more convert functions (use google unit converter to get the values)

Extra point opportunity (10 extra points):

- In addition to the above keep track of car's fuel. You can use the values you want, but make sure to implement the following:
    - Add a new command to refuel
    - Decrease the fuel efficiency linearly as the number of passengers increases
    - Can only refuel when car is stopped.
    - Keep track of fuel for display
    - When out of fuel, stop the car

Sample run (if you follow this printing convention, remove the typos first):

```
You are driving north
You are currently traveling at 0.00000000 m/s (0.00000000 mph)
You have travled 0.00000000 meters (0.00000000 miles)
The lights are turned off
The wipers are off
There are 0 passengers

A passenger has boarded
A passenger has boarded
The car's speed has increased
The car's speed has increased
The car's speed has increased

Use the following keys to interact with the system:
'h' for this help message
't' to turn
'w' to toggle the wipers
'l' to increase the intensity of the lights
'o' to decrease the intensity of the lights
'a' to incrase the speed of the car
'd' to decrease the speed of the car
```

```
'p' to add passengers when stopped
'r' to remove passengers when stopped

Press 'l' to turn left
Press 'r' to turn right
or Press 'q' to stop turning
You are now driving west
The light intensity has increase
The light intensity has increase

You are driving west
You are currently traveling at 3.00000000 m/s (6.71080875 mph)
You have travled 25.59995842 meters (0.01590708 miles)
The high beams are on
The wipers are off
There are 2 passengers

The light intensity has decrease
The wipers status has been toggled

Press 'l' to turn left
Press 'r' to turn right
or Press 'q' to stop turning
You are now driving south

Press 'l' to turn left
Press 'r' to turn right
or Press 'q' to stop turning
Turn canceled
The car's speed has decreased]nThe car's speed has decreased]n
You are driving south
You are currently traveling at 1.00000000 m/s (2.23693633 mph)
You have travled 52.65011215 meters (0.03271526 miles)
The low beams are on
The wipers are on
There are 2 passengers

The car's speed has decreased]nA passenger has left

You are driving south
You are currently traveling at 0.00000000 m/s (0.00000000 mph)
You have travled 56.95004654 meters (0.03538712 miles)
The low beams are on
The wipers are on
There are 1 passengers


You are driving south
You are currently traveling at 0.00000000 m/s (0.00000000 mph)
You have travled 56.95004654 meters (0.03538712 miles)
The low beams are on
The wipers are on
There are 1 passengers
```

Template code:

```
       .text
       .globl main
main:

init:
       # allow hardware interrupts
       mfc0 $t0, $12       # get status register
       ori $t0, $t0, 0xff01
       mtc0 $t0, $12

       # set up keyboard interupts
       li $t0, 0xFFFF0000  # keyboard memory-mapped address
       li $a0, 2
       sw $a0, 0($t0)

       # set up a timer
       li $a0, 5           # get a timer interrupt every 50 ms
       mtc0 $a0, $11       # set up compare register
       mtc0 $0, $9

       li $s0, 100000      # this will be our input-grabing register
       li $s1, 100000      # this resets the input

       li $s2, 0           # our main timer; set to 1 when fired, resets to 0

       li $s3, 200         # this will be our message timer (10 sec = 200 * 50 ms)
       li $s4, 200         # and its default value

       la $a0, north
       jal change_direction
       jal status

main_loop:
main_loop_inputs:
       beq $s0, $s1, main_loop_status
       jal process_input
       move $s0, $s1

main_loop_status:
       bgt $s3, $0, main_loop_tick
       move $s3, $s4
       jal status

main_loop_tick:
       beq $s2, $0, main_loop_end
       move $s2, $0
       addi $s3, $s3, -1
       jal update

main_loop_end:
       j main_loop

main_loop_exit:            # exit that we won't get to
       li $v0, 10
       syscall


#---------------------------INTERRUPT-HANDLER-----------------------------
       .kdata
ksave0:
```

```
       .word 0
ksave1:
       .word 0

       .ktext 0x80000180
kinterrupt_handler:
       sw $a0, ksave0
       sw $v0, ksave1

       mfc0 $k0, $13
       srl $a0, $k0, 2
       andi $a0, $a0, 0x1F

       mfc0 $k0, $9
       mfc0 $k1, $11
       beq $k0, $k1, ktimer_handler

kkeyboard_handler:
       li $v0, 0xFFFF0000
       lw $s0, 4($v0)              # keyboard input
       j kexit

ktimer_handler:
       li $s2, 1                   # timer fired

       # reset timer
       li $k0, 5                   # get a timer interrupt every 50 ms
       mtc0 $k0, $11               # set up compare register
       mtc0 $0, $9

kexit:
       # reenable interrupts
       mtc0 $0 $13                 # Clear Cause register
       mfc0 $k0 $12                # Set Status register
       ori  $k0 0x1                # Interrupts enabled
       mtc0 $k0 $12

       lw $a0, ksave0
       lw $v0, ksave1

       eret
#---------------------------INTERRUPT-HANDLER-----------------------------
```

Grading:

1. You should document your code properly in this homework. You may use the following code as an example for how to style your comments:

```
# Zhenghao Zhang -- 04/10/09
# newton.asm - a simple program calculating the square root of a given
#       is number n using singles. The value of the square root of n,
#       denoted by x, approximated by x=(x+n/x)/2, until the the
#       absolute value of the difference between new x and the previous
#       x is smaller than a given threshold such as 1.0E-3.



# function main -- written by Zhenghao Zhang -- 04/10/09
#       calling the calsqrt function and display the results
# Register use:
#       $a0     syscall parameter
```

```
#       $v0     syscall parameter
#       $f0     exp and syscall return value
#       $f12    exp and syscall parameter

        .text
        .globl main
main:   li.s $f0, 361.0
        mfc1 $a0, $f0
        jal calsqrt

done:   mtc1 $v0, $f12
        li $v0,2
        syscall

eixt:   li $v0,10
        syscall



# function calsqrt -- written by Zhenghao Zhang -- 04/10/09
#       calculating the square root of a given number n using singles.
#       The value of the square root of n, denoted by x, approximated
#       by x=(x+n/x)/2, until the the absolute value of the difference
#       between new x and the previous x is smaller than a given
#       threshold such as 1.0E-3.
# Register use:
#       $a0     parameter from calling routine
#       $v0     return value
#       $f0     storing the value of n as a single precision number
#       $f1     current value of x
#       $f2     next value of x
#       $f3     tempory vaiable
#       $f20    storing constant 2 for dividing
#       $f21    storing constant 0.001 for exit comparision

calsqrt:
        addi $sp, $sp, -24
        swc1 $f0, 20($sp)
        swc1 $f1, 16($sp)
        swc1 $f2, 12($sp)
        swc1 $f3, 8($sp)
        swc1 $f20, 4($sp)
        swc1 $f21, 0($sp)

        mtc1 $a0, $f0          # $f0 gets n
        li.s $f20, 2.0         # $f20 storing constant 2 for dividing
        li.s $f21, 0.001       # $f21 storing constant 0.001 for exit comparision
        div.s $f1, $f0, $f20   # $f1 gets n/2

calsqrtloop:
        div.s $f2, $f0, $f1    # $f2 gets n/x
        add.s $f2, $f2, $f1    # $f2 gets n/x + x
        div.s $f2, $f2, $f20   # $f2 gets x'=(n/x + x)/2
        sub.s $f3, $f2, $f1    # $f3 gets x'-x
        abs.s $f3, $f3         # $f3 gets |x'-x|
        c.lt.s $f3, $f21       # set the flag if |x'-x| < 0.001
        bc1t calsqrtdone
        mov.s $f1, $f2
        j calsqrtloop

calsqrtdone:
        mfc1 $v0, $f2

        lwc1 $f0, 20($sp)
        lwc1 $f1, 16($sp)
        lwc1 $f2, 12($sp)
        lwc1 $f3, 8($sp)
        lwc1 $f20, 4($sp)
        lwc1 $f21, 0($sp)
        addi $sp, $sp, 24
```

```
        jr $ra


        .data
msg_done: .asciiz "done\n"
```

2. Your programs will be evaluated using the following criteria:

   Proper use of registers and memory (0-5,6-10,11-15)
   Proper use of load, store and branch instructions (0-4,5-8,9-10)
   Proper use of integer arithmetic instructions (0-4,5-8,9-10)
   Proper use of floating point arithmetic instructions (0-4,5-8,9-10)
   Proper use of character and string instructions (0-2,3-4,5-5)

   Proper use of subprogram calls [including use of stack] (0-4,5-8,9-10)
   Proper use of return instructions [including use of stack] (0-2,3-4,5-5)
   Proper design of a subprogram meet specified requirements (0-8,9-16,17-20)
   Proper use of appropriate tools to assemble, test and debug program (0-2,3-4,5-5)

   Appropriate documentation (0-4,5-8,9-10)

   The three point ranges next to each item specify the range of scores that will be used to represent ineffective, effective and highly effective performance, respectively.