Report for EE2028 Assignment 1
Sun Qiyang A0242744E
Darryl See A0216216N

Question 1: Knowing the starting address of array points10*[][]*, how to find the memory address of the A-th data point, say A ≤ M? Drawing or equations can be used to explain your answer. *Hint: if A-th data point is located from points10[Ax][Ay], formulate the relation between the memory address and the indices.*

The address of points 10[Ax][Ay] can be found using **[starting address]+(A*2)*4 for [Ax] and [starting address] + (A*2+1)*4 for [Ay], for A starting from 0.**

(i) Compile the "EE2028_AY2122S1S1_Assign1" project and execute the program.

(ii) Comment the PUSH {R14} and POP {R14} lines in classification(), recompile and execute the program again.
Question 2: Describe what you observe in (i) and (ii) and explain why there is a difference.

In (i) the program was completed, printing out the points, and the new centroids.
In (ii) the program was halted in classification.s, not being able to return to the main.c file.
The register R14 contains the LR which enables the program to return to main.c. By not pushing and popping R14, the LR in R14 was changed in the subroutine, which caused the program to not be able to leave classification.s.

Question 3: What can you do if you have used up all the general purpose registers and you need to store some more values during processing?

We can use **push** to store the data from the register to the top of stack memory. and when we use those values that are pushed to the memory again, we can use the pop function.

Classification.s
Before the whole code, we should first use PUSH to store the original value in the register. This can help us avoid losing important information in main.c . In the end we use POP to get all the information in main.c back to register.
Firstly, store the value in memory to register so that it's easier for us to use later. The values of old centroids, M,are used frequently, so we use instruction LDR to store them. use LDR R'x' ,[R0, #4]! to move the point from memory to register.
Secondly, we use a loop. BL is for jumping to a loop from the current code.

For Loop. Firstly, the square of distance from point to the **first centroid** is calculated. Calculate the difference in X and use 'MUL' to operate diff_X*diff_X. Do the same step for difference in Y. Then, we add them together and store this value, because we will use it later.
Do the same step to acquire the square of distance from point to the **second centroid.**
We compare the square distance of the point from the first centroid and the second centroid. Using ITE instruction to do classification. and store the value of the class in memory which registers R2 points.

Using SUBS R7, #1, BNE Loop1 to stop the loop. every time the value in R7 minus 1, then when R7=0, the flag Z=0, then the loop stops.

We use BX LR to go back to continue executing the POP function and then using BX LR to go back to main.c.

Find_New_Centroids.s

Using CMP, compare the class of the first point to #1, to create 2 separate counters for the number of points for each centroid.

Using CMP again, compare the class of the first point to #1.
By using an IT block, we can create 4 separate registers to store the values of X and Y for both centroids.
If the class is equal to #1, add the X and Y value of the point to different counters for the first centroid.
If the class is not equal to #1 add the X and Y value of the point to different counters for the second centroid.

Using DIV, divide the X and Y values of both centroids by the counters of the number of points found in the first part to get the coordinates of the new centroids.
Store the new values back into its addresses to return to the main.c

Improvements

1.More coding could be done in Find_New_Centroids to allow for the program to be used for any number of centroids, which is not required in Assignment 1.

Other efforts