INDIAN INSTITUTE OF TECHNOLOGY DELHI

COL216

Report

Assignment – 1

Area Calculator in MIPS

Abstract

This program computes the area of a set of input points given in order of increasing x coordinate. This has been written in MIPS assembly language to better understand the hardware abstractions.

Harsh Agrawal 2019CS10431

Saptarshi Dasgupta 2019CS50447

Contents

1 Approach

We Made an outline of the task we have to complete and broke it down into subtasks. In this case we prepared the following subtasks.

1. Make a procedure (*input*) to take coordinates as input. Also display instructions to the user while asking for input.

2. Make a procedure (*area*) that takes 2 points and returns the area of the trapeziod formed by those points. The area is calculated using **single precision floating point registers**. This procedure assumes that the input points are on the same side of the X axis.

3. In the main procedure, using the procedure *input*, accept coordinates from the user. Call the procedure *area* using the coordinates received from the user. If two successive coordinates (x_1, y_1) and (x_2, y_2) are on different sides of the X-axis (which we determine by checking the sign of the product $y_1 * y_2$), we break the area into 2 triangles, one lying on each side of the X-axis, and call the procedure *area* on each separately.

4. An accumulator keeps track of the sum of all areas calculated, and after all the points have been processed, stores the area, which is to be displayed to the user.

2 Formulae used

- Area of trapeziod formed by (x_1, y_1) and (x_2, y_2) when both points are above the X-axis is given by

$$Area = (y_2 - y_1) * (x_1 + x_2)/2$$

.

- When both of the points are below X-axis then the negative of the above expression gives the absolute area.

- In the third case, we have that one point is above the X-axis and one is below. In this case, we introduce a third point, M on the X-axis where the line joining points A and B meet the X-axis. Then we can

calculate the absolute area covered by A,B to be the sum of the areas of triangles formed by (A, M) and (M, B). The coordinates of M are:

$$M = (x_1 - \frac{y_1.(x_2 - x_1)}{(y_2 - y_1)}, 0)$$

3 User Interface

On running the program, we are first prompted to enter the number of points. Then on each line we ask for the X or Y coordinates of the i^{th} point. After calculating the area is printed on the console with a message.

4 Testing

We employed both automated and manual modes of testing. The automated mode helped establish the correctness whereas the manual mode helped handle the corner cases.

4.1 Automated

We wrote a **python3** script that generates random coordinates, executes the asm file in a MIPS emulator (*spim*), communicates with the executed process through unix pipes, and compares the output with the expected area within a specified *error tolerance*. To run the script, we enter the following commands once inside root project folder

```
$ sudo apt install spim
$ python3 -m pip install -r requirements.txt
$ python3 tester.py -n 20 -m 100 -b 10 -e 0.0001
```

This will run 20 random test cases with 100 coordinates of at max 10 bits each, with an error tolerance of 0.0001 percent.

4.1.1 Results

The script developed is heavily customizable as evident from the command line flags available. We tested with different values of error tolerance and bitsizes, and found that overflow occured at 16 bit inputs and with 15 bit inputs, we were able to achieve an accuracy of about 10^{-4} %.

4.2 Manual

We also checked a few corner cases manually in QtSpim, which are listed below :

- Number of points 0 or 1
- All points on X-axis.
- All points on Y-axis.
- Duplicate points.

We can execute custom test cases using the same tester script that was developed for automated testing. Execute the command below with a custom test case file. The test case file should begin with the number of points and the following lines will contain the coordinates of each point in a separate line (x and y coordinate on separate line as well).

```
$ python3 tester.py -i <testcase_file_path>
```

4.2.1 Results

In all these cases we got the expected output without overflows, that is 0.

5 Assumptions

- We have used 32 bit floating point registers for our computation. So this program can only be used for calculating areas that take 32 bits in memory i.e. upto $2^{32} = 4294967296$ sq. units.
- We have assumed that points input to the program are sorted according to their x-coordinate values. If this is not the case then our program will double count certain areas leading to a higher value than expected.
- The tester script written is *assumed* to be correct as it is a much easier to verify the correctness of a program written in a high level programming language like python than a much lower level programming language like assembly. This comes partly due to the fact that compilers for high level programming languages have been thouroughly verified and partly due to the fact that the problem in hand is a farily easy one.