

# CSE 3113 - Microprocessor and Assembly Lab

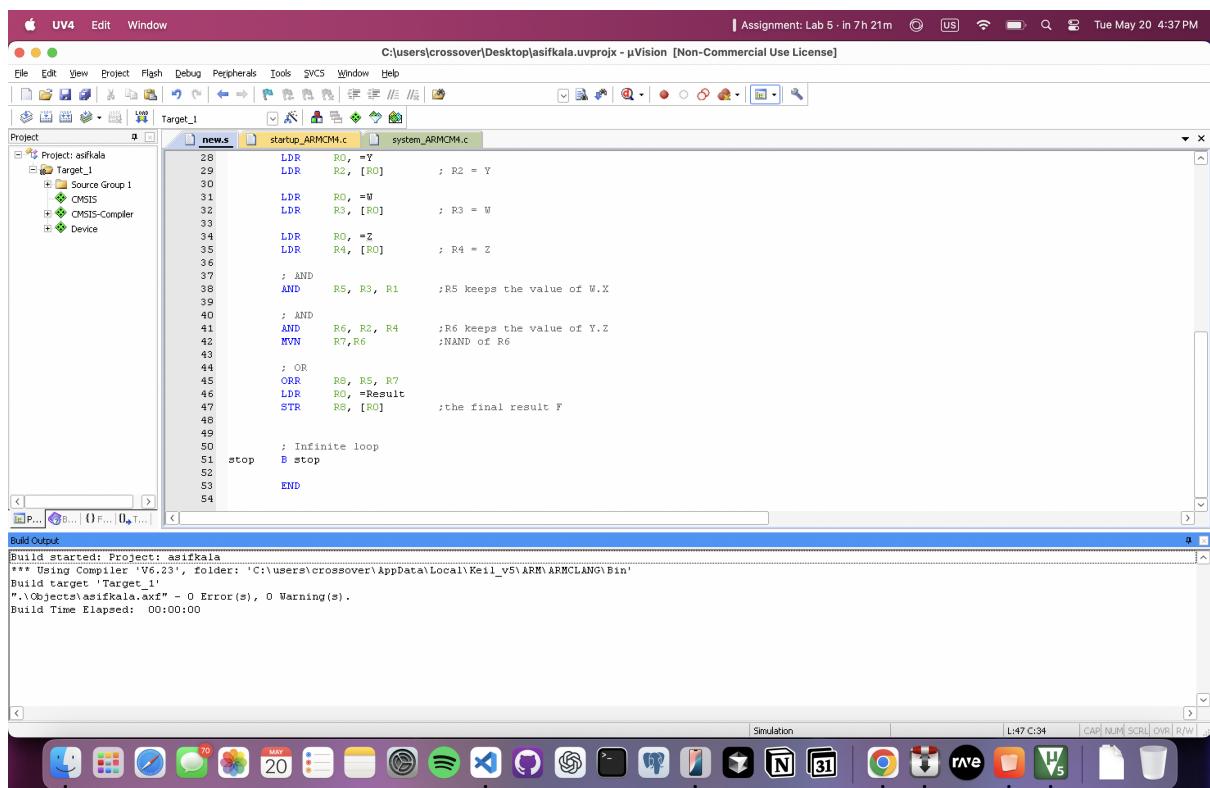
## Lab Report

Tazkia Malik  
Class Roll: 07

May 20, 2025

### 1 Write assembly language to perform a simple Boolean operation to calculate the bitwise calculation of F = W.X + NOT(Y.Z)

#### Build Status



The screenshot shows the Keil µVision IDE interface. The assembly code for the file 'new.s' is displayed in the main window:

```
28    LDR   R0, =Y
29    LDR   R2, [R0]      ; R2 = Y
30
31    LDR   R0, =#0
32    LDR   R3, [R0]      ; R3 = 0
33
34    LDR   R0, =Z
35    LDR   R4, [R0]      ; R4 = Z
36
37    ; AND
38    AND   R5, R3, R1    ; R5 keeps the value of W.X
39
40    ; AND
41    AND   R6, R2, R4    ; R6 keeps the value of Y.Z
42    MVN   R7, R6          ; NAND of R6
43
44    ; OR
45    ORR   R8, R5, R7
46    LDR   R0, =Result
47    STR   R8, [R0]      ; the final result F
48
49
50    ; Infinite loop
51    stop  B stop
52
53    END
```

The build output window at the bottom shows the following message:

```
Build started: Project: asifkala
*** Using Compiler 'V6.23', folder: 'C:\users\crossover\AppData\Local\Keil_v5\ARM\ARMCLANG\Bin'
Build target: 'Target_1'
".\Objects\asifkala.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:00
```

Figure 1: State of the system after the project is built.

## Register Status After Execution

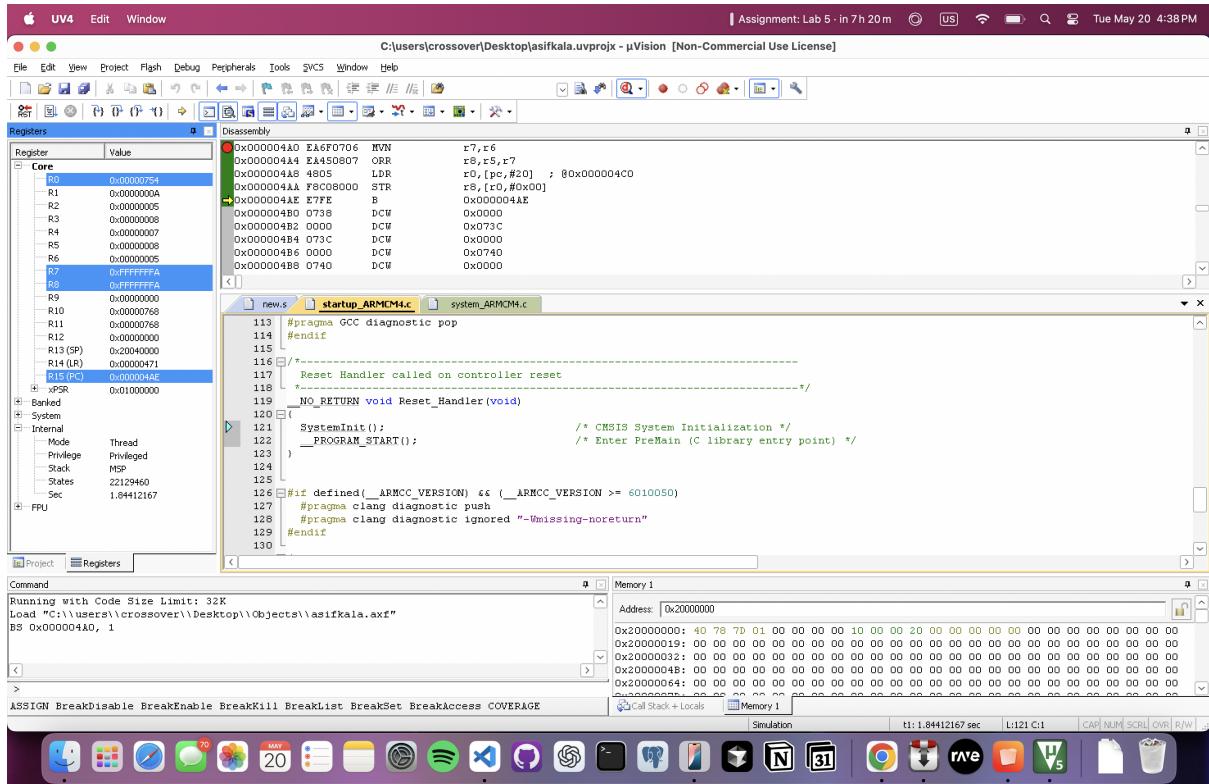
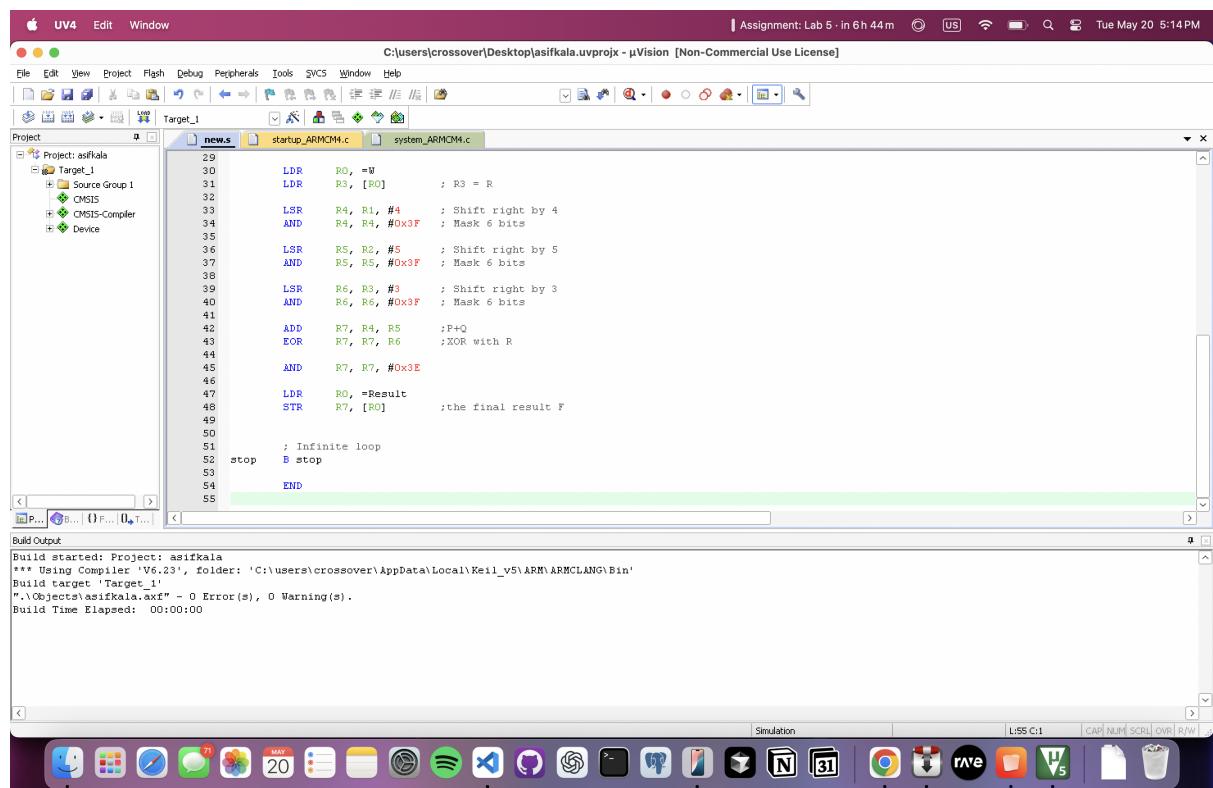


Figure 2: State of the system after the code has been executed.

**2 Suppose we have three words P, Q and R. We are going to apply logical operations to subfields (bit fields) of these registers. We'll use 16-bit arithmetic for simplicity. Suppose that we have three 6-bit bit fields in P, Q, and R as illustrated below. The bit fields are in green and are not in the same position in each word. A bit field is a consecutive sequence of bits that forms a logic entity. Often they are data fields packed in a register, or they may be graphical elements in a display (a row of pixels). However, the following example demonstrates the type of operation you may have to perform on bits.**

## Build Status



```

UV4 Edit Window
Assignment: Lab 5 - in 6 h 44 m ① US Tue May 20 5:14 PM
File Edit View Project Flash Debug Peripherals Tools SWCS Window Help
Target_1
Project new.s startup_ARMCM4.c system_ARMCM4.c
29
30     LDR   R0, =W
31     LDR   R3, [R0]      ; R3 = R
32
33     LSR   R4, R1, #4    ; Shift right by 4
34     AND   R4, R4, #0x3F ; Mask 6 bits
35
36     LSR   R5, R2, #5    ; Shift right by 5
37     AND   R5, R5, #0x3F ; Mask 6 bits
38
39     LSR   R6, R3, #3    ; Shift right by 3
40     AND   R6, R6, #0x3F ; Mask 6 bits
41
42     ADD   R7, R4, R5    ;P+Q
43     EOR   R7, R7, R6    ;XOR with R
44
45     AND   R7, R7, #0x3E
46
47     LDR   R0, =Result
48     STR   R7, [R0]      ;the final result F
49
50
51     ; Infinite loop
52     B stop
53
54     END
55
Build Output
Build started: Project: asifkala
*** Using Compiler 'V6.23', folder: 'C:\users\crossover\AppData\Local\Keil_v5\ARM\ARMCLANG\Bin'
Build target: 'Target_1'
".\Objects\asifkala.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:00

```

Figure 3: State of the system after the project is built.

## Register Status After Execution

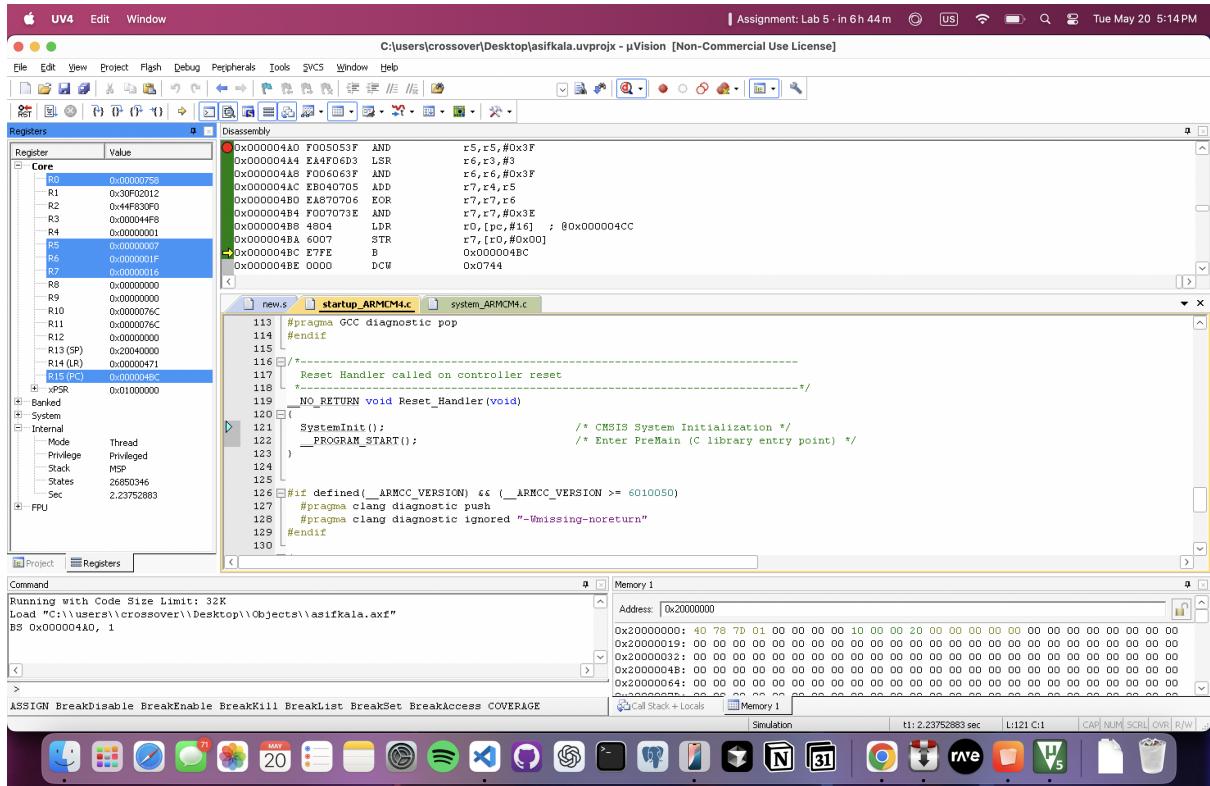


Figure 4: State of the system after the code has been executed.

### 3 Write an assembly language to find the one's complement of a number.

#### Build Status

The screenshot shows the µVision IDE interface. The main window displays the source code for the startup file, specifically the CMSIS initialization code. Below the code editor is a 'Build Output' window showing the build log:

```
Build started: Project: aeifkala
*** Using Compiler 'V6.23', folder: 'C:\users\crossover\AppData\Local\Keil_v5\ARM\ARMCLANG\Bin'
Build target 'Target_1'
".\Object\aeifkala.aifx" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:01
```

The system status bar at the bottom indicates the date and time: 'Tue May 20 4:45PM'.

Figure 5: State of the system after the project is built.

## Register Status After Execution

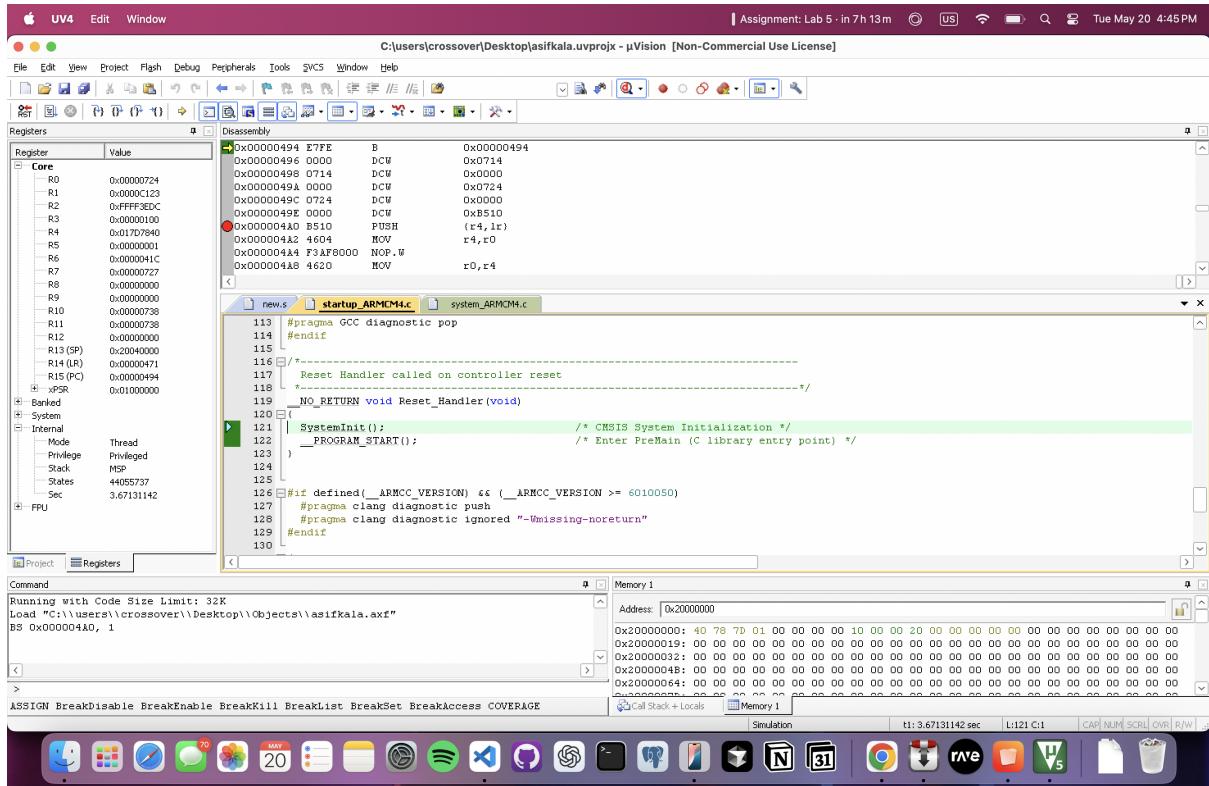
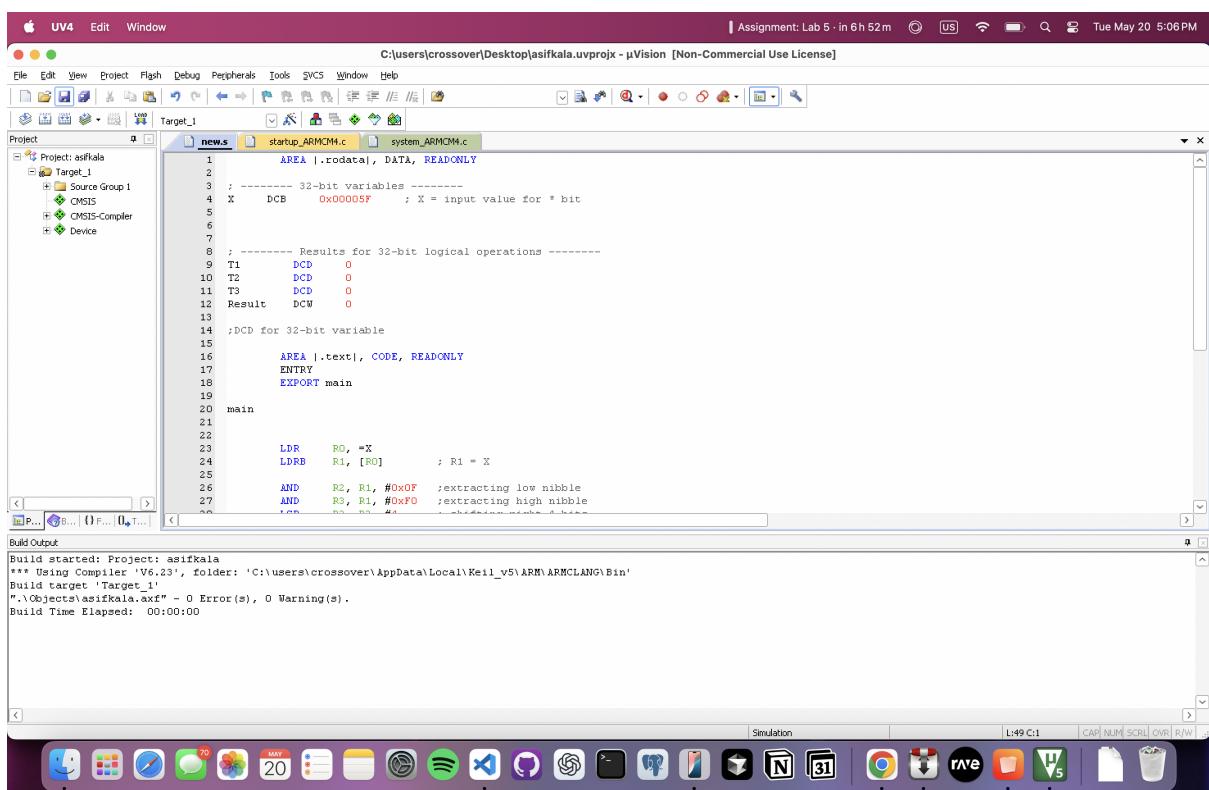


Figure 6: State of the system after the code has been executed.

**4 Write an assembly language program to disassemble a byte into its high and low order nibbles. Divide the least significant byte of the 8-bit variable Value into two 4-bit nibbles and store one nibble in each byte of the 16-bit variable Result. The low-order four bits of the byte will be stored in the low-order four bits of the least significant byte of Result. The high-order four bits of the byte will be stored in the low-order four bits of the most significant byte of Result.**

## Build Status



```

1 AREA .rodata, DATA, READONLY
2
3 ; ----- 32-bit variables -----
4 X    DCB  Ox00005F ; X = input value for * bit
5
6
7 ; ----- Results for 32-bit logical operations -----
8 T1   DCD  0
9 T2   DCD  0
10 T3  DCD  0
11 Result DCW  0
12
13 ;DCD for 32-bit variable
14
15 AREA .text, CODE, READONLY
16 ENTRY
17 EXPORT main
18
19
20 main
21
22
23 LDR R0, =X
24 LDRB R1, [R0]      ; R1 = X
25
26 AND R2, R1, #0x0F ;extracting low nibble
27 AND R3, R1, #0xF0 ;extracting high nibble
28
29
30

```

Build Output

```

Build started: Project: asifkala
*** Using Compiler 'V6.23', folder: 'C:\users\crossover\AppData\Local\Keil_v5\ARM\ARMCLANG\Bin'
Build target 'Target_1'
".\Objects\asifkala.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:00

```

Figure 7: State of the system after the project is built.

## Register Status After Execution

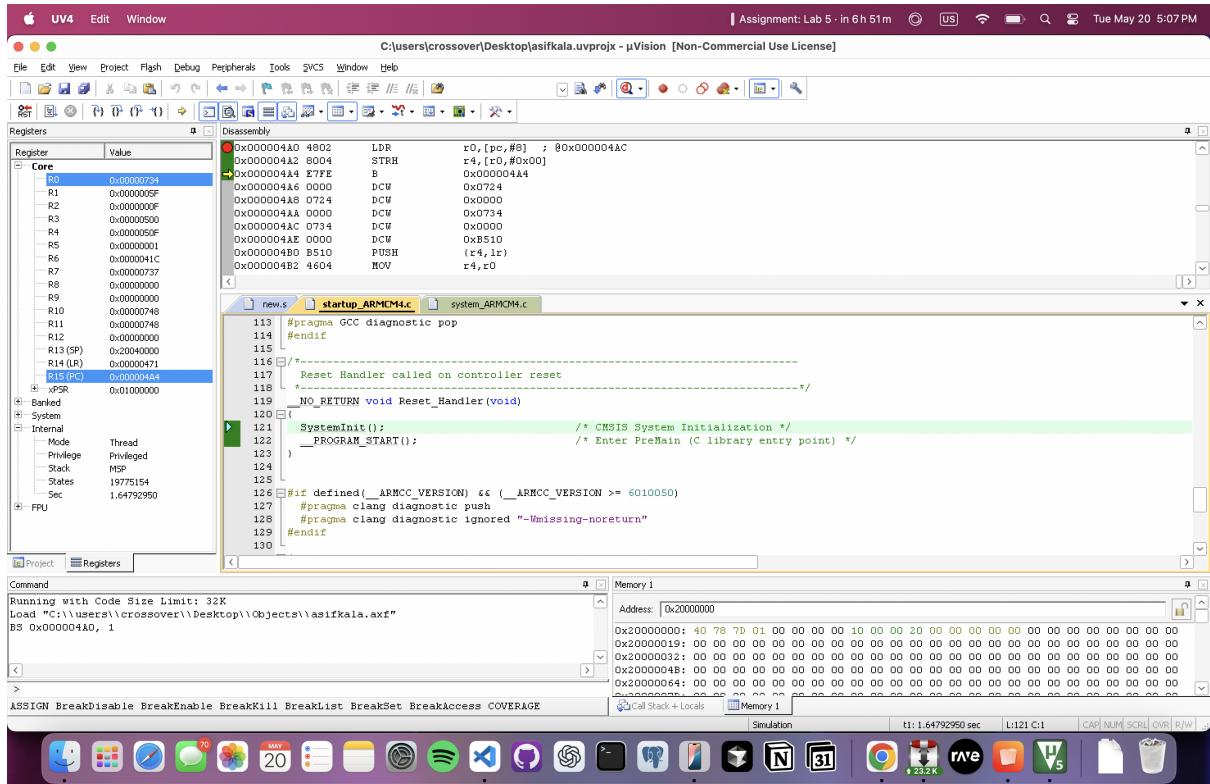


Figure 8: State of the system after the code has been executed.