F28HS – ARM ASSEMBLY REPORT

The aim of the arm assembly part of F28HS coursework was to create a counter and connect it to the Hex display using the buttons on the CPUlator. The main functions of the arm code will attempt to achieve this by first initializing the hexdisplay then looping through the program and observing the state of the button at which point depending on if the buttons are on, the appropriatecounters are incremented and stored in a location in memory. Other parts of the program will then translate the values of the counter and display the on the hex display.

Registers

This section will explain how each registers are used in the program.

R0 and R1 will be used to track the state of the buttons and the hexdisplay, initially assigned to the button base and hex base, which is both their addresses, when comparing the state of the button or sending the new counters to the hex display R0 and R1 will be used and they are not changed throughout the program.

R2 will load the value at the button base, this will be used to view the value show in the button.

R4 will be used to obtain each counter value when it needs to be viewed or modified. It is also used as a local variable for any other addresses in memory that needs to be accessed temporarily. R4 is also used to hold the hex value of any digits that needs to be displayed on the hex.

R5-R7 are all local variables with varying uses across the program.

R8 is used to determine which hex display segment needs to be modified. Depending on the state of the button or the state of the current counter, R8 will be assigned a value which other functions will compare and use to point to a function that will change the bitmask of the hex value to be passed onto the display.

Functions

_start: This function serves to assign the button base and the hex base to their respective registers then a execute a branch link to the init function.

Init: The job of the init function is to display 4 zeros to the hex display. It first pushes registers R4 to R7 as local variables but only R4 is used. Hex value 3f3f3f3f is assigned to R4 then this is displayed on the hex display by storing this value to the address in R1 which we know to be the address of the hex display. The POP operation is used to restore the state of the local variables then the branch is exited

Loop0: This function will load the value in the address of the button base and read it. A cmp operation is used to compare the value at this address. Depending on the value shown by R2 the function will branch to a function that will increment and update the appropriate counter.

Incr_c1: This function will load the value at the first counter by assigning the address to R4 then loading the value at this address into R5, R5 will be compared to 9 to ensure it has not reached capacity. If capacity is reached, then a branch is triggered to incr_c2 which will deal with the result of

this behaviour. If this condition is not met, then this function will go on to increment the current counter then branch to chks to work towards displaying this value. Registers R4 – R7 and pushed to be stack so that they can be used in this branch then popped at the end of the function.

Incr_c2: This function will address the behaviour of the first counter being exhausted. This function will increment and update the second counter, c2, then send it to the chks function so that the new value can be displayed. After the value is displayed then the segment corresponding to c1 will also be sent to a "show_zero" function which will display a 0 at the first segment. This serves to imitate getting a value of 10 in the first two hex displays.

Incr_c3: This function works similar to incr_1 but instead of pointing to the first segment it points to the third segment. The registers used are the same as in incr_1 but instead R4 will point to a different counter in memory and R8 will be amended to trigger the subsequent branching functions to point to the correct display as the value in R8 will be compared in those branches/functions to ensure the correct functions are called.

Incr_c4: This function works similar to incr_2 but just like incr_3 to incr_1, it will load in a different counter in memory as well as update R8 so that it's display points to the correct segment.

Chks: This function will compare the value at R5 with numbers then branch to the correct function that will handle the display. The value inside R5 will be assigned by the upper branch that calls the chks function so this will just be like an intermediate function to handle the different values in R5.

Show_one, show_two…show_zero etc: These functions named with 'show_one' up to 'show_nine' will handle the assignment of the hex codes that corresponds to the value at each counter. When each branch/function is called R4 will be assigned to the digit that goes with the name of the branch then R8 is compared to 0, 1, 2 or 3 to determine which display function to branch to.

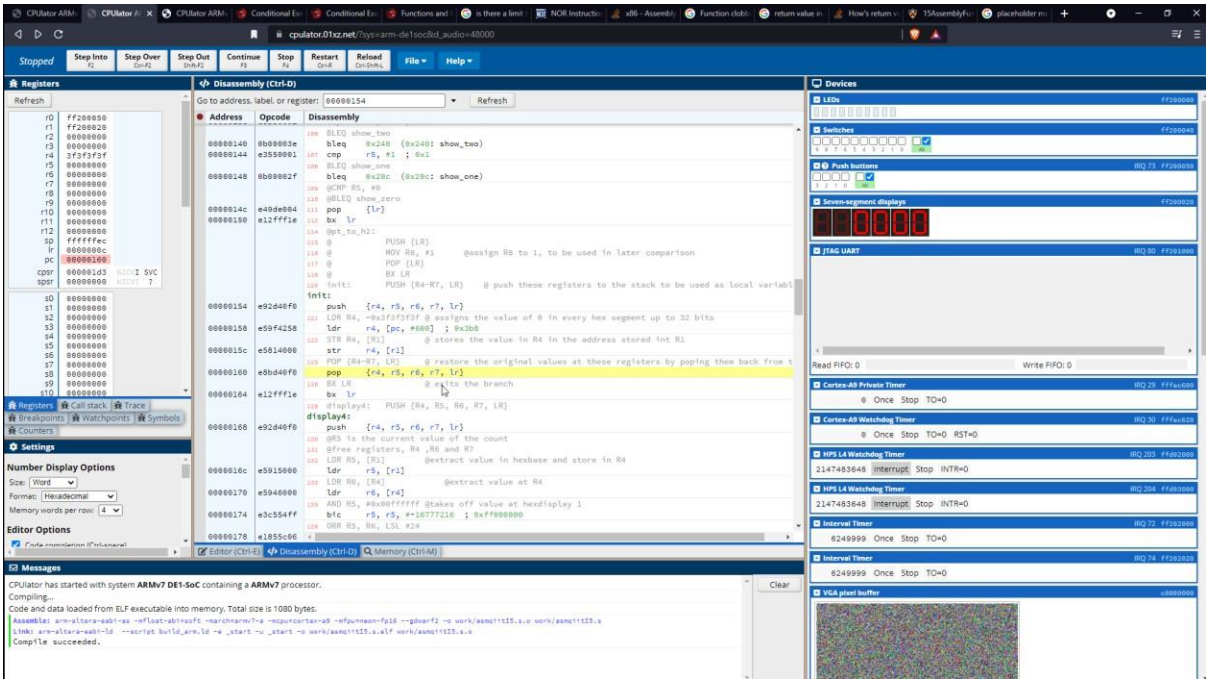
Show_zero: This function will handle whenever a counter reaches its capacity, 9. It will assigned R4 to the address stored in "zero " in memory and will compare the value at R8 with 0 and 2. Only 0 and 2 are compared with R8 because upon starting the program we assume that the hex segments 2 and 4 will not need to be modified as they are only incremented when hex segment 1 and 3 are exhausted, reach value 9, and segments 2 and 4 will never be modified to 0. When either of them reach 9 then the whole counter is exhausted and we cannot count anymore.

Display1: This function will handle displaying unto the first segment of the hex display. The value to be displayed is handled by previous functions and by the time previous functions. The current state of the hex display is loaded into R5 then using the AND operator the value at the first segment is erased. The decimal value of the desired value is loaded into R6 from the value stored in the address R4 which was handled by the previous branch, then using the ORR operator R5 and R6 are combined so that the value can be appended to the correct position before storing R5, which is now the hex code for the updated count, to the hex base.

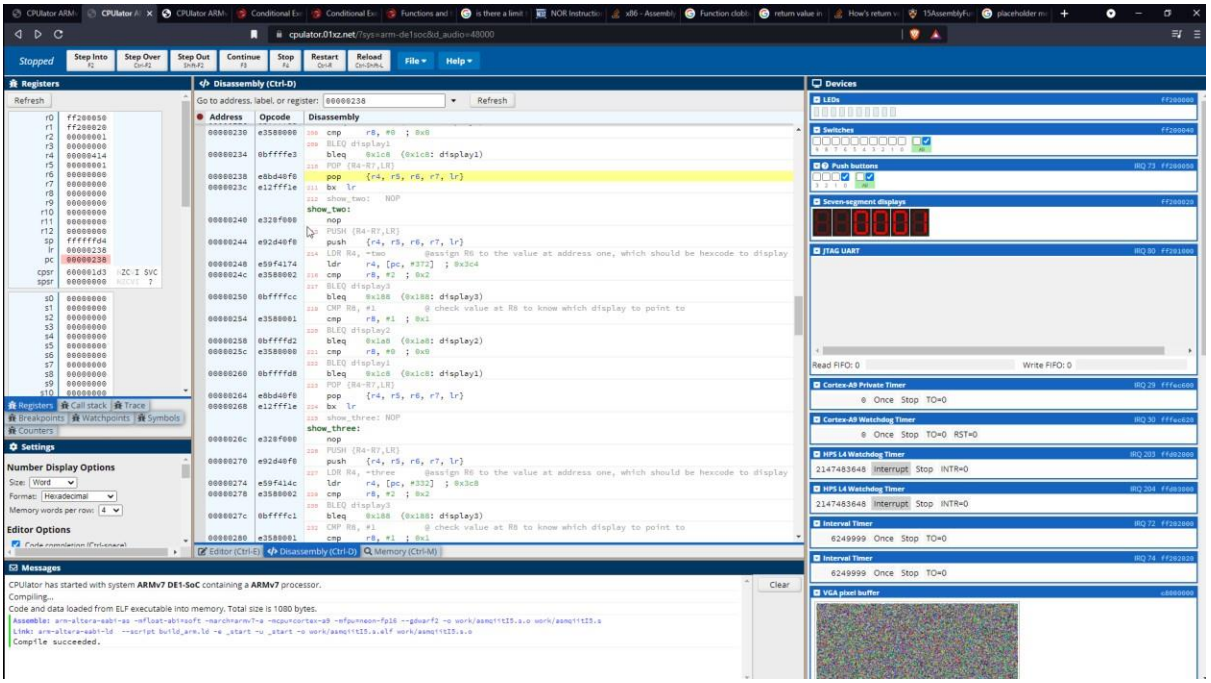
Display2, Display3, Display4: These functions work pretty much the same as display 1 but the masking is different depending on which segment that needs to be pointed to a well as when applying the ORR operation, LSL is also used so that the value in R6 will be appended to the correct position. For example, the masking for display 2 is 0xffff00ff then when performing ORR, an LSL operation is performed on R6 to ensure the value appears in the correct position when ORR is applied.

TESTS

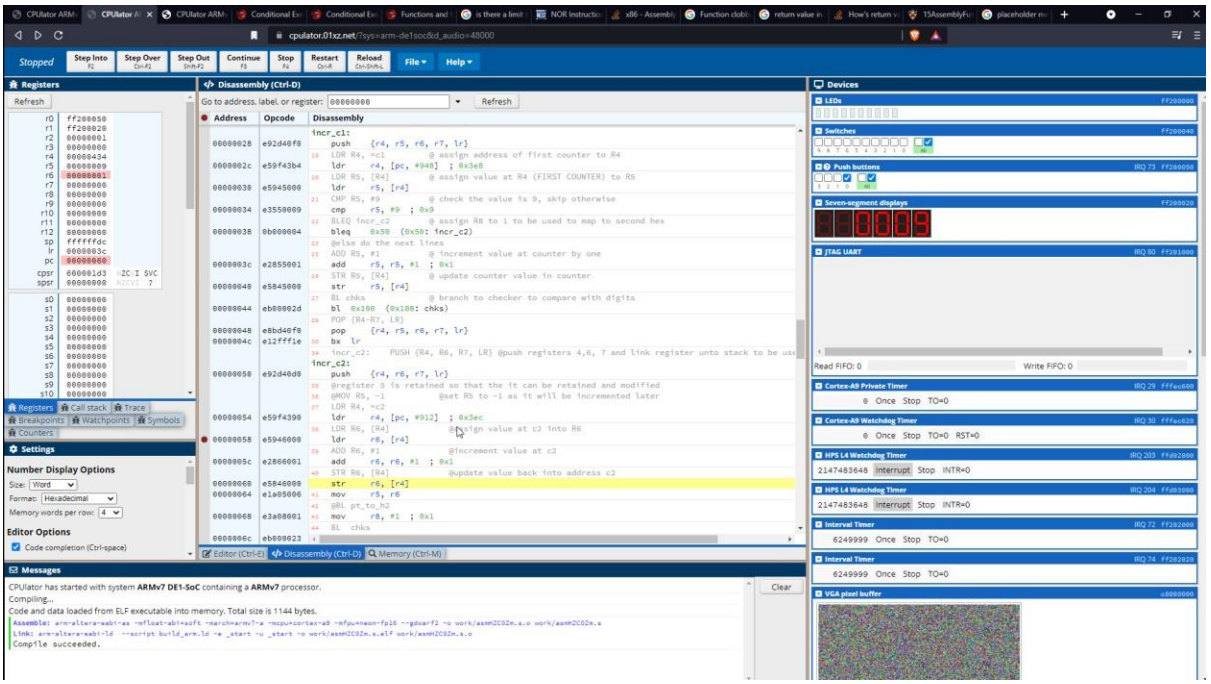
After init function



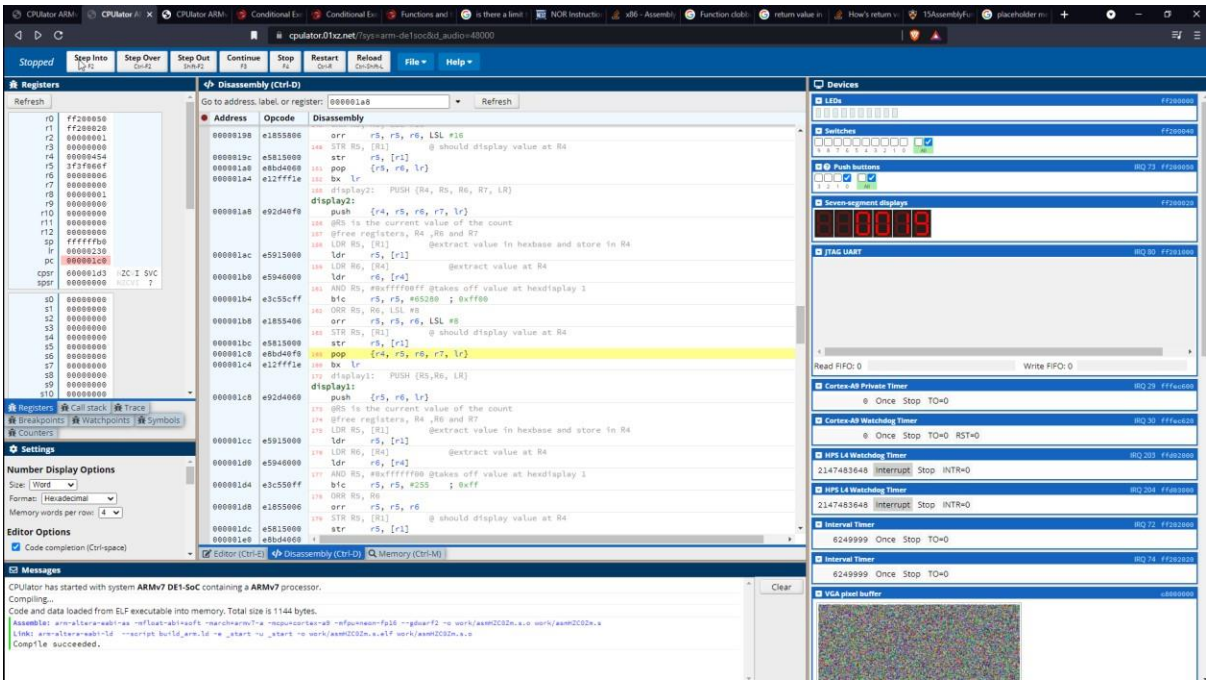
Updating c1



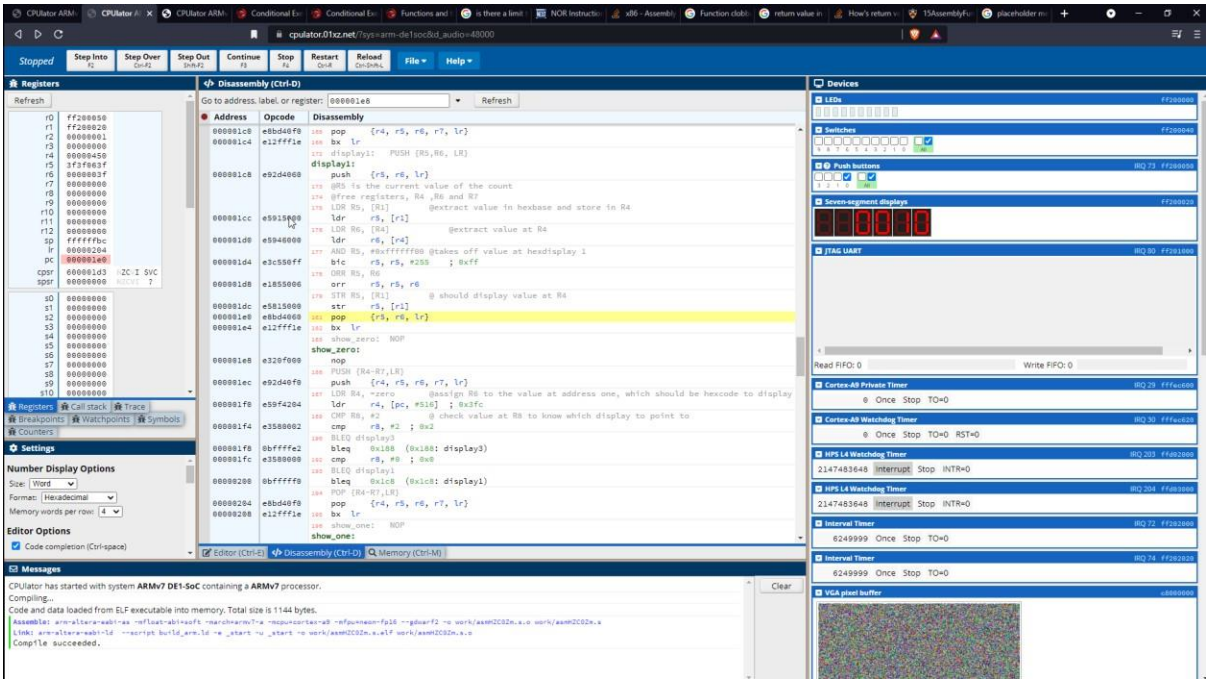
Exhausting c9/ reaching 9 on hex seg 1



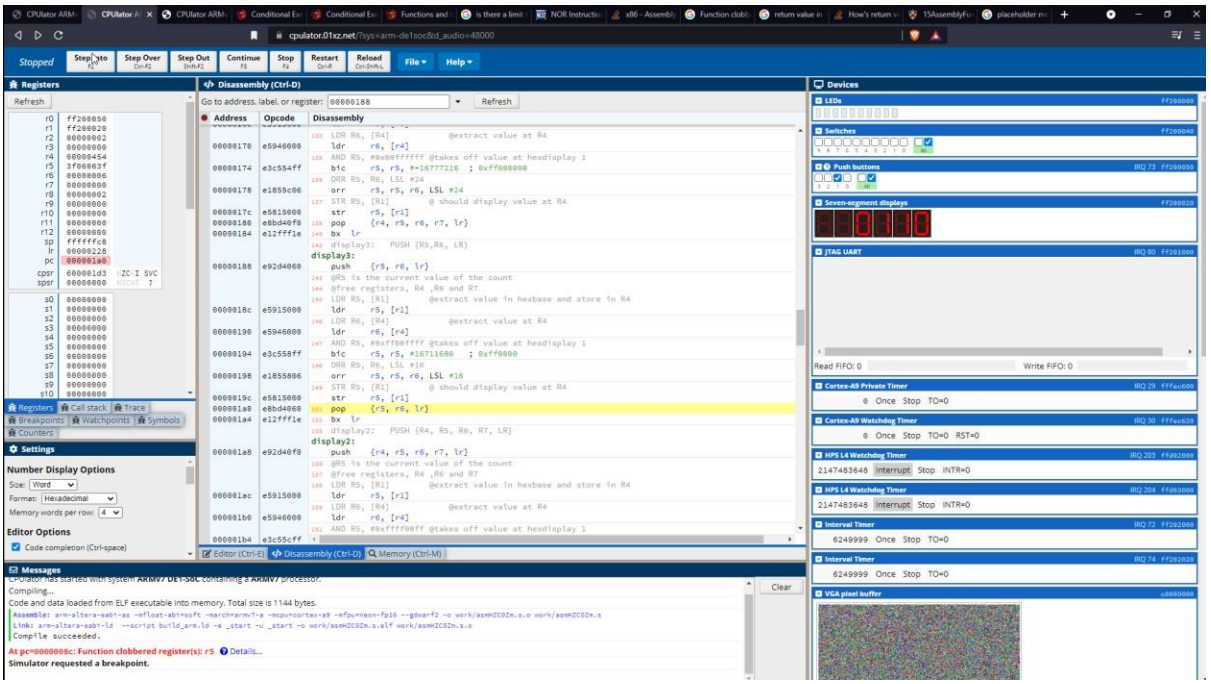
Updating c2 and showing on hex display



C1 and hex display 1 are set to zero on next loop



Updating c3 and hex display 3, second button on



Summary

To summarise, with this coursework I learned the basics of arm assembly and how to manipulate locations in the computer's memory to do some interesting things. Arm takes a slightly different approach in programming in that every command is executed from top to bottom and you are limited to a few commands and variables than are available in c. This coursework I was able to communicate with the button and the hex display in order to update values in memory by the state of the button press and also extract and display those values on the hex.