Lab 4: Introduction to MIPS Assembly

EE 234: Section 2

Michael Mentele

Partner: Jonathan Quissenberry

October 1, 2012

*Abstract*

This paper describes a simple assembly program that will be used to implement a calculator with four operands and four operators. We will be using the peripheral switches 1 & 2 to select our operator switches 3 & 4 to select our operands. The peripheral LEDs are used to display our results and provide feedback to the user.

**Introduction**

This lab is an introduction to MIPS assembly. Assembly uses a different project type than the previous labs. After creating the project using the proper tool suites you must save the code file as a “.s” file type.

We will be performing operations we have done before in C but now in assembly. We will need to implement branches for loops and if statements. We will use the assembly instruction LW (Load Word) to read from port and SW (Store Word) to write to different ports. To determine whether pins on different ports are read and write we use the TRIS register where 1 is an input and 0 is an output. This is review but the operations are slightly different when using assembly.

For this lab we will be using both the switch and LED peripherals to construct a simple calculator. We will be using four different values for the four combinatory states of SW3 & 4. Those values are: 4, 5, 6, and 7. The tables below show how the operands are selected.

|  |  |  |
| --- | --- | --- |
| SW 3 | SW 4 | Operand |
| 0 | 0 | 4 |
| 0 | 1 | 5 |
| 1 | 0 | 6 |
| 1 | 1 | 7 |

**Table 1: Operand Selection**

|  |  |  |
| --- | --- | --- |
| SW 1 | SW 2 | Operation |
| 0 | 0 | Add |
| 0 | 1 | Subtract |
| 1 | 0 | Shift Left by 1 |
| 1 | 1 | Shift Right by 1 |

**Table 2: Operation Selection**

**Hardware**

Simply connect the USB cable to your PC and to the debug port on the Cerebot board. Plug in the switch module to the JK-01:04 connector of the Cerebot board plug the LED module into the JJ-01:04 connector.

**Software**

**Design**

When we enter the code it begins by specifying several programs in data memory. For this lab I created five programs. The first was from the pre-lab and the other four simply ran through all the commands from each section. So for data it would go through READ, WRITE, LOAD, and STORE respectively ending with a HALT command. All programs end with a HALT command.

Variables are placed at the end of the programs because our simulator would attempt to decode a variable when it came across it in “program memory.”

The memory is set up to use relative locations. Therefore you need to specify the number of cells from the command in question to the required.

Data Transfer Instructions:

For these instructions your always referring to some data memory cell. Much of the code here is recycled for each subroutine.

Arithmetic Instructions:

Branch Instructions:

Robot Control Instructions:

**Test Procedure and Results**

**Methodology**

Testing was performed from a “Top Down” viewpoint where functionality is tested first followed by a debugging phase before finally being tested by an independent user. We also made great use CPU registers display and the disassembly listing to test our design and ensure operations were being correctly performed. Doing this in conjunction with the step through this greatly reduced the time it took to debug the program before getting a testable prototype.

**Procedure**

1. Test Correct Operation
   1. Test each possible case
      1. Does it add?
      2. Does it subtract?
      3. Does it shift left and right?
      4. Do these operations work with all operands?
2. Attempt to Break Current Build
   1. Adding
      1. It isn’t possible to have overflow here due to the operands that we seleceted. So we simply tested all combinations of number.
   2. Subtracting
      1. Check negative numbers.
   3. Shifting
      1. Different shift cases.
3. Let someone else play it!

Using Murphy ’s Law… and another perspective this is an effective way to achieve unexpected results.

**Results**

When starting the board it waited with the LEDs off for two seconds and then it reads the states of the switches. With neither switch pressed it added and so required two inputs from switches 3 & 4 as operands. So if you never switch a switch it will automatically add four to four.

All the add and shift operations worked correctly with no strange “corner” cases. However when its we don’t care so the result was ignored. Interestingly it seemed to display two complement at least for the case of 4 – 7 = -3. However I am unsure whether this is coincidental or not. Solutions and expansions for negative numbers are covered in the conclusion.

**Answers to Questions**

1. You can use the stack to preserve variables or registers. A typical application would be passing values between functions. The stack order is best used for applications where you know the order that variables need to be accessed.
2. Beginning of SFR is BF80\_0000

TRISB => BF88\_6040 The offset is 8\_6040

PORTB => BF88\_6050 The offset is 8\_6050

LATB =>BF88\_6060 The offset is 8\_6060

1. It is necessary to write in assembly at times when you want precise control of how your operation is executed within the controller. This is likely due to space or speed constraints where timing or resource allocation is especially critical.

**Conclusion**

Assembly is useful when you need complete control of your system and is especially important for time critical applications. A disadvantage of assembly is that it is much easier to make mistakes such as corrupting memory especially registers.

The code could be improved to be more robust and modular by using the stack pointer and variables instead of storing everything into registers. That way if you were to use this code with some other functions you don’t have to worry about whether registers are being preserved or not.

We could add functionality to display negative numbers. The case simply wasn’t required as part of this lab but that would be the next logical step. Also being able to ‘set’ your operands would be a nice feature. You could, for example, use one of the buttons to increment or decrement your operands so as to set them to arbitrary values. You would most likely then need to use more LEDs than the four peripherals.