Experiment No. 2:

TUTOR COMMAND UTILIZATION and PROGRAM EXPERIMENTATAION

By: Haomin Shi

Lab Partner: Anh Nguyen

Instructor: Dr. Jafar Saniie

ECE 441-003

Lab Date: 09-14-2017

Due Date: 09-21-2017

Acknowledgment: I acknowledge all of the work (including figures and codes) belongs to me and/or persons who are referenced.

Signature : _____

I. Introduction

A. Purpose

The purpose of this lab is to let students become familiarized with TUTOR software and the MC68k instruction set.

B. Background

This lab allowed students to learn more about the MC68k instruction set by programming, executing and debugging. In this lab, students will be program the SANPER unit by using TUTOR and the EASy68k software to load lengthy program into the SANPER unit with ease

II. Lab Procedure and Equipment List

A. Equipment

Equipment

- SANPER-1 system
- PC with TUTOR software

B. Procedure

1. The student will follow the lab instructions and input the program into the SANPER unit and record the data when needed.

III. Results and Analysis

A. Sample Program 2.1

The source code of program 2.1 (post-incremented):

```
ORG $300C               ;PROGRAM STARTED
START:
    MOVE.W D0, (A0)+    ;WRITE WORD FROM D0 TO A0, POSTINCREMENTED
    CMP.W A0, A1        ;CHECK BOUNDRY - GIVEN INFO
    BGE $300C           ;IF HAVENT HIT BOUNDRY, BRANCH TO START
    MOVE.B #228, D7     ;RETURN TO TUTOR
    TRAP #14            ;
    END START           ;END OF THE PROGRAM
```

A Modified version of program 2.1 (pre-incremented):

```
ORG $300C               ;PROGRAM STARTED
START:
    MOVE.W D0, -(A0)    ;WRITE WORD FROM D0 TO A0, PreDECREMENTED
    CMP.W A0, A1        ;CHECK BOUNDRY - GIVEN INFO
    BGE $300C           ;IF HAVENT HIT BOUNDRY, BRANCH TO START
    MOVE.B #228, D7     ;RETURN TO TUTOR
    TRAP #14            ;
    END START           ;END OF THE PROGRAM
```

The only difference between these two programs is that one is post-incremented and one is pre-incremented. In both programs, the address register a0 holds the address of the word that is being compared, and address register a1 has the address of the other word that is going to be compared against. Lastly, data register d7 holds the trap function.

The difference between pre-decrement and post-increment to me is just a personal preference. They are almost the same.

B. Sample Program 2.2

The source code of program 2.2:

```
ORG $900                ;PROGRAM STARTED
START:
    MOVE.B #$41, D0     ;ASCII CODE FOR 'A'
    MOVE.B #248, D7     ;FUNC CODE
    TRAP #14            ;
    MOVE.L #$FFFF, D5   ;SET D5 UP FOR A LARGE NUMBER
    DBEQ D5, $910       ;USE CYCLE AS A TIMER
    BRA $900            ;INFINITE LOOP
    END START           ;END OF THE PROGRAM
```

The result of procedure 10 is based on the modification made in procedure 9. Reason being that in procedure 9 we changed the $FFFF to $000F, which decreased the number, thus made the print become faster.

```
;Demo of a subroutine that outputs any character once, passed through the d1
    MOVE.B D1, D0   ;GET CHAR TO D0, PASSED THROUGH D1
    MOVE.W #248, D7 ;INITIALIZE D7 WITH FUNC #
    TRAP #14        ;
```

If we change the branching from $900 to $904, nothing will happen, because when we branched to $900, we are re-initializing the character(‘A’) again, which almost does the same thing as branching to $904. The only difference is that we did not re-initialize the number “A” again when we branched to $904.

```
    MOVE.L #$FFFF, D5   ;MEM $90A
    DBEQ D5, $910       ;MEM $910
```

This section of the code acts as a timer. And for the TRAP functions, it allowed us to reuse the code and work with the hardware.

C. Sample Program 2.3

The source code of program 2.3:

```
ORG $900                ;PROGRAM STARTED
START:
    MOVE.L #$1000, A5   ;LOAD THE STARTING ADDRESS OF STRING BUF
    MOVE.L #$1018, A6   ;LOAD THE ENDING ADDRESS
    MOVE.B #227, D7     ;PRINT STRING
    TRAP #14            ;
    MOVE.B #228, D7     ;BACK TO TUTOR
    TRAP #14            ;

END START
```

To implement this program without the TRAP function #227, the user must locate the string and input the string byte by byte until the null byte (end of the string), then append a line feed. After the execution of the program, the A5 will simply be pointing to the last byte of the output string plus 1.

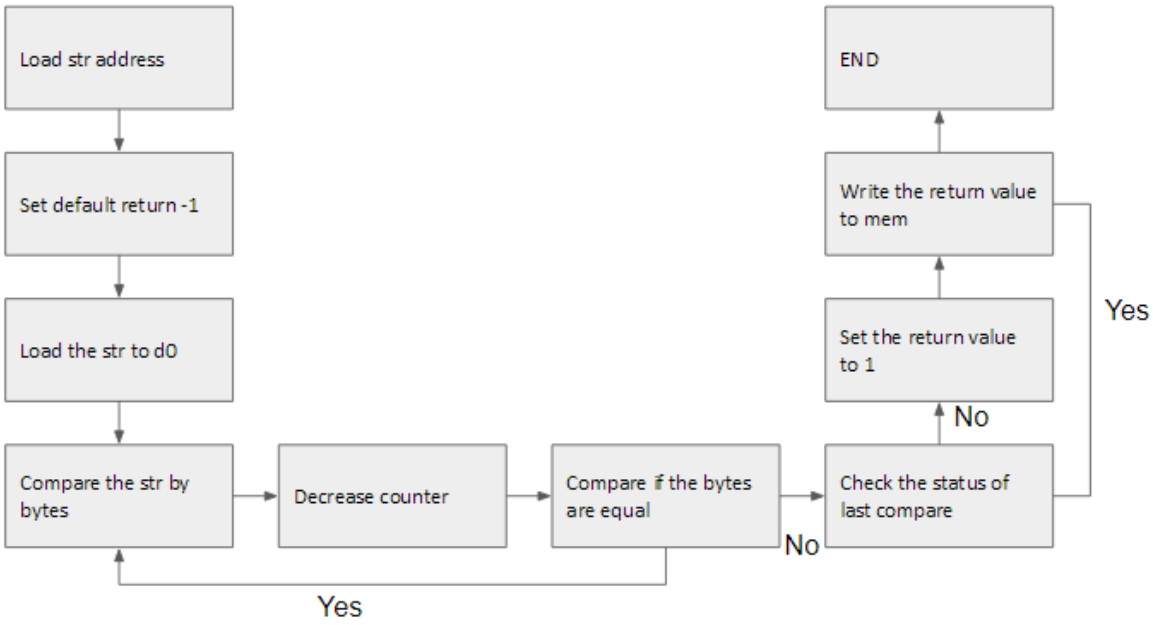
D. Sample Program 2.4

The source code of program 2.4:

```
ORG $1000               ;PROGRAM STARTED
START:
    MOVE.L #$2000, A0   ;LOAD STARTING ADDRESS OF STR1
    MOVE.L #$3000, A1   ;LOAD STARTING ADDRESS OF STR2
    MOVEQ.L #-1, D1     ;SET SEARCH RESULT TO F
    MOVEQ.L #0, D0      ;CLEAR D0
    MOVE.B (A0), D0     ;LOAD THE ADDRESS-STR1
    CMPM.B (A0)+, (A1)+ ;COMP STR LOCACTION
    DBNE D0, $1012      ;LOOP BACK AND CHECK AGAIN, WILL STOP IF NOT EQUAL
    BNE.S $101C         ;F ->JUMP TO MOVE.B D1, $1100
    NOT.B D1,           ;T ->CHANGED D1 FROM -1 TO 1
    MOVE.B D1, $1100    ;OUTPUT T OR F
    MOVE.B #228, D7     ;END
    TRAP #14            ;

END START
```

Flowchart of program 2.4:



The difference between MOVE and MOVEQ is that MOVEQ is faster but the range of it is much less than the MOVE. The CMPM instruction can manipulate multiple data by just using one line, it saves register upstage. The condition code was set by the CMPM instruction since branches don't have condition code.

E. Sample Program 2.5

The source of code program 2.5:

```
ORG $2000               ;PROGRAM STARTED
START:
    MOVE.L A0, A2       ;SET UP
BACK:MOVE.L A2, A0      ;RESET
NOT:CMP.W (A0)+, (A0)+  ;CHECK THE MEM LOCATIONS
    BHI.S $2014         ;IF HIGH BRANCEH->HIGH
    SUBQ.L #2, A0       ;ELSE JUST MOVE TO NEXT
    CMP.L A0, A1        ;CHECK IF DONE
    BNE $2004           ;IF NOT DONE BRA->NOT
    MOVE.B #228, D7     ;ELSE EXIT
    TRAP #14            ;
HIGH:MOVE.L -(A0), D0   ;CREATE BUBBLE
    SWAP.W D0           ;
    MOVE.L D0, (A0)     ;STORE
    BRA $2002           ;BACK TO "BACK"

END START               ;END
```

If we do not have SWAP function on the MC68k, then we can achieve the similar effect by using MOVE and work with registers.

ADDQ and SUBQ are just like MOVEQ from the previous section, they increase the speed of execution.

The difference between ADDQ and ADD is that ADDQ use fewer cycles, but ADD can handle all values. Same thing for the SUBQ and SUB. Lastly, for the line CMP (A0)+, (A0)+, it's content of A0 and A0++ are compared.

F. Sample Program 2.6

The source code of program 2.6:

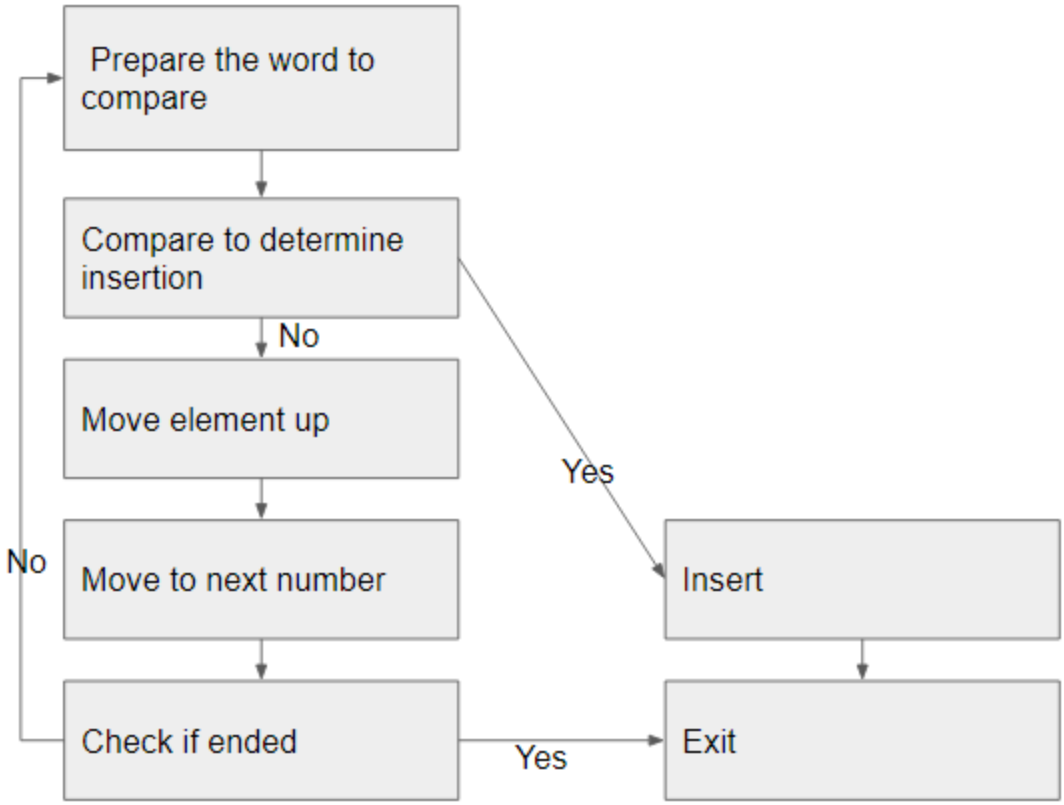
```
ORG $3000               ;PROGRAM STARTED
START:
    CLR.L $700          ;CLEAR OUT $700
    MOVE.W #$700, A6    ;SET A6
    MOVE.W #$700, A5    ;SET A5 THESE 2 ARE THE BUFFER BEGINNING
    MOVE.W #241, D7     ;CALLING THE FUNCTION FOR DATA ENTRY
    TRAP #14

    MOVE.B #226, D7
    TRAP #14

    CMP.W (A0), D0      ;COMPARE
    BCC $3024           ;BRA TO MOVE.B #228, D7
    MOVE.W (A0), -(A0)  ;DECREMENT A0 AND MOVE THE VALUE TO A NEW ADDRESS
    ADDQ #4, A0         ;POINT TO THE NEXT NUMBER
    CMPA.L A0, A1       ;CHECK THE SIZE
    BCC $3018           ;BRA TO CMP.W (A0), -(A0)
    MOVE.W D0, -(A0)    ;LOAD TO MEM
    MOVE.B #228, D7     ;EXIT
    TRAP #14

END START
```

Flowchart for program 2.6:



IV. Conclusions

At the end of this lab, students became familiarized with TUTOR and EASy68k program. The experiment was complete.

References

[1] Experiment 2 Lab Manual

[2] Educational Computer Board manual appendix