Experiment 2: tutor command utilization and program experimentation

Adam Sumner

Illinois Insititute of Technology

ECE 441-01

Lab Date: January 27th, 2015 **Due Date:** February 3rd, 2015

1 Introduction

The purpose of this experiment is to acquaint the user(student) with the TUTOR monitor program along with the MC68k instruction set. The user will enter several sample programs into the SANPER-1 ELU, execute them, and debug them.

2 Background

The Motorola MC68k is a CISC microprocessor that was popular during the 1980's. It has a 16-bit data bus and a 24-bit address bus. It has 8 data registers(D7-D0) for bit and bit field(1-32 bits), byte(8 bits), word(16 bits), and long-word(32 bits) operations. It also has 8 address registers (A7-A0) which can be used as software stack pointers, index registers, or base address registers. They are 32 bits wide and hold a full 32-bit address[3]. The microprocessor has an instruction set that will not be included in this report due to its size, but several commands will be showcased that are used in the programs in this experiment. These instructions consist of at least one word, but some have as many as 11. The first word of the instruction specifies the length of the instruction, the effective addressing mode, and the operation to be performed. The remaining words, called the brief and full extension words, further specify the instruction and immediate operands. An instruction specifies the function to be performed with an OP code and defines the location of every operand. This is done through register specification, effective address, or by implicit reference. Some commonly used instructions are:

- MOVE.X -*Load and store. Copies an 8, 16, or 32 bit value from one memory location or register to another memory location or register*
- BRA -*Branch. Program execution continues at location + displacement*
- Bcc -*Branch Conditionally. If the specified condition is true, program execution continues at location + displacement*
- CMP.X -*Compare. Subtracts the source operand from the destination data register and sets the condition codes according to the result.*

3 Equipment/Procedure

3.1 Equipment

- SANPER-1 EDUCATIONAL LAB UNIT
- Computer with TUTOR software

3.2 Procedure

Due to the length of each procedure, it will not be covered in depth in this report. The procedure for this experiment involved loading several pieces of code given in the experiment guide and following the instructions for each piece respectively.

4 Results

The software was implemented into the SANPER-1 ELU and the procedure was followed correctly. The final results of each piece of code can be found in Section 7.

5 Discussion

5.1 Answers to Follow Up Questions

5.1.1 Program 1

1. A fully commented version of the original program

   Answer: Program is located in Section 7.1.1

2. A fully commented version of the program written for Procedure #9

   Answer: Program is located in Section 7.1.2

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

   Answer: A0 and A1 are the boundary addresses. D7 is used to call the trap function and D0 holds the value $FFFF.

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

   Answer: There are no major differences/advantages to using either other than one adds and the other subtracts. In the end, they can accomplish the same thing. The key is to pick one that suits the implementation of the design

5.2 Program 2

1. A fully commented version of the original program

   Answer: Program is located in Section 7.1.3

2. List and explain the result of Procedure #10

   Answer: It changed the count down from FFFF to 000F which sped up printing of the character to the terminal.

3. Write a subroutine that outputs any character once. The character to be outputted will be passed to this subroutine through Data Register D1.

   Answer: Program is located in Section 7.1.4

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

   Answer: No difference, the character will still be initialized in D0

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

   Answer: The instruction is to test the condition, decrement the data register, then branch to the specified label. In this code, it essentially acts as a timer as it branches to the same DBEQ instruction until the condition fails. It can be used to create a "hack" timer function

6. List the major benefits of using TRAP instructions.

   Answer: Trap functions allow for an abstraction of very useful commonly used functionalities such as reading in data or outputting something to the terminal. It allows the programmer to easily use things like i/o without having to redefine them every single time

5.3 Program 3

1. A fully commented version of the original program

   Answer: Program is located in Section 7.1.5

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

   Answer: Without Trap 227, the user would have to do it character by character. This would involve using Trap 248

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

   Answer: A5 points to the last byte in the output string plus one. This is because the string also includes a null byte.

5.4 Program 4

1. A fully commented version of the original program

   Answer: Program is located in Section 7.1.6

2. Draw a flowchart for the program.

   Answer: Flowchart is located in Section 7.2.1

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

   Answer: MOVEQ is used to move immediate data while MOVE is general purpose and can move data from registers or immediate data. MOVEQ should always be used when immediate data is being loaded/stored. It is much more efficient than using the MOVE instruction. MOVE should be used when loading/storing data from a register/memory location.

4. Discuss the usefulness of the CMPM instruction.

   Answer: CMPM allows for condensing multiple operations into one instruction. It lets the developer write less lines of code for efficiency.

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

   Answer: CMPM sets the condition codes for BNE

5.5 Program 5

1. A fully commented version of the original program

   Answer: Program is located in Section 7.1.7

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

   Answer: It's a classic bubble sort. The program iterates through the list checking each value in the list to each other. Starting with the first value in the list, it then "floats" up if it is less than the adjacent value in the list. This iteration continues until the list is sorted. It should be noted that this is a terrible sorting algorithm to implement with a time complexity of $\mathcal{O}(n^2)$. The best algorithm to use should be merge sort which has time complexity $\mathcal{O}(n \log n)$.

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.

   Answer: The SWAP function is what allows the values to "float" into their correct spot in the list easily without having to use an extra data register to store temporary data and manually implement a swap algorithm. Without SWAP, use MOVE to copy temp data of first number, MOVE adjacent data into memory location of what was just saved into the register, then MOVE the data from the register into the memory location that was just moved to the location where the temporary data was taken.

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

   Answer: They are quick instructions designed for immediate values. They use less bus cycles to execute.

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

   Answer: ADDQ can only handle immediate values while ADD is general purpose. ADDQ is a faster instruction.

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

   Answer: SUBQ can only handle immediate values while SUB is general purpose. SUBQ is a faster instruction.

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

   Answer: The value at memory address A0 is compared to the value at memory address (A0 + word size).

5.6 Program 6

1. A fully commented version of both programs

   Answer: The original program is located in Section 7.1.8 and the modified version is in Section 7.1.9.

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

   Answer: The flowchart is shown in Section 7.2.2. It's an algorithm that shifts the list as it compares the values to find where an insertion is appropriate. This enables it to preserve the integrity of the list without having to overwrite data.

5.7 Analysis

There is little to discuss on the results obtained, as the results obtained were merely knowledge gained by the student on how to successfully load and execute programs into the SANPER-1 ELU. However, this knowledge should carry with the student for future lab experiments as debugging and executing programs is a must have skill.

6 Conclusion

Overall this lab was a success. The student was successfully able to load, debug, and execute MC68k programs using the TUTOR software. From here, the student can build upon this knowledge to develop more complex programs for analysis of the machine.

7 Appendix

7.1 Code

7.1.1 Program 1: Fill a Block of Memory Original

```
*-----------------------------------------------------------
* Title      :  Program 2.1
* Written by :  Adam Sumner and Ryan Jenkins
* Date       :  1/27/2015
* Description:  Fills block of memory with any desired value
*-----------------------------------------------------------
    ORG     $300C
START:
            MOVE.W  D0,(A0)+    ; Copy value in D0 to value located
                                ;in memory
            CMP.W   A0,A1       ; Check if at end of memory block
            BGE     $300C       ; Go back to first instruction
            MOVE.B  #228,D7     ; Return to TUTOR
            TRAP    #14


    END     START           ; last line of source
```

7.1.2 Program 1: Fill a Block of Memory Modified

```
*-----------------------------------------------------------
* Title      :  Program 2.1
* Written by :  Adam Sumner and Ryan Jenkins
* Date       :  1/27/2015
* Description:  Fills block of memory with any desired value
*-----------------------------------------------------------
    ORG     $300C
START:
            MOVE.W  D0,-(A1)    ; Copy value in D0 to value located
                                ;in memory
```

```
        CMP.W   A0,A1          ; Check if at end of memory block
        BGE     $300C          ; Go back to first instruction
        MOVE.B  #228,D7        ; Return to TUTOR
        TRAP    #14


    END     START              ; last line of source
```

7.1.3 Program 2: Outputting a Character to the Terminal

```
*-----------------------------------------------------------
* Title      : Program 2.2
* Written by : Adam Sumner and Ryan Jenkins
* Date       : 1/27/2014
* Description: Outputs a character to the terminal
*-----------------------------------------------------------
    ORG     $900
START:                         ; first instruction of program
        MOVE.B #$41,D0         ; hex for value 'A'
        MOVE.B #248,D7         ; set up OUTCH function
        TRAP #14
        MOVE.L #$FFFF,D5; store large number in D5
        DBEQ D5,$910
        BRA $900               ; infinite loop

    END     START              ; last line of source
```

7.1.4 Program 2: Subroutine

```
MOVE.B  D1,D0    ;Move character from D1 into D0
MOVE.W  #248,D7  ;Initialize trap function for OUTCH
Trap    #14      ; output character to terminal
```

7.1.5 Program 3: Outputting a String to the Terminal

```
*-----------------------------------------------------------
* Title      : Program 2.3
* Written by : Adam Sumner and Ryan Jenkins
* Date       : 1/27/2015
* Description: Outputs a String to the Terminal
*-----------------------------------------------------------
    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 buffer
        MOVE.B #227,D7   ; print string out
        TRAP #14
        MOVE.B #228,D7   ; go to TUTOR
        TRAP #14

* Put variables and constants here

    END     START           ; last line of source
```

7.1.6 Program 4: Pattern Match

```
*-----------------------------------------------------------
* Title      : Program 2.4
* Written by : Adam Sumner and Ryan Jenkins
* Date       : 1/27/2015
* Description: Pattern Match Program
*-----------------------------------------------------------
    ORG     $1000
START:                  ; first instruction of program

        MOVE.L #$2000,A0 ; load address of string 1
        MOVE.L #$3000,A1 ; load address of string 2
        MOVEQ.L #-1,D1   ; initialize to fail
        MOVEQ.L #0,D0    ; clear D0
        MOVE.B (A0),D0   ; Move first value to D0 to test
        CMPM.B (A0)+,(A1)+  ; compare the two string at the same location
        DBNE D0,$1012    ; Break if something is not equaivalent
        BNE.S $101C      ; if fail, jump over to end
        NOT.B D1         ; succeed, flip D1 to be a success
        MOVE.B D1,$1100  ; write success or failure to memory
        MOVE.B #228,D7   ; return to TUTOR
        TRAP #14

* Put variables and constants here

    END     START           ; last line of source
```

7.1.7 Program 5: Bubble Sort

```
*-----------------------------------------------------------
* Title      : Program 2.5
* Written by : Adam Sumner and Ryan Jenkins
* Date       : 1/27/2015
```

```
* Description: Bubble Sort
*-----------------------------------------------------------
    ORG     $2000
START:                          ; first instruction of program

        MOVE.L  A0,A2           ; initialize
        MOVE.L  A2,A0           ; reset to top of loop
        CMP.W   (A0)+,(A0)+     ; check adjacent memory location
        BHI.S   $2014           ; if higher, branch
        SUBQ.L  #2,A0           ; move to next element
        CMP.L   A0,A1           ; done?
        BNE     $2004           ; if not, go back to CMP.W
        MOVE.B  #228,D7         ; if done, exit TUTOR
        TRAP    #14
        MOVE.L  -(A0),D0        ; Bubble it up the list
        SWAP.W  D0              ; flip D0
        MOVE.L  D0,(A0)         ; Store D0
        BRA     $2002           ; next iteration

    END     START           ; last line of source
```

7.1.8 Program 6: Adding a Number to a Sorted List Original Code

```
*-----------------------------------------------------------
* Title      : Program 2.6
* Written by : Adam Sumner and Ryan Jenkins
* Date       : 1/27/2015
* Description: Adding a number to a Sorted List
*-----------------------------------------------------------
    ORG     $3000
START:                          ; first instruction of program

        CMP.W   (A0),D0
        BCC     $300C           ; branch to MOVE.W D0, -(A0)
        MOVE.W  (A0),-(A0)      ; move data to A0 down one word
        ADDQ    #4,A0           ; move to next word
        CMPA.L  A0,A1           ; 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

* Put variables and constants here
```

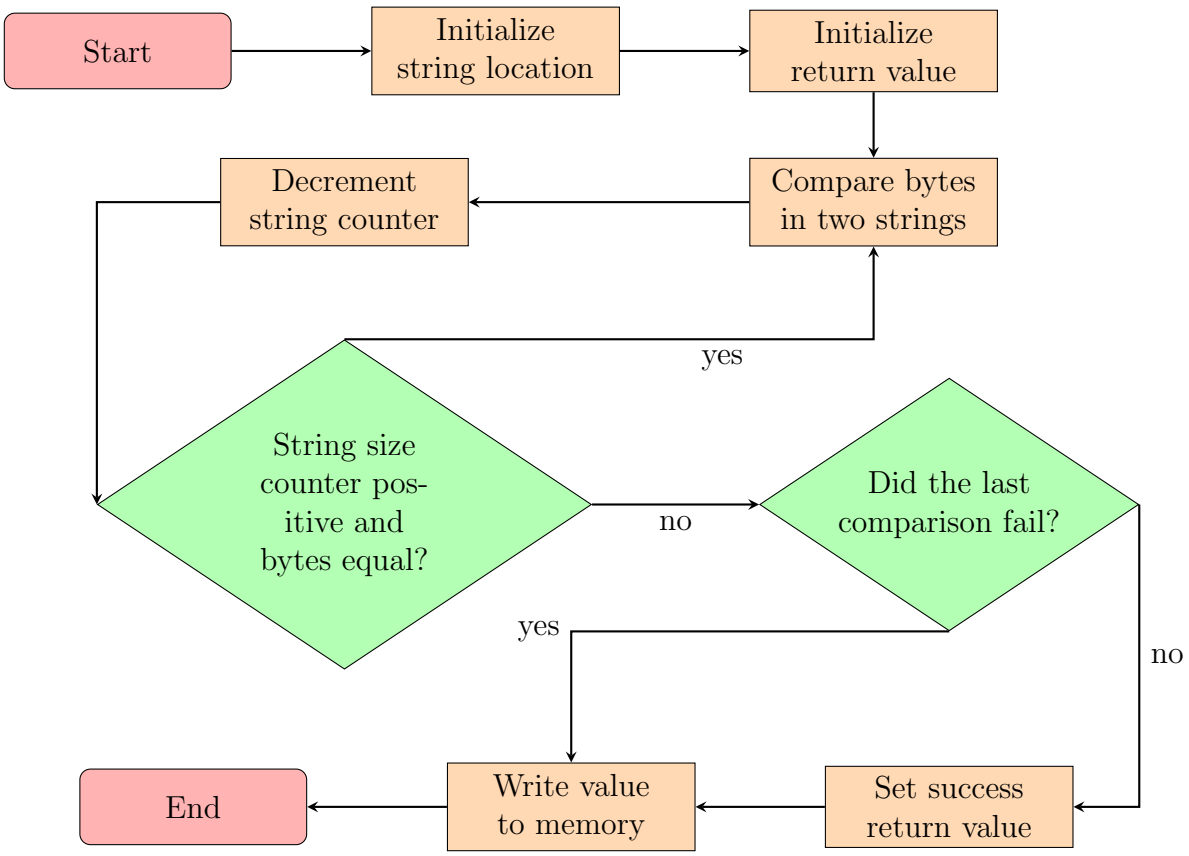
```
    END     START           ; last line of source
```

7.1.9 **Program 6:** Procedure 8 Code

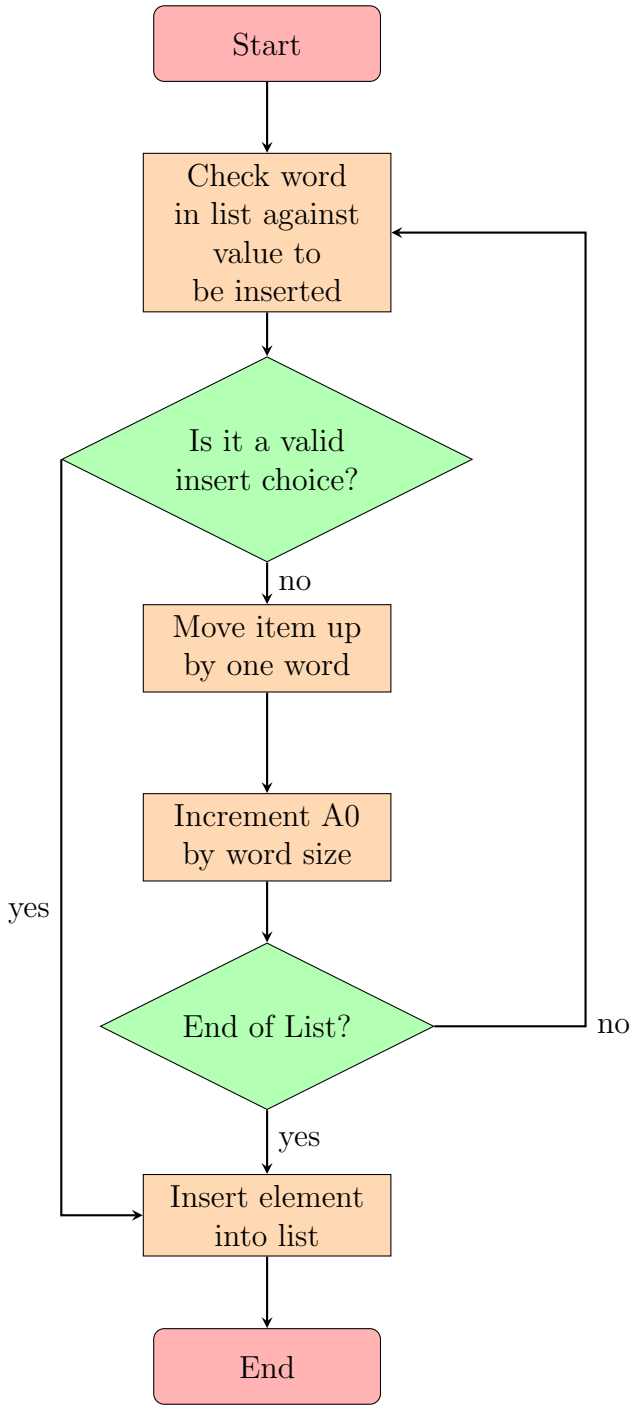
```
*-----------------------------------------------------------
* Title      : Program 2.6.8
* Written by : Adam Sumner and Ryan Jenkins
* Date       : 1/27/2015
* Description: Insert number into list given by user
*-----------------------------------------------------------
    ORG     $3000
START:                  ; first instruction of program

        LEA.L   $2100,A0    ; load initial addresses
        LEA.L   $210E,A1
        CLR.L D0            ;clear register
        CLR.L D7            ;clear trap register
        LEA.L   $1000, A6   ;set up buffer for input
        LEA.L   $1000, A5   ; set up buffer for input
        MOVE.B #241,D7      ; set up trap function for data entry
        TRAP #14
        LEA.L   $1000,A5
        LEA.L   $1004,A6
        MOVE.B #226,D7
        TRAP #14

LOOP:   CMP.W (A0),D0
        BCC SKIP
        MOVE.W (A0),-(A0)
        ADDQ #4,A0      ; move to next word
        CMPA.L A0,A1    ; done?
        BCC LOOP        ; branch if carry clear
SKIP:   MOVE.W D0,-(A0) ; insert into list
        MOVE.B #228,D7  ; exit to TUTOR
        TRAP #14

* Put variables and constants here

    END     START           ; last line of source
```

7.2 Flowcharts

7.2.1 Program 4



7.2.2 Program 6



References

[1] Experiment 2 Lab Manual

[2] Educational Computer Board Manual

[3] MC68K User Manual

[4] SANPER-1 ELU User Manual