CSCI 592
LAB ASSIGNMENT – 5

Written by

DINESH SEVETI

Date: 03-1-2025

OBJECTIVE:

The objective of this lab is to explore various addressing modes in 68K assembly language, including immediate, register direct, absolute, indexed, and PC-relative addressing. By executing instructions in trace mode, students will analyze their impact on registers and memory, enhancing their understanding of effective address calculation and memory manipulation. This lab builds on previous concepts of memory partitioning, providing hands-on experience in debugging and predicting instruction outcomes in low-level programming.

TECHNOLOGY USED

- Easy68K Assembler software to run the code.
- Hypothetical or real CPU with registers and memory
- Hexadecimal Memory Addressing

PROCEDURE

- Initialize registers (D0–D7, A1) to 00000000 and load specified values into memory.
- Execute the program in trace mode, stepping through each instruction.
- Observe how instructions modify registers and memory, identifying addressing modes.
- Compute effective addresses and verify results against simulator execution.
- Analyze each instruction's effect on registers and memory.

OPERATIONS

- LEA (Load Effective Address) stores the address $2100 in register A1.
- MOVE.L (Move Long Word) assigns immediate values to registers (D7 = 10, D4 = 16843009).
- MOVEQ (Move Quick Byte) loads D0 with 27 using a short-form immediate move.
- MOVE.B (Move Byte) modifies only the least significant byte of registers (D1 = 27, D2 = -28 (E4)).
- MOVE.W (Move Word) stores 1001 (03E9) into D3.
- Indexed Addressing (MOVE.B 16(A1,D7),D5) calculates an effective memory address, retrieving a byte value (33) into D5.
- PC-Relative Add

ALGORITHM

- Initialize registers D0-D7 and A1 to 00000000, and load values into memory ($2100 - $211C).
- Set A1 to 2100 using the **LEA** instruction.
- Move immediate value 10 into D7.
- Set D0 to 27 using **MOVEQ**.
- Move immediate value 27 into D1.
- Set D2 to -28 (signed byte E4).
- Set D3 to 1001 (word 03E9).
- Move immediate value 16843009 (long 01010101) into D4.

- Retrieve a byte from memory at the address A1 + D7 + 16 and store it in D5.
- Use PC-relative addressing to move a byte from memory at 2110 to D6.
- End the program with SIMHALT.

CODE LISTING

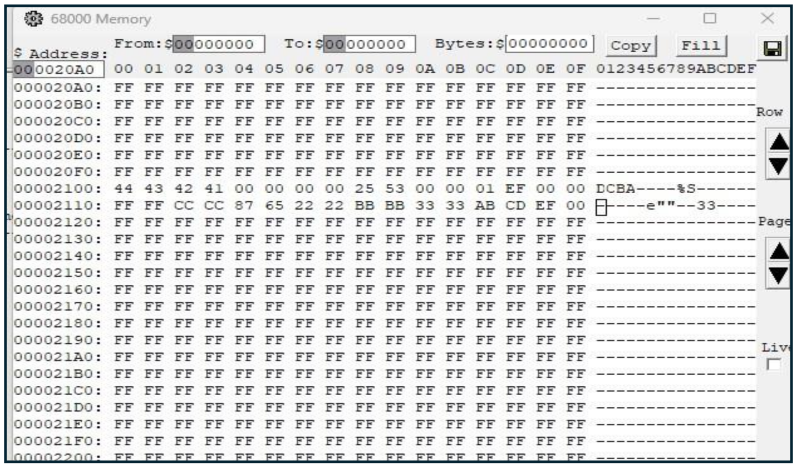
```
START: ; first instruction of program

 MOVE.L #00000000, D0
 MOVE.L #00000000, D1
 MOVE.L #00000000, D2
 MOVE.L #00000000, D3
 MOVE.L #00000000, D4
 MOVE.L #00000000, D5
 MOVE.L #00000000, D6
 MOVE.L #00000000, D7
 MOVE.L #$44434241, $2100
 MOVE.L #$00000000, $2104
 MOVE.L #$25530000, $2108
 MOVE.L #$01EF0000, $210C
 MOVE.L #$FFFFCCCC, $2110
 MOVE.L #$87652222, $2114
 MOVE.L #$BBBB3333, $2118
 MOVE.L #$ABCDEF00, $211C
 LEA.L $00002100,A1
 MOVE.L #10,D7
 MOVEQ.L #27,D0
 MOVE.B #27,D1
 MOVE.B #-28,D2
 MOVE.W #1001,D3
 MOVE.L #16843009,D4
 MOVE.B 16(A1,D7),D5
 MOVE.B $00002110(PC),D6

 SIMHALT ; halt simulator

 END START ; last line of source
```

DESCRIPTION
The objective of this lab is to explore and understand various addressing modes in 68K assembly language, including immediate, indexed, and PC-relative addressing. The lab involves initializing registers and memory, then executing a series of instructions to manipulate registers and memory. Key tasks include loading values into memory, retrieving data using different addressing modes, and observing how these instructions affect the registers. Students will gain practical experience in stepping through the code in trace mode, computing effective addresses, and verifying the results of each operation. This lab provides a hands-on opportunity to deepen understanding of low-level memory operations and assembly programming.

OBSERVATIONS

- The trace shows step-by-step instruction execution with register and memory state changes.
- Memory access operations like LDR and STR are visible.
- Branch instructions (B) control program flow by modifying the Program Counter.
- Stack operations (PUSH, POP) manage memory values.
- The trace helps analyze the program's behavior and validate correctness.

RESULTS



```
$ Address:  From:$00000000   To:$00000000   Bytes:$00000000
000020A0   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF
000020A0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000020B0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000020C0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000020D0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000020E0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000020F0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00002100: 44 43 42 41 00 00 00 00 25 53 00 00 01 EF 00 00 DCBA----%S------
00002110: FF FF CC CC 87 65 22 22 BB BB 33 33 AB CD EF 00 ⊟----e""--33----
00002120: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00002130: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00002140: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00002150: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00002160: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00002170: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00002180: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00002190: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000021A0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000021B0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000021C0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000021D0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000021E0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000021F0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00002200: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
```

CONCLUSIONS
The lab demonstrates the importance of understanding low-level assembly operations, such as memory access, branching, and stack management, by analyzing the program's execution trace. It highlights how each instruction modifies registers and memory, providing insights into program behavior. The trace analysis ensures program correctness and helps identify potential errors, enhancing debugging and optimization skills in assembly programming.