



ARM ASSEMBLY LAB-1

By

Swapneel Pimparkar (CS18M516)

CS6620a

Prof. Madhu Mutyam

STATUS UPDATE

All the three items namely,

1. Setting up the environment
2. Setting up the tool
3. Exercise – Your Mnemonics

were attempted.

EXERCISE – YOUR MNEMONICS

There are multiple ways to produce the mnemonics for same operation.

For this lab, not very fine tuned or optimized mnemonic sequence is used. Rather focus was to get the output correct.

Wherever feasible, multiple logics are mentioned and attempted too.

The mnemonics can be generated with multiple addressing modes. Code is not with all possible addressing modes & permutation/combination thereof.

Small numbers are assumed for function verification and coding at this moment.

In some files, multiple logics are attempted and only one is kept enabled.

XNOR R1, R2, R3

The optimal logic (in terms of instruction cycles) is –

`CMP R2,R3`

`BEQ _EQUAL`

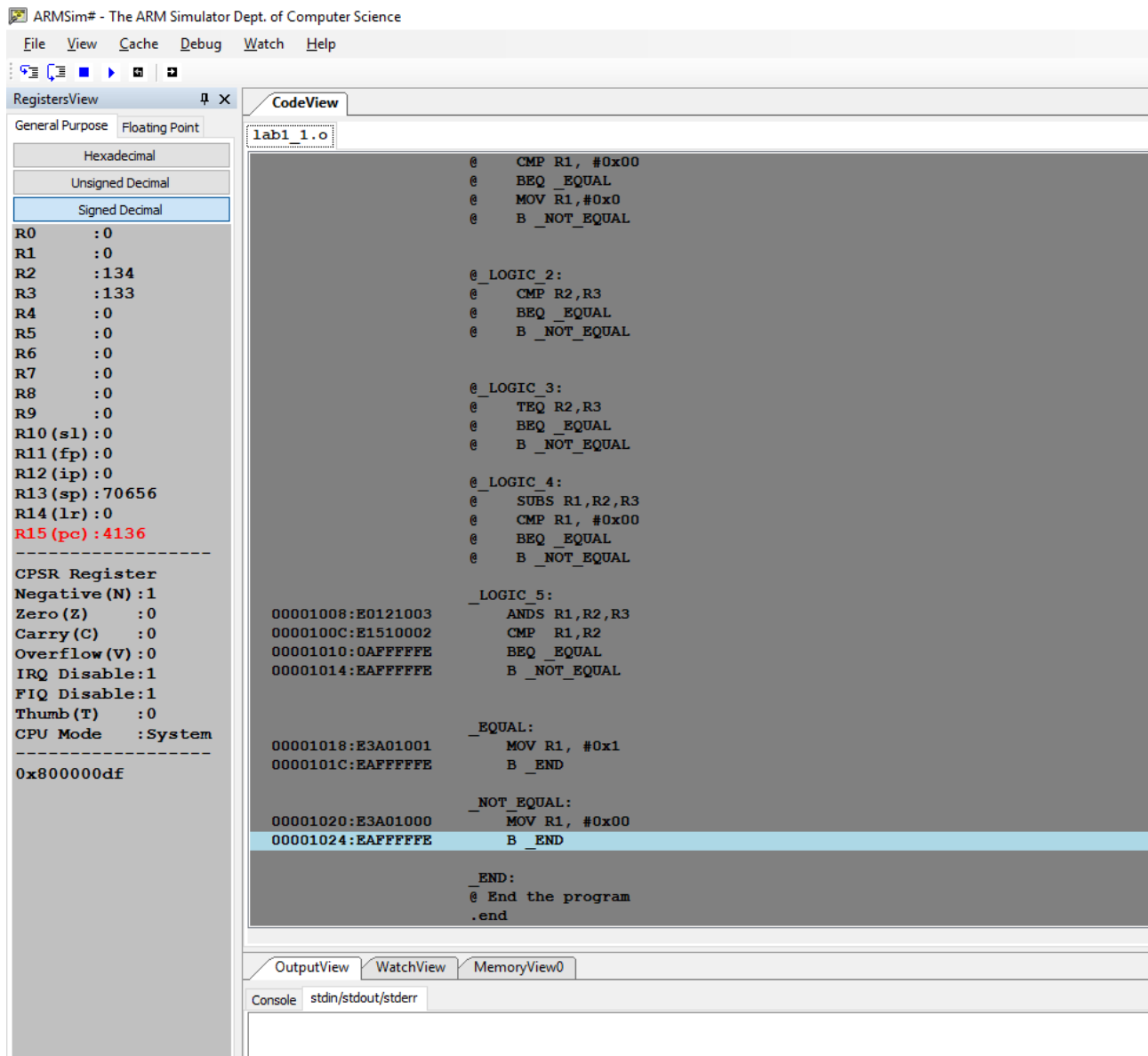
`B _NOT_EQUAL`

However, multiple logics are mentioned in the code file (**lab1_1.s**).

Here, logical XNOR is assumed.

This essentially means – the output would be either true (1) or false (0) and not the bitwise XNOR value.

The sample output screenshot for one of the logic is –



BITWISE AND OF R2 & COMPLEMENT OF R3

The essential logic in corresponding file (**lab1_2.s**) is -

@ Complement R3

EORS R3,R3,#0xFF

@ AND with R2 and store result in R1

ANDS R1,R2,R3

And the corresponding sample output screenshot is –

The screenshot displays the ARMSim# - The ARM Simulator interface. The left pane shows the Register View with the following values:

| Register | Value |
|----------|---------|
| R0 | : 0 |
| R1 | : 4 |
| R2 | : 15 |
| R3 | : 244 |
| R4 | : 0 |
| R5 | : 0 |
| R6 | : 0 |
| R7 | : 0 |
| R8 | : 0 |
| R9 | : 0 |
| R10 (s1) | : 0 |
| R11 (fp) | : 0 |
| R12 (ip) | : 0 |
| R13 (sp) | : 70656 |
| R14 (lr) | : 0 |
| R15 (pc) | : 4136 |

The right pane shows the Code View for the file lab1_2.o. The assembly code is as follows:

```
@ -----  
@ 7 Sep 2019 Swapneel Pimparkar Added and Tested Logic 1  
@ 7 Sep 2019 Swapneel Pimparkar (Bengaluru) First Draft  
@ -----  
  
@ -----  
@ Logic:  
@ -----  
  
@ Text Section Begins  
.text  
  
@ Global labels declared. For description of logic_x please refer above comments.  
@ related to associated logic.  
  
    .global _START  
    .global _END  
    .global _LOGIC_1  
    @ .global _INPUT1  
    @ .global _INPUT2  
  
    @ _INPUT1: .word 15  
    @ _INPUT2: .word 5  
  
_START:  
    MOV R2, #0xF  
    MOV R3, #0xB  
  
    @ Alternatively we can use LDR instructions too instead of MOV.  
    @ LDR R2, =_INPUT1  
    @ LDR R3, =_INPUT2  
  
_LOGIC_1:  
    @ Complement R3  
    EORS R3,R3,#0xFF  
    @ AND with R2 and store result in R1  
    ANDS R1,R2,R3  
  
_END:  
    @ End the program  
    .end
```

The bottom pane shows the Output View, WatchView, and MemoryView0. The Console tab is selected, showing the output: 0x000000df.

FAST MULTIPLICATION WITHOUT MUL

Assumed is Integer Multiplication only. First a C program without '*' operator was written so that MUL variants of instructions are not generated by compiler.

The code file for this exercise is named – **lab1_3.s**

Here is the sample output –

ARMSim# - The ARM Simulator Dept. of Computer Science

File View Cache Debug Watch Help

RegistersView

General Purpose Floating Point

Hexadecimal
Unsigned Decimal
Signed Decimal

R0 : 0
R1 : 162
R2 : 3
R3 : 54
R4 : 0
R5 : 0
R6 : 0
R7 : 0
R8 : 0
R9 : 0
R10 (s1) : 0
R11 (fp) : 0
R12 (ip) : 0
R13 (sp) : 70656
R14 (lr) : 0
R15 (pc) : 70656

CPSR Register
Negative (N) : 0
Zero (Z) : 1
Carry (C) : 1
Overflow (V) : 0
IRQ Disable : 1
FIQ Disable : 1
Thumb (T) : 0
CPU Mode : System

0x600000df

lab1_3.o

```
@      tuned for 32 bit ARMv4. This is in essence using shift & add method.
@      This is verified for positive multiplication only.
@-----

@ Text Section Begins
.text

@ Global labels declared. For description of logic_x please refer above comments.
@ related to associated logic.

.global _START
.global _END
.global _LOOP
.global _COMMON

_START:
00001000:E3A03036    MOV     R3, #54 @Input value 1 (taken from exercise)
00001004:E3A02003    MOV     R2, #3  @Input value 2 (assuming 3)

00001008:E3A01000    MOV     R1, #0  @Output will be stored in R1
0000100C:E1A04003    MOV     R4, R3

@ Following two MOV instructions are not necessary but we are using R5 and R6
@ so that input values in R2 and R3 can be preserved. Just for our visualization.
@ And hence all the operations are done on R5 and R6 below.

00001010:E1A05003    MOV     R5, R3
00001014:E1A06002    MOV     R6, R2

_LOOP:
00001018:E3160001    TST     R6, #1
0000101C:10811004    ADDNE   R1, R1, R4
00001020:E1A04084    MOV     R4, R4, ASL #1
00001024:E0866FA6    ADD     R6, R6, R6, LSR #31
00001028:E1A060C6    MOV     R6, R6, ASR #1
0000102C:E2555001    SUBS    R5, R5, #1
00001030:1AFFFFFEE    BNE     _LOOP

_END:
@ End the program
.end
```

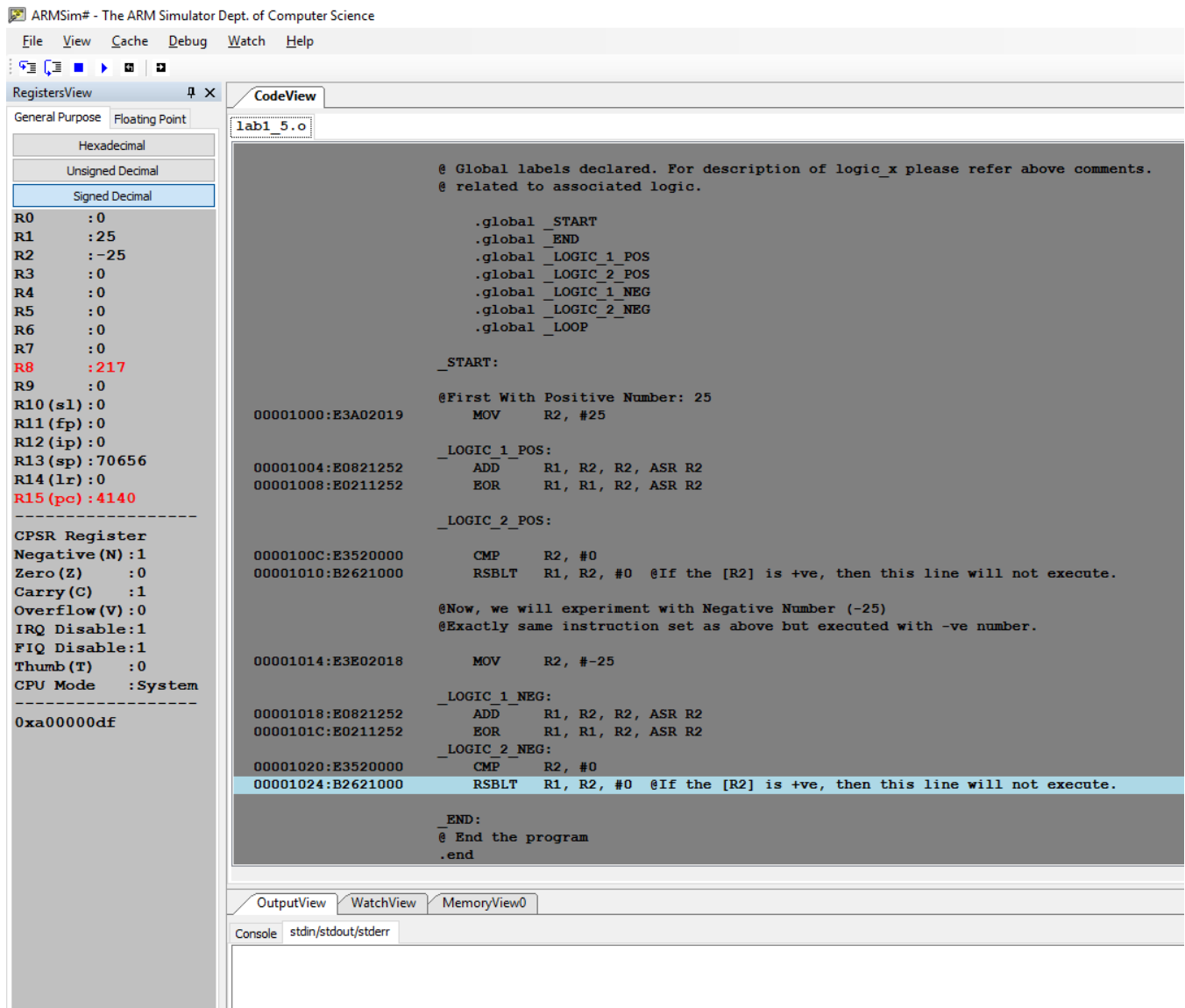
OutputView WatchView MemoryView0

Console stdin/stdout/stderr

Fast Division

The logic is simple here. For simplicity (and understanding), two loops are used instead of one. First loop is essentially for adjusting smaller divisor to that of larger dividend. And second loop is essentially with successive subtract. Only one loop could have been used too. The code file is **lab1_5.s**.

Corresponding sample output is –



```
ARMSim# - The ARM Simulator Dept. of Computer Science
File View Cache Debug Watch Help

RegistersView
General Purpose Floating Point
Hexadecimal
Unsigned Decimal
Signed Decimal
R0 : 0
R1 : 25
R2 : -25
R3 : 0
R4 : 0
R5 : 0
R6 : 0
R7 : 0
R8 : 217
R9 : 0
R10 (s1) : 0
R11 (fp) : 0
R12 (ip) : 0
R13 (sp) : 70656
R14 (lr) : 0
R15 (pc) : 4140
-----
CPSR Register
Negative (N) : 1
Zero (Z) : 0
Carry (C) : 1
Overflow (V) : 0
IRQ Disable : 1
FIQ Disable : 1
Thumb (T) : 0
CPU Mode : System
-----
0xa00000df

lab1_5.o

@ Global labels declared. For description of logic_x please refer above comments.
@ related to associated logic.

.global _START
.global _END
.global _LOGIC_1_POS
.global _LOGIC_2_POS
.global _LOGIC_1_NEG
.global _LOGIC_2_NEG
.global _LOOP

_START:

@First With Positive Number: 25
00001000:E3A02019 MOV R2, #25

_LOGIC_1_POS:
00001004:E0821252 ADD R1, R2, R2, ASR R2
00001008:E0211252 EOR R1, R1, R2, ASR R2

_LOGIC_2_POS:

0000100C:E3520000 CMP R2, #0
00001010:B2621000 RSBLT R1, R2, #0 @If the [R2] is +ve, then this line will not execute.

@Now, we will experiment with Negative Number (-25)
@Exactly same instruction set as above but executed with -ve number.

00001014:E3E02018 MOV R2, #-25

_LOGIC_1_NEG:
00001018:E0821252 ADD R1, R2, R2, ASR R2
0000101C:E0211252 EOR R1, R1, R2, ASR R2

_LOGIC_2_NEG:
00001020:E3520000 CMP R2, #0
00001024:B2621000 RSBLT R1, R2, #0 @If the [R2] is +ve, then this line will not execute.

_END:
@ End the program
.end

OutputView WatchView MemoryView0
Console stdin/stdout/stderr
```