ARM ASSEMBLY LAB-5

By

Swapneel Pimparkar (CS18M516)
CS6620a
Prof. Madhu Mutyam

STATUS UPDATE

The ARM assembly code was written for 32 bit processor and verified using ARMSim simulator successfully.

EXERCISE 1 – CONVERT FROM AN ASCII CHARACTER TO A HEXADECIMAL DIGIT.

Convert the contents of a given A_DIGIT variable from an ASCII character to a hexadecimal digit and store the result in H_DIGIT. Assume that A_DIGIT contains the ASCII representation of a hexadecimal digit (i.e., 7 bits with MSB=0).

The logic to solve this exercise is hand-coded in 32 bit ARM Assembly and verified on ARMSim Simulator.

The logic used is mentioned in the code file itself and all the necessary instructions are supplied with comments.

Various unit tests are also part of the code file itself. Need to uncomment one by one and run. All the unit tests passed including two error cases.

All the output locations are initialized with 0xFF.

SCREENSHOT - TEST 1

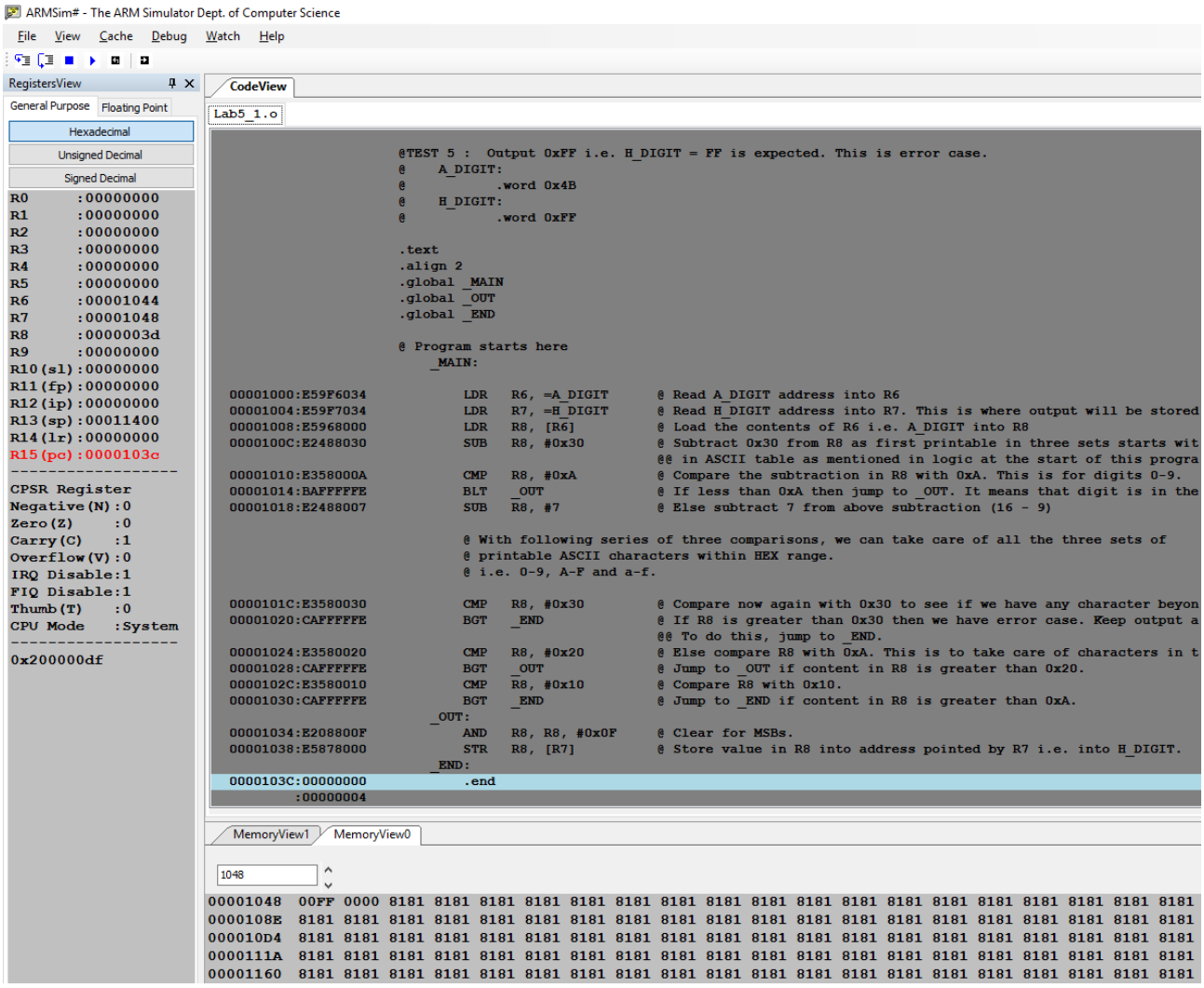
The sample output screenshot for the logic is as below for input (which is essentially error case) –

A_DIGIT:

.word 0x74

H_DIGIT:

.word 0xFF



SCREENSHOT - TEST 2

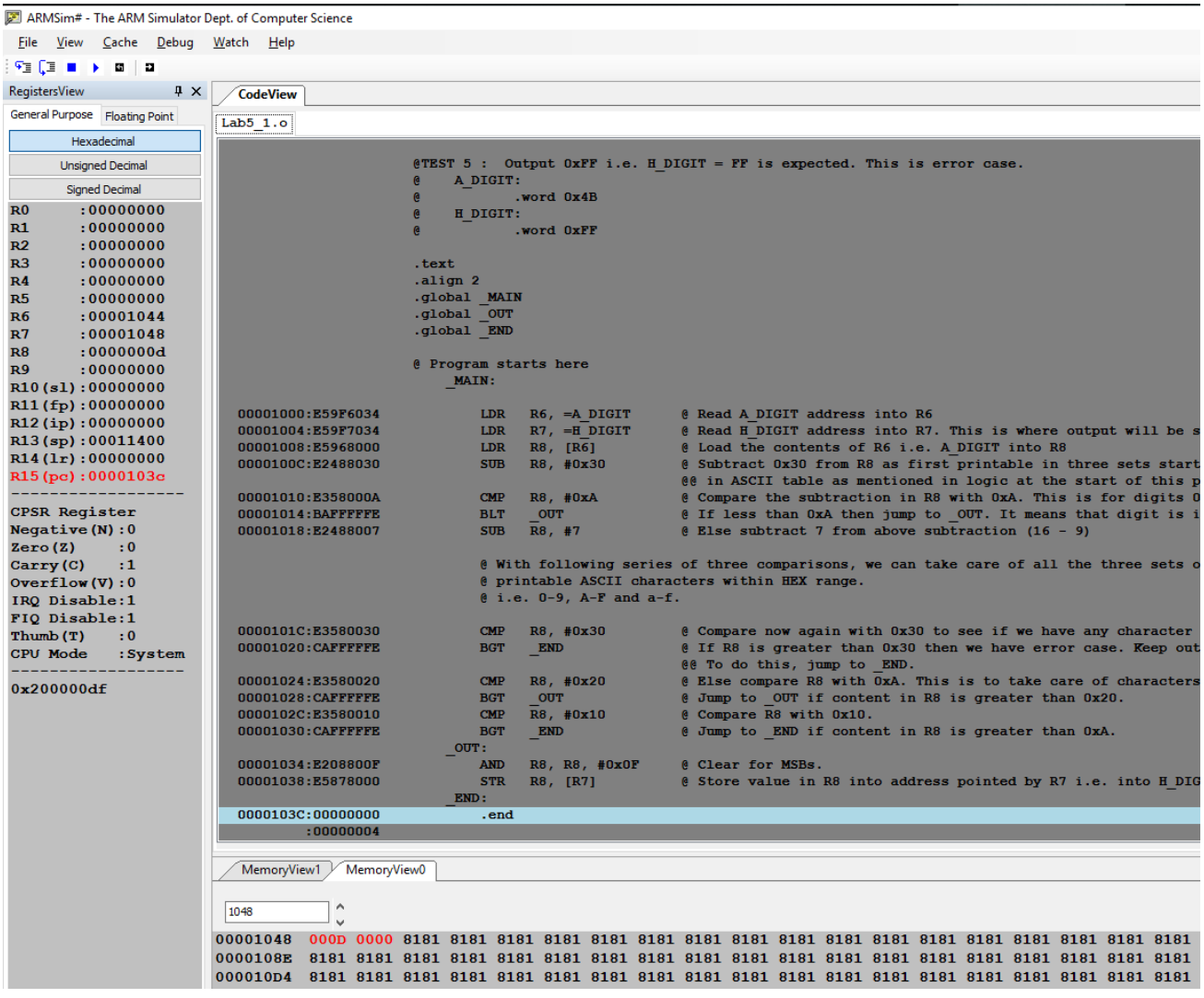
The sample output screenshot for the logic is as below for input (which is essentially good case for small case letter) –

A_DIGIT:

.word 0x64

H_DIGIT:

.word 0xFF



SCREENSHOT- TEST 3

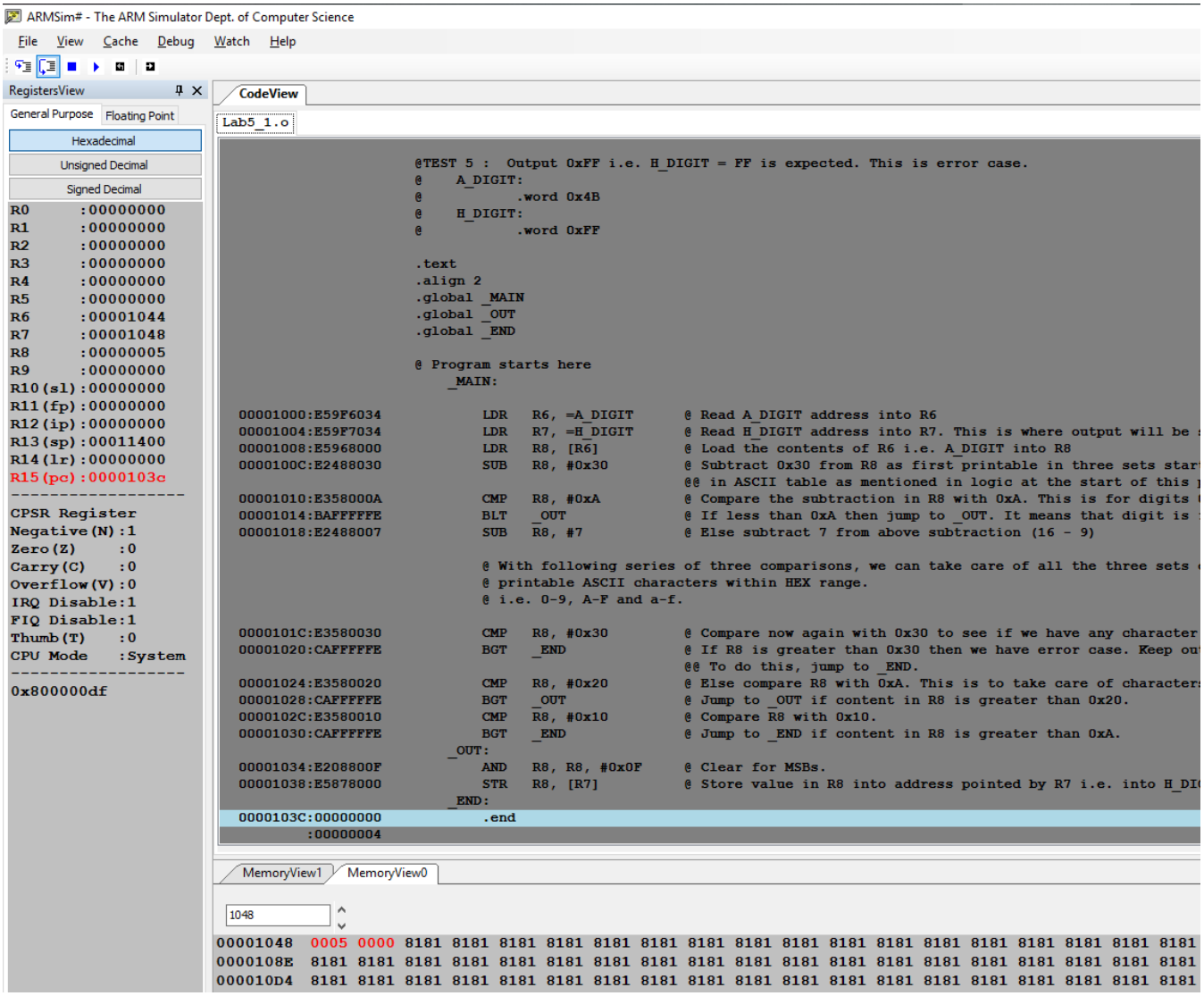
The sample output screenshot for the logic is as below for input (which is essentially good case for decimal digit) –

A_DIGIT:

.word 0x35

H_DIGIT:

.word 0xFF



SCREENSHOT - TEST 4

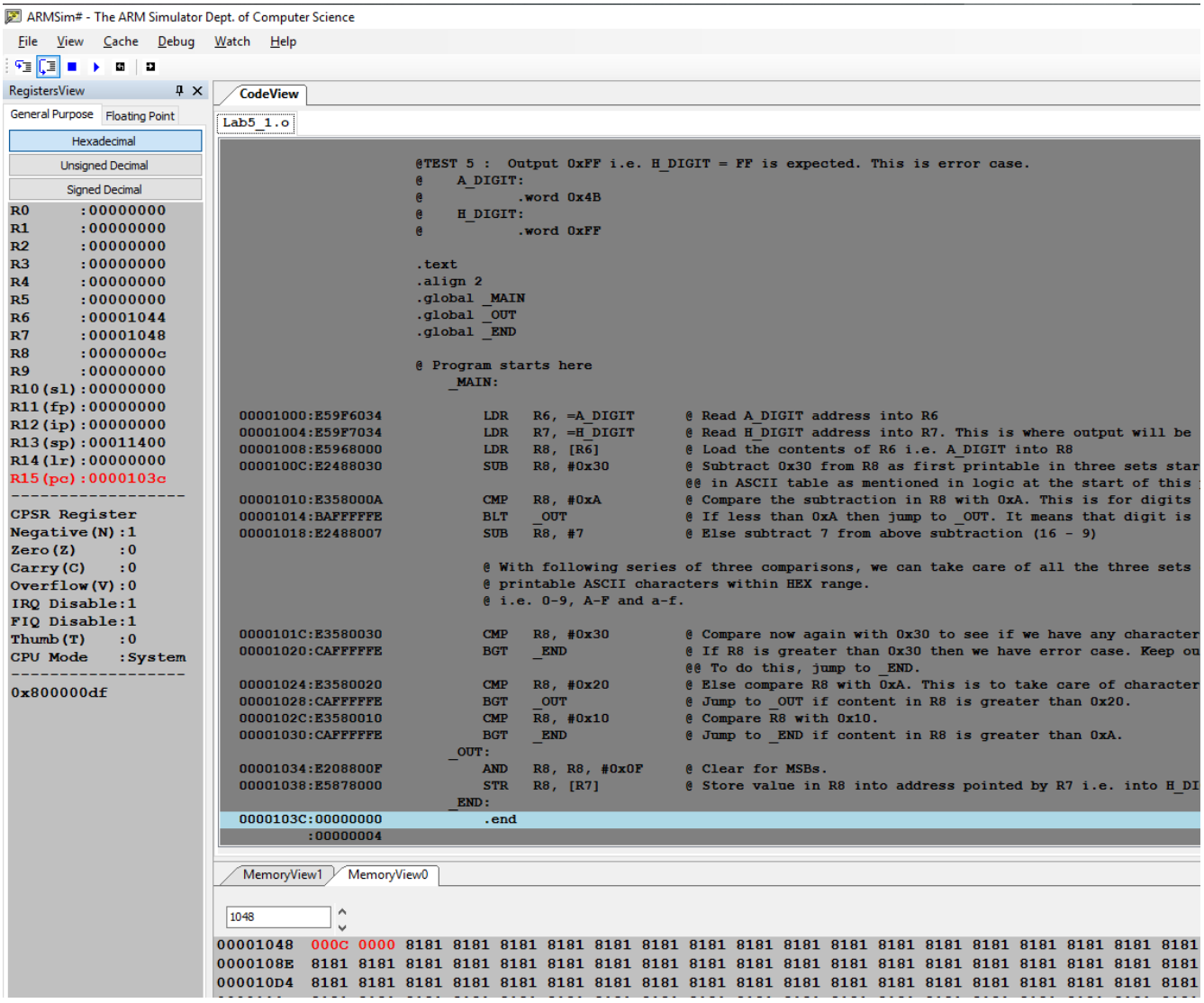
The sample output screenshot for the logic is as below for input (which is essentially good case for capital letter ‘C’) –

A_DIGIT:

.word 0x43

H_DIGIT:

.word 0xFF



SCREENSHOT - TEST 5

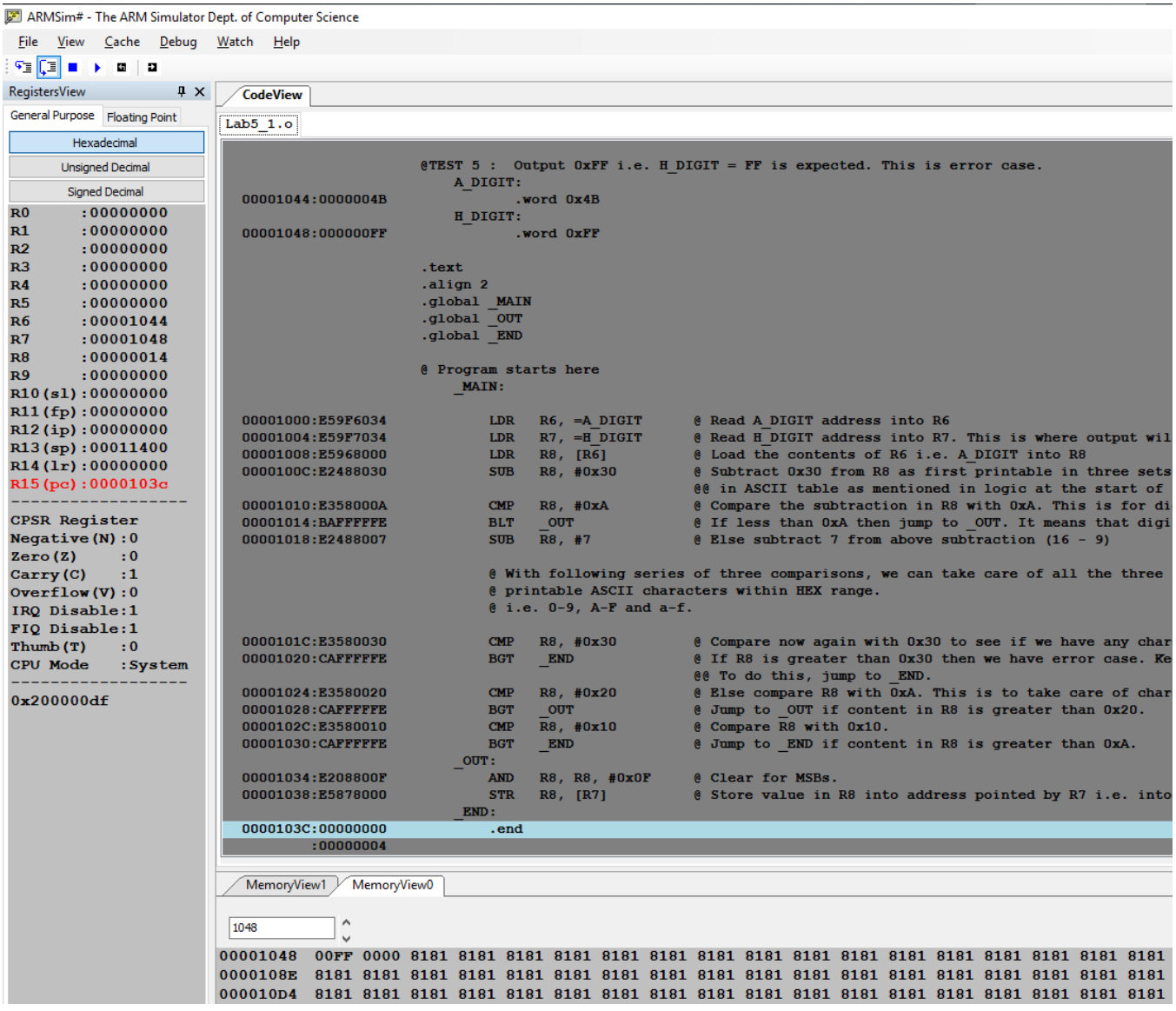
The sample output screenshot for the logic is as below for input (which is essentially error case) –

A_DIGIT:

.word 0x4B

H_DIGIT:

.word 0xFF



EXERCISE – CONVERT A GIVEN EIGHT ASCII CHARACTERS TO AN 8-BIT BINARY NUMBER.

Convert a given eight ASCII characters in the variable STRING to an 8-bit binary number in the variable NUMBER. Clear the byte variable ERROR if all the ASICC characters are either ASCII “1” or ASCII “0”; otherwise set ERROR to all ones (0xFF).

The logic to solve this exercise is hand-coded in 32 bit ARM Assembly and verified on ARMSim Simulator.

The logic used is mentioned in the code file itself and all the necessary instructions are supplied with comments.

Two unit tests are also part of the code file itself. Need to uncomment and run.

Both the unit tests passed.

SCREENSHOT – TEST 1

The sample output screenshot for the logic is as below for input (which is essentially good case)

STRING:

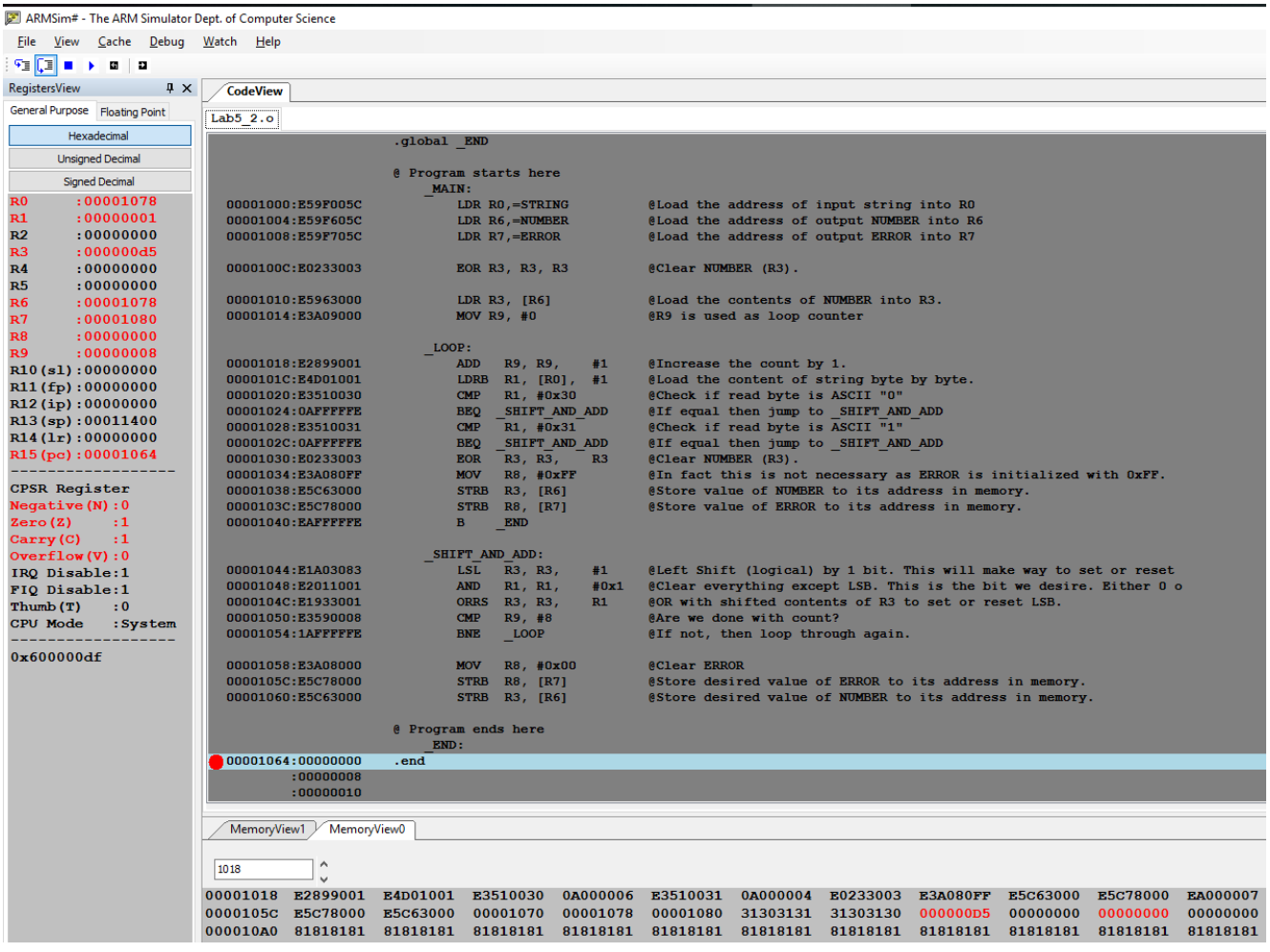
.ascii "11010101"

NUMBER:

.byte 0x0

ERROR:

.byte 0xFF



SCREENSHOT – TEST 2

The sample output screenshot for the logic is as below for input (which is essentially bad case)

STRING:

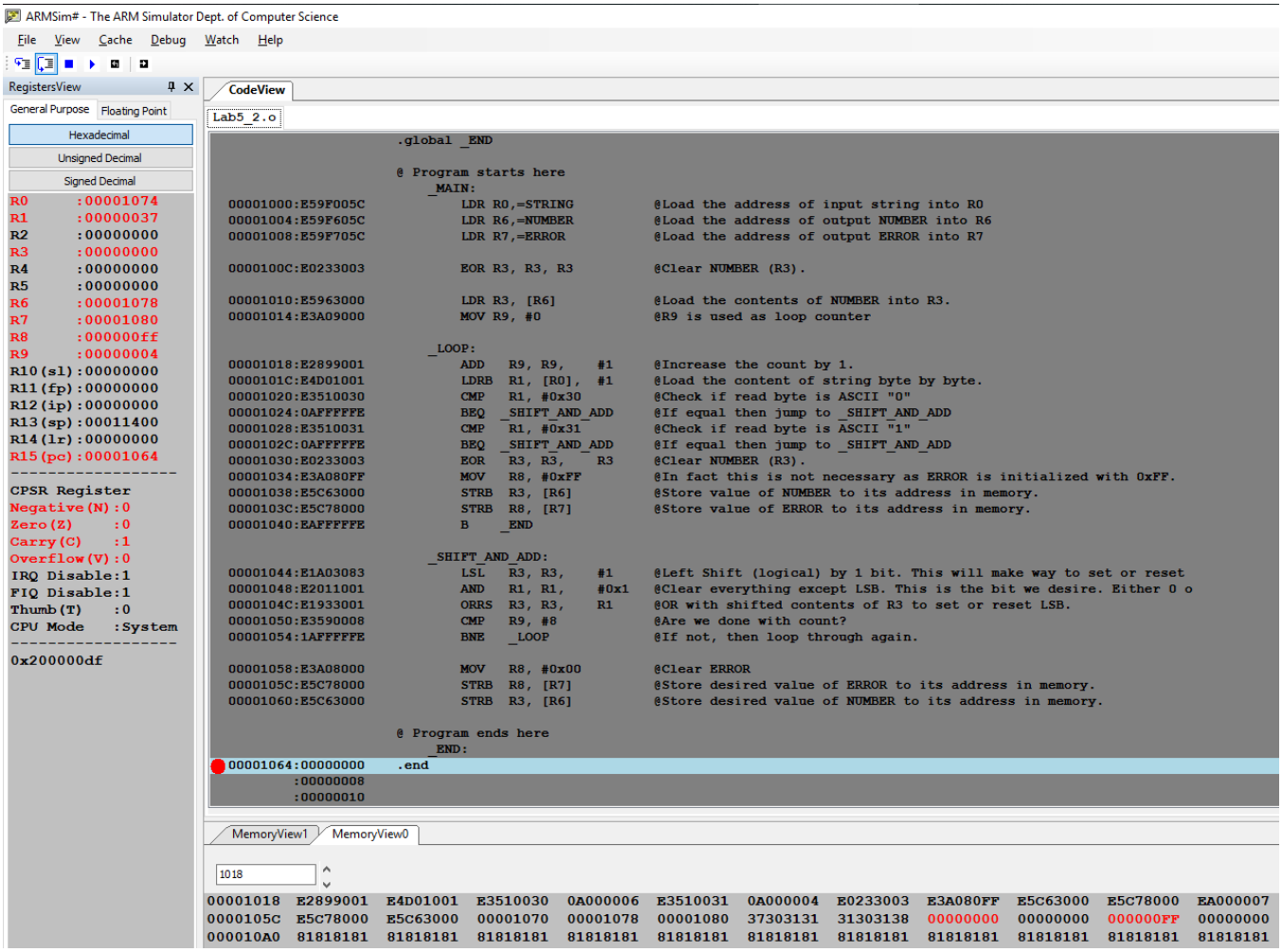
.ascii " 11078101"

NUMBER:

.byte 0x0

ERROR:

.byte 0xFF



EXERCISE – CONVERT A GIVEN EIGHT ASCII CHARACTERS TO AN 8-BIT BINARY NUMBER.

Convert a given eight-digit packed binary-coded-decimal number in the BCDNUM variable into a 32-bit number in a NUMBER variable.

The logic is of two types. One inherent from ARM and one hand computed.

First is that ARM assembly inherently stores number as hex. So, no extra effort is needed. Just store it after reading back to destination.

Second is convert the packed BCD (i.e. decimal) to hex. This can be achieved using positional logic.

Using the first one for ease and optimal coding.

SCREENSHOT – TEST 1

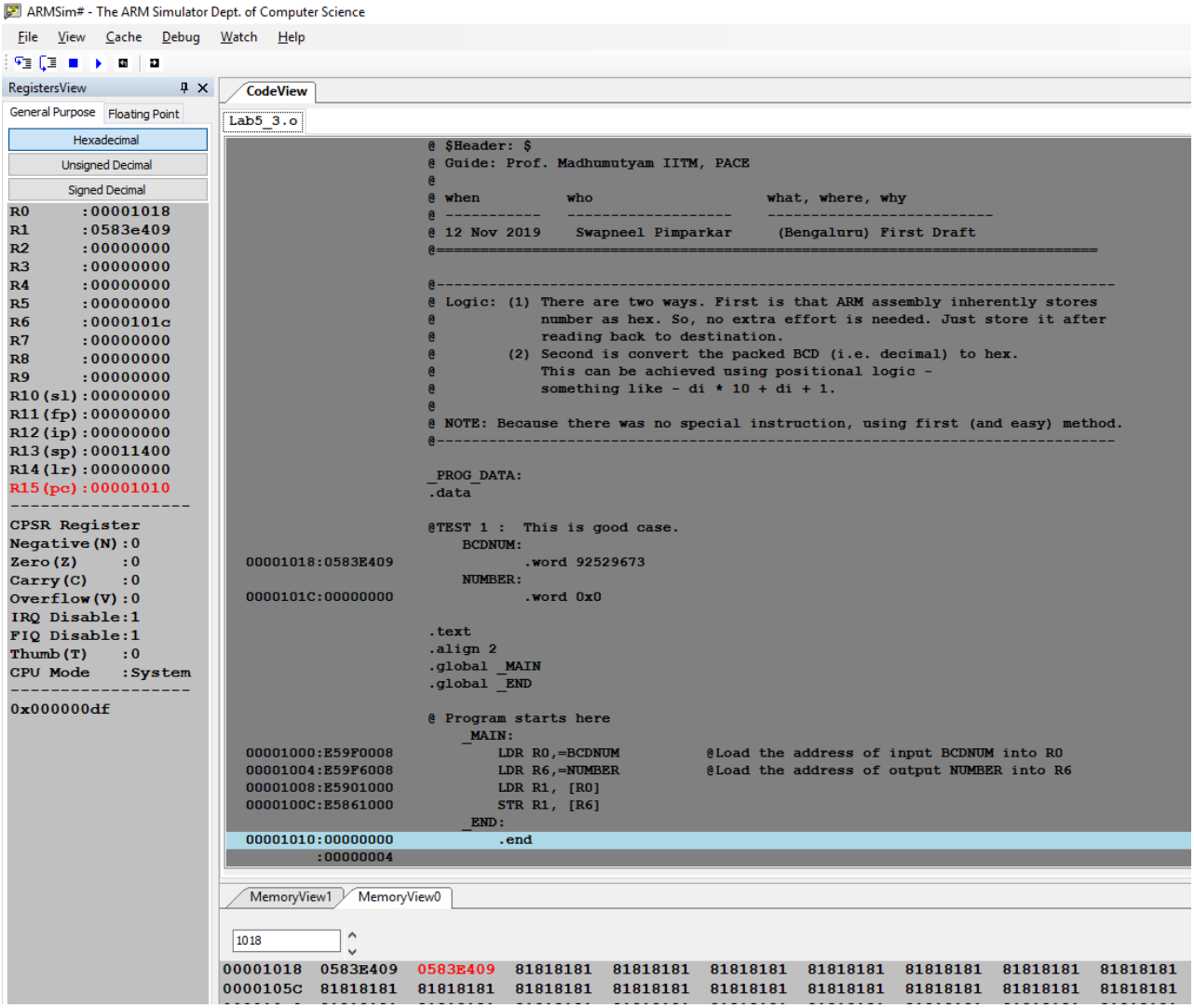
The sample output screenshot for the logic is as below for input (which is essentially bad case)

BCDNUM:

```
.word 92529673
```

NUMBER:

```
.word 0x0
```


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 :00001018
R1 :0583e409
R2 :00000000
R3 :00000000
R4 :00000000
R5 :00000000
R6 :0000101c
R7 :00000000
R8 :00000000
R9 :00000000
R10(sl):00000000
R11(fp):00000000
R12(ip):00000000
R13(sp):00011400
R14(lr):00000000
R15(pc):00001010
CPSR Register
Negative(N):0
Zero(Z) :0
Carry(C) :0
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T) :0
CPU Mode :System
0x000000df
CodeView
Lab5_3.o
@ $Header: $
@ Guide: Prof. Madhumutyam IITM, PACE
@ when who what, where, why
@ 12 Nov 2019 Swapneel Pimparkar (Bengaluru) First Draft
@ Logic: (1) There are two ways. First is that ARM assembly inherently stores
@ number as hex. So, no extra effort is needed. Just store it after
@ reading back to destination.
@ (2) Second is convert the packed BCD (i.e. decimal) to hex.
@ This can be achieved using positional logic -
@ something like - di * 10 + di + 1.
@ NOTE: Because there was no special instruction, using first (and easy) method.
_PROG_DATA:
.data
@TEST 1 : This is good case.
BCDNUM:
00001018:0583E409 .word 92529673
NUMBER:
0000101C:00000000 .word 0x0
.text
.align 2
.global _MAIN
.global _END
@ Program starts here
_MAIN:
00001000:E59F0008 LDR R0,=BCDNUM @Load the address of input BCDNUM into R0
00001004:E59F6008 LDR R6,=NUMBER @Load the address of output NUMBER into R6
00001008:E5901000 LDR R1, [R0]
0000100C:E5861000 STR R1, [R6]
_END:
00001010:00000000 .end
:00000004
MemoryView1
MemoryView0
1018
00001018 0583E409 0583E409 81818181 81818181 81818181 81818181 81818181 81818181 81818181
0000105C 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181 81818181