Experiment No. 02:

TUTOR Command Utilization and Program Experimentation

By: Theo Guidroz

Lab Partner: Daniyal Khan, Gregory Bonnema

Instructor: Dr. Jafar Saniie

ECE 441-001

Lab Date: 02-29-2020

Due Date: 02-05-2020

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 experiment is to help the users to familiarize furthermore with SANPER-1 unit and TUTOR firmware. The users will also learn how to load M68000 language to the SANPER-1 unit. Several programs will be executed and tested during this lab.

**B. Background**

Students had previous experience working with M68000 language. The SANPER-1 ELU and the TUTOR firmware can be used to strengthen their knowledge about this processor and programming language. The 68000 is a popular CISC microprocessor that has a 16-bit data bus and 24-bit address bus. It possesses 8 data registers for byte/word/long-word size operations. It also has 8 address registers, one of which is used as a stack pointer (A7). The 68000 instruction set will be demonstrated in the programs used in this experiment.

**II. Lab Procedure and Equipment List**

**A. Equipment**

* SANPER-1 system
* PC with TUTOR software

**B. Procedure**

Program 2.1

1. Power on SANPER-1 ELU and open TUTOR software.
2. Using the MM command, enter program 2.1.
3. Use the BF command to initialize the memory block to a known value.
4. Initialize the A0 and A1 registers with their appropriate values.
5. Run the program using the GO command.
6. Use the MD command to ensure the program worked properly. $FF should be stored in each memory location from $3100 to $3200.
7. Modify the program to use pre-decrement addressing mode instead of post-increment.
8. Use TUTOR's Block Fill (BF) Command to initialize the memory block to some known value
9. Run the program, and $BB should be stored in each memory location from $3100 to $3200 if done properly.

Program 2.2

1. Type program 2.2 into EASy68k and run it.
2. Use the LO1 command and the “send ASCII” tab to transfer and program the code to SANPER-01.
3. Determine appropriate value for ‘??’ (#$41) and run the program.
4. The letter ‘A’ will display repeatedly, press the ABORT switch on the SANPER-1 ELU to break the loop.
5. Change ‘A’ to ‘5’ using appropriate ASCII value. Run the program again.
6. Press ABORT to break the loop.
7. Change the value $FFFF to $000F at address $90E. Use MM command to do this.
8. Run the program again, determine the differences between $FFFF and $000F.
9. Press ABORT to break the loop.

Program 2.3

1. Type program 2.2 into EASy68k and run it.
2. Use the LO1 command and the “send ASCII” tab to transfer and program the code to SANPER-01.
3. Determine appropriate value for ‘??’ at $950, $956 (#$1000, #$1018) and run the program.
4. Use the MS command to load a string into memory.
5. Use the BS command to search user memory for the string.
6. Run the program, and string should appear once.

Program 2.4

1. Type program 2.2 into EASy68k and run it.
2. Use the LO1 command and the “send ASCII” tab to transfer and program the code to SANPER-01.
3. Determine appropriate value for ‘??’(#$2000,#$3000,$1012) at $1000, $1006, $1014.
4. Load two strings (same value) into memory using the MS command. Use the BS command to see if strings were properly stored; two address will appear.
5. Load string length into the first byte of each string using the MM command for each string’s respective address.
6. Enter $AAAA into location $1100 using the MM command. Run the program.
7. Byte at $1100 should be 00 to indicate strings are equal.
8. Modify string #2 to make the two strings different.
9. Run the program. Since the two strings are different, $FF should be the first byte at $1100.

Program 2.5

1. Type program 2.2 into EASy68k and run it.
2. Use the LO1 command and the “send ASCII” tab to transfer and program the code to SANPER-01.
3. Use the MM command to load a table of unsorted data into memory.
4. Initialize A0 and A1 to start and end the string, respectively.
5. Run the program.
6. Use the MM command to examine the table entries; they should be sorted.

Program 2.6

1. Type program 2.2 into EASy68k and run it.
2. Use the LO1 command and the “send ASCII” tab to transfer and program the code to SANPER-01.
3. Initialize D0, A0, and A1.
4. Run the program; examine the list to see if the insert was performed correctly.
5. Write a program that inputs a 4-digit hex number from the terminal and inserts it into the proper location in the sorted list.
6. Run the program and check if the input has been correctly placed.

**III. Results and Analysis**

1. **Discussion**

Programs

\* This program fills any size block of memory with any desired value.

MOVE.W D0, (A0)+ ; Move word size value from D0 register to memory address written in A0 register. The value in A0 is then incremented by 2.

CMP.W A0, A1 ; Compare A0 and A1.

BGE $300C ; If the value of A1 is greater or equal to the value of A0, branch to memory address $300C (The first line).

MOVE.B #228, D7 ; Syscall for going back to TUTOR

TRAP # 14 ; TRAP 14 Handler

. ; End of program

Figure 1: Sample Program 2.1

\* This program will use one of the TRAP routines to repeatedly output single

\* character to the terminal.

MOVE.B #$41, D0 ; Copy ‘A’ (ASCII) into D0

MOVE.B #248, D7 ; Syscall for outputting single character

TRAP #14 ; TRAP 14 Handler

MOVE.L #$FFFF, D5 ; move long word hex immediate value $FFFF to D5

DBEQ D5, $910 ; If the value in D5 register equals 0, move on to next instruction. Else, the value of D5 register decrements 1 and go back to address $910 (this line itself).

BRA $900 ; Branch back to address $900

Figure 2: Sample Program 2.2

Note: The function ‘.’ Is not used since the code is imported from EASy68k

\* The purpose of this program is to demonstrate the capability of another \* TRAP 14 Handler

function available within the TUTOR firmware.

MOVE.L #$1000, A5 ; Move $1000 into register A5

MOVE.L #$1016, A6 ; Move $1016 into register A6

MOVE.B #227, D7 ; Syscall for outputting string from A5 to A6

TRAP #14 ; TRAP 14 Handler

MOVE.B #228, D7 ; Syscall for returning to TUTOR

TRAP #14 ; TRAP 14 Handler

Figure 3: Sample Program 2.3

Note: The function ‘.’ Is not used since the code is imported from EASy68k

\* This program compares two ASCII strings in memory to determine if they are

\* identical.

MOVE.L #$2000, A0 ; Move $2000 into A0 register

MOVE.L #$3000, A1 ; Move $3000 into A1 register

MOVEQ.L #-1, D1 ; Move quick, move immediate -1 into D1

MOVEQ.L #0, D0 ; Move quick, move immediate 0 into D0

MOVE.B (A0), D0 ; Copy byte size of the value from the memory address that A0 address register points into D0

CMPM.B (A0)+, (A1)+ ; Subtract byte size value from the memory address that A1 address register points from byte size value from the memory address that A0 address register points

; Values in A0 and A1 increments by byte size

; The condition codes will be set accordingly

DBNE D0, $1012 ; If the value in D0 register does not equal to 0, move on to next instruction

; Else, subtract 1 from the value in D0 register and go back to address $1012

BNE.S $101C ; If condition code Z is 0, move to address $101C

; Else, move on to next instruction

NOT.B D1 ; Take complement of byte size of value in D1 register and store it back to D1 register

MOVE.B D1, $1100 ; Copy byte size of value in D1 register to memory address $1100

MOVE.B #228, D7 ; Copy byte size of decimal immediate value 228 to D7 register

TRAP #14 ; TRAP 14 Handler

Figure 4: Sample Program 2.4

Note: The function ‘.’ Is not used since the code is imported from EASy68k

\* This program implements bubble sort.

MOVE.L A0, A2 ; Move long size value from A0 to A2

MOVE.L A2, A0 ; Move long size value from A2 to A0

CMP.W (A0)+, (A0)+ ; Subtract word size value, which the memory address that A0 address register points, from byte size value from the memory address that A0 address register points

; Values in A0 and A0 increments by 1

; The condition codes will be set accordingly

BHI.S $2014 ; If right (A0) is higher than left (A0), branch to address $2014

; Else, move onto next instruction

SUBQ.L #2, A0 ; Subtract Quick - Subtract long word decimal value 2 from the value in A0 address register

CMP.L A0, A1 ; Subtract long word size value in A1 address register from the value in A0 address register

; The condition codes will be set accordingly

BNE $2004 ; If Z is 1, move on to next instruction

; Else, branch to address $2004

MOVE.B #228, D7 ; Syscall for returning to TUTOR

TRAP #14 ; TRAP 14 Handler

MOVE.L -(A0), D0 ; Subtract 4 from the value of A0, then copy long word size value that A0-4 points into D0 register

SWAP.W D0 ; Exchange the upper and lower 16-bit words of D0 register

MOVE.L D0, (A0) ; Copy long word size value of D0 to the memory address A0 address register points

BRA $2002 ; Branch to address $2002

Figure 5: Sample Program 2.5

Note: The function ‘.’ Is not used since the code is imported from EASy68k

\* This program inserts a new number into the proper location within a sorted \* or sequenced list

MOVE.B #241, D7 ; Syscall for getting string type user input

TRAP #14 ; TRAP 14 Handler

MOVE.B #226, D7 ; Syscall for converting ASCII to Hex

TRAP #14 ; TRAP 14 Handler

CMP.W (A0), D0 ; Subtract word size value in D0 register from the value that A0 address points from

; The condition codes will be set accordingly

BCC $2020 ; If C bit is 0, branch to address $202

; Else,move on to the next instruction

MOVE.W (A0), -(A0) ; Copy word size value that A0 address register points into A0-4 memory address

ADDQ #4, A0 ; Add Quick, add immediate value 4 to A0

CMP.L A0, A1 ; Subtract long word size value in A1 address register from A0 address register

BCC.S $2000 ; If C bit is 0, branch to $2000

; Else, move on to the next instruction

MOVE.W D0, -(A0) ; Copy word size value from D0 register to A0-4 memory address

MOVE.B #228, D7 ; Syscall for returning to TUTOR

TRAP #14 ; TRAP 14 Handler

Figure 6: Sample Program 2.6

Note: The function ‘.’ Is not used since the code is imported from EASy68k

Questions

Sample Program 2.1

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

See Please check the **Discussion** section in **Results and Analysis.**

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

See Please check the **Discussion** section in **Results and Analysis.**

1. **Discuss the function of each register used in the original program.**

D0 – holds value $FFFF

A0 – holds the starting address of the memory block to be filled.

A1 – holds the ending address of the memory block to be filled.

D7 – used to catch the Trap function.

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

The pre-decrement and post-increment address modes allow for faster and easier access to memory regions such as arrays. They can be performed in the same instruction as the actual operating that is being performed. They can be done at several rates. On MC68000, they can be done at rates of 1,2 or 4 bytes.

Sample Program 2.2

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

See Please check the **Discussion** section in **Results and Analysis.**

1. **Describe the output results of procedure #10 and explain what caused the difference.**

It changed the count down for $FFFF to $000F, which sped up the printing of the character to the terminal.

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

\* Subroutine to output single character

\* Different functions of TRAP 14 Handler

MOVEM.L D0/D1,-(SP) ; Save registers

MOVE.B D1,D0 ; Move value at D1 into D0

MOVE.B #248,D7 ; System call for outputting a single character

TRAP #14 ; TRAP 14 Handler

MOVEM.L (SP)+,D0/D1 ; Restore Registers

Subroutine to output single character

1. **What is the effect of changing the instruction at address $914 to “BRA 904”?**

No change since the value in D0 is not changed in the program after line 900.

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

The instructions move the value $FFFF to data register D5, tests the condition, decrements the data register, and then branch to the label/memory address specified. The 2 lines act like a timer where the instruction at $910 decrements the value of D5 until it fails.

1. **List the major benefits of using TRAP instructions.**

TRAP instructions are useful when dealing with I/O, text file and accessing data from the terminal. The user can deal with those features without writing a subroutine but just by calling the correct TRAP function.

Sample Program 2.3

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

See Please check the **Discussion** section in **Results and Analysis.**

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

#248 would have been used to output the string letter by letter.

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

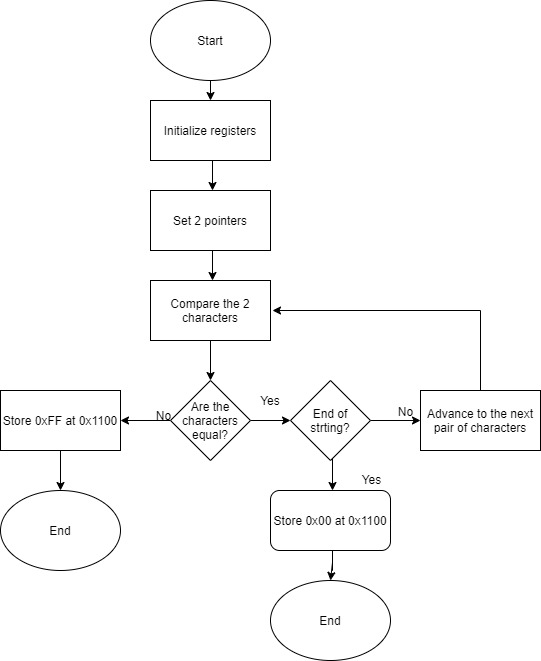
A5 points to byte located after the last value of the string because a string includes a null terminating byte.

Sample Program 2.4

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

See Please check the **Discussion** section in **Results and Analysis.**

1. **Draw a flowchart for the program.**



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

The MOVE instruction can move data to/from registers or move immediate data to a register. Whereas the MOVEQ instruction only moves the immediate value into a data register. Since MOVEQ instruction uses fewer bus cycles, it executes faster than MOVE. If the program requires to move immediate values, MOVEQ should be used otherwise, MOVE is the only option.

1. **Discuss the usefulness of the CMPM instruction.**

The CMPM is a useful instruction that combines multiple operations into one. It allows for comparing two source operands specified using the post-increment addressing mode. It can be used to compare arrays.

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

DBNE instruction at $1014.

Sample Program 2.5

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

See Please check the **Discussion** section in **Results and Analysis.**

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

The program iterates through the list and compares every value. If a pair of values are in the wrong order, they are swapped. The program iterates through the whole array again until it has been completely sorted.

1. **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.**

SWAP the 16-bit halves of a single data register. In bubble sort, it can be used to swap to elements that are out of order. This is assuming that the elements are of word size.

\* Set of instructions necessary to replace SWAP

MOVE.L D0,D1 ; Move words to SWAP at D0 to D1

LSL.L #$16,D0 ; Shift D0 4 bytes to the left to isolate lower byte

LSR.L #$16,D1 ; Shift D1 4 bytes to the right to isolate higher byte

ADD.W D1,D0 ; Using ADD of word size will fill the last 4 bytes of D0 with the value of D1 without modifying the upper word.

Equivalence of SWAP instruction

1. **Describe the function and advantages of the ADDQ and SUBQ instructions.**

They are mathematical functions that are used when the source operand is an immediate value. Since the Q instructions use fewer bus cycles, they use less time to execute. Programs can run faster if Q commands are used however, they cannot be used if the source operand is not an immediate value.

1. **Describe the differences between the ADD and ADDQ instructions.**

This is very similar to the difference between the MOVE and MOVEQ instructions. ADD is

general purpose and can be used among registers and use immediate values when needed. However, the ADDQ is a faster instruction (for the reasons described in #4 above) used strictly for when an immediate value is used as a source value.

1. **Describe the differences between the SUB and SUBQ instructions.**

The SUB instruction can subtract data to/from registers or subtract immediate data to a register. Whereas the SUBQ instruction only subtract immediate values into a data register. Since SUBQ instruction uses fewer bus cycles, it executes faster than SUB. If the program requires to subtract immediate values, SUBQ should be used otherwise, SUB is the only option.

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

The value stored at (A0) is compared to the value at the memory address specified by (A0 + word-size increment).

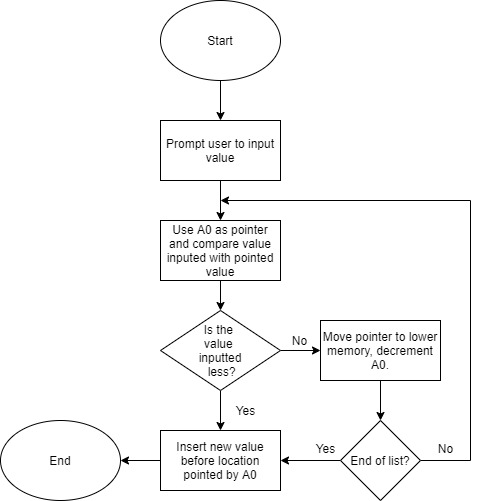
Sample Program 2.6

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

See Please check the **Discussion** section in **Results and Analysis.**

1. **Draw a flowchart of the program and discuss the insertion algorithm.**

The algorithm works by iterating over the array and comparing the input with the existing values of the array. When it gets to a value higher than the itself in the array, the input is inserted before location pointed by A0.



**IV. Conclusions**

This lab strengthened the students’ knowledge about the SANPER unit and TUTOR. It mainly helped to understand how to transfer programs from MC68000 to the transfer unit before executing them. Debugging was also implemented for the success of this lab. The programs typed could perform tasks such list sorting. The increase of familiarity with the lab equipment’s proves that this lab was a success.

**References**

[1] Experiment 2 Lab Manual

[2] Educational Computer Board manual appendix

**Attachments**