In this assignment you are to write the framework for a simple graphics system using text characters as pixels. The basic graphics screen will be 32 raster lines (rows) by 96 pixels per line (columns) so that it will fit on the screen of the ARM debugger. Each pixel will occupy a single bit of a word, so each memory word will contain 32 packed pixels. Thus, the entire memory footprint of the graphics “screen” will be a two-dimensional array containing 96 words, using three words per raster line. The PrintBinary routine that you developed earlier must be used here again, but modified to print “.” instead of “0” and “*” instead of “1”.

Type in the file on the next page, and then add your name and ID number in the indicated places. Your task at this time is to add the bodies of the subroutines SetPixel, ClearScreen, PrintScreen and the modified version of PrintBinary. You’ll also need the PrintLF code from your earlier projects. You may not add any global symbols, code, or variables. As usual, all subroutines are required to be transparent with respect to register usage and must use the stack for temporary register storage.

Neither the ClearScreen nor the PrintScreen subroutines require any parameters. The ClearScreen routine clears the Screen array to all zeroes, and the PrintScreen routine prints out the screen as 32 rows of 96 characters per row. In both cases you must use the symbols Rows, Cols, Words, and Bytes in your code instead of special constants; I want to be able to change the number of rows and the number of columns in your code and still have your program work after assembly with the new values. Note that Cols represents the number of 32-bit words per raster line, not the number of pixels. You may use the explicit constant 32 when dealing with the number of bits per integer word, but not as the number of raster rows.

The SetPixel subroutine gets two input parameters through the registers: register R0 contains the X value (column) and register R1 contains the Y value (row) of the pixel bit to be set to 1. The upper-left pixel is at location <0,0>. X and Y values must be clipped to the screen; i.e., if the X and Y values are off-screen (less than zero or greater than the maximum column or row) then no pixel will be set. In the code on the next page the pixel to set is at location <45,13>. You should change these values for testing the SetPixel routine, but you must return the final values to <45,13> before turning in the assignment.

When working, print out the .ALI assembly listing and a screenshot containing the source code after execution with your name visible, and the console window showing the resulting graphics screen. The graphics screen will occupy most of the window. Staple the screenshot on top of the listing. Here are the point penalties for this 10-point assignment (no assignment will score less than zero):

1. -1 for cosmetic errors: printouts not stapled, or screenshot not on top.
2. -10 for name not visible on .ALI listing (did you write the code?)
3. -5 for not printing the screen at all (does Print_Screen work?)
4. -5 for not correctly setting pixel <45,13> (Clear_Screen & Set_Pixel)
5. -2 for non-transparent subroutines (are registers saved and restored?)
6. -2 for modifying any existing code (did you change the assignment?)
7. -2 for using explicit memory locations (did you use the stack?)
Rows EQU 32  ; Total rows of text
Cols EQU 3   ; Memory words per row
Words EQU Rows*Cols ; Total words to reserve
Bytes EQU Words*4 ; Total bytes to reserve

AREA PROGRAM3, CODE
ENTRY

Start   BL  ClearScreen
       MOV R0,#45 ; X
       MOV R1,#13 ; Y
       BL  SetPixel
       BL  PrintScreen
       BL  PrintLF
       SWI &11

; {Insert your name and ID here}
;
;
SetPixel

; {Insert your name and ID here}
;
;
ClearScreen

; {Insert your name and ID here}
;
;
PrintScreen

; {Insert your name and ID here}
;
;
PrintBinary

; {Insert your name and ID here}
;
;
PrintLF

; {Insert your name and ID here}
;
;
Screen % Bytes

END
EXTRA CREDIT

Each of these suggestions is optional and worth +5 points. You must complete and print the regular assignment before including any new code. You must test any new code to show that it works completely.

1. Implement HorizontalLine

Write a subroutine called HorizontalLine to plot horizontal lines from coordinate \(<X_1,Y>\) to coordinate \(<X_2,Y>\). In this case the coordinates are passed in through integer registers R0, R1, and R2, where \(R0=X_1\), \(R1=X_2\), and \(R2=Y\). As usual, all registers must be preserved (on the stack). You should use the code you developed as part of the second midterm.

2. Implement PlotLine correctly

Change the PlotLine routine you developed earlier so that it draws a straight line on the grid, between the two points passed in, instead of printing out just the endpoints.

3. Implement Bézier Curves

Take the recursive Bézier curve code you developed in the earlier project and insert it here, using the modified PlotLine routine (above). You must also change Threshold to 10.0 so that Bézier curves will appear as a series of coarse segments instead of as a smooth curve. Demonstrate this with a Bézier curve that fills most of the graphics screen area.

4. Implement Solid Circles

Write a subroutine called SolidCircle so that it will draw a circle centered at coordinate \(<X,Y>\) with radius \(R\). The values are passed in through integer registers R0, R1, and R2, where \(R0=X\), \(R1=Y\), and \(R2=R\). Here is a pseudocode template for creating solid circles:

```
Procedure SolidCircle (X,Y,R:Integer)
  Var XX, YY, SS : Integer  { Local variables }
  Begin
    XX := R
    YY := 0
    SS := -R
    While (XX >= YY) Do
      Begin
        HorizontalLine (X-XX, X+XX, Y-YY)
        HorizontalLine (X-XX, X+XX, Y+YY)
        HorizontalLine (X-YY, X+YY, Y-XX)
        HorizontalLine (X-YY, X+YY, Y+XX)
        SS := SS + YY + YY + 1
        YY := YY + 1
        If SS > 0 Then
          Begin
            SS := SS - XX - XX + 2
            XX := XX - 1
          End
      End
  End
```
5. Implement Outlined Circles

A variation that plots circle outlines is as follows:

Procedure OutlineCircle (X,Y,R:Integer)

Var XX, YY, SS : Integer { Local variables }

Begin

XX := R
YY := 0
SS := -R

While (XX >= YY) Do

Begin

SetPixel (X+XX, Y+YY)
SetPixel (X-XX, Y+YY)
SetPixel (X+XX, Y-YY)
SetPixel (X-XX, Y-YY)
SetPixel (X+YY, Y+XX)
SetPixel (X-YY, Y+XX)
SetPixel (X+YY, Y-XX)
SetPixel (X-YY, Y-XX)

SS := SS + YY + YY + 1
YY := YY + 1

If SS > 0 Then

Begin

SS := SS – XX – XX + 2
XX := XX – 1

End

End

End