ECE 212: Microprocessors, Hardware, Software, and Interfacing

Lab 4: Interfacing with Peripherals

Karysse Hay
College of Engineering
University of Miami
Coral Gables, United States
kdh85@miami.edu

***Abstract*— ARM assembly language is a low-level language that many professionals in the engineering field use. To better understand the specific components of ARM programming, such as data movement, arithmetic, code control, etc. The objective of this lab is to also learn how to interface with peripherals that are connected to the microprocessor on the DE1-SoC board.**

Keywords*— *assembly language, data movement, arithmetic, code control, peripherals, and microprocessor

I. Challenge

In previous labs, the memory of DE1-SoC board was mostly explored. This included what type of data it holds and how memory addresses work. However, in this lab, the online emulator was used and other peripheral components, namely the LED's, switches, and push buttons were studied [1]

II. Approach

The provided lab manual the arm cortex a-9 manual [1] was looked at beforehand. For part one, the '.equ' instruction was used to gain access to the LEDs and switches so that we can represent a value on them. For part two, multiple 'ldr' instructions were used to display multiple numbers using the switches [1]. Eventually, the 'cmp' instruction was used to do different operations between an integer X and two manually inputted values [1].

III. Experiment Setup

A. Overview

The online emulator was opened, and the ARMv7 DE1-Soc option was selected. Assembly code for each part of the lab will be inserted in the results section. When problems occurred, the program was debugged by checking the memory tab. It should be noted that '.equ' was used to equate the addresses of the switches and LEDs to "SWITCH" and "LED" respectively [1].

B. Part 1

A short program was written that read a single value for an integer, X, and that read the values of two switches. This value from the switches were then displayed on the LEDs using a 'ldr' instruction and 'str' instruction. This program was also read and written from the addresses of these components.

```
.global _start
_start:
.data

.equ LED, 0xff200000  @base address
.equ SWITCH, 0xff200040  @switches
.equ BUTTONS, 0xff200050  @push buttons


.text

ldr r0, =LED
//ldr r0, [r0]

ldr r1, =SWITCH
ldr r1, [r1]

str r1, [r0] @the value of the switch at the
address
```

Figure I: Assembly Code for Part I

C. Part II- 1-2

It was ensured that X could be a value up to 16 (decimal). X was read into r2. Two different values were read from the switches that was manually inputted. They were also stored in registers namely r3 and r4. Each time the values were inputted, they would be displayed on the LEDs in binary form. A breakpoint was added so that the program could have been paused. This can be seen in the Results and Analysis section.

D. Part II- 3-4

After it was positive that Part I – 1-2 was working, the program was modified. Values such as '00', '01', '10' and '11' were assigned to different operations [2]. These included adding, ANDing, multiplying and shifting left. This part made it possible to input one value on the switches, insert an operation and read it, insert the second number and perform the operation. The result was displayed on the LEDs and if the result was zero, all LEDs were lit up to represent zero in 2s complement. This can be seen in the Results and Analysis section.

```
.global _start
_start:
.data

.equ LED, 0xff200000  @base address
.equ SWITCH, 0xff200040  @switches
.equ BUTTONS, 0xff200050  @push buttons

x: .byte 5
.text

ldr r0, =LED
//ldr r0, [r0]

ldr r1, =SWITCH

ldr r2, =x  @address of x
ldr r8, [r2]  @value of x

first: ldr r3, [r1] @loads switch into r3
str r3, [r0] @store r3 into r0

second: ldr r4, [r1] @breakpoint to get second
number
str r4, [r0] @changes number

operation: ldr r5, [r1] @breakpoint to get thrird
number
str r5, [r0] @displays op
add: cmp r5, #0x0
addeq r6, r3, r4 @adds first and second
bne and
add r8, r6
and: cmp r5, #0x1
andeq r8, r3, r4 @ands the two numbers
multiply: cmp r5, #0x2
 muleq r8, r3, r4

shift: cmp r5, #0x3
  lsleq r8, r3
 str r8, [r2]

str r8, [r0] @displays new number
```

Figure 2: Assembly Code for Part II

E. *Part II- 5*

It was found to be that X was located at address 0x00000080. Using the addresses given for the switches and LEDs which were 0xFF200040 and 0xFF200000 respectively, it was calculates that X was 0xFF1FFFC0 away from the switches and 0xFF1FFF80 away from the LEDs. This can be seen in the Results and Analysis section.

IV. Results and Analysis

A. *Part I Results*

a) Results

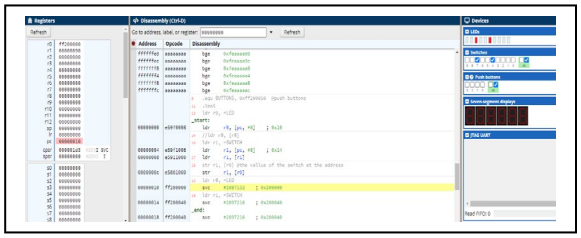


Figure 3: Part I Results on the Emulator

b) Comments

It can be seen that the number selected on the switches is 0010010000 and the LEDs reflected that.

B. *Part II Results (a)*

a) Results

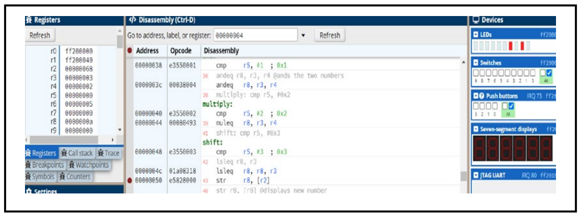


Figure 4: Part II(a) Results on the Emulator

b) Comments

It can be seen that the result on the LEDs is 0000001010 in binary. This was because the first two numbers inputted was 0000000010 and 0000000011 which is 2 and 3 in decimal. This can be seen in the registers 3 and 4. It should be noted that the operation selected was ‘00’ which is to perform addition. Therefore, the result correctly reflects that. The value of X can also be seen in r6 which is 0x5.

C. *Part II Results (b)*

a) Results

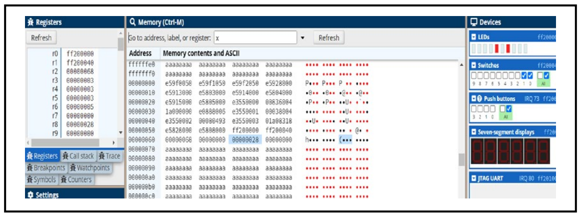


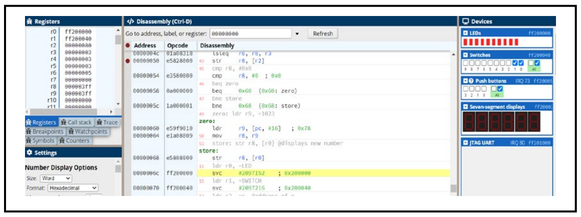
Figure 4: Part II(b) Results on the Emulator

b) Comments

It can be seen that the result on the LEDs is 0000101000 in binary. This is the logical shift left by 2^3- ‘lsl’ of the number 0x5. This happens to be 40 in decimal which is displayed on the LEDs in binary.

D. *Part II Resutls (c)*

a) Results



```
r0   ff200000
r1   ff200040
r2   00000080
r3   00000003
r4   00000003
r5   00000003
r6   00000005
r7   00000000
r8   000003ff
r9   000003ff
r10  00000000
```

Number Display Options
Size: Word
Format: Hexadecimal

Go to address, label, or register: 00000000

```
Address   Opcode    Disassembly
0000004c  01a08318      lsleq   r8, r8, r3
00000050  e5828000  43  str     r8, [r2]
                    45  cmp r8, #0x0
00000054  e3580000      cmp     r8, #0  ; 0x0
                    46  beq zero
00000058  0a000000      beq     0x60  (0x60: zero)
                    47  bne store
0000005c  1a000001      bne     0x68  (0x68: store)
                    49  zero: ldr r9, =1023
zero:
00000060  e59f9010      ldr     r9, [pc, #16]   ; 0x78
00000064  e1a08009  50  mov     r8, r9
                    52  store: str r8, [r0] @displays new number
store:
00000068  e5808000      str     r8, [r0]
                    12  ldr r0, =LED
0000006c  ff200000      svc     #2097152    ; 0x200000
                    15  ldr r1, =SWITCH
00000070  ff200040      svc     #2097216    ; 0x200040
```

Figure 4: Part II(c) Results on the Emulator

b) Comments

It can be seen that the result on the LEDs is 1111111111 in binary. This is because the operation performed resulted in 0 and as the lab instructed, a result in zero should be displayed as all LEDs lit.

V. Conclusion

This lab allowed for numbers to be manually inputted using the switches and to be displayed on the LEDs. Also, with pre-defined numbers assigned to different operations, we were able to manipulate these numbers. This lab gave us a better understanding of the switches and LEDs of the DE1-SoC board and how the ‘ldr’ and ‘str’ instructions can be used to facilitate the displaying of numbers [1]. Assemble code was better understood upon the completion of this lab.

References

[1] A. Holdings, “Cortex-a9 technical reference manual, revision r2p0,” ARM Holdings. Report number: DDI0388E, 2009.

[2] ARM Ltd and ARM Germany GmbH, “ Writing ARM Assembly Language,” Assembler User Guide: Writing ARM Assembly Language. [Online]. Available: https://www.keil.com/support/man/docs/armasm/armasm_dom1359731144635.htm. [Accessed: 20-Mar-2021].

[3] “ASCII,” Wikipedia, 19-Aug-2020. [Online]. Available: https://simple.wikipedia.org/wiki/ASCII#/media/File:ASCII-Tablewide.svg. [Accessed: 20-Mar-2021].

Appendix

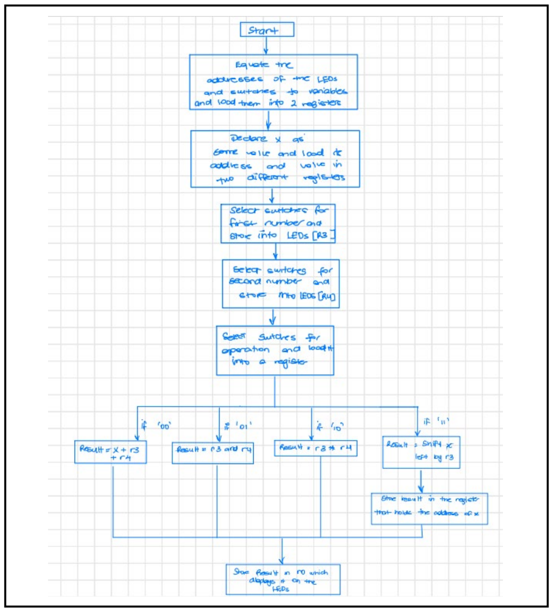

Start
Equate the addresses of the LEDs and switches to variables and load them into 2 registers
Declare x as some value and load its address and value in two different registers
Select switches for first number and store into LEDs [R3]
Select switches for second number and store into LEDs [R4]
Select switches for operation and load it into a register
if '00'
if '01'
if '10'
if '11'
Result = x + r3 + r4
Result = r3 and r4
Result = r3 * r4
Result = shift x left by r3
Store result in the register that holds the address of x
Store Result in r0 which displays it on the LEDs

Figure 5: Annotated Flow Chart of Part II Code