I. Learner Objectives:

At the conclusion of this lab, participants should be able to:

- Compose a simple machine language simulator
- Analyze the fetch/decode/execute cycle

II. Prerequisites:

Before reading this lab, participants should be able to:

- Summarize the basic MIPS instruction formats
- Implement a basic MIPS program

III. Keywords:

Machine language, fetch/decode/execute cycle, load/store architecture

IV. Pre-lab:

Write a ROBO-MAL program to trace out a square shape and stop. Please see below in the Background section for more information about ROBO-MAL. You may work on this in lab today!

V. Background:

Designing a simple *machine language* and simulator is a great way to comprehend the *fetch/decode/execute cycle* and *load/store architecture*. Many ideas exist for designing and simulating a simple machine language. One of my favorites is the Simpletron computer simulator created, as an academic exercise, by Deitel and Deitel. This Simpletron computer simulator executes Simpletron Machine Language (SML). Read, write, load, store, add, subtract, division, multiply, branch, branchneg, branchzero, and halt instructions exist in SML. For this lab you will create a similar language that may apply to robotics applications. We will create a primitive machine language, for robotic applications, called ROBO-MAL (ROBOtics MAchine Language).

You will design ROBO-MAL to define data transfer instructions READ, WRITE, LOAD, and STORE. You will need to define arithmetic operations ADD, SUBTRACT, and MULTIPLY. Branch instructions BRANCH, BRANCHEQ, BRANCHNE, and HALT should also be included. Lastly, robot control instructions (RCI) LEFT, RIGHT, FORWARD, BACKWARD, and BRAKE need to be defined.

The instruction operations codes and meanings are listed below in Table 1. All values shown are in decimal. **NOTE: In the future you may want to add to this instruction set so be sure to implement your simulator with as many independent subroutines as you see fit!**

Operation Code		**Description**
Data Transfer Instructions:		
READ	10	Reads PORTE 7:0 and stores it into a specific data memory cell.
WRITE	11	Writes to PORTE 7:0 from a specific data memory cell.
LOAD	12	Loads a word from a specific data memory cell into s0
STORE	13	Stores a word from s0 into a specific data memory cell.
Arithmetic Instructions:		
ADD	20	Adds a word from a cell in data memory to s0. The result is stored in s0.
SUBTRACT	21	Subtracts a word from a cell in data memory from s0. The result is stored in s0.
MULTIPLY	22	Multiplies the word in s0 by a word in a specific data memory cell. The result is stored in s5:s0.
Branch Instructions:		
BRANCH	30	Branch to a specific address in data memory.
BRANCHEQ	31	Branch to a specific address in data memory if s0 is zero.
BRANCHNE	32	Branch to a specific address in data memory if s0 is not zero.
HALT	33	End of the program, robot stops.
Robot Control Instructions (RCI):		
LEFT	40	Turn the robot left some specified number of degrees between [0:99].
RIGHT	41	Turn the robot right some specified number of degrees between [0:99].
FORWARD	42	Move the robot forward at slow (00), medium (01), or fast speed (10).
BACKWARD	43	Move the robot backward at slow (00), medium (01), or fast speed (10).
BRAKE	44	Slow the robot down for some number of seconds between [0:99].

Table 1: ROBO-MAL Instructions

An example program written in ROBO-MAL is below. Notice that each instruction is 32-bits wide (opcode = 16 bits, operand = 16 bits) and the data values are only 32-bits wide.

Memory Cell	Instruction or Data Value	Description
00	1011	READ num2
01	1211	LOAD num2
02	2110	SUBTRACT num1
03	3106	BRANCHEQ to cell 06
04	4190	RIGHT 90 degrees
05	3007	BRANCH to cell 07
06	4090	LEFT 90 degrees
07	4201	FORWARD at medium speed
08	4403	BRAKE for 3 seconds
09	3300	HALT program, robot stops
10	80	Variable num1
11	00	Variable num2

Table 2: Sample ROBO-MAL Program

The program in Table 2 reads the status of PORTE 7:0 and loads the value into s0. If PE7:0 is 80_{10} then the program will cause the robot to turn left at 90 degrees. If the status of PE7:0 is not 80_{10} then the robot will turn right at 90 degrees. Once the robot has turned, it will proceed forward at medium speed. It will then BRAKE for 3 seconds. Once the program is finished executing the robot stops.

VI. Lab:

This lab will require that you write the simulator for running ROBO-MAL programs. You will need to plug your LED module into the JJ-01:04 connector before you start the lab. Before you write the simulator part of the program, you will need to load your program into data memory. Thus, the first part of the MIPS assembly program is allocated just for loading ROBO-MAL programs into data memory. After you have written the code for loading ROBO-MAL programs into memory, you may start to write the simulator.

You will not be able to write the simulator to execute all instructions until you learn about timers and interrupts and have access to a robot. Thus for this lab you will only be able to execute some of the instructions completely.

In order to complete the simulator you will need to define the accumulator register (s0), instruction counter (s1), instruction register (s2), operation code (s3), and operand (s4). Remember the accumulator register is the implicit register used with many of the ROBO-MAL instructions. The instruction counter keeps track of the address of the next instruction to be executed. The instruction counter is generally incremented by one to get to the next instruction. The instruction register contains the current 32-bit instruction to be executed. The operation code register contains the 16-bit opcode part of the instruction and the operand register contains the 16-bit operand part of the current instruction.

The simulator follows the fetch/decode/execute cycle. A ROBO-MAL instruction is fetched from data memory and placed into the instruction register. The instruction is decoded by extracting the opcode and operand from it. The opcode is then executed. The instruction counter may be affected by the operand of the instruction. The cycle is repeated.

To show that LEFT, RIGHT, FORWARD, and BACKWARD instructions are simulated correctly, since you do not currently have a robot or understand timers/interrupts, turn peripherals LD3 on, LD0 on, LDs 1 and 2 to blink, and LDs 1 and 2 on, respectively, when the corresponding instruction is executed. All of the other instructions may be shown to work through the AVR Studio debugger.

After you have finished the simulator, load multiple ROBO-MAL programs into memory to be simulated. Make sure that all instructions simulate as expected.

Lastly, but most important, have fun and learn!!!

VII. Questions:

1. Explain how the operation code of each instruction indicates the category of instruction to which it belongs.
2. Describe how your design would change if instruction were 16-bits instead of 32-bits.
3. Imagine that you where going to control a robotic car. Expand on the ROBO-MAL language and design five other instructions that would be helpful in controlling the robotic car. These do NOT have to be robot control instructions; they may be data transfer, arithmetic, or branch instructions. Also, provide the operation code for the instructions. Feel free to be creative!!!