Microprocessor Systems

ELE 271

Laboratory 1:

Memory Allocation and Intro to Debugging

Due 21JAN2023

By: Nicohlas Bean , Toussaint Falangani ,

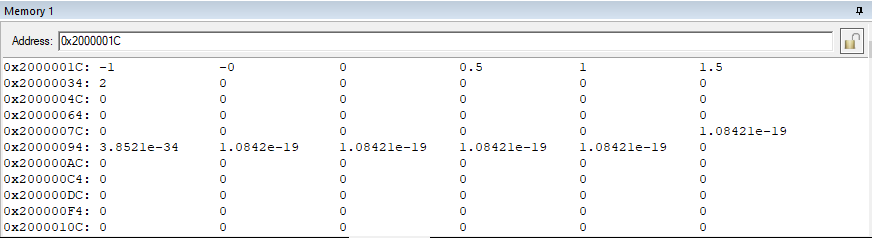
**Introduction**

The overall goal of this lab was to introduce the process of making a program in the c language using the Keil uVision5 software to create a useable project that stored memory onto the microprocessor, examine the memory that is stored in various locations, and also to utilize the debugging function to examine how memory changes over time as a program executes. We used some provided sample code to accomplish this task….

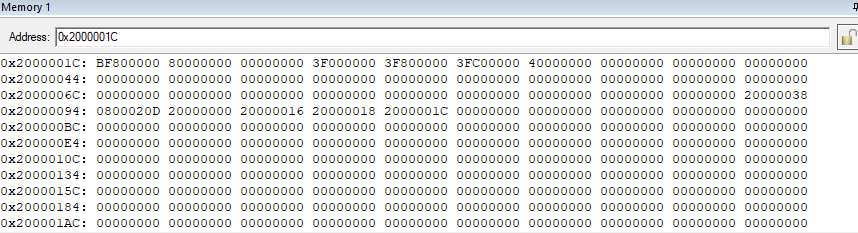
**Experimental Findings**

**Float Decimal and bit values**

The first fundamental difference that was examined was the difference between an integer byte value and a floating point decimal value, below we can see one example of each of the different numerical format occurring at the same place in memory.



(this format shows the information in floating decimal)

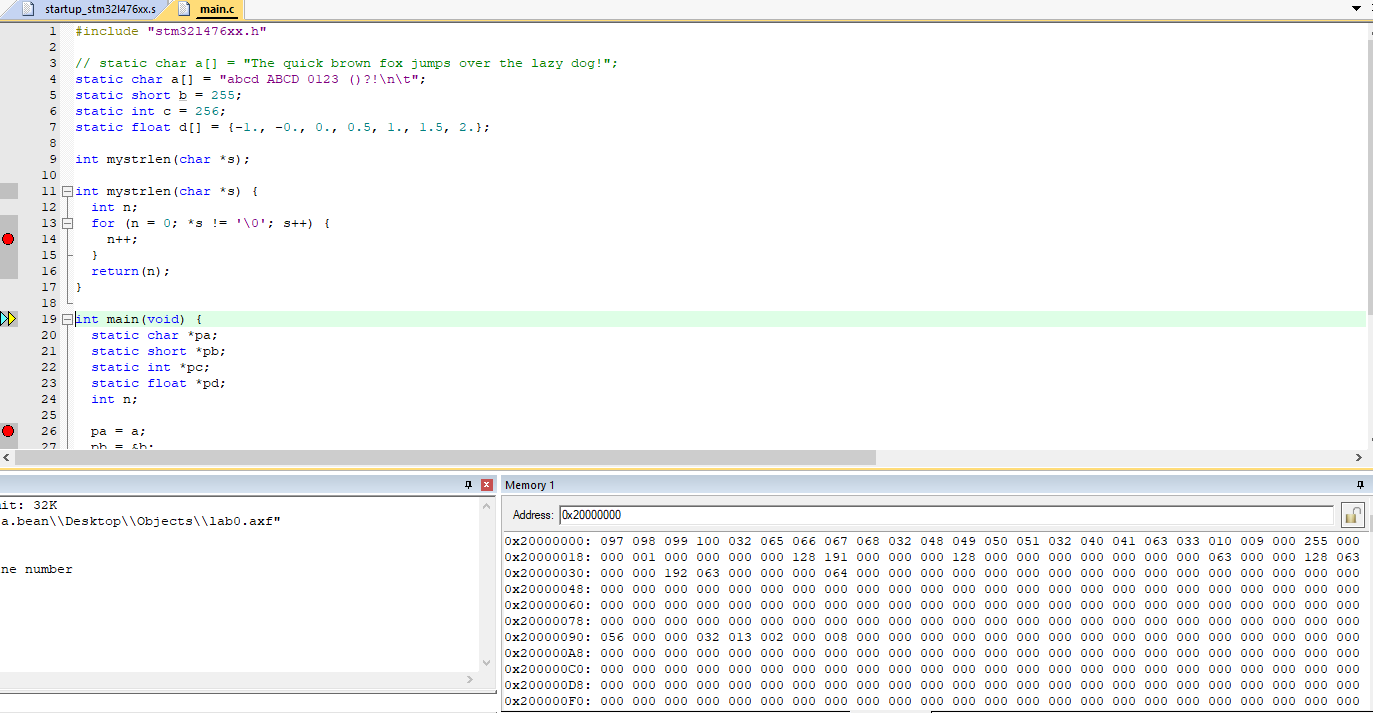


(this format shows the same information in integer byte values, displayed in base 16)

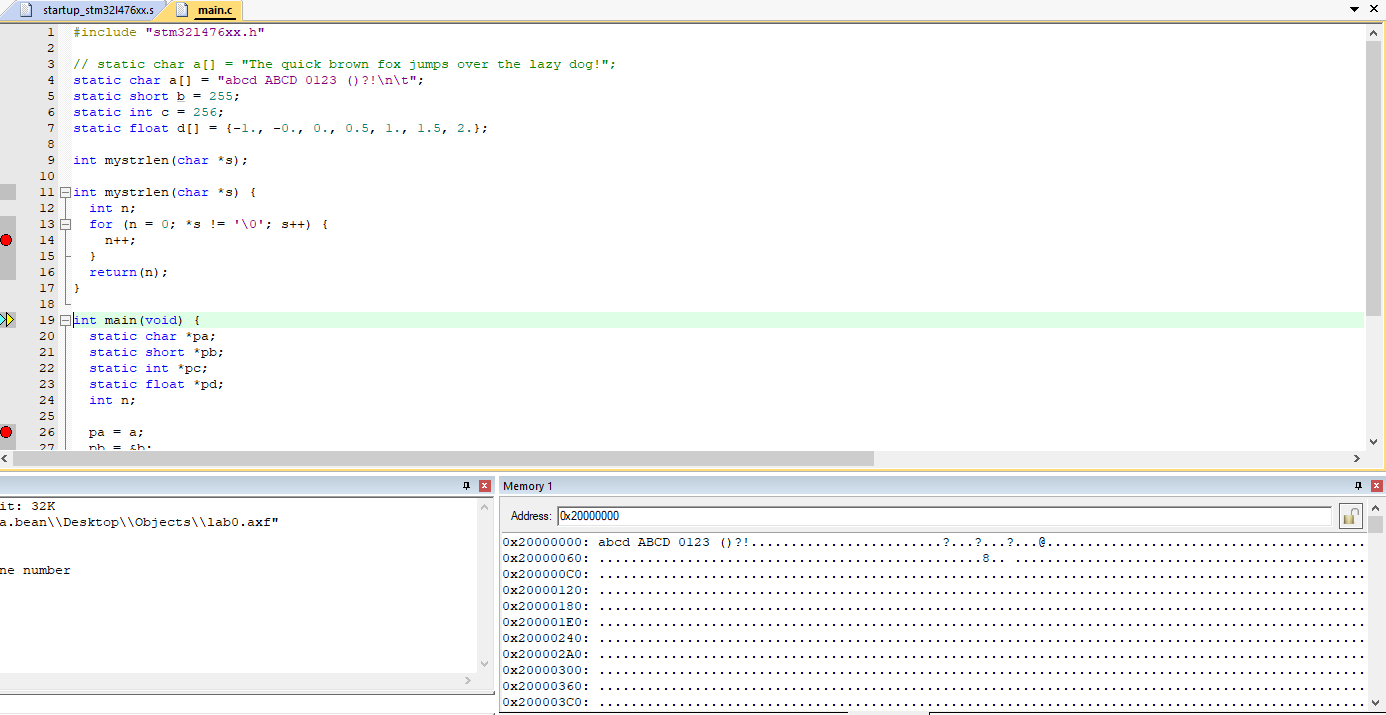
With these two examples we can examine the similarities at various locations to prove the congruence between the various display modes. Starting with perhaps the most obvious being -0 and its twin representation of 80000000. Knowing that a negative value is identified with the first value of a 1 in bit sequence we can convert the first byte of the hex code to ‘10000000’ and the following values are all ‘0’. The floating decimal represents this as -0 because of the first bit value and the subsequent 0’s.

**Char Value Examination**

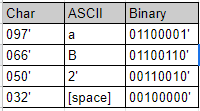
The char value in C is stored as an unsigned 8 bit value ranging between 0 and 255, these values have a corresponding character symbology that can be widely interpreted by many languages and is a national standard. Below we can see the same location in memory in various representations both of the Char value and the ASCII interpretation of that value which is readable to most people, we can even take this a step further and translate the character into 8-bit binary code to see another representation of the memory storage.



(Char values stored as a base 10 number between 0-255)



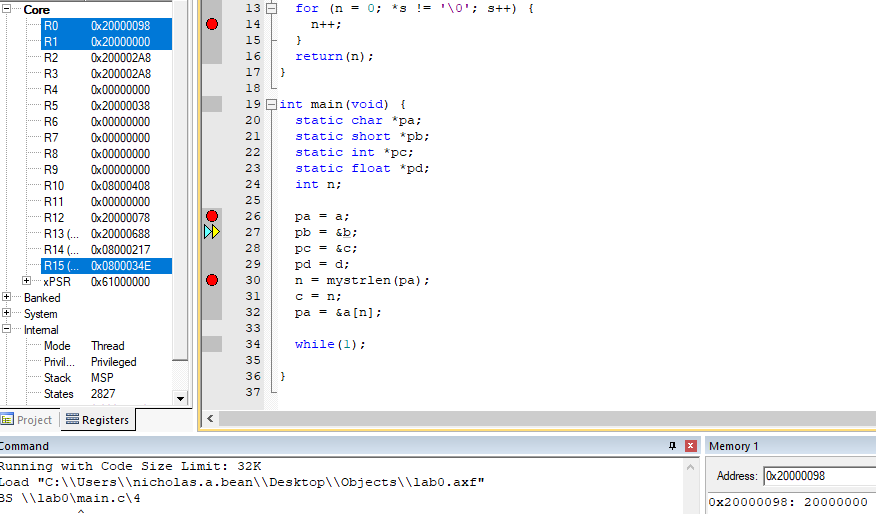
( The same location in memory, displayed as the more readable ASCII values )



This is a simple chart explaining how some of the various locations in memory can be converted between various forms of information display, and how one can use easy markers within the code to avoid laborious translation (like knowing that 032, is the char for [space]). We have also learned that although these landmarks can be helpful, it is generally more useful to view information through whatever lens it will be primarily used as, but not always.

**Memory Addresses within the Stack**

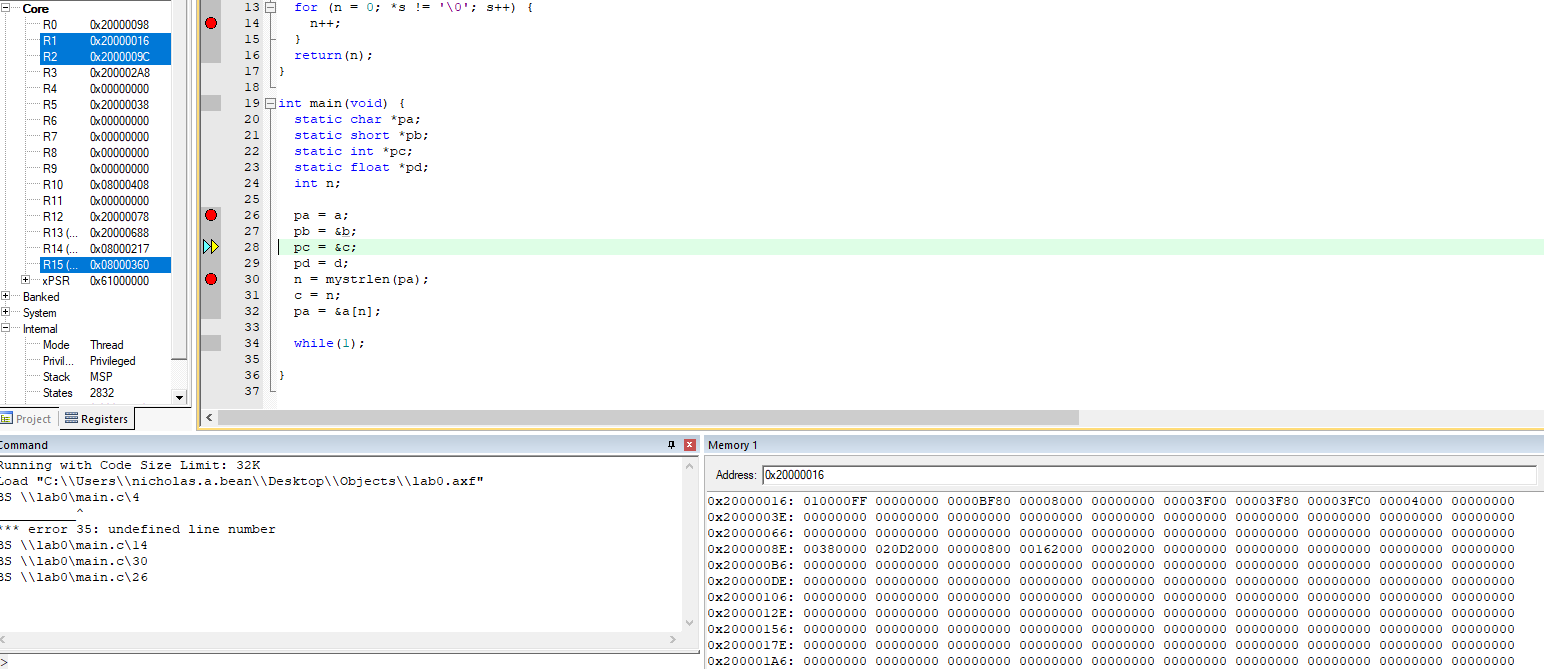
Examining the memory stack within the programming environment, we can see that the various instructions are stored in the stack R0-R15 (it should be noted that the last place in the stack is always valued with a null to signify that the program is ‘finished’). These values tell the processor where to find the data for the allocated command, so the first thing to be called into the stack is always the main function of a program, and from there various auxiliary functions are added to the memory stack as they are called, when the stack ‘executes’ it proceeds in the order of top to bottom, meaning that the last application added to the stack is the first operation executed. Within each location, the various variables created within the function are stored in sub-segments within that memory register, which is stored within the stack.



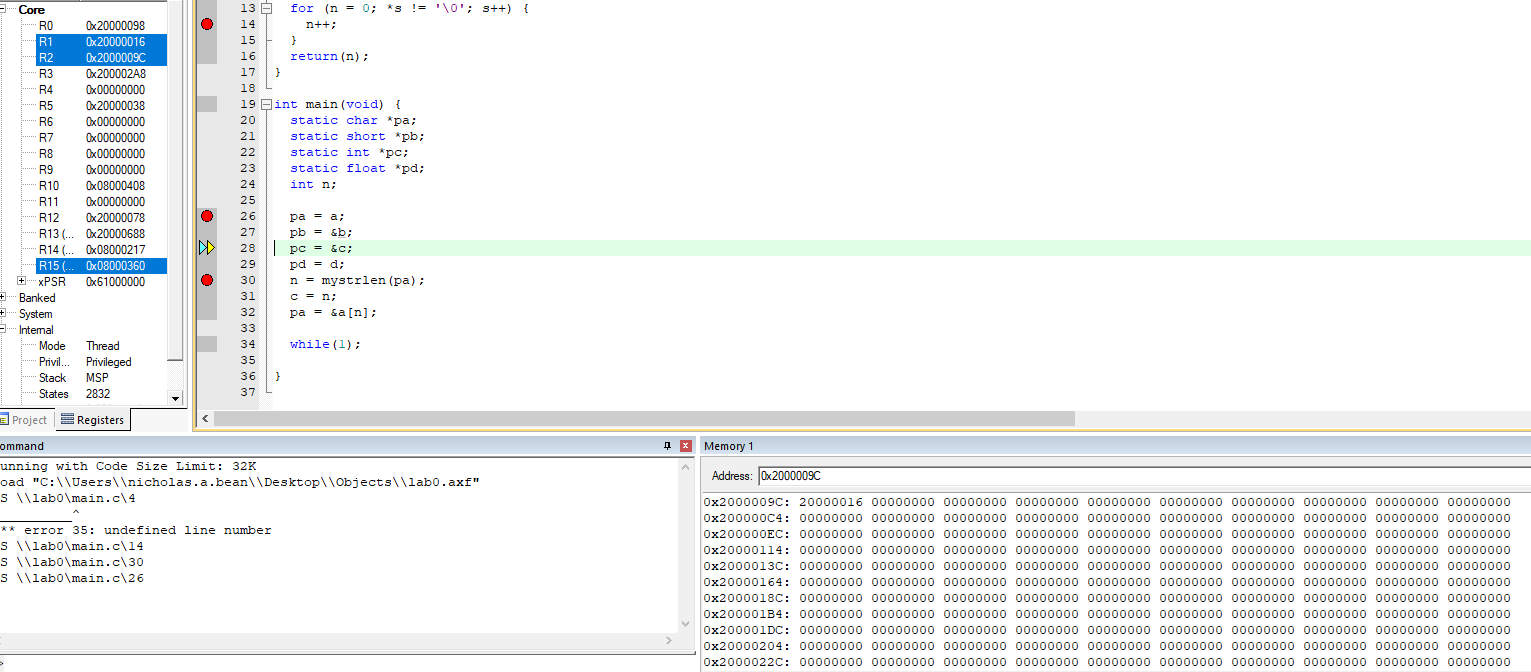
( here we can see that we are currently executing R0 within the stack, and in the lower right hand corner, we can see that that address points to the register 0x20000000 which is the register where further variables can be accessed)

**Memory Pointers**

Memory pointers, as per its definition and use, are variables that store memory addresses. We have observed how the stack R0 to R15 stores the memory addresses of respective values. Additionally we need to understand that the stack pointer is the value of the next used address for assigning stack memory. As a program continues to run ( this program runs indefinitely due to the while(1) loop) it will need new places to store values within memory as they are created, the stack pointer tells the program where the next value is to be inserted into the program.



(This format shows R1 storing or pointing to the memory address for the value given.)



(This format shows R2 storing or pointing to the memory address for the value given.)

**Little vs. Big Endian Storage**

Evaluating the system to discover weather it operates on a big or little endian system we need only evaluate a singular set of binary numbers, in a system that uses big endian the LSB (least significant bit) is located of the left most side of a number series [ meaning 2 would be evaluated as 0100], where a a little endian system would have them represented in the more common method of the LSB being the rightmost digit in a series [ 2 is valued as 0010]. We have determined through all of these various conversions that the kiel system, as well as our individual processors operate on the little endian system, so the left most bit in a sequence is the most significant value and the rightmost is the least.

**Conclusion**

We were successfully able to identify the storage locations of various bits of memory within the microprocessor, and using the ‘next step’ function of the debugging program we were able to see the data increment in a predictable manner. Additionally we attained the skills necessary to convert between various forms of information displayed in the software and are able to convert int to float, char to ascii and vise versa. This lab was successful in its goal to provide a basic introductory ability to predict where various values will be stored within the memory system, and also to fundamentally set-up and begin a project in the Keil software.