

## 68K Closed Labs

In the beginning, the use of closed labs is beneficial – closed labs have a .c suffix. These labs will be conducted in a group atmosphere, and this will help students become familiar with lab partners. Subsequent labs will be of a more complex nature and knowing your lab partners will be helpful. The concepts covered in the closed labs are concerned with the instruction set and the underlying hardware. It isn't our goal to devote very much time to assembly language programming. Therefore, we will execute simple instructions and observe the results in the hardware by using trace mode and memory dumps. These labs may or may not be the actual labs assigned – variations may be useful at times.

## Lab 1.C - Copy Program

Your first assignment is simply a copy program. That is, you are to duplicate the code that is supplied on the last page of this assignment (page 42). A goal here is to have you interact with the SBC68000 board to create data in memory and execute a small program.

When you do the portion which constructs the program in memory **MM 2060;DI** on this page, the computer will first inform you of the contents of the current memory location, and then print a "?". You then enter the instruction which you want in that location in memory, **but** make sure to put a space after the "?". After you enter an instruction, the next instruction that is in memory is displayed followed by a "?". This process continues until you have completed entering instructions. Then you enter a "." followed by a carriage return. This returns you to TUTOR. The actual code you are to enter is contained on the last page of this lab and follows the command

**MD 2060 128;DI.**

You enter only the instruction shown. Your session with the MC68000 is as follows where your input is shown in boldface type. Follow this exactly.

TUTOR 1.32> **MS 2000 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'**

TUTOR 1.32> **MS 2020 'abcdefghijklmnopqrstuvwxyz'**

TUTOR 1.32> **MS 2040 '0123456789'**

TUTOR 1.32> **MM 2060;DI**

*(enter the code here)*

Once the code has been entered, your session is to continue with the following commands. All commands and the output generated by them will appear on the printer output. That output is to be submitted for credit on or before the due date. Again, the information in boldface type is that which you are to enter and the other is produced by the MC68000. Consult the Board User's Manual for the meaning of all the commands used here. If you have any questions, ask your instructor.

TUTOR 1.32> **PA**

TUTOR 1.32> **BR 20CA**

BREAKPOINTS  
0020CA

TUTOR 1.32> **GO 2060**

TUTOR 1.32> **MD 2000 256**

Code to be inserted:

```
LEA.L    $002000,A2
LEA.L    $002020,A3
LEA.L    $002040,A4
LEA.L    $002200,A1
MOVE.B   $002016,(A1)+
MOVE.B   $002024,(A1)+
MOVE.B   $00202B,(A1)+
MOVE.B   $002022,(A1)+
MOVE.B   $00202E,(A1)+
MOVE.B   $00202C,(A1)+
MOVE.B   $002024,(A1)+
MOVE.B   #32,(A1)+
MOVE.B   19(A3) ,(A1)+
MOVE.B   14(A3) ,(A1)+
MOVE.B   #32,(A1)+
MOVE.B   2(A2) ,(A1)+
MOVE.B   18(A2) ,(A1)+
MOVE.B   2(A2) ,(A1)+
MOVE.B   8(A2) ,(A1)+
MOVE.B   #32,(A1)+
MOVE.B   2(A4) ,(A1)+
MOVE.B   8(A4) ,(A1)+
MOVE.B   (A4) ,(A1)+
MOVE.B   #33,(A1)+
MOVE.B   #33,(A1)+
MOVE.B   #33,(A1)+
MOVE.B   #32,(A1)+
```

## Lab 2.C - Addressing Modes

The goal of this experiment is to study memory partitions and addressing modes. This is accomplished by creating specific items in memory and then executing instructions which operate on those items. You will also be required to initialize some of the processor registers.

The following information represents memory setup and processor registers. Your task is to determine the changes in all respects caused by executing the following instructions using pencil and paper. The results of the pencil and paper portion are to be turned in at the end of the lab along with all of the other materials which are required.

		addr.	contents
A3	002468FA	2518	4433
A4	00002544		4241
A6	00002518		0000
D3	00000000		0000
D5	FFFFFFFF		2553
D6	00000000		0000
			01EF
			ABCD
			5476
			CC22
			FF34
			12FF
			.
			.
			.
		2544	A267
			1FEE
			FFFF
			FFFF
			0100
			0000
			ABCD
			FFFF
			0000
			0000

### MEMORY

00260E	162E0003	MOVE.B	3(A6) ,D3
002612	3D6C00040006	MOVE.W	4(A4) ,6(A6)
002618	2A1E	MOVE.L	(A6)+,D5
00261A	31DC2522	MOVE.W	(A4)+,\$002522
00261E	1938252E	MOVE.B	\$00252E,-(A4)
002622	47F8252A	LEA.L	\$00252A,A3
002626	3C13	MOVE.W	(A3) ,D6

## INSTRUCTIONS

You are to now actually execute your instructions in **TRace** mode. That is, each instruction will be executed one at a time at your command from the keyboard.

Prior to entering **TRace** mode, one must first initialize the Program Counter (PC). This can be accomplished by entering

```
.PC 260E
```

and then going into **TRace** mode.

After execution, you need to display the contents of memory to determine all that has happened. You need to turn in a **copy of memory before and after execution** as well as the **TRace** mode listing.

## Lab 3.C - More Addressing Modes

The goal of this experiment is to continue the study of memory partitions and addressing modes. This is accomplished in the same manner as Closed Lab 2 by creating specific items in memory and then executing instructions which operate on those items.

The following information represents memory setup and processor registers. Your task is to determine the changes in all respects caused by executing the given instructions. You are to compute the results of the following instructions prior to the lab on pencil and paper. Those results are to be turned in at the end of the lab along with all other required materials.

A1	00000000	2100	44434241
D0	00000000		00000000
D1	00000000		25530000
D2	00000000		01EF0000
D3	00000000		FFFFCCCC
D4	00000000		87652222
D5	00000000		BBBB3333
D6	00000000		ABCDEF00
D7	00000000		

### REGISTERS

### MEMORY

002010	43F82100	LEA	\$002100,A1
002014	2E3C0000000A	MOVE.L	#10,D7
00201A	701B	MOVEQ	#27,D0
00201C	123C001B	MOVE.B	#27,D1
002020	143CFFE4	MOVE.B	#-28,D2
002024	363C03E9	MOVE.W	#1001,D3
002028	283C01010101	MOVE.L	#\$01010101,D4
00202E	1A317010	MOVE.B	16(A1,D7),D5
002032	1C3A00DC	MOVE.B	\$002110(PC),D6

### INSTRUCTIONS

You are to actually execute your instructions in **TRace** mode. That is, each instruction will be executed one at a time at your command from the keyboard. At the end of the closed lab hour, you must turn in the pencil and paper results as well as the **TRace** mode listing.

## Lab 4.C - Linked Data Structures

The goal of this lab is to introduce the concepts of pointer types and linked data structures at the architecture level.

Construct a MC68000 memory image of **Figure 2.2** found in the chapter on addressing modes and data structures. Load A5 with the address \$0074B0 (do **not** do this in the code, do this before you run your program). Insert the following code which establishes pointers to  $e_2$  and  $e_3$ :

```
LEA      $0074A8,A6      .A6 points to e1
MOVE.L   4(A6),A1        .A1 points to e2
MOVE.L   4(A1),A2        .A2 points to e3
```

Insert the element that A5 points to between those two elements. You must turn in a copy of the linked list (memory display) prior to and after inserting the new element. You are also required to turn in a copy of your instructions for this operation.

## Lab 5.C - Control Structures

The goal of this lab is to investigate the ability of the architecture to alter flow of control through memory space.

Compose a sequence of operations which will find the largest of three bytes. These bytes are located in D3, D4 and D5. Choose any appropriate values you want for these registers. But make sure to test the code thoroughly

Use a flow chart to lay out your plans prior to entering code. Try all possible combinations of values to check out your algorithm! Put the greatest value in D<sub>0</sub>.

Turn in one TRace mode execution of your program and your program listing. Also turn in your test runs (not TRace mode). Don't forget to turn in your flow chart.

## Lab 6.C - Logical and Shifting Instructions

The goal of this lab is to examine the logical and shifting instructions available in the architecture. The lab also illustrates a typical problem space in which these commands are useful.

This lab is concerned with logical/shift operations. You are to begin by creating data in memory as follows:

.D4 A59BD2FC (remember that these are **hex** values)

The remainder of your assignment is to decompose this long word into “nibbles”. A “nibble” is four bits. You are to do this using the logical and shift operators. When you are completed, D4 should be as it started. The decomposed nibbles are to appear in bytes in memory beginning at location \$002200. The nibbles are to be left-justified in those bytes.

You are to turn in the following at the end of this lab:

1. **TR**ace mode operation for decomposition of the first two nibbles. After the first two nibbles are decomposed, type **GO** to complete the decomposition.
2. Memory display of the decomposed longword.

## Lab 7.C - Procedure Protocol

This lab is concerned with calling protocols used in communicating with procedures. Your assignment here is to call a procedure which will add the two arguments which are passed using the stack convention. The two arguments are passed differently. The first is passed by reference, and the second is passed by value. Both arguments are bytes.

For the first argument, create a byte at memory location \$002100 which has the value 2. The second argument to be passed by value also has the value 2.

Your procedure is to act as a function. It will simply add the two values and return the result in a manner typical of function behavior.

The calling procedure is to reside at memory location \$002000 and the called procedure at location \$002050. Turn in a listing of both the calling and called program segments. Also, turn in a run of the calling program showing that it did execute. Prior to running the calling program, clear D0 as follows:

```
.D0 00000000
```

## Open Labs

These labs are sufficiently large or complex enough that they cannot be accomplished in the closed lab atmosphere. As these labs become more complex, your group will be required to meet and plan your strategy for solving the assigned problem. One particular item necessary here then is a “division of labor”. That is, exactly what each group member’s responsibility is for solving the problem. For more details, see the section on “Group Policy” found in the manual for Architecture I on page 26.

## 68K

### Lab 1.0 - Name and Social Security Number

Rewrite the copy program (Lab 1.C) so that it will create your name beginning at memory location \$02100 and social security number beginning at location \$02160. You are to do this using the following addressing modes in your program:

Address register direct	An
Address register indirect	(An)
postincrement	(An)+
predecrement	-(An)
displacement	d(An)

You can use these anywhere in your program. You need to submit the following for your grade:

1. a printout of your code;
2. a printout of memory locations \$2100 through \$2180 after execution;

No documentation of the code is required.

## Lab 2.0 - Program Relocatable Code

This lab is concerned with program relocatable code. PC relative addressing modes are useful here. In a multiple user environment, a user's program should be capable of residing anywhere in memory. That is, user programs should be address independent.

Your assignment here is to reverse the elements of a vector. The procedure is as follows:

1. Create a vector of the ASCII characters 'A ... J' at location \$004AC4.
2. Construct a sequence of operations beginning at location \$004ACE which will reverse the elements of the vector.
3. **Display** both the ASCII vector and the instructions.
4. Execute your program.
5. **Display** the vector to insure that it was indeed reversed.
6. Perform a **Block Move (BM)** of the entire package (vector and instructions) to a new location at \$002000.
7. **Display** your program in its new location and then execute it.
8. **Display** the vector to insure that it was again reversed. The vector should now be as it was initially, only in a new location.

In order for all of this to work, references to data must be PC relative. Turn in all items which you were asked to display along with your evaluation of the lab. Consider the consequences of not being able to relocate your programs or having to specify a particular memory address to insure proper program execution.

## Lab 3.0 - Condition codes and Bcc

This lab is concerned with the fundamentals of control structures. The 68000 manuals contain all the Bcc information in concise tables. However, our interest in architecture includes a hardware insight. We will design some of the hardware for facilitating control structures in this lab.

Suppose that a Bcc instruction exists in the Instruction Register (IR). Then, bits 11 through 8 represent the condition to be evaluated. We can then decode those four bits and thus, determine which condition is to be evaluated. We will use four data switches to represent bits 11 through 8 in the IR. The other players here are the condition codes. We consider here only N, Z, V and C. Use the other four data switches to represent these entities.

Although there are fourteen different conditions, we will consider only five here to make the problem reasonable. Those five are:

EQ,  
GE,  
HI,  
LS and  
MI.

Consider a Bcc instruction in the IR and the condition codes in the Status Register - when the condition is true, the instruction at PC + d is executed next; otherwise the instruction at PC is executed. Your task in this lab is to design and implement in B<sup>2</sup> Logic the combinatorial logic to determine whether the condition is true or false and register that information using an led. You are to use the leds on the Digital Designer to illustrate bits 11 through 8 and N, Z, V and C. For this reason, you will need an external led to demonstrate true or false. Information regarding led wiring is found in the section on Led and Switch Connections.

Once your circuit is complete and working properly, you are to demonstrate its operations. Again, be prepared to answer questions regarding your design. You need to turn in a neatly documented circuit diagram and your report.

You must also turn in a disk or e-mail your circuit to the grader at the address provided.

## Lab 4.0 - Hamming Code

The Hamming code is a special technique for encoding and decoding information to enable error detection and correction. Richard Hamming published his work in 1951. The high cost of hardware in the past made this technique costly and, therefore, not practical. Recently, a CRAY computer design was released to the marketplace which uses single error correcting techniques as described in the Hamming paper.

### Encode

A significant feature of the Hamming code is that all components of the algorithm are mathematically sound and practical. The encoding procedure is to compute "even parity" bits based on the information to be transmitted and send those parity bits along with the transmitted information. If the information contains 4 bits (abcd), then three parity bits (rst) are needed. These parity bits are called check bits. The encoded message is then structured as:

position	7	6	5	4	3	2	1
bit	a	b	c	r	d	s	t

where:

r is set to create even parity for bits 7, 6, 5 and 4;  
 s is set to create even parity for bits 7, 6, 3 and 2;  
 t is set to create even parity for bits 7, 5, 3 and 1.

You are to encode the contents of a byte in memory where the information bits a, b, c and d are located as shown in the following:

xxxabcd

For example, suppose that you have the following memory configuration, your task would be to compose the encoded bytes as shown:

	memory	encoded message
2274	xxx1011	01010101
2275	xxx1001	01001100
2276	xxx0011	00111110

### **Decode**

The process of decoding is to examine the received binary code and determine the position of any possible error. The position of the error is indicated after evaluating the check bits (rst). Thus, if the bit pattern (7, 6, 5, 4) has even parity,  $r = 0$ ; otherwise,  $r = 1$ . If the bit pattern (7, 6, 3, 2) has even parity, the check bit  $s = 0$ ; otherwise,  $s = 1$ . If the bit pattern (7, 5, 3, 1) has even parity,  $t = 0$ ; otherwise,  $t = 1$ . The pattern (rst) is then the binary equivalent of the position

where the error exists. The case where  $r = s = t = 0$  is that where no error occurred in transmission.

Suppose you have received the following encoded signals and they were stored in memory as indicated, then your analysis of those memory locations would produce the error position information as shown in the following:

	received message	error position
3675	01010101	000
3676	01000101	101
3677	00111110	110

Your task here is to write two separate packages of code. One which will encode the information (abcd) found in the lower nibble of the byte whose memory location is \$002400. The encoded result is placed in D1. The other will decode the information found in the byte at location \$002500 and place the position of any error in D0.

Document your code! Execute each package using an example from this handout. Try your code with other groups if you want. Finally, your code will be tested by the instructor or the grader for the course! Don't forget your evaluation of the lab.

## Lab 5.0 - Shifting

This lab is concerned with the shifting operations provided by the architecture. In this lab you will implement some of the available shifting operations using 74194 chips. The 74194 chip is a universal shift register. Its circuit diagram is on the data sheet section of Appendix A. Its operations are facilitated by using control lines. These are labeled  $s_0$  and  $s_1$  and their functions are also given on the data sheet.

You are to implement all of the shifting functions in B<sup>2</sup> Logic:

ROL Left circular shift,  
 ROR Right circular shift,  
 ASL Left arithmetic shift,  
 ASR Right arithmetic shift,  
 LSL Left logical shift, and  
 LSR Right logical shift.

Your construction will consist of two 74194 chips connected to make an 8-bit shift register. You can think of this as  $D_n$  with the constraint that  $D_n$  has only 8 bits. When implementing these functions, it would be a neat trick to implement each function simply by setting data switches and then use a logic switch on the designer to manually clock the shifter the required number of times. The switches act as bits in the instruction word of the shifting functions. There is a certain set of bits for which the instruction word of each function possesses a unique combination. The 68000 has internal control lines which are activated by decoders, etc., to decide which operation to perform. You need to design your own logic to activate control lines and thus, implement these functions.

Meet to discuss your strategy. When you have completed your design, submit a list of logic devices which are needed in addition to the 74194's. Use the leds on the circuit designer to display the contents of the shift register. Also, use data switches on the designers to load information into the shift register for testing purposes.

A report for the lab as well as a neatly drawn and well documented logic diagram are to be submitted for a grade. You must also turn in your B<sup>2</sup> Logic circuit via disk or e-mail to the grader and include instructions as to it's operation in your lab report.

## Lab 6.0 - Parity Function

In Lab 4 it was necessary to find the number of 1's in a particular memory partition in order to determine even or odd parity. It would have been nice to have had a function to accomplish that for us. Now that we know how to create functions in the architecture, we will do just that.

The function you are to create will determine the parity of a byte (1), word (2) or longword (4). The numbers in parentheses are used to indicate the size of the argument to be investigated. The argument itself is passed by value. The stack convention is used to pass parameters. Thus, a call to your function would be of the form:

Parity(value,size).

The function in turn returns the value "1" if there are an odd number of 1's in the passed value and "0", otherwise. In particular, consider the following:

Parity (01100101,1)

which has the value 0.

The code to use to call your function is the following:

```

002000      2F00      MOVE.L    D0,-(A7)
002002      2F01      MOVE.L    D1,-(A7)
002004      6100FFA   BSR      $003000
002008      4E71      NOP

```

You can then do

```
.DO xxxxxxxx
```

and

```
.D1 yyyyyyyy
```

to set up different values to test your function completely.

It is reasonable to consider a hardware solution of this problem. In fact, you have already done so. What is it? Our goal here is function protocol. Therefore, we will compose the appropriate function and execute it.

## Lab 7.0 - Recursion

This assignment is concerned with recursive procedures. Code to implement **Fibonacci(n)** in a recursive manner can be found on the following page. **Fibonacci(n)** is defined as follows:

n	Fibonacci(n)
0	1
1	1
2	2
3	3
4	5
5	8
6	13
.	.
.	.
.	.
n	fibonacci(n-1) + fibonacci(n-2)

You are to test the 68000 architecture **as we know it** to determine the greatest n which will correctly run on our system. This may vary from one run to another depending on the amount of stack available. Determine how many recursive calls are made for your max data and the number of longwords stacked. Determine the maximum number of calls and longwords as a function of n. What can you do to improve computability as far as maximum n? Try some things and report on them in your evaluation. Experiment! You are to provide adequate comments in the appropriate area of the function.

The following code can be used for experimenting with the architecture. The purpose of experimenting is to witness the overhead required in order for the 68000 architecture to facilitate recursive procedures. Given this code, complete Lab 7.0 as stated on the previous page. You can pass values to the main program for testing by using D7 in following manner.

```
.D7      xxxxxxxx                                comments

002000      MOVE.L      D7,-(A7)
002002      BSR        $002010
002004      ADDQ       #4,A7
002006      NOP
002008      NOP
00200A      NOP
00200C      NOP
00200E      NOP
002010      MOVEM.L    D1-D2,-(A7)
002014      MOVE.L    #1,D0
00201A      MOVE.L    12(A7),D1
00201E      CMP.L     #1,D1
002024      BLE       $00203A
002026      SUBQ     #1,D1
002028      MOVE.L    D1,-(A7)
00202A      BSR        $002010
00202C      ADDQ     #4,A7
00202E      MOVE.L    D0,D2
002030      SUBQ     #1,D1
002032      MOVE.L    D1,-(A7)
002034      BSR        $002010
002036      ADDQ     #4,A7
002038      ADD.L    D2,D0
00203A      MOVEM.L    (A7)+, D1-D2
00203E      RTS
```

Set a Breakpoint at \$002006.

## RISC Labs

### Lab 1.0 – Single Cycle Processor Design

The purpose of this lab is to acquaint the architecture student with the design of datapath and control in the single cycle RISC processor. To help you understand the integrated processor architecture design the following:

1. DataPath and control signals to run a program consists of many R type instructions such as Add \$1 \$2 \$3
2. DataPath and control signals to run a program consists of many SW \$1 offset (\$2) instructions
3. Data path and control signals to run a program consists of many branch instructions such as beq \$1 \$2 offset
4. Datapath and control path to run a program consists of many R format and I Format instructions such as lw, add, sub. Your design should put together all independent data paths. To share many elements in the data path use a mux with a data selector.
5. Datapath and control path to run a program consists of R format and I Format instructions such as sw, add, sub. Your design should put together all independent data paths. To share many elements in the data path use a mux with a data selector.
6. Datapath and control path for R format, I Format and J Format instructions such as lw, sw, add, sub and beq. Your design should put together all independent data paths. To share many elements in the data path use a mux with a data selector.

You should design your circuits for verification and submit a **neatly** detailed and documented circuit diagrams.

## Lab 2.0 – Design of Controllers for Single Cycle Processor

The purpose of this lab is to acquaint the architecture student with the design of controllers in the single cycle RISC processor. Now we will be dealing with hardwired control. However, we are only concerned with the hardware of several instructions. Your assignment here is to design controller circuits using PLAs and the logic circuitry to generate control signals for the following or similar R, I and J instructions.

- Add
- Lw
- Sw
- Branch
- Jump

For this assignment use the complete set of control signals discussed in the class.

You should design your PLA circuits for verification and submit a **neatly** detailed and documented circuit diagram.

## Lab 3.0 – Multi Cycle processor Design

The purpose of this lab is to acquaint the architecture student with the design of datapath and control in the Multi Cycle RISC processor. To help you understand the integrated processor architecture design the following:

1. DataPath and control signals to run a program consists of many R type instructions such as Add \$1 \$2 \$3
2. DataPath and control signals to run a program consists of many SW \$1 offset (\$2) instructions
3. Data path and control signals to run a program consists of many branch instructions such as beq \$1 \$2 offset
4. Datapath and control path to run a program consists of many R format and I Format instructions such as lw, add, sub. Your design should put together all independent data paths. To share many elements in the data path use a mux with a data selector.
5. Datapath and control path to run a program consists of R format and I Format instructions such as sw, add, sub. Your design should put together all independent data paths. To share many elements in the data path use a mux with a data selector.
6. Datapath and control path for R format, I Format and J Format instructions such as lw, sw, add, sub and beq. Your design should put together all independent data paths. To share many elements in the data path use a mux with a data selector.

You should design your circuits for verification and submit a **neatly** detailed and documented circuit diagrams.

## Lab 4.0 – Design of Controllers for Multi Cycle processors

The purpose of this lab is to acquaint the architecture student with the design of controllers in the multi cycle RISC microprocessor. Now we will be dealing with hardwired control. A complete hardwired controller could be constructed using finite state machines or microprogramming. However, we are only concerned with the hardware of several instructions. Your assignment here is to design controller circuits using PLAs and the logic circuitry to generate control signals for the following or similar R, I and J instructions.

- Add
- Lw
- Sw
- Branch
- Jump

For this assignment use the complete set of control signals discussed in the class.

You should design your PLA circuits for verification and submit a **neatly** detailed and documented circuit diagram.

## **Lab 5.0 – Microprocessor Design using a Simulation Language**

Using a hardware simulation language such as Verilog implement a functional simulator for

1. single cycle processor
2. multi cycle processor
3. Pipeline processor

## **Lab 6.0 – Microprocessor Design**

Using standard parts build

1. single cycle processor
2. or a multi cycle processor

to implement ten instructions.

## X86 Labs [1]

### Lab 1 I/O processing with X86

The objective of this lab is to analyze, edit, assemble, link and run X86 programs.

#### In Lab Exercise:

Following instructions will help you to edit, assemble, link and run X86 programs.

To edit a program use notepad  
Save the file with .asm extension.

To assemble the program

C: tasm pgm4\_1;

To link

C:tlink pgm4\_1;

To run

Pgm4\_1

#### In lab Exercise:

Copy and run

1. ECHO PROGRAM
2. PRINT STRING PROGRAM
3. CASE CONVERSION PROGRAM

#### Post Lab Exercise:

1. Write a program to (a) display a "?", (b) read two decimal digits whose sum is less than 10, (c) Display them and their sum on the next line, with an appropriate message.

Sample Execution:

?54

THE SUM OF 5 AND 4 IS 9

2. Write a program to (a) prompt the user, (b) read first, middle and last initials of a person's name, and (c) display them down the left margin.

Sample Execution:

ENTER THREE INITIALS: JFK

J

F

K

### Lab 2 X86 Debugging

The objective of this Lab is to use the DEBUGGING environment to learn the way instructions affect the flags.

In Lab Practice Exercise

Write a program to illustrate the use of DEBUG to check the flags.

To enter DEBUG with demonstration program, type

C:\DEBUG A:\PGM.EXE

DEBUG responds by its prompt, “-“, and waits for a command to be entered. To view the registers type “R”. The display shows the contents of the registers in hex.

To step through the program, use the “T” (trace) command.

To complete execution of the program, type “G” (go).

And to exit DEBUG, type “Q” (quit).

**Post Lab Exercise:**

**Write programs to do two of the following.**

1. For each of the ADD, SUB, DEC, NEG, XCHG following instructions, give the new destination contents and the new settings of CF, SF, ZF, PF, and OF. Flags are initially 0 in each part of the question.
2.
  - a. Both AX and BX contain positive numbers, and ADD AX,BX is executed. Show that there is a carry into the msb but no carry out of the msb if, and only if, signed overflow occurs.
  - b. Both AX and BX contain negative numbers, and ADD AX,BX is executed. Show that there is a carry out of the msb but no carry into the msb if, and only if, signed overflow occurs.
3. In the following, the first number is the contents of AX, and the second number is the contents of BX and ADD AX,BX is executed. Find the resulting value of AX and tell whether signed or unsigned overflow occurred.

$$\begin{array}{r} \text{a.} \quad 512\text{Ch} \\ + 4185\text{h} \end{array}$$

$$\begin{array}{r} \text{b.} \quad \text{FE}12\text{h} \\ + 1\text{ACBh} \end{array}$$

$$\begin{array}{r} \text{c.} \quad \text{E}1\text{E}4\text{h} \\ + \text{DAB}3\text{h} \end{array}$$

4. The first number of the following is the initial contents of AX and the second number is the contents of BX. SUB AX,BX is executed. Give the resulting value of AX and tell whether signed or unsigned overflow occurred.

$$\begin{array}{r} \text{a.} \quad 2143\text{h} \\ - 1986\text{h} \end{array}$$

$$\begin{array}{r} \text{b.} \quad 81\text{FEh} \\ - 1986\text{h} \end{array}$$

$$\begin{array}{r} \text{c.} \quad 19\text{BCh} \\ - 81\text{FEh} \end{array}$$

## **Lab 3      X86 Jump Instructions**

The objective of this Lab is to learn and apply X86 Jump Instructions.

### **In lab Exercise 1**

Write a program to display the entire IBM character set. Use jump instructions in your program.

### **In lab exercise 2**

Write a program that prompts the user to enter a line of text. On the next line, display the capital letter entered that comes first alphabetically and the one that comes last. If no capital letters are entered, display “No capital letters”.

### **Post Lab Exercise:**

Write a program to display the extended ASCII characters (ASCII codes 80h to FFh). Display 10 characters per line, with the characters separated by blanks. Stop after the extended characters have been displayed once.

## Lab 4 Logic, Shift and Rotate Instructions

The objective of this Lab is to learn and apply X86 Logic Shift and Rotate Instructions.

Logic Instructions have the ability to manipulate individual bits. The shift and rotate instructions shift bits in the destination operand by one or more positions either to the left or right. For a shift instruction, the bits shifted out are lost; for a rotate instruction, bits shifted out from one end of the operand are put back into the other end. The instructions have two possible formats.

### In Lab Exercise

Write a program that prompts the user to enter a character and prints the ASCII code of the character in HEX on the next line. Repeat the process until the user types a carriage return.

### Post Lab Exercise:

Write a program that prompts the user to enter a character, and on subsequent lines prints its ASCII code in binary, and the number of 1 bits in its ASCII code.

*Sample Execution:*

TYPE A CHARACTER: C

**THE ASCII CODE OF C IN BINARY IS: 01000011**

THE NUMBER OF 1 BITS IS: 3

## Lab 5 Stack and Procedures

The objective of this Lab is to learn and apply X86 Stack and Procedure instructions.

### In lab Exercise 1

Write a program using the stack's LIFO property to read a sequence of characters and display them in reverse order on the next line. Push, pushf, pop and popf instructions are used to manipulate the stack in “last-in, first-out” manner.

### In lab Exercise 2

Unsigned multiplication can be implemented on the computer by addition and bit shifting. Write a program with a procedure for finding the product of two positive integers A and B. To invoke a procedure, the **CALL** instruction is used. To return from the procedure, the instruction RET is used.

### Post Lab Exercise:

Write a program that evaluates an algebraic expression which contains ( ), { } [] and ending with a carriage return. The program should evaluate each character as the expression is being typed in. After the carriage return is typed, if the expression is not correct, the program displays “expression is incorrect.” Else the program displays “expression is correct”. In both cases, the program asks the user if he or she wants to continue. If the user types ‘Y’, the program runs again.

You **must** turn in a disk or e-mail your program code in a “.asm” file.

## Lab 6 Display and Key Board Programming

The objective of this Lab is to learn and apply Display and Key Board Programming techniques.

### In lab Exercise

Write a program to do the following:

1. Set the display to mode 3 (80 x 25 16-color text).
2. Clear a window with upper left corner at column 26, row 8, and lower right corner at column 52, row 16, to red.
3. Move the cursor to column 39, row 12.
4. Print a blinking, cyan "A" at the cursor position.

To control the screen use the BIOS video screen routine which is invoked by the INT 10h instruction; a video function is selected by putting a function number in the AH register.

### In Lab Exercise 2

Write a program to implement a Screen Editor with basic functions of a word processor. It first clears the screen and puts the cursor in the upper left corner, then lets the user type text on the screen, operate some of the function keys, and finally exits when the Esc key is pressed.

### Post Lab Exercise:

#### Write a program to

- a. Clear the screen, make the cursor as large as possible, and move it to the upper left corner.
- b. Program the following function keys

Home key: Cursor moves to the upper left corner.

End key: Cursor moves to the lower left corner.

PgUp key: Cursor moves to the upper right corner.

PgDn key: Cursor moves to the lower right corner.

Esc key: Program terminates

Any other key: Nothing happens.

You **must** turn in a disk or e-mail your program code in a ".asm" file.

**References:**

Maruth, IBM PC Assembly Language, McGraw Hill  
Embedded Controllers, Barry Brey, Prentice Hall

## Web Assignments

1. . Your first assignment is to investigate the latest developments of new processors through the web.
2. Search the web to locate a 68K software simulator that runs assembly language programs for 68K architecture. Download the simulator and run several assembly language programs on the simulator. Generate the results and write a report.
3. SPIM simulator that runs assembly language programs for MIPS architecture is available through [www.mkp.com/code.htm](http://www.mkp.com/code.htm). Download SPIM simulator and run several assembly language programs on SPIM.
4. Search the web to locate a x86 software simulator that runs assembly language programs for Intel x86 microprocessor architectures. Download the simulator and run several assembly language programs on the simulator. Generate the results and write a report.
5. Through the web survey information for several RISC architectures. Write a one-page article based on your survey.
6. Through the web find information on instructions and addressing modes for several CISC architectures. Write a one-page article based on your survey.
7. Through the web survey information for several multiprocessor architectures. Write a one-page article based on your survey.
8. Your last assignment is to find documentation on the fastest and/or largest cpu, cache, main memory, disk drive and anything else having to do with time and space. These items must currently be for sale. They don't have to be from the same manufacturer. The idea is that we will purchase those parts and put them together to produce the largest and fastest machine ever. Use the web, ACS employees and anything else that comes to mind. You must make sure that the information is real and documented.

## Appendix A

### More 68K Labs

#### Lab 12.0 – 68K Microprocessor Hardwired Control

The purpose of this lab is to acquaint the architecture student with the design of control circuits in the microprocessor. We will be dealing with hardwired control. A complete hardwired controller could be constructed. However, we are only concerned with the hardware of several instructions. Your assignment here is to construct a *Control step counter* and the encoding logic circuitry to generate control signals for the following or similar instructions.

```
MOVE.L (A3)+,D6
```

and

```
MOVE.W D5,12(A5)
```

You are to assemble the instructions into MC68000 hexadecimal code and then determine the control sequences for execution. The microprocessor architecture used is that shown in **Figure 1.1** in chapter 1 of the manual. Use **PC<sub>increment</sub>** as discussed in class to update the PC as required. Also, use **SIZE<sub>select</sub>** to adjust address registers when needed. A complete set of control lines for this assignment is given on page 58.

You will treat MFC as an **asynchronous** signal which can be controlled by a logic switch on the digital designer. Since MFC is asynchronous, your logic will need to stop the control step counter until MFC is actuated. The entire sequence of control signals should also be initiated by an asynchronous reset signal which you have at your control. The Control step counter can be designed using 74194 chips as a Johnson counter to generate the steps. See page 36 of the manual for 74194 details.

You are required to display the control step times – perhaps using leds. Using the clock at its lowest speed, you can observe the required signals more readily than at higher speeds. In fact, you may want to use a manual clock (logic switch) in initial testing.

Using leds (light-emitting diodes), create a vector of control signals as follows. See Appendix A of the Architecture I manual for led wiring information.

PC<sub>in</sub>  
PC<sub>out</sub>  
PC<sub>increment</sub>  
SIZE<sub>select</sub>  
Z<sub>in</sub>  
Z<sub>out</sub>  
MAR<sub>in</sub>  
MDR<sub>in</sub>  
MDR<sub>out</sub>  
Add  
A<sub>in</sub>  
IR<sub>in</sub>  
A3<sub>in</sub>  
A3<sub>out</sub>  
A5<sub>out</sub>  
D6<sub>in</sub>  
D5<sub>out</sub>  
read  
write  
WMFC  
MFC  
run  
end

Your group is to meet and determine its division of labor, i.e., who will be doing what. Your decision, the assembled instruction, the control sequences of microprocessor operations, and a parts list it due no later than two school days after assignment of this project.

You will prepare your circuit for verification and when you demonstrate your design, your team will submit:

1. a **neatly** detailed and documented circuit diagram.
2. individual evaluations of the lab.

## Lab 13.0 – Distributed Bus Arbitration

The goal of this lab is to design that portion of an I/O interface that manages bus mastership using *distributed arbitration*. Each group is to design, construct and test a circuit as described in the text. Use the description of circuit activities given in the text to assist you in designing your circuit. The scope of the project is reduced in order to make the project practical. The reduction is to eliminate ARB3. You can use either the 7409 or 7436 chips as they have open collector outputs as required in implementing the arbitration process. Data sheets were distributed in class.

Use data switches to implement ARB0, ARB1 and ARB2. This is done in order to allow you to change the 3-bit address for your device address. Changing device addresses facilitates testing.

When you have your arbitration circuit working, find two other willing groups to visit you on a bus which has ARB0, ARB1, ARB2, and START-ARBITRATION. Once you and your willing bus partners have the bus working, it will be checked out.

## Lab 14.0 - Disk Formatting

Construct a program which will format a 5-1/4" diskette such that it contains 9 sectors and 512 bytes per sector. Use the track structure handed out in class as a guide when you create a track in memory for formatting a diskette. Use the Floppy Disk Functions to guide you in writing instructions to execute your format procedure.

When you have formatted your diskette, store the names of all of the members of your group on the disk. Do this on track 13, sector 6, and side 1. Also, turn in a documented listing of the program which you used to format the diskette and the program segment used to store the names. Also, a copy of the track created for this assignment. Finally, your report.

## Disk Operations

The MC68000 Educational Computer Boards are really neat for studying computer architecture. However, there are some disadvantages. In order to keep the boards reasonably priced, there are no frills. In fact, when you turn the power off, all of the information stored in registers and memory vanishes. This is known as volatile memory. In order to make life slightly easier when composing programs on the Mc68000, it will behoove you to store your programs on a diskette. An alternative is to retype your program each session on the computer. Another alternative is to stay with the computer until you have completed all work.

### Storing a sector on a formatted diskette:

A disk address is a 3-tuple (track, side, sector) which specifies the physical location where information is stored on the disk. A sector is 512 bytes. More details will follow. There are three steps in saving a sector on a formatted diskette using our system. You must **select** the drive, **write sector** the information and then **unselect** the drive. When storing more than one sector, the **write sector** portion has to be executed enough times to complete the task. Each time it is executed, appropriate addressing values need to be supplied.

#### **select**

The following program segment is used to select the disk drive. You will probably only need to do this once each session unless you **unselect** at sometime during a session.

```

MOVE.B #00,D0      .specifies drive 0 for double density
MOVE.B #223,D7     .specifies the select function
TRAP   #14         .call the function

```

#### **write sector**

The following program segment is used to write a sector to the selected disk.

```

LEA    buffer,A5   .specify the beginning address of the
                    .sector to be written
MOVE.L #$02009486,D0 .see note on disk address
MOVE.B #218,D7     .specify the write sector function
TRAP   #14         .call the function

```

#### **unselect**

The following program segment is used to **unselect** the disk drive. This segment needn't be run until you are finished and are ready to terminate your session with the **MC68000**. If you were to **unselect** the disk drive and then wanted to read or write after that time, you would need to **select** that disk drive again.

```

MOVE.B #215,D7     .specify the unselect function
TRAP   #14         .call the function

```

**Retrieving a sector from a diskette:**

The process of retrieving a sector from a diskette is the same as storing except that you do a **read sector** operation in place of the **write sector** operation.

**read sector**

The following program segment is used to read a sector from the diskette.

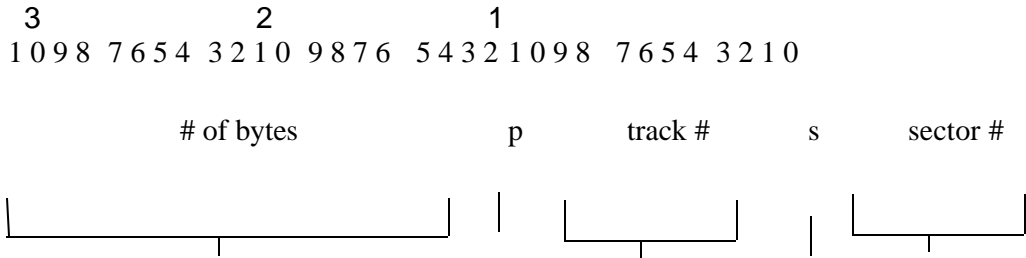
LEA	<i>buffer</i> ,A5	.specify the address in memory where .you want the sector to appear
MOVE.L	#\$02009486,D0	.see note on disk address
MOVE.B	#221,D7	.specify the read sector function
TRAP	#14	.call the function

**Binary/decimal/hexadecimal**

<u>binary</u>	<u>decimal</u>	<u>hexadecimal</u>
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

**Information in D0 for *read/write sector***

When reading or writing a sector from or to the disk, the 32-bit contents of data register D0 are used to indicate to the disk controller which sector on the disk is desired. The format for D0 is:



where p indicates precompensation and s indicates which side. the values for all fields in the format are in binary.

0, enabled for tracks 24-40  
 precompensation = {  
 1, disabled for tracks 0-23

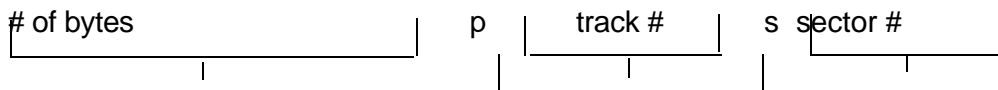
0, side 0  
 side = {  
 1, side 1

The instruction

```
MOVE.L #$02009486,D0
```

sets D0 to

```
0000 00100000 000 1 0010100 1 0000110
```



Thus, the address is sector #6 on track #20 of side #1. The value of p is 1 and thus, precompensation is disabled. There are 512 bytes.

**Warning:**

Since the MC68000 is on a tight budget, there isn't a directory or other software to keep track of the location and name of your information for you! Thus, that task is strictly yours. If you forget where you stored your program, too bad! You will now have to search the entire disk for it or do it over.