Laboratory 1
Introduction to MX4ck Trainer System & Simple Programming

Due Date: February 3, 2015
Points: 20 Points
Work individually

Objective: When you have finished the work associated with this lab, you will know how to utilize the MX4ck PIC32MX460F512L processor board, examine memory locations, register values, and system status, as well as enter sets of instructions using the assembler and loader, and make use of the built-in breakpoint facility.

Activities: (Note: The term 'monitor' in this context refers to a program that runs on the host PC system that provides control for the user activities, such as creating a program in memory or running that program. During the semester we will work in various ways with different aspects of the monitor system.)

Use an editor and the assembler to enter and create executables the following sets of statements. Each set of statements illustrates how to perform some common task associated with a higher-level language like C or C++.

1) Assignment Statements – In general, values in a program are stored in the memory system, and when work is needed, these values are loaded into the registers and used in appropriate ways. When a new value has been calculated for a variable, the value is stored – moved from the register where the calculated value is located to the assigned location in memory. In a RISC architecture, all work is done in the registers, and the memory is used as storage only. Create a text file (use the "<file>.s" convention) that contains eight store statements, plus a couple of nop's and a forever loop. These statements take the form of **`<ins> rt,offset(base),`** where **`<ins>`** is an appropriate instruction. For this laboratory use **`sb`** to store bytes, **`sh`** to store halfwords, and **`sw`** to store words. **`rt`** is to identify the source register, where the data is coming from. The **`offset`** is a displacement, and the **`base`** is the base address. For this laboratory, select a memory location that is appropriate, someplace in the RAM. The steps, then, in this first experiment are:
   a. Set up a register to point at the base address of the data area that you have selected. This can be done in your file of instructions with a '**`lui`**' instruction followed by an '**`ori`**' instruction, for addresses larger than 64K, or simply a **`'li'`** instruction (directive). Use the monitor to clear this area to begin.
   b. Fill eight other registers of your choice with values (using monitor) that utilize the values 0x11FFEEDD, 0x22EEDDCC, 0x33DDCCBB, and continue this pattern for 8 values in 8 different registers.
   c. Use store statements (again, in your file of instructions) to place these values in contiguous locations in memory. Note that the target location will be identified by a combination of the pointer register (base) and the displacement value (offset).
   d. Use assembler to create an executable file. (Assembly process – create .elf file)
   e. Download the file, set a breakpoint, clean out the target memory locations, and execute your program.
   f. Report on your results.

   Perform the above process three times, once for word (**`sw`**), once for halfword (**`sh`**), and once for byte (**`sb`**).

2) Once part (1) has been completed, repeat the process, but moving information from memory to registers, with word, halfword, and byte instructions. This time the monitor will be used to place values in memory, and the program will use load instructions to move the data to registers. Hence, the values stored in memory will be the 0x11FFEEDD, 0x22EEDDCC, etc., patterns. Use **`.long`** directives to set up the memory area.

3) Work statements – The work in a RISC system is done in the registers. Create a program that will calculate the addition, subtraction, logical AND, and logical NAND of the values 0xEF6587CD and 0x1234AE57. That is, the program will point at locations containing these values, move the values to registers of your choice (this will take two registers), do the four operations (four more registers) and store the values in memory. Then, use the monitor to run the program and report on the results.

4) Simple loop – Looping mechanisms are needed for the solution to many problems. The MIPS architecture uses a conditional branch mechanism, with a variety of tests, to implement the loop. Use a looping construct to fill 100 locations in data memory with ascending values, starting at 0x1101. Make the size change from one value to the next be 7.

5) Create a program that will perform the vector dot-product on two vectors and send the final value to a specified location in memory. A “vector” in this case is a one dimensional array, and hence is identified by a starting location in memory and a length. Let the two vectors be identified as VECA and VECB. Choose a starting address (in data memory) for VECA and fill memory with values from 54 to 1081 in steps of 13 (80 elements in all). Choose a starting address (in data memory) for VECB and fill memory with values from 157 to 1579 in steps of 18 (80 elements in all). Note that one easy way to create the vectors used as input is to use an editor and the .long directive to include the values in the program. Next, create a program to do the vector dot product: $\sum_{i=0}^{79} VECA_i \times VECB_i$ Store the result of the dot-product in another data memory location of your choosing.