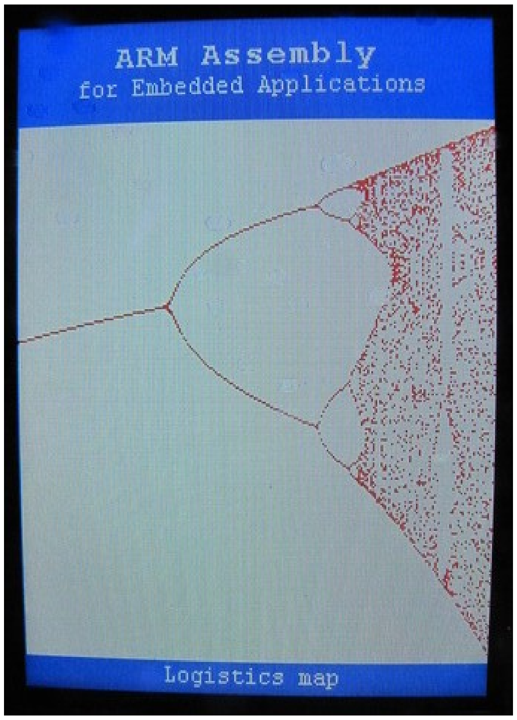
CS2240 Fall 2022
Programming Project #3 – The Chaotic Logistics Map

Due: Friday, November 18, 2022, 11:59pm

Hint: Remember the helpful '`SVC #0`' and '`SVC #1`' debugging instructions available with your Discover Board library! That can be inserted anywhere in your assembly code any time you need to clarify exactly what is in the core and floating point registers.

First study the basics of the Logistics Map at https://en.wikipedia.org/wiki/Logistic_map . This "map", or recurrence relation, is characterized by a single real-valued parameter '*r*'. As discussed in this article, we are interested in the sequence of real-valued numbers x_0, x_1, x_2, ... that result from first initializing the sequence with some x_0, and then generating a sequence of x_i values with the iteration formula $x_{i+1} = r \cdot x_i \cdot (1 - x_i)$, for some fixed value of *r*. For lower values of the parameter *r* in a certain range, the values in the sequence of x_i converge quickly to a single value. For higher values of the parameter *r*, the sequence of x_i will alternate between two or four or eight fixed values. Then for a higher range of *r*, the sequence of x_i values demonstrate numerical chaos. These three regimes of behavior can be seen in the plot captioned "Bifurcation Diagram" in the Wikipedia article, and this is the plot we will be duplicating on the display screen of our Discovery Boards in this project. The various values of the parameter *r* are along the horizontal plot axis, and the sequence of x_i values that result from a given *r* value are represented by multiple points on the plot in a vertical column, with x along the vertical axis. This nonlinear map has long been an example of the complexity of mathematical behavior that is observable with a simple polynomial recurrence relation of only second order.

Download the C code file Project3-LogisticsMain.c and the template assembly source file Project3-logistics.s. Of course, the values of *r* and x_i are real-valued, so you must use floating point arithmetic in your assembly code to compute them. 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.



Your project is to complete the 'map' function in the assembly source file to generate the Logistics Map sequence. There should be four function arguments accepted. These are, in order, the floating point *r* parameter governing the sequence, the floating point initial sequence value x_0, the pointer to a floating point array where the subsequent generated sequence values x_i are to be stored, and the integer size of the array.

Note that in the main C program a loop is formed over the column coordinate of the display pixel. In each iteration, a corresponding value of the float parameter *r* is calculated from the display column that will result in a plot ranging from *r* values of 2.5 to 4 across the screen. Your 'map' function will be called with each *r* value to fill in `NUM_POINTS` array values in a float array representing the x_i sequence. Important: In order to let the recurrence relation reach its terminal value or values for the non-chaotic regimes, first iterate 100 values of x_i in the sequence but **do not** store them in the array.

Simply run the relation $x_{i+1} = r \cdot x_i \cdot (1 - x_i)$ in a first loop starting with x_0 to generate x_1 through x_{100} that are **not** stored. Then taking off from x_{100}, run the number of iterations of the map in a second loop needed to generate additional values starting with x_{101} that fill the float array in the C program, using the size of the array that is provided as the fourth argument. At each display column, the C main program then loops through your generated array and plots points for each x_i value in the column for the r parameter. Note from the C code the plot scale on the vertical axis will be x=0.0 for screen row 300 at the bottom and x=1.0 for screen row 50 the top of the plot, as in the plot in the Wikipedia article.

Make use of your code from Lab #10 as example floating point arithmetic to generate each successive x_i sequence value. Make use of your code from Lab #9 and for Lab #10 addressing elements in an array whose pointer is specified from the C main program as a function argument. Note that the size of this array is also a function argument, so the second loop in your function must use this argument to respect the array length and only store the number of array values using allocated array memory! Of course, Lab #9 worked with C character arrays, or arrays with each element requiring only one byte of memory. The array in this C program is of float values, with each element requiring four bytes like the array in Lab #10, so write your memory addressing accordingly.

Run your code and verify that it produces a display similar to that shown above. 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.