Experiment #2

TUTOR COMMAND UTILIZATION and PROGRAM EXPERIMENTATION

Alan Palayil

TA: Xinrui

ECE 441-L02

Lab Date: 02-03-2022

Due Date: 02-10-2022

I. Introduction

A. Purpose

The purpose of this lab is to utilize SANPER-1-ELU to write 6 programs which will help in understanding MC68K Assembly language better.

B. Background

This experiment will have users write 6 programs and have them executed and debugged. SANPER-1-ELU is used to write the TUTOR commands.

II. Lab Procedure and Equipment List

A. Equipment

- SANPER-1-ELU
- TUTOR software
- Easy 68K

B. Procedure

1. Perform the Sample Program 2.1 to 2.6.
2. Record the results gathered by running the programs.
3. Have the TA check the executed codes.

III. Results and Analysis

Sample 2.1
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>**	

Sample 2.2
TUTOR 1.X > MM $900; DI <CR>

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

Sample 2.3

TUTOR 1.X > 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>**	

Sample 2.4

TUTOR 1.X > 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>**	

Sample 2.5

TUTOR 1.X > 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>**	

Sample 2.6
TUTOR 1.X > 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>**	

IV. Discussion

Sample 2.1

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

```
MOVE.W D0,(A0)+ ; Word sized D0 is moved to memory location specified in A0 with
;postincrement
CMP.W A0,A1 ; Compare addresses A0 and A1
BGE $300C ; If A1 greater than or equal to A0 branch to $300C
MOVE.B #228,D7 ; Syscall to return to TUTOR
TRAP #14 ; T RAP Handler
```

2.

```
MOVE.W  D0,-(A0)      ; Word sized D0 is moved to memory location specified in A0  with
;predecrement
CMP.W   A0,A1             ; Compare addresses A0 and A1
BGE     $300C             ; If A1 greater than or equal to A0 branch to $300C
MOVE.B  #228,D7           ; Syscall to return to TUTOR
TRAP    #14               ; TRAP Handler
```

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

- D0 saves new contents so it can be moved to memory later.
- A0 starting memory address
- A1 end memory address

- D7 Saves Trap #14.

4. The advantages of the pre-decrementing and post-incrementing addressing modes.
   - Pre-decrementing allows instructions to flow without using another line of code to tell the register to move to the next one. It will automatically increase the index value of program instructions while other modes execute regularly.
   - Post-increment allows program to access memory before it increments index value. It will automatically keep receiving the next high memory address.

Sample 2.2

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

```
MOVE.B  #$41,D0       ; Move ASCII value #$41 which corresponds to 'A' to
D0
MOVE.B  #248,D7          ; Syscall to output a character
TRAP    #14              ; Trap handler part of syscall
MOVE.L  #$FFFF,D5        ; Move #$FFFF to D5
DBEQ D5,$910           ; If value in D5 = 0, continue to next line, if not equal decrement D5 by
1 and   ;branch to $910 which is the same line
BRA     $900             ; branch to $900 which is start of program
```

2.The output results of procedure #10, and explain what caused the difference.
   - Procedure 10 changes $000F to $FFFF which in original code it prints infinite 55555... But changing the code prints only one 5. This is because the infinite loop is changed.

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

```
MOVEM.L D0/D1,-(SP)         ; Move D0 and D1 on to the stack (required for subroutine)
MOVE.B D1,D0                ; Move D1 to D0 which is used by the syscall
MOVE.B #248,D7        ; Syscall for outputting a single character
TRAP #14
MOVEM.L (SP)+,D0/D1         ; Restore from stack4.
```

4. The effect of changing the instruction at address $914 to "BRA 904":
   - Nothing will happen to the output until D0 register is changed.

5. The steps involved in the execution of the instructions at addresses $90A and $910.
   - $90A is where the counter is located while $910 is decrementing the D5 register. This function delays the program.

6. List the major benefits of using TRAP instructions.
   - TRAP function is where you can use I/O instructions.
   - TRAP automatically converts majority of ASCII codes.

Sample 2.3

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

```
MOVE.L #$1000, A5        ; Move $1000 to A5 which is starting location
MOVE.L #$1018, A6        ; Move $1018 to A5 which is ending location
MOVE.B #227, D7        ; Syscall to output string
TRAP #14
```

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

2. Discuss how you would have implemented this program were TRAP Function No. 227 not available.
   - TRAP 242 is an alternate function to the TRAP 227.

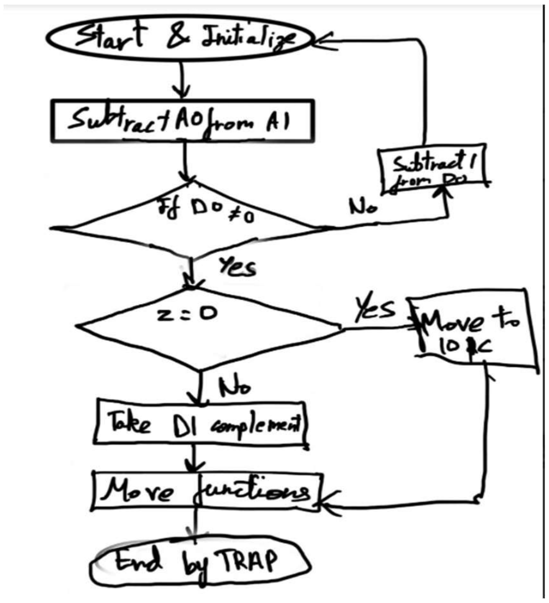
3. After executing the program, what is the final value of A5, and why?
   - Final value will be $1018, following the program logic.

Sample 2.4

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

```
        MOVE.L #$2000, A0      ; Move $2000 to A0
        MOVE.L #$3000, A1      ; Move $3000 to A1
        MOVEQ.L #-1, D1        ; Move -1 to D1
        MOVEQ.L #0, D0         ; Move 0 to D0
        MOVE.B (A0),D0         ; Move value in A0 to D0
        CMPM.B (A0)+, (A1)+    ; Subtract value in A1 from value in A0 both have post
        increment condition codes set accordingly
        DBNE D0, $1012         ; If D0 not equal to 0 move to next instruction else subtract 1
        from D0 and go back $1012
        BNE.S $101C            ; If condition code z = 0 move to $101C else move to next
        instruction
        NOT.B D1               ; Take complement of D1
        MOVE.B D1, $1100       ; Move D1 to $1100
        MOVE.B #228, D7        ; Syscall to output string
        TRAP #14               ;
```

2. Flowchart for the program.



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

- MOVEQ is better when you need to move data immediately to the data register.

4. Discuss the usefulness of the CMPM instruction

- CMPM is useful because it allows user to post increment both side of the source and destination.

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

- CMPM.B (A0) +, (A1)+

Sample 2.5

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

```
MOVE.L A0, A2          ; Move A0 to A2
MOVE.L A2, A0          ; Move A2 to A0
CMP.W (A0)+, (A0)+     ; Compare A0 with A0 increment word size
BHI.S $2014            ; If second A0 higher than first A0 branch to $2014
                       ; else continue to next instruction
SUBQ.L #2, A0          ; Subtract 2 from A0
CMP.L A0, A1           ; Compare A0 and A1

BNE $2004              ; If A1 – A0 is 0 continue to next instruction
                      ; else branch to $2004
MOVE.B #228, D7        ; Syscall to return to tutor
TRAP #14
MOVE.L -(A0), D0       ; Predecrement long size then move value in A0 to D0
SWAP.W D0              ; Swap upper 16 bits with lower  bits in D0
MOVE.L D0, (A0)        ; Move D0 to memory specified by A0
BRA $2002              ; Branch to $2002
```

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

- It adds value to two different address and starts comparing until it reaches the end
- Then it swaps to $2014

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.

```
SWAP.W D0
MOVE.W D0, D1 ; Save D0 to D1
MOVE.L #$FFFFFFFF, D2 ; Register temporary D2
ROL #16, D0 Lowest goes to highest
ROL #16, D1 Highest goes to lowest
AND.L D0,D2
AND.L D1,D2
MOVE.L D2,D0 Result goes to D0
```

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

- Like MOVEQ it moves the value immediately to the location basically making it faster.

5. Describe the differences between the ADD and ADDQ instructions.
   - This takes immediate data and moves it right to the destination.
6. Describe the differences between the SUB and SUBQ instructions.
   - This takes immediate data and moves it right to the destination.
7. Describe the sequence of events that occurs during the execution of the instruction located at address $2004.
   - Comparison of2000 and 2002 and A0 loads to 2004
   - High goes to 2014
   - 2000 and 2002 swaps
   - Then it repeats the comparison process.

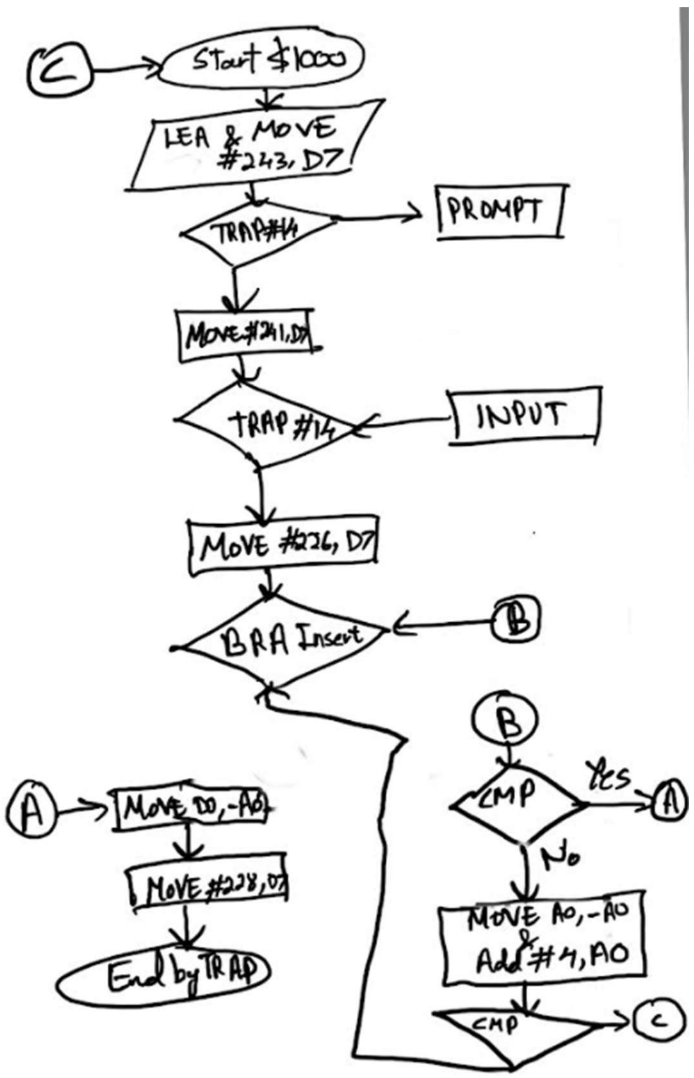
Sample 2.6

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

The code was incomplete and the TA will give us time to run it later on.

```
  ORG $1000
        LEA STR1, A5
        LEA END1, A6
        MOVE.B #243,D7
        TRAP #14 ; TRAP #14 ready prompt
        MOVE #241,D7
        TRAP #14 ; TRAP #14 input collections
        MOVE #226,D7
        TRAP #14 ; TRAP #14 store in D0 after conversion
        BRA INSERT
  ORG $1500
INSERT: CMP.W (A0),D0 ; compare A0 and D0
        BCC LBL1 ; If table value is less than inset value, branch to LBL1
        MOVE.W (A0),-(A0) ; increment the value
        ADDQ #4,A0
        CMPA.L A0,A1 ; Compare the memory address
        BCC START ; if its in the table Loop
LABEL1: MOVE.W D0,-(A0) ; insert the value into A0
        MOVE.B #228,D7
        TRAP #14 ; Trap #14 : Return to TUTOR
        SIMHALT ; halt simulator
  ORG $900
STR1 DC.B 'Enter 4 Hex Digits: '
END1 DC.B 0
  END START ; last line of source
```

2. Flowchart of the program and the insertion algorithm.



V. Conclusion

All the programs were successfully compiled apart from 2.6. This has increased our knowledge about the MC68K Assembly language.

VI. References

Experiment #2 Lab Manual

VII. Attachments

None