Lab Report 2.O – Reverse Vector

Description:

In this lab, we were asked to reverse a vector of characters stored in memory. However, we needed to make our data references PC relative and store our program at memory address $4ACE.

Equipment Used:

- EASy 68K simulator

Procedure:

I started by initializing memory address $4AC4 with a vector of characters (A-J). Then I set the program counter to $4ACE using the ORG operation so my program would be stored at the specified location in memory.

Next, I initialized the address registers with their PC-relative values. A0 was initialized to the start of the vector, and A1 was initialized to the end of the vector, plus one. I was unsure how to do this at first, so I referenced the Reverse Vector document from Week 5 in D2L. This is where I saw “LEA.L $004AC4(PC),A0”, which I used in my program.

Finally, the program was executed in the simulator and the memory window was checked to ensure the vector was properly reversed.

Program Description:

This program starts by storing a vector of characters at memory address $4AC4. Then it initializes A0 and A1 with the beginning and end (plus one) of the vector, respectively. This is done relative to the PC.

Next, the program copies the value in (A0) and stores it in D0. Then the value in (A1) is copied into (A0). Here, A1 and A0 are pre decremented and post incremented, respectively. Finally, the value in D0 is copied into (A1). This process is repeated five times since the vector is 10 characters long. This effectively swaps the characters at each end of the vector and moves inward over time.

Results:

The results can be seen in the screenshots below (figures 1 and 2). They show the vector before and after execution. They also show the instructions stored after the vector, starting at memory address $4ACE. That's why there are seemingly random characters near the vector.

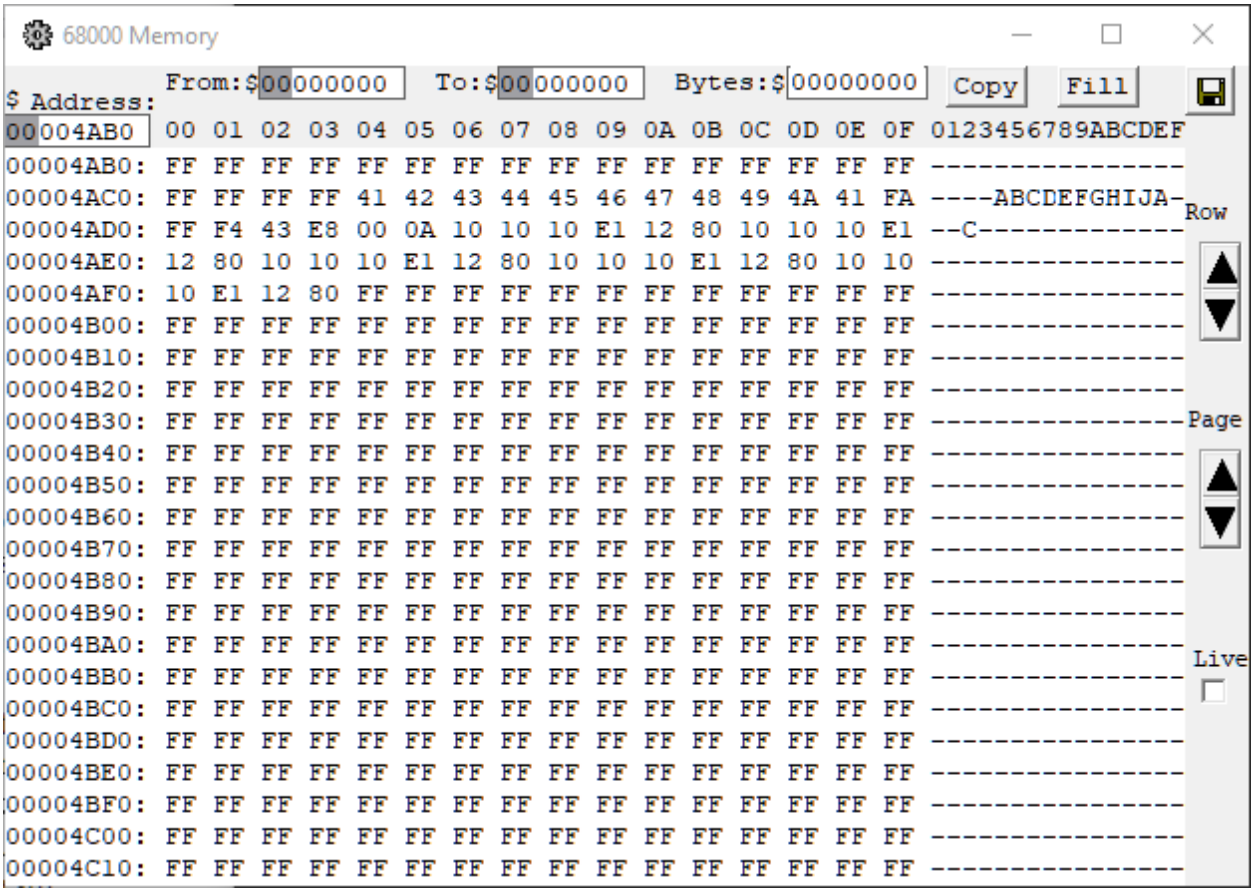

Figure 1 The memory window before execution.

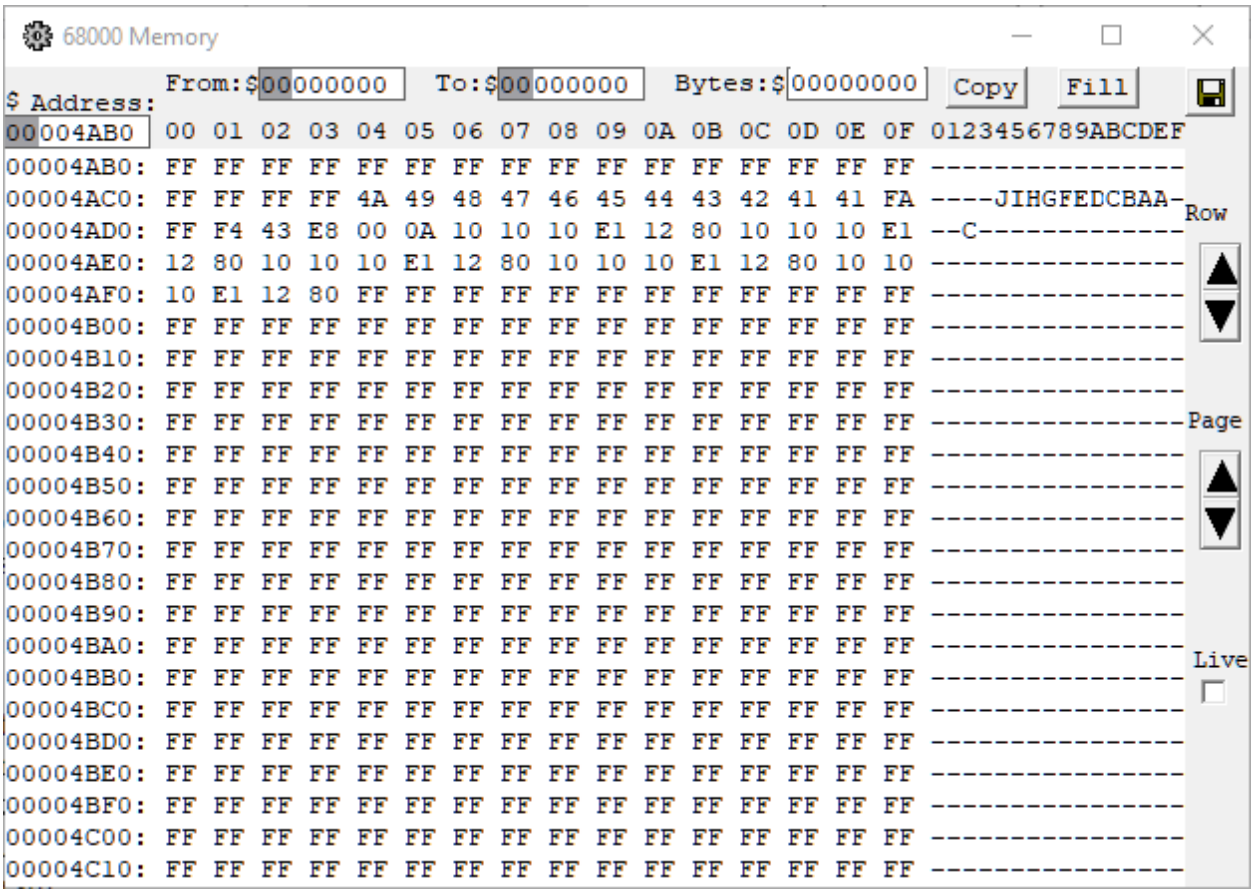

Figure 2 The memory window after execution.

Conclusions:

This lab helped me learn more about PC addressing modes in 68K Assembly. It also made me consider how non-relocatable code would affect a program. For example, most operating systems swap around pieces of memory to improve CPU utilization. If a program only references one part of memory, it may reference invalid data after things are swapped around. Of course, the OS and hardware are usually responsible for this, but it's still important to consider.

Code Listing:

```
    ORG     $4AC4
    DC.L 'ABCDEFGHIJ'
    ORG     $4ACE
START:                  ; first instruction of program
    LEA.L $004AC4(PC),A0
```

```
LEA.L 10(A0),A1

MOVE.B (A0),D0
MOVE.B -(A1),(A0)+
MOVE.B D0,(A1)

MOVE.B (A0),D0
MOVE.B -(A1),(A0)+
MOVE.B D0,(A1)

MOVE.B (A0),D0
MOVE.B -(A1),(A0)+
MOVE.B D0,(A1)

MOVE.B (A0),D0
MOVE.B -(A1),(A0)+
MOVE.B D0,(A1)

MOVE.B (A0),D0
MOVE.B -(A1),(A0)+
MOVE.B D0,(A1)

SIMHALT             ; halt simulator
END    START        ; last line of source
```