EXPERIMENT #2

TUTOR COMMAND UTILIZATION and PROGRAM EXPERIMENTATION

1.0 Purpose

The purpose of this experiment is to acquaint the user with the TUTOR monitor program and with the 68000 instruction set.

2.0 Component Requirements

None.

3.0 Background

One of the best methods for learning the MC68000 Instruction Set is to write an assembly language program, enter it into the development system, execute it and debug it. In this experiment, the user will be introduced to various MC68000 instructions and addressing modes. The user will enter the code into the SANPER-1 ELU through the use of the TUTOR Commands.

4.0 Statement of Problem

In this experiment, several sample programs are presented that may require modifications. The user will be entering, executing, and debugging these programs, and then answering questions that are raised throughout the programs.

5.0 Preliminary Assignment

1. Completely read all seven sample programs to familiarize yourself with their objectives.
2. Review the 68000's instructions and addressing modes. The user should read both the "Data Organization and Address Capabilities" section and the "Instruction Set Summary" section in the M68000 Programmer's Reference Manual.
3. Review the TUTOR Command Set, which is described in the MC68000 Educational Computer Board User's Manual.
4. Procedure #8 of sample Program 2.6 requires that the student write a short MC68000 Assembly Language program. This student should have this program written before coming to the lab.

6.0 Procedure

Execute each sample program and record data when requested.

7.0 Discussion

Answer any questions that were asked in each sample program, and submit your answers as a final report to your Lab Instructor.

SAMPLE PROGRAM 2.1 - Fill a Block of Memory

Purpose:

This program fills any size block of memory with any desired value. In addition, this program introduces the user to TUTOR's program entering, executing and debugging commands.

Given Information:

1. Starting address of memory block to be filled = $3100
   (use an Address Register as a pointer)
2. Ending address of memory block to be filled = $3200
   (use an Address Register as a pointer)
3. Data to be stored = $FFFF
   (use a Data Register to hold the data)
4. Starting address of program = $300C

Procedure:

1. Enter the following program by using TUTOR's Memory Modify (MM) Command, and invoking the disassembler.

```
TUTOR 1.X > MM $300C;DI <CR>
```

Address	Instruction	Comments
00300C	**MOVE.W D0,(A0)+ <CR>**	To be determined by user
00300E	**CMP.W A0,A1 <CR>**	
003010	**BGE $300C <CR>**	
003012	**MOVE.B #228,D7 <CR>**	
003016	**TRAP #14 <CR>**	
003018	**. <CR>**	

2. Examine the program and determine what registers need to be initialized, and what values to initialize them to. Use TUTOR's Individual Register Set/Change Command for this purpose.

3. Use TUTOR's Block Fill (BF) Command to initialize the memory block to a known value.

   ```
   TUTOR 1.3 > BF $3100 $3300 $AAAA <CR>
   ```

   TUTOR Responds by outputting the following:

   ```
   PHYSICAL ADDRESS=00003100 00003300
   ```

   This statement indicates that the command was successfully completed.

4. Execute a Memory Display (MD) Command to ensure that the memory fill operation was performed properly.

   ```
   TUTOR 1.3 > MD $3100 <CR>
   ```

   16 bytes of data will be displayed. Each byte should be $AA. By typing in just a <CR>, the user can continuously display 256 bytes of data. Verify that all locations from $3100 to $3300 are filled with $AA.

5. Run the program using the GO (G) Command.

   ```
   TUTOR 1.3 > G $300C <CR>
   ```

   TUTOR will display the following on the screen.

   ```
   PHYSICAL ADDRESS=0000300C
   ```

   and the TUTOR prompt will appear.

6. Execute a Memory Display (MD) Command to ensure that the program worked properly.

   ```
   TUTOR 1.3 > MD $3100 <CR>
   ```

   If the program executed correctly, $FF should be stored in each memory location from $3100 to $3200, and $AA should be stored in each memory location from $3202 to $3300.

7. Demonstrate to your Lab Instructor that the program works correctly.

8. Trace through the program by first initializing the Program Counter (PC), and then invoking TUTOR's Trace (T) Command.

   ```
   TUTOR 1.3 > .PC $300C <CR>
   ```

   and then typing in:

   ```
   TUTOR 1.3 > T <CR>
   ```

   Note that TUTOR will display all the registers and will show the next instruction to be executed. Notice that the prompt now changes to:

   ```
   TUTOR 1.3 :>
   ```

   Note: It is not necessary to trace completely through the TRAP #14 instruction.

9. Modify the program so that it performs the same function but uses the pre-decrement addressing mode instead of the post-increment addressing mode for the instruction at

address $300C. To enter changes to the existing program, use TUTOR's Memory Modify (MM) Command, and invoke the disassembler.

10. Use TUTOR's Block Fill (BF) Command to initialize the memory block to some known value.

    ```
    TUTOR 1.3 > BF $3100 $3300 $BBBB <CR>
    ```

    TUTOR responds by displaying the following:

    ```
    PHYSICAL ADDRESS=00003100 00003300
    ```

    This statement indicates that the command was successfully completed.

11. Execute a Memory Display (MD) Command to ensure that the memory fill operation was performed properly.

    ```
    TUTOR 1.3 > MD $3100 <CR>
    ```

    16 bytes of data will be displayed. Each byte should be $BB. By typing in just a <CR>, the user can continuously display 256 bytes of data. Verify that all locations from $3100 to $3300 are filled with $BB.

12. Run the program using the GO (G) Command.

    ```
    TUTOR 1.3 > G $300C <CR>
    ```

    TUTOR will display the following on the screen.

    ```
    PHYSICAL ADDRESS=0000300C
    ```

    and the TUTOR prompt will appear.

13. Execute a Memory Display (MD) Command to ensure that the program worked properly.

    ```
    TUTOR 1.3 > MD $3100 <CR>
    ```

    If the program executed correctly, $FF should be stored in each memory location from $3100 to $3200, and $BB should be stored in each memory location from $3202 to $3300.

14. Demonstrate to your Lab Instructor that the program works correctly.

Discussion:

Submit the following to your Lab Instructor as part of the Final Report:

1. A fully commented version of the original program (it must include both global and local comments)

2. A fully commented version of the program written for Procedure #9. (it must include both global and local comments)

3. Discuss the function of each register used in the original program.

4. Discuss the advantages of the pre-decrementing and post-incrementing addressing modes.

SAMPLE PROGRAM 2.2 - Outputting a Character to the Terminal

Purpose:

This program introduces the user to the concept of TRAPs, and to TUTOR's TRAP 14 Handler. This program will use one of the TRAP routines to repeatedly output a single character to the terminal.

Given Information:

The character to be outputted is the capital letter 'A'.

Procedure:

1. Enter the following program by using TUTOR's Memory Modify (MM) Command, and invoking the disassembler.

```
TUTOR 1.3 > MM $900;DI <CR>
```

Address	Instruction	Comments
000900	**MOVE.B #$??,D0 <CR>**	To be determined by user
000904	**MOVE.B #248,D7 <CR>**	
000908	**TRAP #14 <CR>**	
00090A	**MOVE.L #$FFFF,D5 <CR>**	
000910	**DBEQ D5,$910 <CR>**	
000914	**BRA $900 <CR>**	
000916	**. <CR>**	

2. Determine the value of the expression ?? in the instruction at address $900, and insert the proper value into the instruction.

3. Run the program using the GO (G) Command.

   ```
   TUTOR 1.3 > G $900 <CR>
   ```

   TUTOR will display the following on the screen.

   ```
   PHYSICAL ADDRESS=00000900
   ```

   and the letter A should repeatedly appear on the screen.

4. Press the ABORT Switch on the SANPER-1 Unit's Front Panel in order to break out of the program's loop.

5. Change the original character (A) to another ASCII character by sequencing through the program byte by byte until the character A is encountered and modify this

location to the ASCII value for the number 5. Use the Memory Modify (MM) Command to do this but to not invoke the disassembler.

```
TUTOR 1.3 > MM $900 <CR>
```

6. Run the program using the GO (GO) Command.

```
TUTOR 1.3 > G $900 <CR>
```

TUTOR will display the following on the screen.

```
PHYSICAL ADDRESS=00000900
```

and the number 5 should repeatedly appear on the screen.

7. Press the ABORT Switch on the SANPER-1 Unit's Front Panel in order to break out of the program's loop.

8. Demonstrate to your Lab Instructor that the program works correctly.

9. Change the value $FFFF to $000F in the instruction at address $90E. You may either retype the entire instruction or simply change the bytes of code where $FFFF is located as shown below.

```
TUTOR 1.3 > MM $90E <CR>

00090E  FF ? 00 <CR>
00090F  FF ? 0F <CR>
000910  XX ? . <CR>
```

10. Run the program using the GO (G) Command.

```
TUTOR 1.3 > G $900 <CR>
```

TUTOR will display the following on the screen.

```
PHYSICAL ADDRESS=00000900
```

By examining the terminal screen, determine what differences, if any, have occurred since the last time you ran the program. In other words, what effect did procedure #9 have on the program? Record your findings.

11. Press the ABORT Switch on the SANPER-1 Unit's Front Panel in order to break out of the program's loop.

Discussion:

Submit the following to your Lab Instructor as part of the Final Report:

1. A fully commented version of the original program (it must include both local and global comments).

2. List the and explain the results of procedure #10

3. Write a subroutine that outputs any character once. The character to be outputted will be passed to this subroutine through Data Register D1. Note: you are not required to execute this program on the SANPER-1 ELU.

4. What is the effect of changing the instruction at address $914 to "BRA 904"?

5. Outline the steps involved in the execution of the instructions at addresses $90A and $910. Discuss the usefulness of this combination of instructions.

6. List the major benefits of using TRAP instructions.

SAMPLE PROGRAM 2.3 - Outputting a String to the Terminal

Purpose:

The purpose of this program is to demonstrate the capability of another TRAP 14 Handler function available within the TUTOR firmware. We will use a function that outputs a string to the terminal.

Given Information:

1. Starting address of the string buffer = $1000
2. Ending address of the string buffer = $1017
3. TUTOR's TRAP Function No. 227 outputs a string followed by a carriage return (CR) and a line feed (LF) to Port 1 (the terminal port).
4. TUTOR's TRAP Function No. 228 returns program control from the user program back to TUTOR.
5. Starting address of the program = $950

Procedure:

1. Enter the following program by using TUTOR's Memory Modify (MM) Command, and invoking the disassembler.

```
TUTOR 1.3 > MM $950;DI <CR>
```

Address	Instruction	Comments
000950	**MOVE.L #$??,A5 <CR>**	To be determined by user
000956	**MOVE.L #$??,A6 <CR>**	
00095C	**MOVE.B #227,D7 <CR>**	
000960	**TRAP #14 <CR>**	
000962	**MOVE.B #228,D7 <CR>**	
000966	**TRAP #14 <CR>**	
000968	**. <CR>**	

2. Determine the value of the expression ?? in the instructions at addresses $950 and $956, and insert the value into the instruction.

3. Use the Memory Set (MS) Command to load a string into memory.

```
TUTOR 1.3 > MS $1000 'WELCOME TO THE 68000 LAB'
```

This command loads the equivalent ASCII character value of each letter into memory. This is the string that will be outputted to the terminal.

4. To ensure that the string was entered into memory correctly, use TUTOR's Block of Memory Search (BS) Command to search user memory for the string.

```
TUTOR 1.3 > BS $900 $3FFF 'WELCOME TO THE 68000 LAB'
```

If the string was properly stored, TUTOR will display the following:

```
PHYSICAL ADDRESS=00000900 00003FFF
001000 'WELCOME TO THE 68000 LAB'
```

5. Run the program using the GO (G) Command.

```
TUTOR 1.3 > G $950 <CR>
```

TUTOR will display the following on the screen.

```
PHYSICAL ADDRESS=00000950
```

and the string 'WELCOME TO THE 68000 LAB' should appear once, followed by the TUTOR prompt on the next line.

6. If your program is not working properly, use the Trace (T) and Breakpoint (BR) Commands to debug it.

7. Demonstrate to your Lab Instructor that your program works correctly.

Discussion:

Submit the following to your Lab Instructor as part of the Final Report:

1. A fully commented version of the original program (it must include both global and local comments)

2. Discuss how you would have implemented this program were TRAP Function No. 227 not available.

3. After executing the program, what is the final value of A5, and why?

SAMPLE PROGRAM 2.4 - Pattern Match

Purpose:

This program compares two ASCII strings in memory to determine if they are identical.

Given Information:

1. The starting address of string #1 is $2000.
2. The starting address of string #2 is $3000.
3. The first byte of each string specifies the string length.
4. If the strings match, store $00 in location $1100.
5. If the strings do not match, store $FF in location $1100.
6. Data register D0 holds the length of the string.
7. Data register D1 holds the result of the comparison.
8. The starting address of the program is $1000.

Procedure:

1. Enter the following program by using TUTOR's Memory Modify (MM) Command, and invoking the disassembler.

```
TUTOR 1.3 > MM $1000;DI <CR>
```

```
Address   Instruction                Comments
001000    MOVE.L #$????,A0 <CR>      To be determined by user
001006    MOVE.L #$????,A1 <CR>
00100C    MOVEQ.L #-1,D1 <CR>
00100E    MOVEQ.L #0,D0 <CR>
001010    MOVE.B (A0),D0 <CR>
001012    CMPM.B (A0)+,(A1)+ <CR>
001014    DBNE D0,$???? <CR>
001018    BNE.S $101C <CR>
00101A    NOT.B D1 <CR>
00101C    MOVE.B D1,$1100 <CR>
001020    MOVE.B #228,D7 <CR>
001024    TRAP #14 <CR>
001026    . <CR>
```

2. Determine the value of the expression ???? in the instructions at addresses $1000, 1006, and $1014 and insert the correct value into the appropriate instruction.

3. Load two strings into memory using the Memory Set (MS) Command.

```
TUTOR 1.3 > MS $2001 'MC68000 MICROPROCESSOR'
TUTOR 1.3 > MS $3001 'MC68000 MICROPROCESSOR'
```

4. Verify that the strings were properly stored by searching all of user memory to determine if the strings are present. Use TUTOR's Block of Memory Search (BS) Command.

```
TUTOR 1.3 > BS $900 $3FFF 'MC68000 MICROPROCESSOR'
```

If the strings were properly stored, TUTOR will display the following:

```
PHYSICAL ADDRESS=00000900 00003FFF
002001 'MC68000 MICROPROCESSOR'
003001 'MC68000 MICROPROCESSOR'
```

5. Load the string length into the first byte of each string.

   A. Load in the number of bytes (in hexadecimal) for string #1.

```
TUTOR 1.3 > MM $2000 <CR>

002000   XX ? 16 <CR>
002001   xx ? . <CR>
```

   B. Load in the number of bytes (in hexadecimal) for string #2.

```
TUTOR 1.3 > MM $3000 <CR>

003000   XX ? 16 <CR>
003001   XX ? . <CR>
```

6. To ensure that the program works properly, we will initialize the comparison result location before running the program. We will load $AAAA into location $1100 using the Memory Modify (MM) Command.

```
TUTOR 1.3 > MM $1100 <CR>
001100  XX ? AA <CR>
001101  XX ? AA <CR>
001102  XX ? . <CR>
```

7. Run the program using the GO (G) Command.

```
TUTOR 1.3 > G $1000 <CR>
```

TUTOR will display the following on the screen:

```
PHYSICAL ADDDRESS=00001000
```

and then the TUTOR prompt will appear.

8. Examine the contents of location $1100 to determine the result of the comparison.

```
TUTOR 1.3 > MM $1100 <CR>

001100  00 ? <CR>
001101  AA ? . <CR>
```

Did the strings match or not?

9. Modify the text of string #2 from "68000" to "68001" by changing the contents of location $3007 from $30 to $31.

```
TUTOR 1.3 > MM $3007 <CR>

003007  30 ? 31 <CR>
003008  20 ? . <CR>
```

10. To verify that the program works properly, we will once again initialize the comparison result location before running the program.

```
TUTOR 1.3 > MM $1100 <CR>

001100  XX ? AA <CR>
001101  XX ? AA <CR>
001102  XX ? . <CR>
```

11. Run the program using the GO (G) Command.

```
TUTOR 1.3 > G $1000 <CR>
```

TUTOR will display the following on the terminal:

```
PHYSICAL ADDDRESS=00001000
```

and then the TUTOR prompt will appear.

12. Examine the contents of location $1100 to determine the result of the comparison.

```
TUTOR 1.3 > MM $1100 <CR>

001100  FF ? <CR>
001101  XX ? . <CR>
```

Did the strings match or not?

13. Demonstrate to your Lab Instructor that your program works correctly.

Discussion:

Submit the following to your Lab Instructor as part of the Final Report:

1. A fully commented version of the original program (it must include both global and local comments).

2. Draw a flowchart for the program.

3. Describe the differences between the MOVE and MOVEQ instructions. Under what conditions is it advantageous to use one instruction over the other?

4. Discuss the usefulness of the CMPM instruction.

5. What instruction sets the Condition Code bits for the BNE instruction at address $1018?

SAMPLE PROGRAM 2.5 - Bubble Sort

Purpose:

This program sorts a string of elements (each 16-bits wide), in such a manner that when the sort is completed, the largest number will be at the lowest memory location, and the smallest number will be at the highest memory location. Thus the large numbers bubble up to low memory.

Given Information:

1. Address register A0 points to the first element in the list, which is located at address $2100.
2. Address register A1 points to the last element in the list, which is located at address $210E.
3. Address register A2 holds a copy of the starting address of the list.
4. Starting address of the program = $2000.

Procedure:

1. Enter the following program by using TUTOR's Memory Modify (MM) Command, and invoking the disassembler.

```
TUTOR 1.3 > MM $2000;DI <CR>
```

Address	Instruction	Comments
002000	**MOVE.L A0,A2 <CR>**	To be determined by user
002002	**MOVE.L A2,A0 <CR>**	
002004	**CMP.W (A0)+,(A0)+ <CR>**	
002006	**BHI.S $2014 <CR>**	
002008	**SUBQ.L #2,A0 <CR>**	
00200A	**CMP.L A0,A1 <CR>**	
00200C	**BNE $2004 <CR>**	
00200E	**MOVE.B #228,D7 <CR>**	
002012	**TRAP #14 <CR>**	
002014	**MOVE.L -(A0),D0 <CR>**	
002016	**SWAP.W D0 <CR>**	
002018	**MOVE.L D0,(A0) <CR>**	
00201A	**BRA $2002 <CR>**	
00201C	**. <CR>**	

2. Use the Memory Modify (MM) Command to load a table of data into memory.

```
TUTOR 1.3 > MM $2100;W <CR>

002100   XXXX ? 0000 <CR>
002102   XXXX ? 0010 <CR>
002104   XXXX ? 0020 <CR>
002106   XXXX ? 0030 <CR>
002108   XXXX ? 0040 <CR>
```

```
00210A   XXXX ? 0050 <CR>
00210C   XXXX ? 0060 <CR>
00210E   XXXX ? 0070 <CR>
002110   XXXX ? . <CR>
```

where XXXX = previous data values which we treat as don't cares.

3. Initialize Address Register A0 to the start of the string and Address Register A1 to the end of the string, by using TUTOR's Individual Display / Set Register Command.

```
TUTOR 1.3 > .A0 $2100 <CR>
TUTOR 1.3 > .A1 $210E <CR>
```

4. Examine the contents of Address Registers A0 and A1 to ensure that they were properly initialized.

```
TUTOR 1.3 > .A0 <CR>
```

TUTOR responds with:

```
A0=00002100
```

```
TUTOR 1.3 > .A1 <CR>
```

TUTOR responds with:

```
A1=0000210E
```

5. Run the program using the GO (G) Command.

```
TUTOR 1.3 > G $2000 <CR>
```

TUTOR will display the following on the screen:

```
PHYSICAL ADDRESS=00002000
```

and then the TUTOR prompt will appear.

6. Examine the contents of memory to determine if the sort operation was successfully completed. If the sort was performed correctly, the data should appear in memory as shown below.

Use the Memory Modify (MM) Command to examine the table entries.

```
TUTOR 1.3 > MM $2100;W <CR>

002100   0070 ? <CR>
002102   0060 ? <CR>
002104   0050 ? <CR>
002106   0040 ? <CR>
```

```
002108   0030 ? <CR>
00210A   0020 ? <CR>
00210C   0010 ? <CR>
00210E   0000 ? <CR>
002110   XXXX ? . <CR>
```

7. Demonstrate to you Lab Instructor that you program works correctly.

Discussion:

Submit the following to your Lab Instructor as part of the Final Report:

1. A fully commented version of the original program (it must include both global and local comments).

2. Examine the program and describe how the sorting algorithm has been implemented.

3. Describe the significance of the SWAP instruction. Assume for a moment that the 68000 does not have a SWAP instruction. List the set of instructions, in the proper sequence that are necessary to replace the SWAP instruction.

4. Describe the function and advantages of the ADDQ and SUBQ instructions.

5. Describe the differences between the ADD and ADDQ instructions.

6. Describe the differences between the SUB and SUBQ instructions.

7. Describe the sequence of events that occurs during the execution of the instruction located at address $2004.

SAMPLE PROGRAM 2.6 – Adding a Number to a Sorted List

Purpose:

This program inserts a new number into the proper location within a sorted or sequenced list. This program is to be run only after executing Sample Program 2.5.

Given Information:

1. Address register A0 points to the first element in the list.
2. Address register A1 points to the last element in the list.
3. Data is moved up in memory (towards lower addresses) to make room for the new number.
4. The starting address of the program is $3000.

Procedure:

1. Enter the following program by using TUTOR's Memory Modify (MM) Command, and invoking the disassembler.

```
TUTOR 1.3 > MM $3000;DI <CR>
```

Address	Instruction	Comments
003000	**CMP.W (A0),D0 <CR>**	To be determined by user
003002	**BCC $300C <CR>**	
003004	**MOVE.W (A0),-(A0) <CR>**	
003006	**ADDQ #4,A0 <CR>**	
003008	**CMPA.L A0,A1 <CR>**	
00300A	**BCC $3000 <CR>**	
00300C	**MOVE.W D0,-(A0) <CR>**	
00300E	**MOVE.B #228,D7 <CR>**	
003012	**TRAP #14 <CR>**	
003014	**. <CR>**	

2. Initialize Data Register D0 to the value of the new word of data.

```
TUTOR 1.3 > .D0 $0055 <CR>
```

Examine the data register.

```
TUTOR 1.3 > .D0 <CR>
```

TUTOR displays the following:

```
D0=00000055
```

3. Initialize Address Registers A0 and A1 by using TUTOR's Individual Display/Set Register Command.

```
TUTOR 1.3 > .A0 $2100 <CR>
TUTOR 1.3 > .A1 $210E <CR>
```

4. Using TUTOR's Individual Display / Set Register Command, verify that Address Registers A0 and A1 were initialized correctly.

   ```
   TUTOR 1.3 > .A0 <CR>
   ```

   TUTOR responds as follows:

   ```
   A0=00002100
   ```

   ```
   TUTOR 1.3 > .A1 <CR>
   ```

   TUTOR responds as follows:

   ```
   A1=0000210E
   ```

5. Run the program using the GO (G) Command.

   ```
   TUTOR 1.3 > G $3000 <CR>
   ```

   TUTOR displays the following:

   ```
   PHYSICAL ADDRESS=00003000
   ```

   and then the TUTOR prompt appears.

6. Examine the sequenced list to determine if the insert operation was performed properly. The list should appear in memory as follows:

   ```
   TUTOR 1.3 > MM $20FE;W <CR>

   0020FE  0070 ? <CR>
   002100  0060 ? <CR>
   002102  0055 ? <CR>
   002104  0050 ? <CR>
   002106  0040 ? <CR>
   00210A  0020 ? <CR>
   00210C  0010 ? <CR>
   00210E  0000 ? <CR>
   002110  XXXX ? . <CR>
   ```

   Notice that the list grew towards lower memory addresses.

7. Demonstrate to your Lab Instructor that the program works properly.

8. Write a program that inputs a 4-digit hexadecimal number (word) from the terminal and inserts it into the proper location in the sorted list. You will have to input the

number from the keyboard, convert it from ASCII to hexadecimal and then call the original insertion routine. Hint: Use TUTOR's Trap 14 Handler.

9. Demonstrate to your Lab Instructor that the program from Procedure Step #8 works correctly.

Discussion:

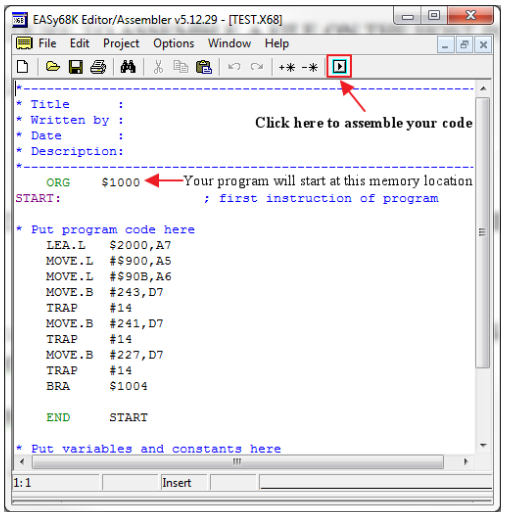
Submit the following to your Lab Instructor as part of the Final Report:

1. A fully commented version of both programs (they must include both global and local comments)

2. Draw a flowchart of the program, and discuss the insertion algorithm.

PROCEDURES FOR INTERFACING A HOST PC TO THE SANPER-1 ELU

1. Type in your code on EASy68K Editor/Assembler and save it in a desired folder.
2. Press ▶ button to assemble your code


EASy68K Editor/Assembler v5.12.29 - [TEST.X68]
File Edit Project Options Window Help
Click here to assemble your code
Your program will start at this memory location
1: 1
Insert

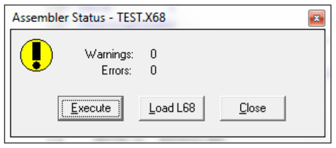
```
*-----------------------------------------------------------
* Title      :
* Written by :
* Date       :
* Description:
*-----------------------------------------------------------
    ORG    $1000
START:                  ; first instruction of program

* Put program code here
    LEA.L   $2000,A7
    MOVE.L  #$900,A5
    MOVE.L  #$90B,A6
    MOVE.B  #243,D7
    TRAP    #14
    MOVE.B  #241,D7
    TRAP    #14
    MOVE.B  #227,D7
    TRAP    #14
    BRA     $1004

    END     START

* Put variables and constants here
```

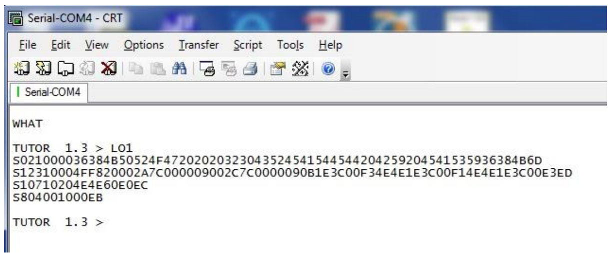
3. If you see the following pop-up window, your code is flawless. If the window indicates errors or warnings, please go back and correct your code.



4. Close the window by clicking 'Close' button. Now, go to the folder where you saved your code in Step 1. You'll see a file with an extension of .S68 created in the folder. This is the file that you'll upload to SANPER.
5. Go to CRT 6.1 program and type in the following command

```
TUTOR 1.3> LO1
```

6. Then, under the menu "Transfer", click "Send ASCII"
7. Go to the folder where you saved your code in Step 1, and select the created *.S68 file. (You may have to change Files of type *from Text files (*.txt) to All Files (*.*)* if cannot see in your folder)
8. The S68 file will be now transferred and programmed to SANPER like the following.



```
WHAT

TUTOR  1.3 > LO1
S021000036384B50524F47202020323043524541544544204259204541535936384B6D
S12310004FF820002A7C000009002C7C0000090B1E3C00F34E4E1E3C00F14E4E1E3C00E3ED
S10710204E4E60E0EC
S804001000EB

TUTOR  1.3 >
```

9. Now, you can run your code!