Dylan Thornburg
Sal Martinez
Tuesday 2:15-5:00
10/3/23

SANTA CLARA UNIVERSITY
Electrical and Computer Engineering Department

ELEN 120 – Embedded Computing Systems

Lab 1 – Introduction to ARM Assembly

Andrew Wolfe

Assignment: In this assignment, you will trace the execution of ARM assembly programs and write new assembly programs.

Learning Objectives: Using the Keil simulator. Using ARM's data processing instructions (arithmetic instructions, bitwise logical operations, and load/store instructions) and data transfer instructions (load and store).

Problem 1

Create a new project in Keil and enter the following code:

```
                AREA main, CODE, READONLY
                EXPORT __main
                ENTRY

__main PROC

            LDR r0, =0x15
            LDR r1, =label1
            LDR r2, [r1]
            ADD r3, r0, r2
            ADD r1, #4
            LDR r4, [r1]
            ADD r3, r4
endless B endless
            ENDP

            ALIGN
label1 DCD 0x06
P1 DCD 0x04

                END
```

Build the program in the Keil simulator and then debug it by stepping one instruction at a time. Answer the following questions using the *Disassembly* and *Registers* windows in the simulator:

1. What is the result in R0 after the following pseudoinstruction is executed?
   `LDR r0, =0x15`

Register 0 gets loaded with hex value 0x15

2. What is the address that corresponds to the label `label1`?

`pc; 0x1E0`

3. What is the value in r1 after the second ADD instruction is executed?

`0x1e4`

4. What is the address of the instruction: `ADD r1, #4` ?

`0x1d4 from the pc`

5. What is the value stored in the PC before the instruction: `ADD r1, #4` is executed?

`0x1d4 from the pc`

6. What is the final result in R3 after the program reaches the endless loop?

`0x1f`

7. Add meaningful comments to the program.

`Done`

Code for Problem 1

```
;******************** (C) Andrew Wolfe
*****************************************
; @file    mainproto.s
; @author  Andrew Wolfe
; @date    August 18, 2019
; @note
;          This code is for the book "Embedded Systems with ARM Cortex-M
;          Microcontrollers in Assembly Language and C, Yifeng Zhu,
```

```
;          ISBN-13: 978-0982692639, ISBN-10: 0982692633 as used at Santa Clara
University
;*************************************************************************
******


                AREA    main, CODE, READONLY
                EXPORT      __main
                ENTRY

__main PROC
            LDR         r0, =0x15 ;Loads hex value 15 into r0
            LDR         r1, =label1 ;Loads r1 with the address of label1
            LDR         r2, [r1] ;Loads r2 with r1's value
            ADD         r3, r0, r2 ;Adds r0 and r2 into r3
            ADD         r1, #4 ;Adds four to r1
            LDR         r4, [r1] ;Loads r1 value into r4
            ADD         r3, r4 ;adds r4 to r3 in r3
endless B               endless ;endless loop
            ENDP
            ALIGN
label1      DCD 0x06
P1          DCD 0x04
                        END

;*************************************************************************
*******


;
;*************************************************************************
*******



            ENDP

            ALIGN
```

```
;*******************************************************************
*******
; Put Your Data Here
;*******************************************************************
*******

    END
```

Problem 2

```
                AREA main, CODE, READONLY
                EXPORT __main
                ENTRY

__main PROC

            LDR r1, =value
            LDR r2, [r1]
            MVN r3, r2
            ADD r3, r3, #1
endless B endless

            ENDP

            ALIGN
value DCD 0xFFFFFFFF

                END
```

1. Describe what the program does in a single statement.

Value gets memory initialized with 0xffffffff inside which is then pointed to by r1 and then r2 gets loaded with that value and r3 gets one added to it since MVN r2 into r3 changed nothing.

2. What is the address of the memory loaded into r2 by the LDR instruction?

0x1d8 0x1d9 0x1da 0x1db

3. Can we replace the MVN instruction with NEG and get the same result? Explain.

We could not use neg because it negates a value but uses two's complement so the negation of 0xffffffff would be 0x1 and not 0x0.

4. Can we replace the MVN instruction with NOT and get the same result? Explain.

Not does not exist in our environment.

Code for Problem 2

```
;******************** (C) Andrew Wolfe
*****************************************
; @file    mainproto.s
; @author  Andrew Wolfe
; @date    August 18, 2019
; @note
;         This code is for the book "Embedded Systems with ARM Cortex-M
;         Microcontrollers in Assembly Language and C, Yifeng Zhu,
;         ISBN-13: 978-0982692639, ISBN-10: 0982692633 as used at Santa Clara
University
;*****************************************************************
***********
```

```
                AREA main, CODE, READONLY
                EXPORT          __main
                ENTRY
__main          PROC
                LDR r1, =value ;makes r1 a pointer to value
                LDR r2, [r1]; loads r2 with the memory intialized number of r1
                MVN r3, r2; moves not r2 into r3
                ADD r3, r3, #1; adds 1 to r3
endless     B   endless
                ENDP
                ALIGN
value           DCD 0xFFFFFFFF; allocates and initializes memory for value
                        END
```

Problem 3

In this problem, we will use the STR instruction to change data in memory.

```
                AREA main, CODE, READONLY
                EXPORT __main
                ENTRY

__main PROC

            LDR r1, =list
            LDR r2, [r1]
            LSL r2, r2, #1
            STR r2, [r1]
            ADD r1, r1, #4
            LDR r2, [r1]
            LSL r2, r2, #1
```

```
            STR r2, [r1]
            ADD r1, r1, #4
            LDR r2, [r1]
            MOV r2, r2, LSL #1
            STR r2, [r1]

endless B endless

            ENDP

            ALIGN
list DCD 3, 4, 5 ; three 32-bit numbers stored in memory END
```

Note: To write data into memory, you need to override the linker directives and make the section of memory that holds the data read/write instead of read-only. If you configure the debugger to use my wdefault.ini file as shown in class, this will be taken care of.

Trace the program as usual in the debugger to answer the following questions.

1. What are the addresses of the stored values 3, 4, and 5 in `list`?

`0x1ec 0x1f0 0x1f4`

2. What is loaded into r2 after the first LDR instruction is executed?

`Nothing was loaded into r2 after the first LDR instruction. After the 2nd LDR instruction, 0x3 was loaded into it.`

3. What does the following instruction do?

```
LSL r2, r2, #1
```

`It shifts the binary value of r2 to the left by one (aka multiplies by 2)`

4. Check the contents of the word at memory location 0x1F0 after the program is executed. What is the new value stored in memory at this location?

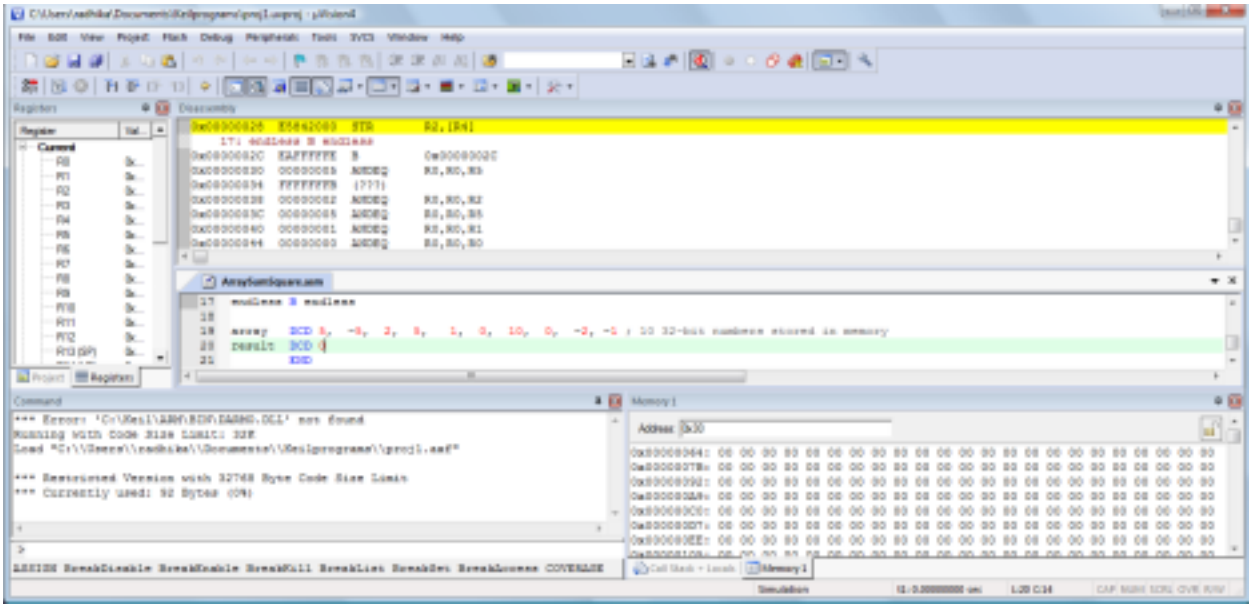
`The new value is 0x8`

5. Do the following instructions produce a different result for the same initial value in register r2?

• MOV r2, r2, LSL #1
• LSL r2, r2, #1

No they do not.

Note: Use **View->Memory Windows->Memory1** to see the contents of memory. You can specify the desired address in the *Address* textfield.



5

Code for Problem 3

```
;******************** (C) Andrew Wolfe ****************************************

; @file    mainproto.s

; @author  Andrew Wolfe
```

```
; @date    August 18, 2019
; @note
;        This code is for the book "Embedded Systems with ARM Cortex-M
;        Microcontrollers in Assembly Language and C, Yifeng Zhu,
;        ISBN-13: 978-0982692639, ISBN-10: 0982692633 as used at Santa Clara University
;*******************************************************************************


                AREA main, CODE, READONLY
                EXPORT __main
                ENTRY
__main          PROC
                LDR r1, =list
                LDR r2, [r1]
                LSL r2, r2, #1
                STR r2, [r1]
                ADD r1, r1, #4
                LDR r2, [r1]
                LSL r2, r2, #1
                STR r2, [r1]
```

```
                    ADD r1, r1, #4

                    LDR r2, [r1]

                    MOV r2, r2, LSL #1

                    STR r2, [r1]

endless     B       endless

                    ENDP

                    ALIGN

list                DCD 3, 4, 5 ; three 32-bit numbers stored in memory

        END
```

Problem 4

Write a program to calculate the sum of the squares of two numbers stored in memory and then store the result back in memory. For example, suppose that the numbers are defined by the labels *num1* and *num2* in the data section of your program as follows:

num1 DCD 0x05

num2 DCD 0x03

result DCD 0x22

The result should be stored back in memory after num2 as shown.

Submission: Prepare a lab report due at the beginning of your next lab. Submit the solution to all the problems including the source code of the programs written by you. Demonstrate your programs for problem 4 to the teaching assistant.

Code for Problem 4

```
;******************** (C) Andrew Wolfe
*****************************************

; @file    mainproto.s

; @author  Andrew Wolfe

; @date    August 18, 2019

; @note

;          This code is for the book "Embedded Systems with ARM Cortex-M

;          Microcontrollers in Assembly Language and C, Yifeng Zhu,

;          ISBN-13: 978-0982692639, ISBN-10: 0982692633 as used at Santa Clara University

;*****************************************************************************
**


                AREA main, CODE, READONLY

                EXPORT __main
```

```
                    ENTRY
__main              PROC
                    LDR r8, =result
                    LDR r1, =num1
                    LDR r2, [r1]
                    LDR r3, =num2
                    LDR r4, [r3]
                    MUL r5, r2, r2
                    MUL r6, r4, r4
                    ADD r7, r5, r6
                    STR r7, [r8]
endless     B       endless
                    ENDP
                    ALIGN
num1        DCD 0x05
num2        DCD 0x03
result      DCD 0x0
        END
```
