CDA 3331C

Introduction to Microcomputers

Laboratory Manual

Summer

2003

Prepared by

Dr. Bassem A. Al-Halabi

Computer Science and Engineering Department

College of Engineering

Florida Atlantic University

FLORIDA ATLANTIC UNIVERSITY
CSE DEPARTMENT
LOGIC DESIGN AND MICROPROCESSOR LAB

LD/MP Lab Rules and Regulations

1. **Use your student's ID card to admit ONLY yourself and KEEP the door always CLOSED.**
2. **No smoking / drinking / eating is allowed in the lab. Keep the entire lab CLEAN.**
3. **Do not leave your own documents and/or papers in the lab.**
4. **Lab documents may not be checked out.**
5. **Do not INSTALL / COPY / DELETE / MODIFY any software in the lab.**
6. **Keep in your possession only the chips you are using in your current (or next) experiment.**
7. **When your lab is graded, return the chips to the recycle bin and wires to the wires pan.**
8. **Be conservative on wires by using short runs (1", 2", 5", ..). Reuse used wires.**
9. **Teaching Assistants will be available in the lab at scheduled times, which are posted.**
10. **Completed experiments must be checked/graded by the TA's during scheduled lab time.**

Lab Usage Limitation

You must be enrolled in Logic Design or Microprocessor Courses to use this lab. Further, your experiment should be all thought of before taking time in the lab.

Security

You are continuously video taped for your security and against lab vandalism.

Problem Reporting

Any hardware/software failure and discovery of any lost or damaged item in the lab must be reported to the lab attendants or the CSE technical support staff.

Penalties

Any violation and/or abuse of the Lab regulations and/or facilities may result in disciplinary actions and loss of lab access privileges.

Lab Access

To obtain access to the LD/MP lab via your student ID card, you must enter your name and ISO number on a special form available on Dr. Alhalabi Web site.
By doing so, you indicate you read, understood, and accept all lab regulations.

Name: **Grade: /10**

[10] 1) This lab will help you get acquainted with the 68000 microprocessor training kit. Type the following sample assembly language program which starts at address $0900. The program adds the contents of three consecutive memory locations starting at address $0A00. The sum is stored at location $0A03. In the following subsections of the question, various commands are listed for you to explore.

```
        ORG     $0900       ;start program at this address
LAB1    MOVE.B  #01,$0A00   ;set a number on location $0A00
        MOVE.B  #02,$0A01   ;set a number on location $0A01
        MOVE.B  #03,$0A02   ;set a number on location $0A02

LINEA   CLR.L   D0          ;clear the entire D0 register
        CLR.L   D1          ;clear the entire D1 register
        CLR.L   D2          ;clear the entire D2 register
        CLR.L   D3          ;clear the entire D3 register

LINEB   MOVE.B  $0A00,D0    ;copy a byte from $0A00 to D0
        MOVE.B  $0A01,D1    ;copy a byte from $0A01 to D1
        MOVE.B  $0A02,D2    ;copy a byte from $0A02 to D2

LINEC   MOVE.B  D0,D3       ;start accumulator D3 with D0 value
        ADD.B   D1,D3       ;add to it the contents of D1
        ADD.B   D2,D3       ;add also the contents of D2
        MOVE.B  D3,$0A03    ;now store the sum in location $0A03

        MOVE.B  #228,D7
        TRAP    #14
        END
```

[3] 1.a) After you assemble and download your program to the training board (follow procedure in the previous section of this manual), you are ready to explore the following TUTOR commands. All commands are uppercase and must be followed by <Enter> key.

> **HE** Provides online help on TUTOR commands

> **.PC 900** Initialize the program counter (PC) to the first address of your program

> **MD 900** Display one line of memory contents (assembled code) staring at $900. Hit another <Enter> for a full-screen display.

> **MD 900;DI** Display memory contents (assembled code) with disassembled instructions. This allows you to know where every instruction resides. Just hit another <Enter> for a full-screen display.

> **T** Trace the program, execute one instruction at a time and observe changes. Hitting <Enter> again will repeat last command.

> **.D3** Display the contents of D3 to check answer; .D for all data registers.

> **MD A00** Display memory contents (values) staring at $0A00 to check results.

Name: **Grade: /10**

[2] 1.b) Here you learn how to manually modify the contents of selected memory location and rerun the program starting at any particular location.

> **MM A00** Manually, fill memory locations (one at a time) $0A00-0A02 all with $5, and $0A03 with 0. Use a dot to stop entering data.

> **MD A00** Check the values you just entered.

> **MD 900;DI** Display memory contents (assembled code) with disassembled instructions. This allows you to know where every instruction resides.

> **.PC 918** Set up program counter to **LINEA** address where **CLR.L D0** instruction resides (i.e. skip the first 3 move instructions).

> **T** Now trace the program again and observe the changes.

> **.D** Display contents of all data registers and verify the new values.

> **MD A00** Display memory contents and verify the new values.

[3] 1.c) Now you change the contents of data registers D0-D2 all to $B2 and rerun the program starting from an other location.

> **.D0 B2** Manually, fill D0 with $B2, repeat for D1 and D2. Make D3 equal to 0.

> **.D** Display contents of all data registers and indicate below the new values.

D0______, D1______, D2______, D3______, SR______, NZVC____

> **.PC 932** Set up program counter to **LINEC** where **MOVE.B D0,D3** instruction resides.

> **T** Trace one time only (one instruction) and indicate below the new values:

D0______, D1______, D2______, D3______, SR______, NZVC____

> **T** Trace a second time and indicate below the new values:

D0______, D1______, D2______, D3______, SR______, NZVC____

> **T** Trace a third time and indicate below the new values:

D0______, D1______, D2______, D3______, SR______, NZVC____

> **.D** Display contents of all data registers and verify the new values.

D0______, D1______, D2______, D3______, SR______, NZVC____

[2] 1.d) Explain the changes in the NZVC flags after each of the above three T commands.

____

____

____

Name:	**Grade:** /10

[10] 2) Implement the following arithmetic function which involves summation and square computations. The only input to the function is variable (a) that is initialized in register D0 at the beginning of the program to 10 and maintained thereafter. The X calculation result (X) is stored in D1 and the Y calculation result (Y) is stored in D2. The final answer (F) is stored in D3. For testing purposes, keep the above registers D0-D3 for their designated assignment (i.e. do not use them for temporary calculations).

$$F = \frac{X}{Y} \quad \text{where } X = \sum_{i=0}^{i=a} \left(5i^2 + \left\lfloor \frac{i}{2} \right\rfloor ! \right) \quad \text{and } Y = a^3$$

[2] 2.a) Write a 68000 assembly language program starting at address \$0900 which implements the above function using the following program layout. After assembly with no errors, make a print of the list file of this program and attach it to this lab experiment.

```
         ORG      $0900       ;start program at this address
LAB2     MOVE.L   #10,D0      ;initialize input variable (a) to 10
CLEAR    CLR.L    D1          ;clear D1 for X result (X)
         CLR.L    D2          ;clear D2 for Y result (Y)
         CLR.L    D3          ;clear D3 for F result (F)

XCALC    ...      ...         ;the X calculation part of your program
         ...      ...         ;taking value of D0 as an input
         ...      ...         ;and returning result (X) in D1

YCALC    ...      ...         ;the Y calculation part of your program
         ...      ...         ;taking value of D0 as an input
         ...      ...         ;and returning result (Y) in D2

FCALC    ...      ...         ;the final part of your program
         ...      ...         ;taking inputs from D1 and D2
         ...      ...         ;and returning result (F) in D3

         MOVE.B   #228,D7
         TRAP     #14
         END
```

[3] 2.b) Run your program and verify the results by examining data registers.

D0 = (a) = __________, in decimal = __10______

D1 = (X) = __________, in decimal = __________, **(X)** by hand = __________,

D2 = (Y) = __________, in decimal = __________, **(X)** by hand = __________,

D3 = (F) = __________, in decimal = __________, **(X)** by hand = __________,

Name: **Grade: /10**

[2] 2.c) Manually change the contents of D0 to 5, a new input value for variable a. Set breakpoints at locations **XCALC**, **YCALC**, and **FCALC**, which will enable you run the program in sessions so that you can observe the results progression.

D0 = (a) = __________, in decimal = _5________

D1 = (X) = __________, in decimal = __________

D2 = (Y) = __________, in decimal = __________

D3 = (F) = __________, in decimal = __________

[3] 2.d) Manually change the contents of D0 to 15, a new input value for variable a. Remove all breakpoints. Set a new breakpoint at an address, which is the first instruction of the main X calculation loop so that when you run the program, it stops at the beginning of each iteration of the loop. Start your program at locations **CLEAR** and run it one iteration at a time. For every iteration, observe the accumulating value of (X) in register D1. At the first iteration when the (X) value exceeds $06FF, stop and indicate the iteration number (**i** in the equation).

D0 = (a) = __________, in decimal = _15_______

D1 = (X) = __________, in decimal = __________

(i) = __________, in decimal = __________

[2] 2.e) **BONUS** Modify your program to automate the process of the previous part 2.d without any breakpoints or tracing. The program stops when the (X) value exceeds $06FF. The number of iterations is returned in any of the unused registers.

D0 = (a) = __________, in decimal = __________

D1 = (X) = __________, in decimal = __________ (first value higher than $06FF)

(i) = __________, in decimal = __________

Name:	Grade: /10

[10] 3) The program of this exercise deals with arrays of numbers and subroutines. First the program defines some random lists of numbers and allocates empty storage for sorted arrays, then it sorts the lists. The overall program structure should be as follows:

```
         ORG      $0900         ;start program at this address
LAB3     CLR.L    D0            ;

ARY1     DC.B     5             ;define number of elements in array1
         DC.B     25,-21,7,19,-5 ;define the elements of array1
ARY1S    DS.B     5             ;reserve location for sorted array1

ARY2     DC.B     10            ;define number of elements in array2
         DC.B     $C2,$F2,$56,$D3,$C3,$F2,$3D,$15,$17,$18
                                ;define the elements of array2
ARY2S    DS.B     10            ;reserve location for sorted array2

SORT1    ...      ...           ;set parameter (D0,A0,A1) to sort array1
         ...      ...           ;then call subroutine SORT

SORT2    ...      ...           ;set parameter (D0,A0,A1) to sort array2
         ...      ...           ;then call subroutine SORT

SORT     ...      ...           ;subroutine SORT reads the address of
         ...      ...           ;the array from A0 and stores the sorted
         ...      ...           ;array at A1. The length of the array
         ...      ...           ;is passed via D0

         MOVE.B   #228,D7
         TRAP     #14
         END
```

[3] 3.a) Complete the above 68000 assembly language program where the **SORT1** part sets the D0/A0/A1 parameters, which are used by the **SORT** subroutine to sort array **ARY1**. D0 holds the length of the array. A0 holds the address of the first element of the array. A1 holds the address where the sorted array will be stored. The **SORT2** part will, in the same analogy, sort array **ARY2**. After assembly with no errors, make a print of the list file of this program and attach it to this lab experiment.

[3] 3.b) Run your program and verify the results by using memory display commands.

[4] 3.c) Manually change the values of the arrays **ARY1** and **ARY2** and rerun your program starting at address **SORT1**. Check your results again by memory display commands.

Name: **Grade: /15**

[15] 4) The program of this exercise deals with image processing and bit manipulation. The program defines an image of 16x16 pixels entered to the memory as a 16 consecutive words each of which represent a 16-bit line of the image. The program also allocates empty storage for the results of the image manipulations performed by the three subroutines. The overall program structure should be as follows:

```
         ORG     $0900      ;start program at this address
LAB4     BSR     NEG        ;call subroutine negative
         BSR     LSR        ;call subroutine left side right
         BSR     ROT        ;call subroutine rotate
         BRA     STOP       ;stop the program

NEG      ...     ...        ;create the negative subroutine
LSR      ...     ...        ;create the left side right subroutine
ROT      ...     ...        ;create the rotate subroutine

         ORG     $0C00      ;at this address, create an image
IMAGE    DC.W    $0000      ;................
         DC.W    $0000      ;................
         DC.W    $3FE0      ;..111111111.....
         DC.W    $3FF0      ;..1111111111....
         DC.W    $3878      ;..111....1111...
         DC.W    $3838      ;..111.....111...
         DC.W    $3838      ;..111.....111...
         DC.W    $3870      ;..111....111....
         DC.W    $3FE0      ;..111111111.....
         DC.W    $3FE0      ;..111111111.....
         DC.W    $3870      ;..111....111....
         DC.W    $3870      ;..111....111....
         DC.W    $3838      ;..111.....111...
         DC.W    $3838      ;..111.....111...
         DC.W    $0000      ;................
         DC.W    $0000      ;................
         DC.L    $COCOCOCO  ;a marker at the end of original image

         ORG     $0C40      ;at this address
IM-NEG   DS.W    16         ;allocate space for negative image
         DC.L    $C4C4C4C4  ;a marker at the end of negative image

         ORG     $0C80      ;at this address
IM-LSR   DS.W    16         ;allocate space for left side right imag
         DC.L    $C8C8C8C8  ;a marker at the end of flipped image

         ORG     $0CC0      ;at this address
IM-ROT   DS.W    16         ;allocate space for rotate image
         DC.L    $CCCCCCCC  ;a marker at the end of rotated image

STOP     MOVE.B  #228,D7
         TRAP    #14
         END
```

Name:	**Grade: /15**

[5] 4.a) Complete the above 68000 assembly language program by creating the three subroutines. The **NEG** subroutine computes the negative form of the original image and stores it in the allocated space. The **LSR** subroutine flips the original image left side right and stores it in the allocated space. The **BRT** subroutine rotates the image 90 degrees clockwise. After assembly with no errors, make a print of the list file of this program and attach it to this lab experiment.

[5] 4.b) Run your program and compare the results obtained from memory with those computed by hand. All of these numbers should be in hex.

IMAGE from program =

___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___

IM-NEG from program =

___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___

IM-NEG by hand =

___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___

IM-LSR from program =

___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___

IM-LSR by hand =

___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___

IM-ROT from program =

___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___

IM-ROT by hand =

___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___

[5] 4.c) Create a new subroutine which adds a frame to the original image. The frame is formed by making the most outer surrounding bits of the original image all 1's. The frame is 1 bits thick.

IMAGE from program =

___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___

IM-FRM from program =

___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___

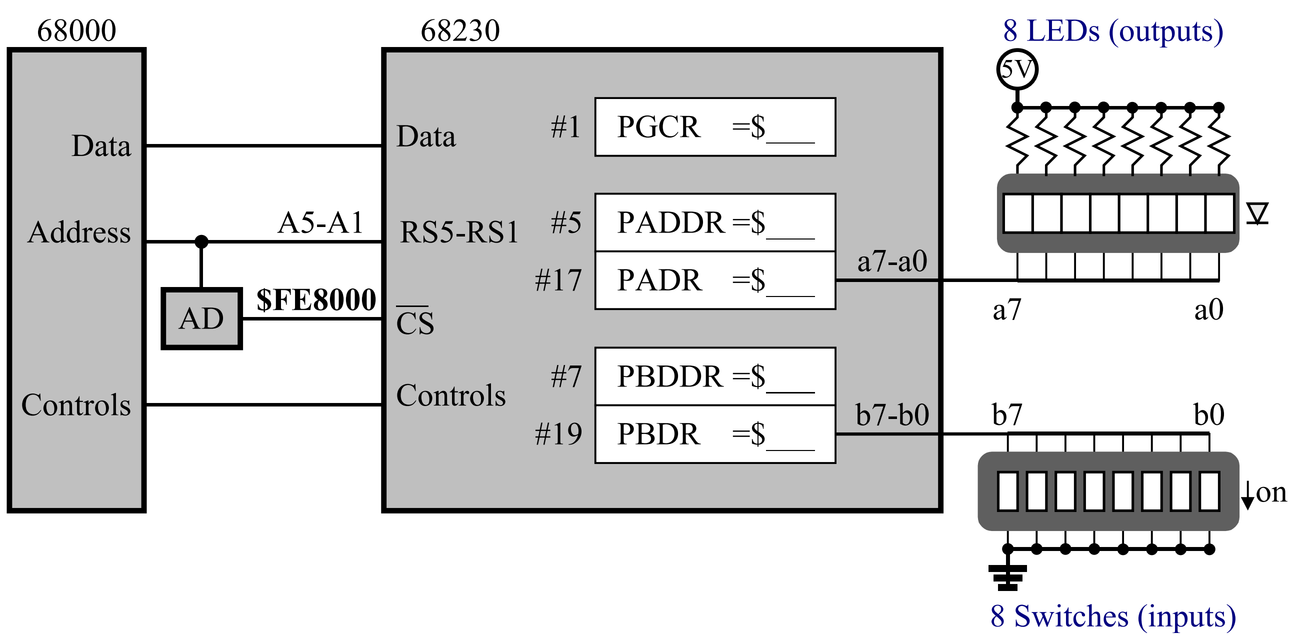
IM-FRM by hand =

___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___

Name: **Grade: /15**

[15] 5) With this lab experiment, you learn how to perform real world interface to the 68000 via a special supporting chip, the 68230 Parallel Interface and Timer (PIT). This chip is permanently wired at base address $FE8000 through address decoding circuit (AD) and high-order address lines. The low-order address lines (A5-A1) are used to select one of the 32 internal registers. Only registers 1,5,7,17 and 19 are used. Ports A and B are used to interface to the external hardware. Your program will read the 8-bit DIP switches from PortB which are used for pattern entry and control commands. The LEDs will display an 8-bit pattern outputted from PortA.

The 8 LEDs are connect to PortA with a0 and a7 as indicated in the diagram. Each LED is connected to the 5V supply through a 220Ω resisters to limit the LED current. The ground connection to the LED comes from the port output pin so that a 0V (logic 0) output will turn the LED on. The DIP switches are connected to PortB as indicated so that when a switch is turned on, it asserts a 0V (logic 0) on the port input pin. When the switch is off, the input pins will float high, (5V or logic 1) due to internal pull-up resistors. With this polarity of connection, the LEDs and switches have an Active-Low operation.



Name: **Grade: /15**

[3] 5.a) Write a small assembly language program which tests your connections as described above. The program will continuously read the 8-bit pattern from the switches and display it on the LEDs. Note that an on-switch (0 input) means a lit-LED (0 output).

```
      ORG      $900          ;start program at address #0900
PIT   EQU      $FE8000       ;initial the starting address of PIT
      LEA      PIT,A0        ;load PIT address to A0
      MOVE.B   #$00,1(A0)    ;set PIT mode0
      MOVE.B   #$FF,5(A0)    ;set port A to all outputs
      MOVE.B   #$00,7(A0)    ;set port B to all inputs
LOOP  MOVE.B   19(A0),D1     ;read switches for port B
      MOVE.B   D1,17(A0)     ;and light LEDs on port A
      BRA      LOOP          ;loop

STOP  MOVE.B   #228,D7
      TRAP     #14
      END
```

[3] 5.b) Modify the program so that switch b7 has the following function:

b7 On: The system is in read mode, i.e., the program continuously reads the switches and takes only 4 bits (b3-b0) as a pattern and display them on the LEDs (a3-a0). The other LEDs are forced off. As long as b7 switch is on, the a3-a0 pattern is continuously read and displayed.

b7 Off: When b7 switch is turned back off, the system is in run mode, i.e., the 4-bit pattern rotates (to the right) on the 8 LEDs. Slow down the rotation by using a delay subroutine.

[3] 5.c) Modify the program so that switch b6 has the following function:

b6 On: The LED pattern rotates to the left.

b6 Off: The LED pattern rotates to the right.

[3] 5.d) Modify the program so that switch b5 has the following function:

b5 On: The rotation is fast.

b5 Off: The rotation is slow.

[3] 5.e) Modify the program so that switch b4 has any other function or feature. Be creative, very creative.