

Lab 3: Microprocessor System Design

Programming in ARM assembly language

Miriam Mnyuku

May 3, 2018

Abstract

This lab is an introduction to Keil MDK integrated development environment and programming in ARM assembly language.

Introduction

We used the Keil ARM simulator to create new projects, assemble and debug programs. We also designed an algorithm that counts the number of 1's in a 32-bit number.

Material Used

Keil MDK version 4.7

Procedure

Part I: Using the Keil ARM Simulator

We created a new project by watching a tutorial youtube video and downloaded the startup.s file found on canvas.

Part II: Writing your first Assembly Program

Given a code that adds two numbers, we copied the code to keil editor and debug the code and watched the changes in the registers.

1. Explain what these lines mean

AREA: instructs the assembler to assemble a new code

|.text|: Name of the area used for codes produced by C compiler

CODE: machine instructions

READONLY: what is stored cannot be changed

ALIGN = 2: adjust location counter to word boundary, in this case is 2²-byte boundary (16-bits)

THUMB: particular set of instruction (library)

2. What is the value of R0, R1, R2, and PC at the start and at the end of the program?

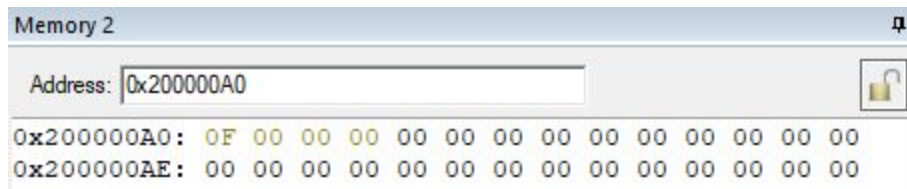
Start: R0 = 0x00000000, R1 = 0x00000000, R2 = 0x00000000, PC = 0x0000026C

End: R0 = 0x00000004, R1 = 0x00000005, R2 = 0x00000009, PC = 0x000002A4

3. Explain the S B S line of code

Means branch to the line labeled S and is used to create an infinite loop which ends the program.

4. Expand the program to solve $4+5+9-3$ and save the result in the 40th word in memory. Take a screenshot of the memory for your lab report.



Part II: Tracing an Assembly Program

Given a new set of code to debug and watch the changes in the registers.

1. What is the value in R0 after the program ends? R0 = 0x00000006

Registers	
Register	Value
Core	
R0	0x00000006
R1	0x00000003
R2	0x00000000

2. If the value initially placed in R0 is equal to 5, what is the value in R0 when the program ends? R0 = 0x00000078

Registers	
Register	Value
Core	
R0	0x00000078
R1	0x00000005
R2	0x00000000

3. What does this program do?

Calculates the factorial of a number, in this case 3! and 5!

4. If you replace the instructions at lines 2 and 10 in Figure 2 with PUSH {R0, R1} and POP {R1, R2} respectively, how will the program behave and why?

The factorial function is a non-leaf function and therefore it must save values in the preserved registers (R0,LR) in the PUSH instruction and restore it in POP because it will be modified as the code runs. Replacing with PUSH{R0,R1} and POP{R1,R2} means that LR is not saved. Saving R1 does not help because R1 is not used, R2 is restored but was not saved in the first place. The program is stuck in an endless loop and sets R0 = 0.

5. Repeat 4 with the following instructions modifications:

a. Replace the instructions at lines 2 and 10 in Figure 2 with PUSH {R3, LR} and POP {R3, LR} respectively.

R3 is not used so it does not have to be saved or restored. The program does not work because the value of preserved register R0 was not saved hence cannot be restored, causing R0 = 0.

b. Replace the instructions at lines 2 and 10 in Figure 2 with PUSH {LR} and POP {LR} respectively.

Stores/restores the value of LR(return address) on to the stack but does not save/restore R0 (return value) so the program returns R0 = 0

c. Delete the instruction at line 6.

Removing ADD SP,SP, #8 the stack pointer goes to the memory address rather than the immediate values. So, it jumps straight to the last step and the program has the same end result.

Part III: Coding in Assembly Language

We designed an algorithm for counting the number of 1's in a 32-bit number.

1. A pseudocode or a flow chart of your design steps

```
int number = -1; //can be any number
int mask = 1;
int onecounter = 0;
for (int i = 0; i < sizeof(int) * 8; i++)//goes to
32-bits
    if (mask & number)
        Onecounter++;
    mask <<= 1; //shift masking number to the left
once
```

2. Implement your algorithm using ARM assembly in the Keil IDE. Provide screenshots of the registers and/or memory locations used by the program. code:

```
1 AREA |.text|,CODE,READONLY,ALIGN=2
2 THUMB
3 EXPORT Start
4 Start
5     ; R0 = number
6     ; R1 = mask
7     ; R2 = counter of ones (Output of the program)
8     ; R3 = loop counter
9     ; R4 = AND result
10
11     ; Initialize registers.
12     MOV r0, #34
13     MOV r1, #1
14     MOV r2, #0
15     MOV r3, #0
16
17 LOOP    AND r4, r0, r1      ; r4 = r0 AND r1
18         CMP r4, #0         ; if r4 == 0...
19         BEQ INC            ; ...then jump to INC, otherwise continue to r2++
20         ADD r2, r2, #1      ; r2++
21 INC     ADD r3, r3, #1
22         LSL r1, r1, #1      ; r1 = r1 << 1
23         CMP r3, #32         ; 32 bits = sizeof(int) * 8 bits per byte
24         BLT LOOP           ; if( r3 < 32 ) goto LOOP; otherwise end program.
25     END
```

Initializing variable: R0 = 0x00000022 Or 34 (decimal)

The screenshot shows a debugger window with two panes. The left pane, titled 'Registers', displays the state of various registers. The right pane, titled 'Disassembly', shows the assembly code being executed.

Register	Value
R0	0x00000022
R1	0x00000001
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000400
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x00000294
xPSR	0x01000000

The disassembly view shows the following code:

```

0x00000290 F04F0300 MOV      r3,#0x00
17: LOOP  AND r4, r0, r1      ; r4 = r0 AND r1
0x00000294 EA000401 AND      r4,r0,r1
18:      CMP r4, #0          ; if r4 == 0...
0x00000298 2C00      CMP      r4,#0x00
19:      BEQ INC              ; ...then jump to INC, otherwise continue to r2++
part3.s
11      ; Initialize registers.
12      MOV r0, #34
13      MOV r1, #1
14      MOV r2, #0
15      MOV r3, #0
17 LOOP AND r4, r0, r1      ; r4 = r0 AND r1
18      CMP r4, #0          ; if r4 == 0...
19      BEQ INC              ; ...then jump to INC, otherwise continue to r2++
20      ADD r2, r2, #1      ; r2++
21      INC                  ; r2++
22      LSL r1, r1, #1      ; r1 = r1 << 1
23      CMP r3, #32          ; 32 bits = sizeof(int) * 8 bits per byte
24      BLT LOOP            ; if( r3 < 32 ) goto LOOP; otherwise end program.
25      END

```

Setting R0 = 34, the program runs 20 times (32 decimal) the computed number of 1's in 34(decimal) Or 0x22 was 2 stored in R2.

The screenshot shows the same debugger window after the program has executed for 20 iterations. The registers have been updated, and the disassembly view shows the program reaching the end of the loop.

Register	Value
R0	0x00000022
R1	0x00000000
R2	0x00000002
R3	0x00000020
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000400
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x000002AA
xPSR	0x61000000

The disassembly view shows the following code:

```

0x000002A8 2B20      CMP      r3,#0x20
24:      BLT LOOP            ; if( r3 < 32 ) goto LOOP; otherwise end program.
0x000002AA DBF3      BLT      0x00000294
0x000002AC 0000      MOV      r0,r0
0x000002AE 0000      MOV      r0,r0
0x000002B0 0000      MOV      r0,r0
part3.s
11      ; Initialize registers.
12      MOV r0, #34
13      MOV r1, #1
14      MOV r2, #0
15      MOV r3, #0
17 LOOP AND r4, r0, r1      ; r4 = r0 AND r1
18      CMP r4, #0          ; if r4 == 0...
19      BEQ INC              ; ...then jump to INC, otherwise continue to r2++
20      ADD r2, r2, #1      ; r2++
21      INC                  ; r2++
22      LSL r1, r1, #1      ; r1 = r1 << 1
23      CMP r3, #32          ; 32 bits = sizeof(int) * 8 bits per byte
24      BLT LOOP            ; if( r3 < 32 ) goto LOOP; otherwise end program.
25      END

```

Setting R0 = -1, the program runs 20 times (32 decimal) the computed number of 1's in -1(decimal) 0r 0xF was 20 stored in R2.

The screenshot displays the Keil MDK integrated development environment. On the left, the 'Registers' window shows the state of various registers. On the right, the 'Disassembly' window shows the assembly code being executed.

Register	Value
R0	0xFFFFFFFF
R1	0x00000000
R2	0x00000020
R3	0x00000020
R4	0x80000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000400
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x000002AA
xPSR	0x61000000

The assembly code in the 'Disassembly' window is as follows:

```

25:          BLT LOOP          ; if( r3 < 32 ) goto LOOP; otherwise end program.
0x000002AA DBF3          BLT          0x00000294
0x000002AC 0000          MOVS          r0,r0
0x000002AE 0000          MOVS          r0,r0
0x000002B0 0000          MOVS          r0,r0
0x000002B2 0000          MOVS          r0,r0
0x000002B4 0000          MOVS          r0,r0

part3.s
1  AREA |.text|,CODE,READONLY,ALIGN=2
2  THUMB
3  EXPORT Start
4  Start
5      ; R0 = number
6      ; R1 = mask
7      ; R2 = counter of ones (Output of the program)
8      ; R3 = loop counter
9      ; R4 = AND result
10
11     ; Initialize registers.
12     MOV r0, #-1
13     MOV r1, #1
14     MOV r2, #0
15     MOV r3, #0
16
17 LOOP  AND r4, r0, r1      ; r4 = r0 AND r1
18       CMP r4, #0         ; if r4 == 0...
19       BEQ INC            ; ...then jump to INC, otherwise continue to r2++
20       ADD r2, r2, #1      ; r2++
21
22 INC   ADD r3, r3, #1
23       LSL r1, r1, #1      ; r1 = r1 << 1
24       CMP r3, #32        ; 32 bits = sizeof(int) * 8 bits per byte
25       BLT LOOP          ; if( r3 < 32 ) goto LOOP; otherwise end program.
26       END

```

Conclusion

This lab has successfully introduced me to Keil MDK integrated development environment. I was able to create new projects, debug code and implement new algorithm that counter the number 1's in a 32-bit number in ARM assembly language.

Bibliography

Badr: Explanation

Tutorial: <https://youtu.be/NwyPuKM1qvw>