Lab #8 – Random Bit Generator

Goals:

Write programs on the Discovery Board using bitwise logical instructions to generate random bit patterns

Use the Discovery Board graphics display to translate random bit patterns into random pixels

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!

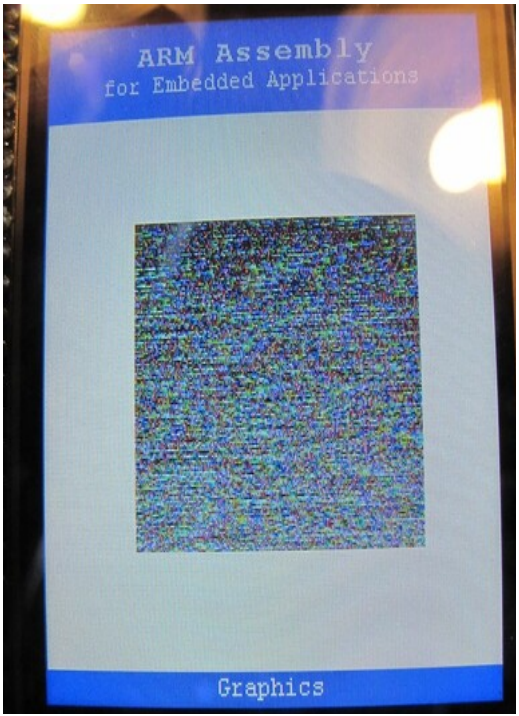
Refer to the Lecture slides for the following:

Part 1: a) Download Lab08-RandomMain.c and Lab08-random.s from the Canvas files area, and in the LabCode folder. Complete the assembly code to implement the random bit algorithm on slide 18 of the Bitwise Logical slides using the bitwise logical assembly instructions. The `seed_random` function is provided, and this simply initializes the random state variable with the value of the function argument. The `random` function should implement one pass through the algorithm in the class slide and return the new random generator state as its function return value. Verify that the main C program displays 20 different pseudorandom words each press of the blue button by iteratively calling your function.

b) Now download Lab08-DrawMain.c and Lab08-draw.s from the Canvas files area, and in the LabCode folder. You can delete Lab08-RandomMain.c, but save your Lab08-random.s in your 'src' directory. Note that the new main program simply calls a function.

Note that the top lines of Lab08-draw.s contain the same .equ definition as in Lab 6 for the memory-mapped display base address:

```
.equ DISPLAY,       0xD0000000
```



Modify your assembly code in the `rect` function from Lab #7 so that instead of filling in a rectangular area on the display with a fixed color, the rectangle will be now filled in with a different random color in each pixel. The (column,row) coordinates of the top left corner of the rectangle are (50,100), and the coordinates of the lower right corner are (199,249), somewhat larger than in Lab #7. Your code from Lab #7 should already be in the form of two nested loops, one over column coordinate and the other over row coordinate. It should already 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. Then inside the inner loop, as in Lab #7, it should include the assembly instructions including a multiply

to compute the current pixel index from the row and column indices from the usual formula $i = (row \cdot 240) + column$.

To modify your `rect` function remove the fixed colored pixel and then **in each iteration of the inner pixel writing loop** create a randomly colored pixel by first calling your `random` function you wrote in a) to get a 32 bit pseudorandom word, and then using a bitwise logical instruction to force bits 31-24 (the opacity field) to be all 1's. 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 as in Lab #7 to store your randomly colored pixel word to the display memory at the proper address. Generate a new random pixel within the inner loop so that all individual pixels inside the rectangle are randomly colored.

Note that **your `rect` function will no longer be a "leaf" function**, since it now in turn calls the `random` function. So you must also modify the use of registers in `rect` to allow for calling another function, with the proper assumptions about which registers can be modified in what function, and with stack instructions that push the LR at the entry of `rect` and pop it back into the PC at the end of `rect`, as discussed in class. You will need three source code files in your 'src' workspace directory, namely Lab08-DrawMain.c, Lab08-draw.s, and Lab08-random.s

Your rectangle should look like the random area in the photo above. Upload your code and a cell phone photo of your display screen showing the filled rectangle.

c) Edit the main C program to surround the call to `rect()` with an infinite loop, as in:

```
while (1) {
  rect();
}
```

Observe the continual stream of random pixel colors on the display screen.

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