CSEE W3827: Fundamentals of Computer Systems

Homework Assignment 3. Stephen Edwards. Columbia University

Due Sunday, June 21 at 11:59 PM EDT via Courseworks

Download and edit the .s files in the hw3.zip file. Edit and make them work in the SPIM simulator `http://spimsimulator.sourceforge.net/` as discussed in class. Package your modified .s files in a .zip file and upload them to Courseworks to submit them. **Do not rename the .s files.** This time, you do not need to annotate or submit this file (hw3.pdf).

1. (25 pts.) In the SPIM simulator in MIPS assembly, write the *itri* routine in the `itri.s` skeleton to make it print an inverted triangle. The height of the triangle will be given in $a0, and will be between 1 and 40 inclusive. For the first three tests, the included test harness should print

```
Testing itri with 1
#
Test complete

Testing itri with 2
###
 #
Test complete

Testing itri with 5
#########
 #######
  #####
   ###
    #
Test complete
```

2. (30 pts.) Write a MIPS assembly routine *sqrt* that uses an iterative Newton's method to calcuate integer square roots. To compute the square root of k, start with $x_0 = k$ and calculate

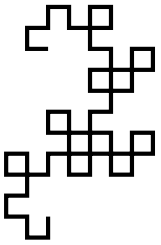
$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{k}{x_n}\right)$$

until either $x_{n+1} = x_n$ or $x_{n+1} = x_n + 1$, at which point you should return x_{n+1}. For example, for $k = 24$, you should observe the sequence $24, 12, 7, 5, 4, 5$ and return 5. Use *divu* to perform k/x_n and *srl* to divide by two.

Our test harness should print (omitted lines print identical outputs)

```
sqrt test harness
sqrt(1) = 1
sqrt(2) = 1
sqrt(3) = 2
...
sqrt(7) = 2
sqrt(8) = 3
...
sqrt(14) = 3
sqrt(15) = 4
...
sqrt(23) = 4
sqrt(24) = 5
...
sqrt(34) = 5
sqrt(35) = 6
...
```

3. (45 pts.) Dragon curves are self-similar fractals that can be approximated recursively. Modify the *dragon.s* file by writing a recursive function called *dragon* that outputs a dragon curve of a given order in SVG format. Here's an order 6 dragon curve generated by my solution:



Below is pseudocode for the dragon curve algorithm written for "turtle graphics," in which the "turtle" has a current point and direction and can either move forward, drawing a line, or turn in place. Note that in the algorithm, *sign* is either 1 or -1, so the turtle only ever turns $90°$ or $-90°$.

procedure DRAGON(order, sign)
  if order $= 0$ **then**
    Move forward
  else
    DRAGON(order $- 1, 1$)
    Turn $90° \times$ sign
    DRAGON(order $- 1, -1$)

SVG is an XML-based text format for vector graphics. For this assigment, you only need to know that its paths can be expressed as a series of horizontal and vertical line segments. For example, `h-10` means "draw a horizontal segment 10 pixels to the left" and `v5` means "draw a vertical line segment 5 pixels down."

Here is the order 3 dragon, which consists of 8 line segments, and its corresponding SVG file:

```
<svg xmlns="http://www.w3.org/2000/svg">
<path stroke="black" fill="none"
      d="M225 225h-5v-5h5v-5h5v5h5v-5" />
</svg>
```

The provided skeleton includes a test harness that prints out everything but the various horizontal and vertical segments of the "d" attribute of the path. Don't change anything in the *main* function except for the order number.

Keep the current direction of the turtle, coded as a byte with values 0, 1, 2, and 3, in the global variable in memory *direction*. To move forward, print a string from the *dirs* array of string pointers, which contain strings encoding the four compass directions in SVG, indexed by *direction*. To turn, update the *direction* byte according to the *sign* argument (passed as a signed integer in register $a1).

Your solution needs to be recursive. You will have to store various registers (such as $ra) on the stack when your function is entered and restore them when you return. My solution was about 40 lines.

I debugged my code by writing the output of my program to a .svg file and previewed it with the Chrome browser. Inkscape can work, but the curves may appear outside the document area.