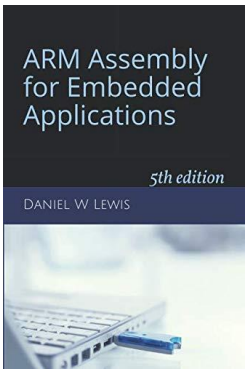


Programming Lab 5B

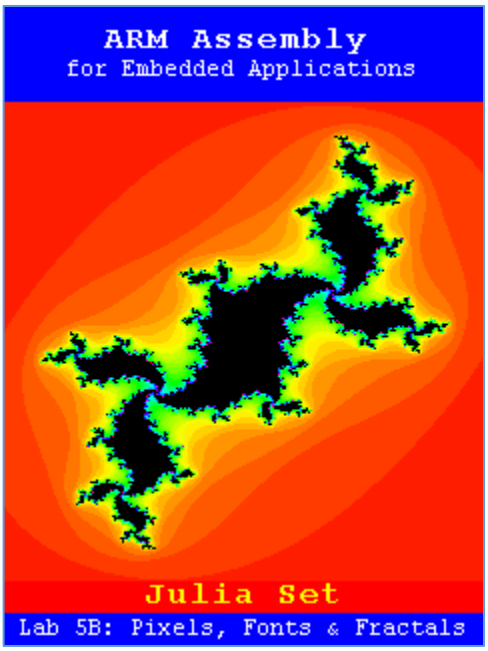
Pixels, Fonts & Fractals

Click to download Lab5B-Main.c

Topics: Integer arithmetic, organization of fonts and pixels in memory, converting the x and y coordinates of a pixel into a memory address, indexing into font tables.

Prerequisite Reading: Chapters 1-5
Revised: January 22, 2021

***Background*[1]:** Fractals are useful in modeling structures (such as eroded coastlines or snowflakes) in which similar patterns recur at progressively smaller scales, and in describing partly random or chaotic phenomena such as crystal growth, fluid turbulence, and galaxy formation. Abstract fractals – such as those displayed in this lab – may be generated by a computer calculating a simple equation over and over.



Pixels: In this lab, images are prepared in a frame buffer; once the image is complete, the frame buffer is copied to the display using direct memory access (DMA). The frame buffer is a 2D array – one byte per pixel. During the DMA transfer, special hardware uses each byte as an index into a color table to lookup the pixel's red, blue, and green components to be written to the display.

Text Fonts: Fonts come in various character sizes, specified by their $height\ x\ width$ in pixels. Our run-time library provides five fonts of 94 characters each starting with the ASCII space. Each character in stored as a 2D bitmap with one bit per pixel. For each character, there are n bytes per row, where n is given by the integer quotient $(font\ width + 7)/8$. For example, the letter **K** of the $16x11$ font is stored as 16 rows of 2 bytes, where each row uses the left-most 11 bits of the 2 bytes to represent the 11 pixels in that row of the character. The bytes are stored in Big-Endian order. I.e., the first byte contains the left-most 8 bits of the row so that when assembled correctly the bits representing the top of the letter **K** should be arranged as shown below:

01111011	11000000
Byte N	Byte N+1

```
0x00, 0x00, //
0x00, 0x00, //
0x7B, 0xC0, //  #### ####
0x31, 0x80, //   ##   ##
0x33, 0x00, //   ##  ##
0x36, 0x00, //   ## ##
0x3C, 0x00, //   ####
0x3E, 0x00, //   #####
0x33, 0x00, //   ##  ##
0x31, 0x80, //   ##   ##
0x79, 0xC0, //  ####  ###
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
0x00, 0x00, //
```

Assignment: The main program will compile and run without writing any assembly. However, your task is to create an equivalent replacement in assembly language for the following three functions found in the C main program. The original C versions have been defined as "weak" so that the linker will automatically replace them in the executable image by those you create in assembly; you do not need to remove the C versions. These functions are called by a main program that uses them to display three animated fractals on the display and the fractal title in white text just below the image. Use the blue pushbutton to sequence through the fractals.

`void WritePixel(int x, int y, uint8_t colorIndex, uint8_t frameBuffer[256][240]) ;`

Stores $colorIndex$ in $frameBuffer[y][x]$.

`uint8_t *BitmapAddress(char ascii, uint8_t *fontTable, int height, int width) ;`

Returns the address of the bitmap for a specified character within a given font table. Parameter `fontTable` is a pointer that contains the starting address of a font table.

`uint32_t GetBitmapRow(uint8_t *rowAdrs) ;`

Returns a left-justified row of bits corresponding to the pixels in one row of a character. Parameter `rowAdrs` is a pointer that contains the address of one row in the font's bitmap of some character. *Hint: This requires a total of only 3 instructions if you use an* `REV` *instruction (see Instruction Set Summary).*

[1] https://fractalfoundation.org/fractivities/WhatIsaFractal-1pager.pdf