Laboratory 2
Working with ARM Assembly Instructions

Introduction

The objective of this laboratory is to provide you with more experience using the ARM assembly language instruction set and to further examine the operation of the ARM processor as it executes those instructions.

Procedure

To begin this laboratory, you will once again be staring with an example program and exploring its operation before writing your own programs.

As you did in Laboratory 1, create a new ARM assembly language project, but this time select the Getting Started sample program (not the Simple Program we used last time). Getting Started will again use the LEDs and switches you encountered last laboratory, but it will also use the four pushbuttons.

Examine the program in the getting_started.s code file to understand what the program does. Step through the code and observe the changes that occur with the registers and CPSR.

You can add breakpoints to the program by left-clicking in the grey region beside the memory address of an instruction. For example, clicking to the left of the address `0x00000024` will add a breakpoint at the `MOV r4,r5` instruction. This will be indicated by the red stop sign which appears. Run the program and observe what happens when a pushbutton is pressed. Step through the code from that point to see how the `ROR` instruction functions.

Modifying the Program

1. The supplied program will load a new pattern if any of the pushbuttons are pressed. You are to modify the program so that each pushbutton has its own function, as described below. For the sake of the following discussion, we shall label the rightmost button as button 0 (labelled Key 0 on the circuit board) and the leftmost button as button 3 (Key 3).

   Modify the program so that the pattern no longer rotates automatically. Instead, the pattern should rotate to the right only when button 0 is pressed, and stop when it is released. Similarly, the pattern should rotate to the left only when button 1 is pressed, and stop when it is released. Pressing button 2 should set the pattern from the switches (as it did initially), while pressing button 3 should reload the original startup pattern stored in memory.

2. Using the getting_started.s and/or simple_program.s as reference, you are to write your own program to implement a pattern matching system which operates as follows.

   (SW values vs [mem add.])

   The system is to compare the 10-bit value set on the switches to a pattern stored in memory. If the two values match, then this should be indicated in some manner using the LEDs. (You can decide how the LEDs indicate the match.) The system should not be comparing continuously, but should only perform the comparison/check when pushbutton 0 is pressed. The LEDs should not indicate a match before pushbutton 0 is pressed to check for a match. If the switch value does not match the pattern stored in memory, nothing should happen.

3. Modify the program in #2 above to permit the user to change the stored pattern once they have correctly matched the current pattern correctly. The user should use one of the pushbuttons to indicate that a new pattern is to be captured from the switches and used as the new target pattern. This pattern should be stored back into memory to replace the value originally set in the code. Note that the main monitor tool window has a Memory tab at its bottom, which will allow you to view the contents of the system memory. Use this to verify that the new pattern is stored correctly. loose mark if not store in mem