

Lab 5: Hardware Interrupts

Joseph Salazar
College of Engineering
University of Miami
Miami, United States
jes409@miami.edu

Abstract— The ARM assembly language is widely used by professionals in the computer engineering field. However, there is very little material that introduces entry-level persons into assembly language. To help individuals better understand assembly for their futures in engineering, we propose following a quick and efficient program provided by the University of Miami's College of Engineering. This program allows individuals to practice assembly by writing a short sequence of code that affects a DE1 board's LED's.

Keywords—ARM, assembly,

I. INTRODUCTION

The purpose of this lab is to interact and familiarize oneself with the DE1's I/O devices. By becoming familiar with the I/O devices, one can utilize their board more effectively and in many more ways. These are both important reasons to become familiar with the I/O as it will benefit the programmer's future in the field of Computer Engineering. To demonstrate the importance of this, a programmer will be asked to write a simple code in ARM assembly that contains four different subroutines that will perform operations on two integers gathered from the switches. The operation is dependent on a 2-bit value that is input. The result will be represented on the LED display, and if the result is zero, all the LED's will turn on. This exercise should force the programmer to learn how the I/O works and it can be utilized in many ways.

II. METHOD/EXPERIENT

A. Overview

Before attempting this lab, one should have Altera Monitor Program installed before continuing. Then, power on the DE1 board and connect it to the computer. Once connected, one should proceed to set-up their coding environment/document so that the code remains organized. After one's ".s" file is created, the programmer should include the data, global, text, start, and end directives to ensure compilation goes smoothly after code has been added. If one is not familiar with this routine, referencing the DE1 manual would be most beneficial [2].

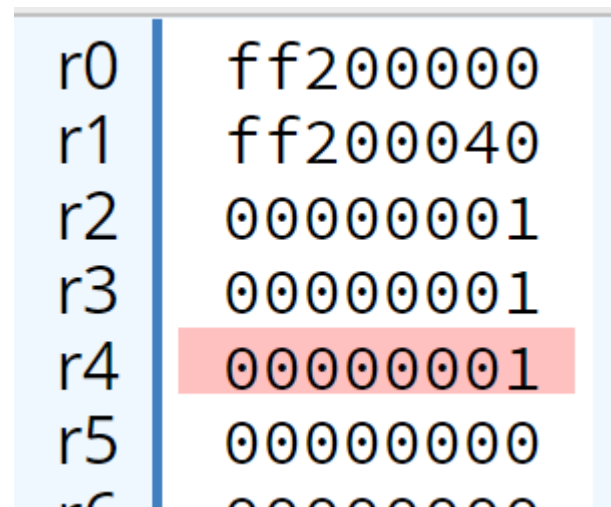
Before attempting the code, the programmer should observe where the LED's and switches are located in memory. In order to utilize them, one will have to initialize them and create a pointer to effectively read and write values. Once this is understood, the programmer can begin writing their program.

To begin this lab, one is asked to write a short program that reads two integers from the board's switches, and then either adds, "AND's," multiplies, or "XOR's" the two. This part of the code should take three inputs from the switches. The first two

are the integers previously discussed, and the third one should be a 2-bit selecting input. The two-bit values should be assigned to the four subroutines respectively to utilize a subroutine. If an invalid selection input is given, all the LED's should turn on. To turn them on, the programmer should observe how many LED's there are, and if a value exists that would activate all of them. This can be observed in Figure 4.

Lastly, the programmer is asked to add another condition where all the LED's turn on if the result is zero. This is like the previous step, and it should reiterate the importance of knowing how to code effectively and efficiently in assembly.

B. Results



r0	ff200000
r1	ff200040
r2	00000001
r3	00000001
r4	00000001
r5	00000000
r6	00000000

Figure 1. The following figure shows r2 is the first integer, r3 is the second, and r4 is the condition for "AND."



Figure 2. The following figure shows the result from the previous figure.

r0	ff200000
r1	ff200040
r2	00000001
r3	00000001
r4	00000007
r5	00000000
r6	00000000

Figure 3. The following figure shows r2 and r3 as integers and r4 as an invalid conditional input.



Figure 4. The following figure shows all of the LED's activated as a result of an invalid conditional input.

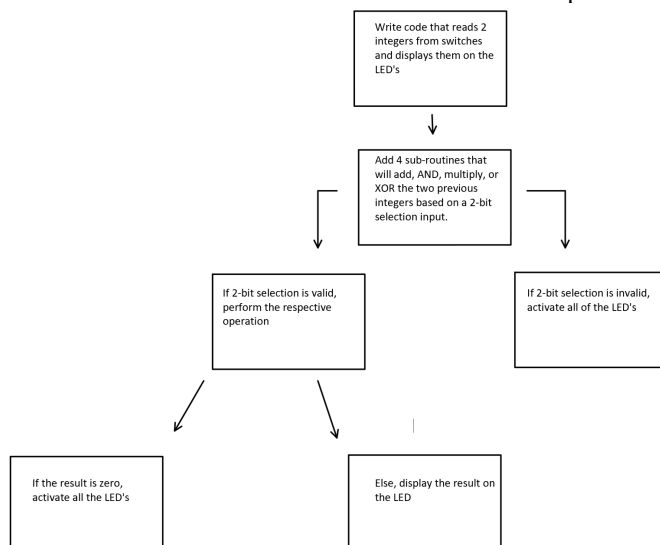


Figure 5. The following is a flow-chart for the steps taken during this exercise.

C. Discussion

After completing the previous tasks, one could reflect and take many lessons away. These exercises emphasize the importance of knowing how to correctly utilize a DE1's switches and LED's to create a visual coding experience. This could be very important to know in case one is tasked with an assignment involving the switches and/or LED's. As a

computer engineer, one should be able to be well-rounded in all aspects of ARM assembly.

Ultimately, this exercise should not provide too much difficulty because it only includes writing a relatively simple program. The most difficult task is determining how to utilize the switches. One should take their time to ensure that they fully understand the memory addressing of the I/O so that their experience is as easy as possible. With this being said, the exercise is very well-written, but a programmer should try to build upon it on his own in order to completely understand the material. The exercise should not change, but extra steps, such as creating a BCD display, should be explored.

III. CONCLUSION

In sum, this exercise introduces instructions in ARM assembly that are essential for more difficult programs. After following the instructions, the programmer will have walked away with a deeper understanding of a DE1's I/O and how to utilize it. The results gathered from this exercise reinforce this idea by forcing the programmer to give input using the board's switches, then displaying results on the LED's. Ultimately, this will help the programmer excel as a computer engineer by learning new and important content in ARM and hand assembly.

REFERENCES

- [1] ARM Limited. ARM Architecture Reference Manual. 1996.
- [2] *Documentation-ArmDeveloper*, developer.arm.com/documentation/ddi0388/latest/.
- [3] "GNU Manuals Online - GNU Project - Free Software Foundation." [A GNU Head] , www.gnu.org/manual/manual.en.html.
- [4] Pyeatt, Larry D. *Modern Assembly Language Programming with the ARM Processor*. Newnes, 2016.

APPENDIX

*****Part One*****

.data

```
led:    .word    0xff200000
sw:     .word    0xff200040
push:   .word    0xff200050
```

.text

.global _start

_start:

```
    ldr r0,=led
    ldr r0, [r0]
```

```
    ldr r1,=sw
    ldr r1,[r1]
```

```
    ldr r2,[r1] //read from switches
    str r2,[r0] //store to led
    ldr r3,[r1] // read from switches
    str r3,[r0] //store to led
    b done
```

done:

```
    b done
```

*****Part Two*****

.data

```
led:    .word    0xff200000
sw:     .word    0xff200040
push:   .word    0xff200050
lights: .word    1023 //value that turns on all LED's
```

.text

.global _start

_start:

```
    ldr r0,=led
    ldr r0, [r0]
    ldr r1,=sw
    ldr r1,[r1]
    mov r9, #0 // add
    mov r10, #1 //and
    mov r11, #2 //mul
    mov r12, #3 //xor
    ldr r8,=lights
    ldr r8,[r8]
```

```
    ldr r2,[r1]// First value
    ldr r3,[r1]// Second value
    ldr r4,[r1] //Condition
```

```

comp:                //determine which operation was chosen
    cmp r4,r9
    beq add
    cmp r4,r10
    beq and
    cmp r4,r11
    beq mul
    cmp r4, r12
    beq xor
    b lightup

add:
    add r2, r2, r3
    cmp r2, #0
    beq lightup
    str r2,[r0]
    b done

and:
    AND r2, r2, r3
    cmp r2, #0
    beq lightup
    str r2,[r0]
    b done

mul:
    mul r2, r2, r3
    cmp r2, #0
    beq lightup
    str r2,[r0]
    b done

xor:
    EOR r2, r2, r3
    cmp r2, #0
    beq lightup
    str r2,[r0]
    b done

lightup:             //turn all led's on
    str r8,[r0]
    b done

done:
    b done

.end

```