CS2240 Spring 2023
Programming Project #2 – Zingl's Enhanced Version of the Bresenham Line Drawing Algorithm
Due: Friday, March 31, 2022, 11:59pm

In the Lab #6 exercise you developed assembly code to draw strictly horizontal and vertical lines on the Discovery Board display. We now explore an efficient algorithm for drawing a line in a raster graphics system at any arbitrary downward slope, not just straight horizontal or vertical. You can find a description of this algorithm in sections 1.6 and 1.7 of Zingl's paper available at http://members.chello.at/~easyfilter/Bresenham.pdf. The algorithm in Listing 2 is what we will be coding in assembly, but simplified for the case of a line that only slants down and to the right on the Discovery Board screen. Note that the coordinate variables *x* and *y* in this algorithm are the 'column' and 'row' coordinates, respectively, we have been using in the labs writing graphics to the display screen.

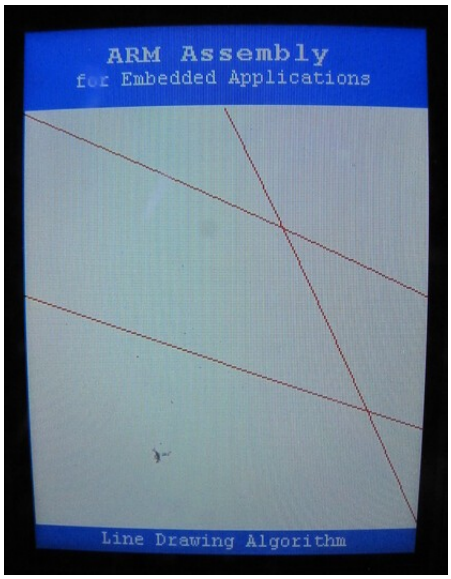
Download the C code file Project2-DrawLineMain.c and the template assembly source file Project2-DrawLine.s. Your project code will resemble the lab exercises with two source code modules, a main program in C and an assembly module. For this project the assembly module will provide one function called by the C code. You will complete the '`drawline`' function in the assembly source file to follow Zingl's enhanced version of the Bresenham algorithm and draw a red line. There should be four function arguments accepted, in the order x0, y0, x1, y1. These are the starting column and row coordinates (x0,y0) and ending coordinates (x1,y1) of the line to be drawn, as in the algorithm description. Write your assembly function to implement the pseudocode:

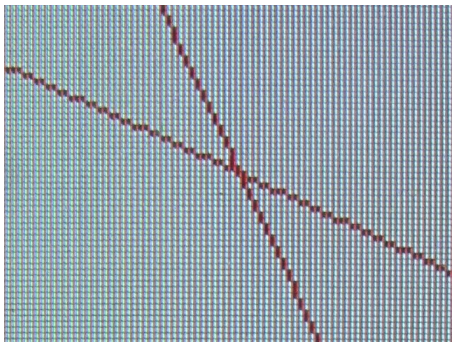
```
drawline(x0, y0, x1, y1)
    dx = x1 - x0
    dy = y0 - y1
    error = dx + dy

    while true
        pixel(x0, y0, color)
        e2 = 2 * error
        if e2 >= dy
            if x0 == x1 break
            error = error + dy
            x0 = x0 + 1
        end if
        if e2 <= dx
            if y0 == y1 break
            error = error + dx
            y0 = y0 + 1
        end if
    end while
return
```

Also include in your code a copy of the `pixel` function from Lab #7 that accepted three arguments, column, row, and pixel color, to store a pixel word to the screen memory. As you can see in the above pseudocode, you will be calling your `pixel` function from within your `drawline` function, similar to what you did in Lab #7 to fill a colored rectangle. You can use any color of your choice, using the `.equ` constants provided in the template source file, to pass as the third argument to `pixel` to set the line color. Be sure to push onto the stack at the entry point of your function any registers other than R0-

R3 and R12 you use and pop them off the stack before returning. Write the `drawline` function managing its use of the registers and with appropriate stack pushes and pops to be a **non-leaf** function.





Note in the main C program three calls are made to your function '`drawline`' in the assembly language module to draw three lines on your display screen with different starting and ending (column,row) coordinates. Run your code and verify that it produces a display similar to that shown here, only with a color of your choice, not necessarily red. Inspection of the display screen with a magnifying glass will reveal the pixel stepping behavior of the Bresenham and Zingl line drawing algorithms. Hint: Remember the helpful '`SVC #0`' debugging instruction available with your Discover Board library! Upload your code to Canvas and a screen photo. Make sure, however, that any debugging breakpoints you inserted for debugging your code have been removed in the version you submit.