Lab #7 – Conditional Execution

Goals:

Write programs on the Discovery Board forming two nested loops
Further develop the use of the Discovery Board graphics display

Reference: 2^0=1 2^1=2 2^2=4 2^3=8 2^4=16 2^5=32 2^6=64 2^7=128 2^8=256 2^9=512 2^{10}=1024 2^{11}=2048 2^{12}=4096 2^{13}=8192 2^{14}=16384 2^{15}=32768 2^{16}=65536 …

For each step enter the necessary responses into the Canvas assignment text entry tool to let the TA follow your work.

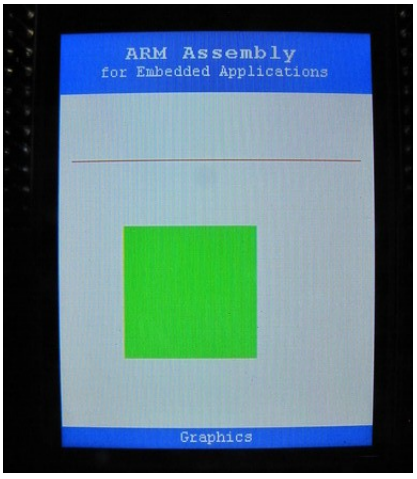
Remember the 'SVC #0' debugging instruction available with your Discover Board library! That can be inserted in your assembly code any time you need to clarify exactly what is in the registers!

Refer to the Weeks 5, 7, and 8 lecture slides for the following:

Part 1: Download the template main C program Lab07-DrawMain.c and the template assembly source code Lab07-draw.s from the Canvas files area, and in the LabCode folder. Note that the main program simply calls a function.

Note that the top lines of Lab07-draw.s contain the same .equ definitions as in Lab 6 for common colors and the memory-mapped display base address:

```
.equ COLOR_BLUE,    0xFF0000FF
...
.equ COLOR_ORANGE,  0xFFFFA500
.equ DISPLAY,       0xD0000000
```



a) Write assembly code in the function to fill a rectangular area on the display of a color of your choosing. The (column,row) coordinates of the top left corner of the rectangle are (50,150), and the coordinates of the lower right corner are (149,249). (Since this is in the white background region of the display, obviously choosing a drawing color other than white would be a good idea!) Write your code in the form of two nested loops, one over column coordinate and the other over row coordinate. You can follow the example nested loop coding shown on slide 14 of the Conditional Execution lecture slides if you like.

Use two variables in registers representing the logical column and row of the pixel to control each loop, that is, to be incremented each loop iteration and tested to loop termination. It will then work easiest to have inside the inner loop assembly instructions including a multiply to compute the current pixel index from the row and column indices. So you may use this example C code as a template for filling in the function assembly code:

```
uint32_t col,row,i,display[76800];

row = 150;
```

```
do {
  col = 50;
  do {
    i = (row * 240) + col;
    display[i] = pixel;
    col++;
  } while (col < 150);
  row++;
} while (row < 250);
```

Then once the display array index ‘i’ is computed in the assembly code, use the `[Rn, Rm, LSL #<n>]` addressing mode in an STR instruction to store your chosen pixel word to the display memory at the proper address. The total number of registers needed in this nested structure will likely be more than the R0-R3 allowed as working registers with the function calling convention, so you must use PUSH and POP instructions to temporarily save some registers from among R4-R11 to use in your function, and then restore before the function returns, as discussed in class. As in Lab #6, no `.rept` and `.endr` repeat blocks allowed in this lab!

Your rectangle should look like the green area in the photo above, except in whatever color you choose. (Ignore the colored line, as we covered drawing that in Lab #6.) Upload your code and a cell phone photo of your display screen showing the filled rectangle.

Note: Be sure you save the assembly code you write for this lab. Lab #8 will involve much of the same code!

Part 2: Time to work on the current programming project assignment.