Experiment No. 2
TUTOR COMMAND UTILIZATION and
PROGRAM EXPERIMENTATION
ECE 441

Peter Chinetti

September 12, 2013

Date Performed:	September 5, 2013
Partners:	Zelin Wu
Instructor:	Professor Saniie

1 Introduction

1.1 Purpose

The purpose of this experiment is to familiarize students with the TUTOR monitor software. It also aims to familiarize the student about the M68K instruction set.

1.2 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.

2 Lab Procedure and Equipment List

2.1 Equipment

- SANPER System
- Computer with TUTOR software

2.2 Procedure

Execute each sample program and record data when requested.

3 Results, Analysis and Discussion

3.1 Sample Program 2.1

3.1.1 Original Source

```
    ORG     $300C
START:                      ; first instruction of program

* Put program code here
    MOVE.W D0,(A0)+  ; Write word from D0 out to postincremented A0
    CMP.W A0,A1  ; Check if we are at our boundry
    BLT $300C  ;If we haven't hit the boundry, branch back to the MOVE
    MOVE.B #228,D7  ;Return to TUTOR
    TRAP #14

    END     START           ; last line of source
```

3.1.2 Modified Source

```
    ORG     $300C
START:                      ; first instruction of program

* Put program code here
    MOVE.W D0,-(A0)  ; Write word from D0 out to predecremented A0
    CMP.W A0,A1  ; Check if we are at our boundry
    BLT $300C  ;If we haven't hit the boundry, branch back to the MOVE
    MOVE.B #228,D7  ;Return to TUTOR
    TRAP #14

    END     START           ; last line of source
```

3.1.3 Discussion of Registers

A0 holds the address of the word to be compared, A1 holds the word to compare against, D7 holds the function we are trapping to.

3.1.4 Predecrement vs Postincrement

They are almost identical, except the limits of the loop might need to be adjusted.

3.2 Sample Program 2.2

3.2.1 Original Source

```
    ORG      $900
START:                          ; first instruction of program

    MOVE.B #$41,D0  ; Character code for 'A'
    MOVE.B #248,D7  ; Function code OUTCH
    TRAP #14
    MOVE.L #$FFFF,D5  ; Initialize a register to a large number
    DBEQ D5,$910  ; Loop while decrementing to act as a timer
    BRA $900      ; Infinte loop

    END      START              ; last line of source
```

3.2.2 Procedure 10

Procedure 10 changed the count down from 0xFFFF to 0x000F, which sped up printing.

3.2.3 Single Character Print

```
MOVE.B D1,D0  ;Copy char to D0, where OUTCH reads from
MOVE.W #248, D7  ;Initialize D7 with OUTCH's function number
TRAP #14  ;Trap out to OUTCH
```

3.2.4 Effect of branching to $904

Nothing, the character will remain initialized.

3.2.5 Effect of $90A and $910

This is a loop that decrements a counter to act as a poor man's timer.

3.2.6 Effect of $90A and $910

Trap functions allow for code reuse and abstraction from direct hardware drivers.

3.3 Sample Program 2.3

3.3.1 Original Source

```
    ORG      $950
START:                          ; first instruction of program

    MOVE.L #$1000,A5  ; Load starting address of string buffer
    MOVE.L #$1018,A6  ; Load ending address of string buffer
    MOVE.B #227,D7    ; Print string with a line feed character
```

```
    TRAP #14
    MOVE.B #228,D7     ; Exit to TUTOR
    TRAP #14

    END     START          ; last line of source
```

3.3.2 Implementation without OUT1CR

Without the ability to print an entire string plus a linefeed, the user would have to manually implement a routine to print each of the characters of the string until the null byte, then append a line feed.

3.3.3 State of A6 and A6 after execution

According to Chapter 5, page 5-7 of the board manual, "A5 is pointing to the last byte in the output string plus one when the output function is complete."

3.4 Sample Program 2.4

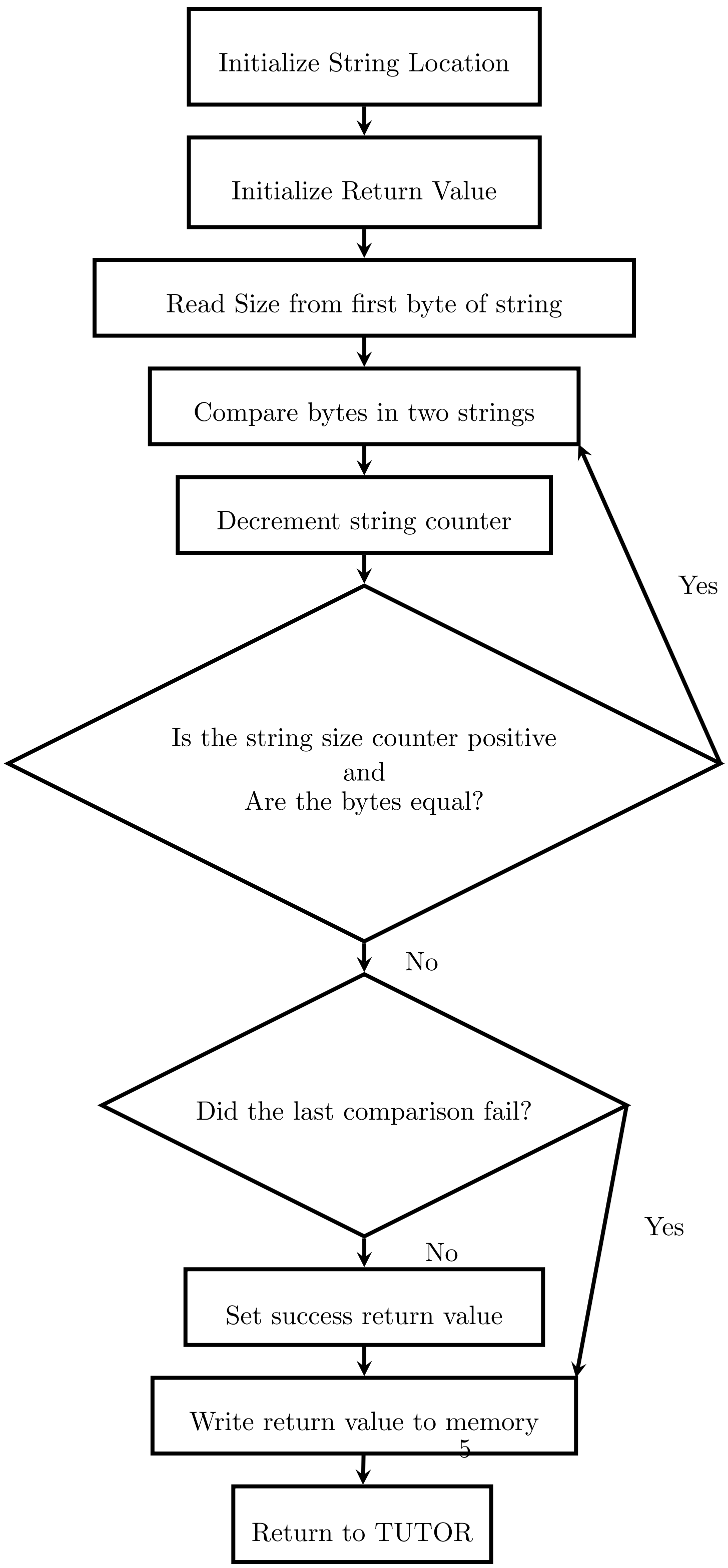
3.4.1 Original Source

```
    ORG     $1000
START:                         ; first instruction of program

    MOVE.L #$2000,A0  ; Load starting address of string1
    MOVE.L #$3000,A1  ; Load Starting address of string2
    MOVEQ.L #-1,D1    ; Initalize search result to failure
    MOVEQ.L #0,D0     ; Clear D0
    MOVE.B (A0),D0    ; Load first value for testing
    CMPM.B (A0)+,(A1)+ ; Check each location of the string
                       ;  to see if it matches
    DBNE D0,$1012     ; Repeat for the number of chars in
                      ; the string, break if not =
    BNE.S $101C       ; Failed, jump over the success instruction
    NOT.B D1          ; Succeed, flip the staus to 1
    MOVE.B D1,$1100   ; Output success or failure
    MOVE.B #228,D7    ; Exit to TUTOR
    TRAP #14

    END     START          ; last line of source
```

3.4.2 Flowchart



3.4.3 MOVE vs. MOVEQ

MOVEQ can move small values faster than MOVE can, but can not handle as wide a range of values as MOVE.

3.4.4 CMPM

CMPM allows for condensing multiple operations into one instruction, which saves on instruction cache usage, named register usage and program size.

3.4.5 Which instruction sets the CC bits for $1018

CMPM two instructions back. Branches do not modify the CC bits.

3.5 Sample Program 2.5

3.5.1 Original Source

```
    ORG     $2000
START:                      ; first instruction of program

    MOVE.L A0,A2            ;Inititailize
    MOVE.L A2,A0            ;Reset at top of loop
    CMP.W (A0)+,(A0)+       ;Check adjacent memory locations
    BHI.S $2014             ;If higher got to MOVE.L -(A0), D0
    SUBQ.L #2,A0            ;If lower move to the next element
    CMP.L A0,A1             ;Are we done?
    BNE $2004               ;If not, go back to CMP.W
    MOVE.B #228,D7          ;If done, exit to TUTOR
    TRAP #14
    MOVE.L -(A0),D0         ;Bubble up
    SWAP.W D0               ;Flip D0
    MOVE.L D0,(A0)          ;Store D0
    BRA $2002               ;On to next iteration


    END     START           ; last line of source
```

3.5.2 See comments for a description of the program operation

3.5.3 SWAP Instruction

Without ROR: Copy register to temp register. Shift it one direction by 16 bits. Shift the original register in the opposite direction by 16 bits. Add the two registers.
With ROR: ROR 16 bits in place

3.5.4 ADDQ and SUBQ

ADDQ and SUBQ are quick instructions, meaning they take fewer cycles to execute.

3.5.5 ADDQ vs ADD

ADDQ can only handle smaller, immediate values. ADD can handle all addressing modes.

3.5.6 SUBQ vs SUB

SUBQ can only handle smaller, immediate values. SUB can handle all addressing modes.

3.5.7 CMP (A0)+ (A0)+

The data at memory locations A0 and A0++ are compared.

3.6 Sample Program 2.6

3.6.1 Original Source

```
    ORG     $3000
START:                        ; first instruction of program

    CMP.W (A0),D0            ;Not used on first cycle
    BCC $300C                ;Branch to MOVE.W D0,-(A0)
    MOVE.W (A0),-(A0)        ;Move the data at A0 down one word
    ADDQ #4,A0               ;Move A0 to the next word
    CMPA.L A0,A1             ;Are we done?
    BCC $3000                ;If carry clear, go back to top
    MOVE.W D0,-(A0)          ;Insert new value
    MOVE.B #228,D7           ;Exit to TUTOR
    TRAP #14

    END     START             ; last line of source
```

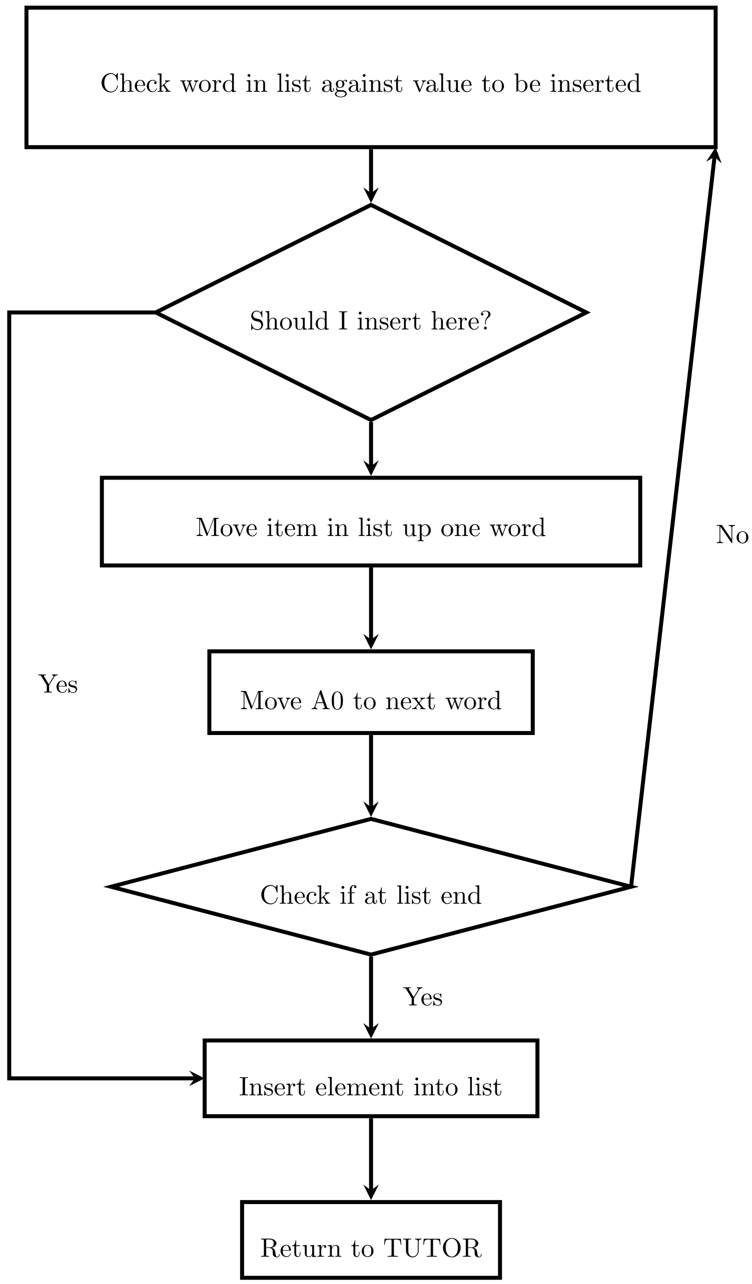
3.6.2 Source for procedure 8

```
        ORG $3000
START:  CLR.L D0                 ;Clears D0
        CLR.L D7                 ;Clears D7
        MOVE.L A0,A2
        LEA.L $1000, A5          ;sets up beginning buffer for input from keyboard
        LEA.L $1000, A6          ;sets up beginning buffer for input from keyboard
        MOVE.B #241,D7           ;traps to the terminal for data entry
        TRAP #14
```

```
        MOVE.B #226,D7
        TRAP #14
LOOP:   CMP.W (A0),D0          ;Compare (A0) with D0
        BGE INSERT             ;Branch to INSERT if CARRY CLEAR
        MOVE.W (A0),-(A0)      ;Decrement A0 and move the value of previous
                               ; content to new address.
        ADDQ #4, A0            ;Increment A0 to point to the next element
        CMPA.L A0, A1          ;Compare the size of A0 and A1
        BGE LOOP               ;Branch to loop if A1>=A0
INSERT: MOVE.W D0, -(A0)       ;Decrement A0 and move D0 to memory address
                               ; specified by A0
        MOVE.B #228, D7
        MOVEA.L -(A2),A0
        TRAP #14

        END     START          ; last line of source
```

3.6.3 Flowchart



4 Conclusions

This experiment was accomplished. TUTOR was introduced, as well as M68k instructions. From this building block, students can work on more and more complex programs for SANPER and can continue to learn about the functioning of the machine.