Benjamin Linam

[Bml42@students.uwf.edu](mailto:Bml42@students.uwf.edu)

EEL 4744L: Microprocessor Applications Laboratory

Lab 5: Writing Subroutines and Using BUFFALO I/O Routines

2/28/2018

**Objective**

Introduce students to writing subroutines in HC11 assembly language and using the BUFFALO I/O routines to display results.

**Introduction/Background/Theory**

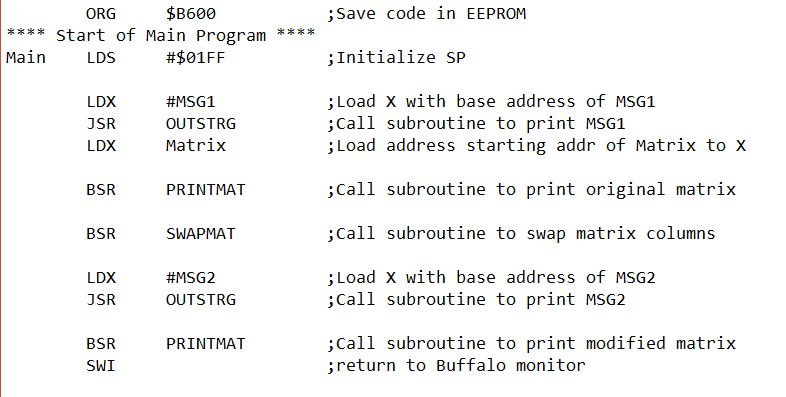
A subroutine is a set of instructions that can be used by a program several times throughout its execution, similar to functions used by other higher-level programming languages such as C. This lab required construction of an assembly program that includes subroutines which when called, display the elements in an NxM matrix in row-major order, as well as swap each of the rows of the array. The point of the lab is to construct each set of instructions to where it may be used several times to output the desired result.

To complete this program, certain BUFFALO monitor I/O subroutines must be used to display information to the console for the user to view. While the subroutines have set addresses in memory, their names are not recognized by the HC11; therefore, variables must be created that refer to their locations. After creating the variable, the desired subroutine can be called upon by using the command JSR to indicate that program execution must “jump” to the desired address of that subroutine.

Jumping between subroutines and the main body of the program is done by use of the stack pointer. The return address for each subroutine is saved to the stack for the program to be able to return to once the subroutine has completed its execution. Not only can addresses be pushed to the stack, but variables and other values can also be saved for easier access.

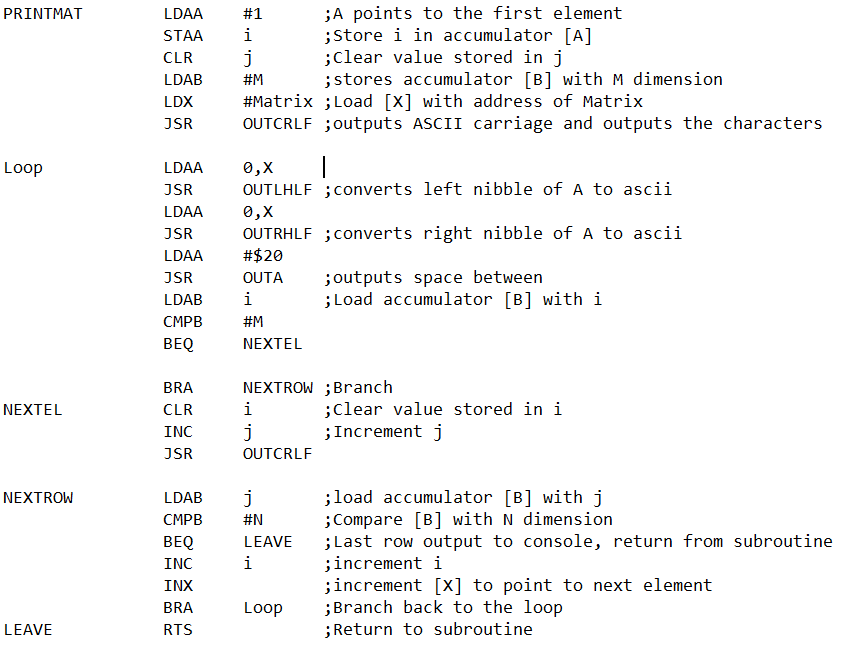
**Procedure**

1. As shown in Fig.1, program execution begins at $B600 with a message that displays that the original matrix will follow before calling the subroutine OUTSTRG to display the elements stored in the matrix.



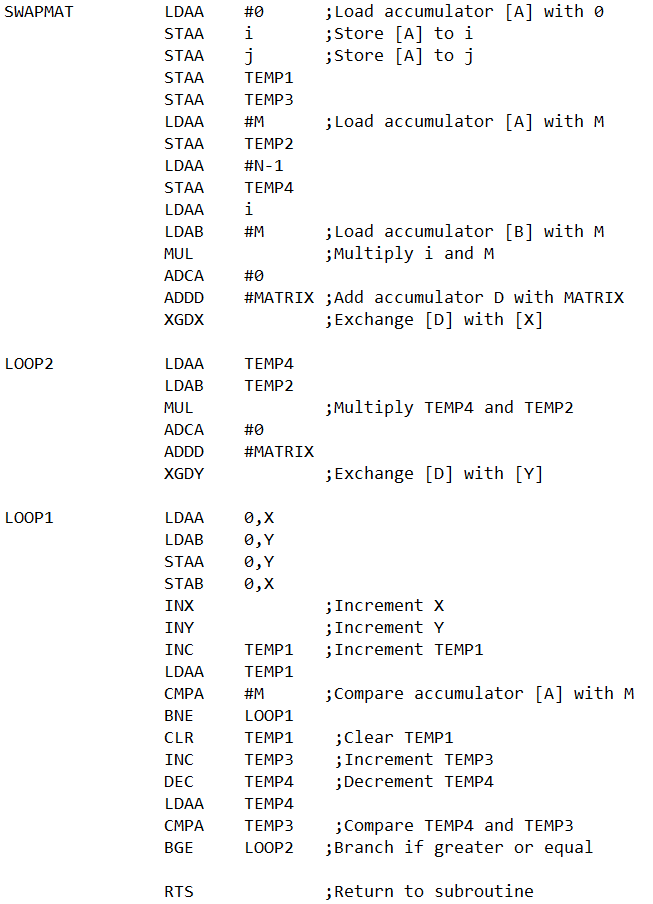
**Figure 1**: Assembly language code showcasing the general layout of the main program which calls subroutines PRINTMAT and SWAPMAT

2. Fig.2 shows that PRINTMAT is then branched to and by incrementing through each element in the matrix as well as calling the OUTLHLF and OUTRHLF subroutines, each value is displayed to the console.



**Figure 2**: Assembly language code for PRINTMAT subroutine

3. After the matrix is displayed and the return is reached at the end of PRINTMAT, program execution resumes in the main portion of the function. The next instruction is for the program to branch to the SWAPMAT subroutine, shown in Fig.3.



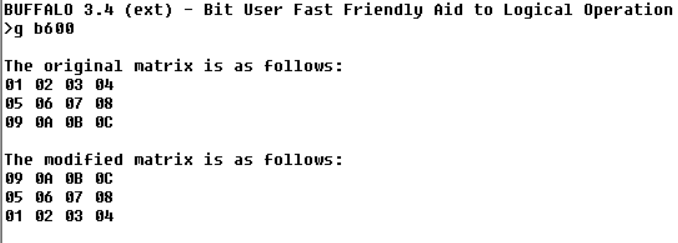
**Figure 3**: Assembly language code for SWAPMAT subroutine

4. The subroutine has full access to the matrix by incrementing through each of its elements, and by use of temporary variables, it swaps each element to its desired location. Elements in the first row are swapped with the last row, second row elements are saved to the second to last row, etc. If there are an odd number of rows, the middle row remains untouched. Once the middle of the matrix is reached and there are no remaining swaps to be made, the subroutine returns to the main program.

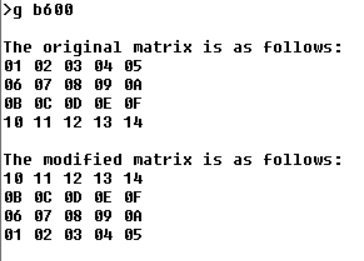
5. OUTSTRG is used again to display a message stating that the modified matrix will follow it. Then the OUTCLRF subroutine moves the cursor to the next line and PRINTMAT is called for the last time to display the values stored in the matrix.

6. After having compiled the .s19 file and uploading it to the HC11, typing “g b600” in the BUFFALO monitor will display the following represented by Fig.4 and Fig.5.

7. The two sets of original matrix values were provided during the lab and had to be manually entered in the .asm file before compiling and uploading it each time.



**Figure 4**: BUFFALO I/O console displaying the 3x4 array before and after the swap



**Figure 5**: BUFFALO I/O console displaying the 4x5 array before and after the swap

**Conclusions**

The program was by far more difficult to complete than the previous two; however, it was demonstrated successfully without error. If this program were to have been written in the C language, it would have been much easier, but with assembly being a lower-level language, the loops had to be entered manually as well as many of the other instructions that C greatly simplifies. Being restricted to using the registers for almost every operation was another point of concern that in C, would have been much more simple.