COMP 273

Assignment 4
School of Computer Science
McGill University
Available On: November 22th, 2017
Due Date: December 6th, 2017. 11:59pm.
Submit your solution in electronic form using MyCourses
Read the submission instructions at the end of the document
(late policy: 25 marks off per day late)
Sharing code is strictly prohibited.
YOU WILL BE GRADED ON WHAT YOU
SUBMIT IN YOUR ASSIGNMENT FOLDER AND
IT IS YOUR RESPONSIBILITY TO CHECK THAT
THE FILES YOU INTENDED TO SUBMIT ARE
ACTUALLY IN THAT FOLDER!

1 FILE I/O (20 marks)

The template for this question is provided in fileio.asm

(a) (15 marks) Using the template fileio.asm as a starting point, write a MIPS procedure *readfile* that takes as its argument the address of a string that contains a valid filename, and then appropriate syscalls to read from that input file and then simply prints the content of that file to the screen. In the template there are two input files called test1.txt and test2.txt which you should test. To accomplish this task we allow you to create a large buffer, i.e., one that is larger than the expected number of ASCII characters (bytes) in the input file. The body of your code should work by calling the procedure you have written. The procedure should open the file, read its content as ASCII characters, store the content in the buffer, print the content, and then close the file. For your reference a more complete set of MIPS syscalls implemented in MARS, along with clear documentation on how to use each, is here: http://courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html . Error statements to should be printed if there are any errors in opening the file, reading from it or closing the file.

(b) (5 marks) We shall now build on the above example. Following the process of reading the file test1.txt, after which the ASCII characters have been read into the buffer, we shall call a second MIPS procedure called *writefile*. This procedure should open a file called test.pgm, should then write the following information to that file:
P2
24 7
15
Then it should write out the content that was read into the buffer. It should then close the file. Error statements to should be printed if there are any errors in opening the file, reading from it or closing the file.

If things are working properly when you view test.pgm with a suitable image viewer, e.g., gimp, you should see something interesting.

2 2D Arrays (30 marks)

1. The template for this question is provided in BlurredLines.asm

2. Read the data in test1.txt into the specified buffer.

3. The data read into the buffer should now be converted to consecutive integers and then stored in a 2D array of length 24×7, i.e., one that has 24 columns and 7 rows. However that 2D array will actually be represented as a 1D array* Finally, take care to convert the entries in the buffer (which are in ASCII) to their numerical values in base 10.

4. Write a procedure named *blur*, which takes every 3x3 block from the 2D array, averages** their values, and then places the rounded result into a new 2D array.

5. The content of the output 2D array should then be written into a file named test-blur.pgm

   (a) Use an image viewer like gimp to view the pgm, this should have taken your original image and blurred it, which should look like a blurred version of the image you created in question 1 b).

   (b) THIS CAN ONLY BE VIEWED if you have properly implemented a). This means that you are expected to write the information as you did in 1.b) and THEN write the completed output 2D array to the test-blur.pgm.

* Normally, in a language like Java, we would simply specify the respective array positions of i and j. (Let i represent the row we are currently at, and j represent the column we are currently at) In MIPS, however, our 2D array is stored as values in a 1D array. It is clear to see that for any position [i,j] in our 2D array, we can retrieve said position by simple doing (i * width) + j. Since i represents rows, whenever we add a width (for this assignment, width is 24) we are essentially going to the next row in our conceptual 2D array. j simply represents which column we are currently looking at. Since we are currently looking at the correct range of data in our 1D array based on i*width, we simply determine which column to look at by adding j to said value. We now have the position of interest.

** Averages are computed by taking the 3x3 block of data, finding the average of those points, and entering the average value to the central position of your 3x3 block, to the corresponding central value position in the new 2D array. In other words, we pick a central position, find the average of it and all its surrounding array values and place the average, at the same chosen position, in another 2D array. There will be slight differences when dealing with edges, which can be accounted for by the known width of 24 and height of 7. You can deal with this in an appropriate way, i.e., don't compute 3x3 averages for the outermost rows and columns. Also, every average is being taken from the original file. The averaged pixels are displayed in the output array.

3 MIPS Calculator(50 marks)

The purpose of this question is to write a simple MIPS calculator that would perform basic calculator functions using memory mapped I/O (you are not allowed to use syscall). The calculator should perform the basic arithmetic operations: addition(+), subtraction (-) , multiplication (*) and division (/) between a pair of numbers. The numbers entered and the mathematical operators (+,-,*,/) should be echoed to the screen and the program should reject any "unknown" key press. The results should be viewed upon pressing Enter key. Use the key 'c' for clear and 'q' for quit. A sample output would look like this :
Calculator for MIPS
press 'c' for clear and 'q' to quit:
35 + 55
Result : 90
You Must:

- We will be entering numbers by using the MIPS keyboard.
- If the user enters 'c' the calculator will be cleared.
- If the user enters 'q' the MIPS program will end.
- There are two situations you must consider:

  - NumberA Operation NumberB <enter is pressed>
    The result is displayed
  - Operation NumberB <enter is pressed>
    the last result is used as NumberA. the new result is displayed.

- For every expression, you delimit the numbers and operations by a ' ' (space).
- As input is being given to the **MIPS** keyboard, the values input will be echoed to the **MIPS** display.
- You are NOT required to account for complex expressions (ex: 1 + 4 / 3 + 2 - 1)
- You will be doing arithmetic on INTEGERS, so processing negative numbers will be required.
- The output is expected to be a REAL number, the displayed result will be rounded up to two decimal places (ex: 1.11)

The program should be submitted in a file called Calculator.asm. Make sure the file runs on MARS and you have your name and studentID inside the file.

4 Assignment Submission Instructions

1. Submit your solution to myCourses before the due date.
2. Highlight the Three template files, and zip them. The zipped file will be named <studentID>.zip. If my student ID is 123456789, then the zip file to submit will be named "123456789.zip".
3. If you have special comments about your code in any questions, feel free to include a "confessions.txt" file in your zip containing your specific comments. Otherwise, simply comment your code as you normally would.
4. Your code *must* run and assemble, even if the final solution is not quite working. Part marks will be awarded for correct high-level control flow and use of conventions. If something is not working, comment-out the broken parts of code and write a comment or two about what you expect to happen, what is happening, and what the problem may be. *If your code does not assemble you will receive very few points*