

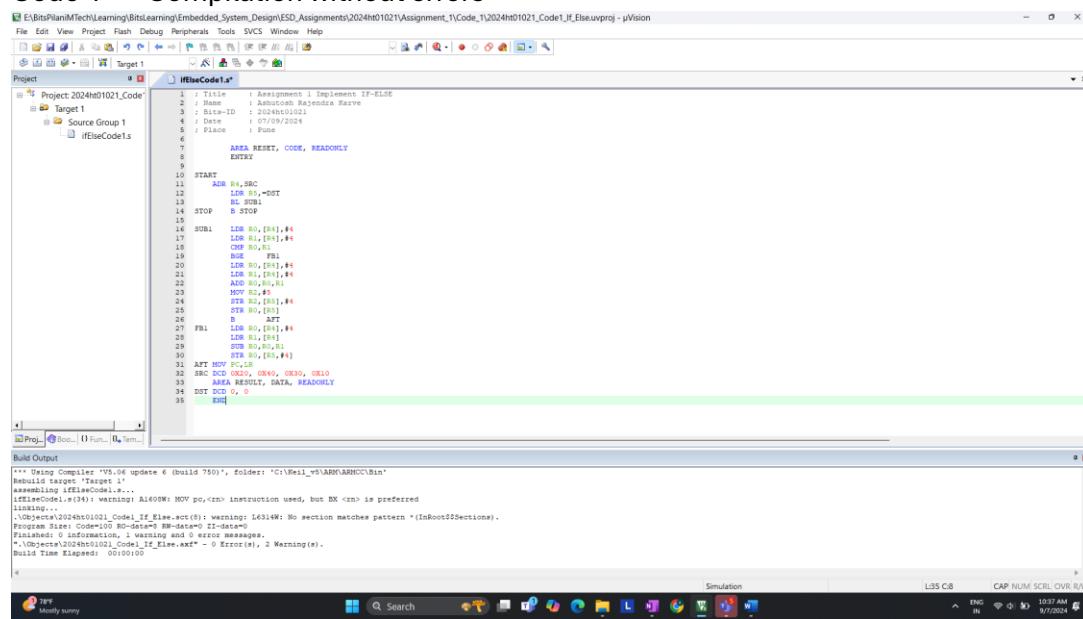
ESD Lab Assignment No.1

Name : Ashutosh Rajendra karve
Bits-ID : 2024ht01021
Mail : 2024ht01021@wilp.bits-pilani.ac.in
Contact : +91 9765541324
Data : 05/09/2024
Place : Pune, Maharashtra

Q.1. Assembly Language Program (ALP) for an LPC2378 processor to implement following IF-ELSE statement are given below:

```
If ( a<b )  
{  
    x=5;  
    y=c+d;  
}  
else  
    y=c-d
```

Code-1 >> Compilation without errors



The screenshot shows the µVision IDE interface with the assembly code for the provided IF-ELSE statement. The code uses R0 and R1 registers and includes instructions for loading immediate values, performing arithmetic operations (ADD, SUB), and branching (BNE, BZ). The assembly code is as follows:

```
1  ;Assignment 1 Implement IF-ELSE
2  ;Name : Ashutosh Rajendra Karve
3  ;Bits-ID : 2024ht01021
4  ;Date : 05/09/2024
5  ;Place : Pune, Maharashtra
6
7  AREA RESET, CODE, READONLY
8
9  ENTRY
10 START    ADR R4,SRC
11 LDR R0,*R0-#0ST
12 BZ    R0,0
13     BL SUB1
14 STOP    B STOP
15
16 SUB1    LDR R0,[R4],#4
17 LDR R1,[R4],#4
18 ADD R0,R0,R1
19 BNE    R0,R1,F1
20 LDR R0,[R1],#4
21 LDR R1,[R1],#4
22 ADD R0,R0,R1
23 BZ    R0,[R0],#4
24     STR R0,[R0],#4
25     STR R0,[R0],#4
26     STR R0,[R0],#4
27 F1      LDR R0,[R1],#4
28 LDR R1,[R1],#4
29     STR R0,[R1],#4
30     STR R0,[R1],#4
31 AFT NOV PCLR
32     STR R0,[R0],#4
33     STR R0,[R0],#4
34     STR R0,[R0],#4
35     STR R0,[R0],#4
36
37 AREA RESULT, DATA, READONLY
38 DST DCD 0, 0
39
40 END
```

The build output window at the bottom shows the compilation process and results:

```
*** Using Compiler 'V3.04 update 6 (build 750)', folder: 'C:\Keil_v5\ARM\GCC\Bin'.
Rebuild targets 'Target 1'.
assembling...
linking...
lcc2024ht01021_Code1_IF_Else.s(8): warning: 16059W: NOV ps,<rn> instruction used, but NX <rn> is preferred
linking...
lcc2024ht01021_Code1_IF_Else.s(8): warning: 16314W: No section matches pattern *(InRout02$Sections).
Program Size: Code=109 RO=0 Data=8 RW=0 Date=0 21-Nov-0
Finished 0 Information, 1 Warning and 0 Error messages.
+1 Object(s) (lcc2024ht01021_Code1_IF_Else.o) - 0 Error(s), 2 Warning(s).
Build Time Elapsed: 00:00:00
```

Code- 2 >> Compilation without error

The screenshot shows the µVision IDE interface. The assembly code window displays the following code:

```

1 ; TITLE           : Assignment 1 Implement and debug IF-ELSE Code 2
2 ; Name            : Abhijeet Rajendra Karve
3 ; File-ID         : 2024h01021
4 ; Date            : 07/09/2024
5 ; Place           : Pune
6
7 AREA RESET, CODE, READONLY
8
9 START          ADR R4, SRC
10             LDR R1,*R0
11             BL SUB1
12             B STOP
13
14             B STOP
15
16 SUB1    LDR R0,[R4],#4
17             LDR R1,[R4],#4
18             CMP R0,R1
19             LDR R1,[R4]
20             MOV R2,R1
21             STRLT R2,R0,R1
22             ADDLT R0,R0,R1
23             STR R0,[R4],#4
24
25
26 AFT MOV PC,LR
27 SRC ICD 0x00, 0x00, 0x00
28             AREA RESULT, DATA, READONLY
29 DST DCD 0, 0
30             END

```

The build output window shows the following message:

```

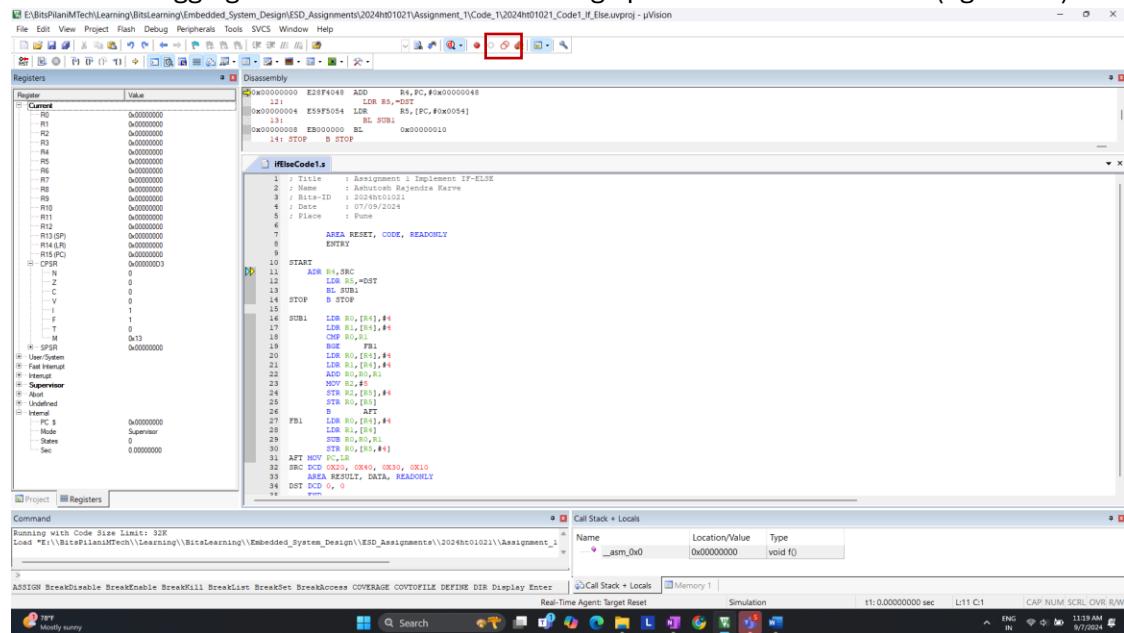
[Object]ctJ024ht01021_Code2.If_Else.sct(0): warning: 14314W: No section matches pattern *(InRoot)Sections.
Program Size:#00 00=0 00=0 00=0 II=0
Finished 0 Information, 1 warning and 0 error messages.
"..\obj\ctJ024ht01021_Code2.If_Else.axd" = 0 error(s), 2 Warning(s).
Build Time Elapsed: 00:00:02

```

a. On reset what is the LPC2378 processor's state and mode of operation?

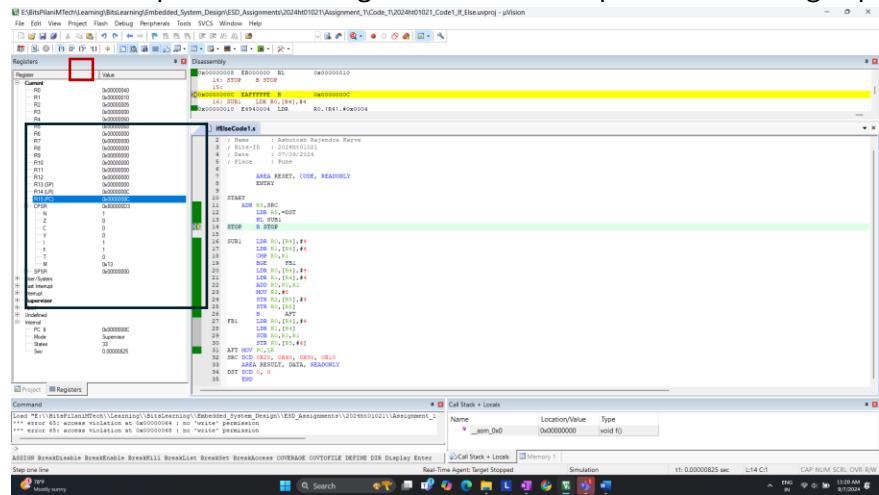
CODE 1 >>

Start of debugging for code 1 >> Click on debug option as shown in below (figure 1.1)



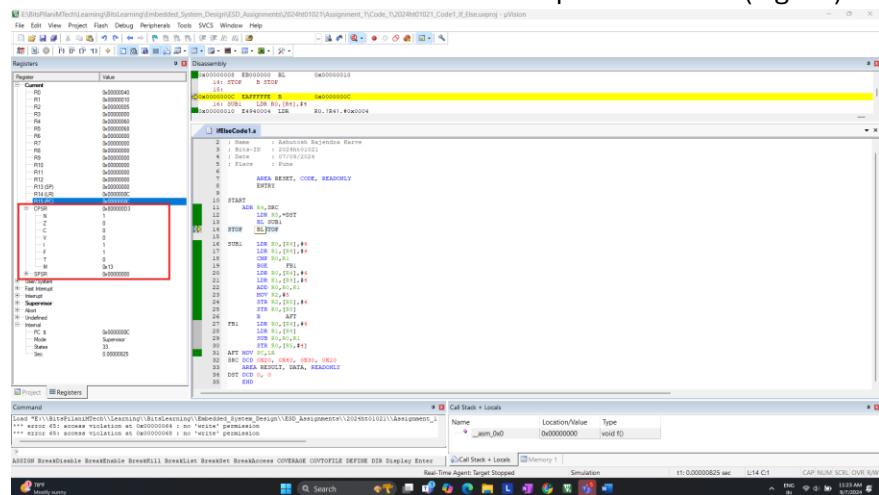
(Figure 1.1)

After >> Step >> Value of Registers have been updated according to program.

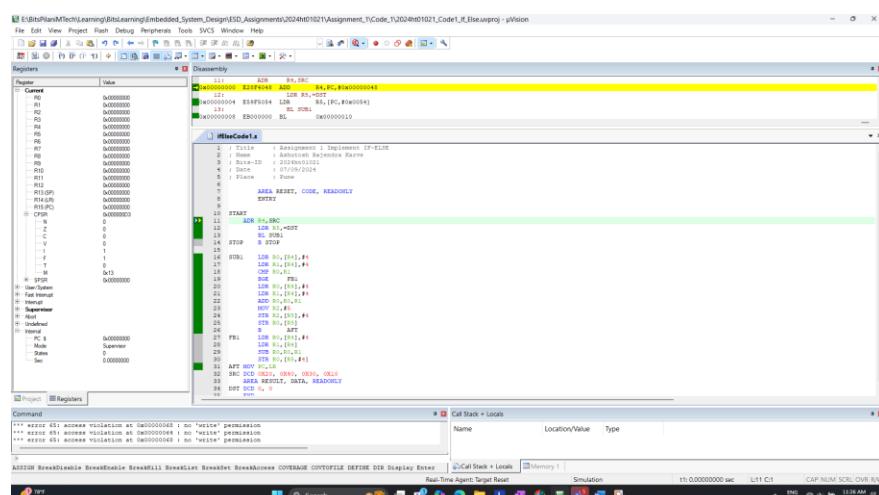


(Figure 1.2)

Observation of Code 1 >> Processor is in supervisor mode (Fig 1.3)



(Figure 1.3)



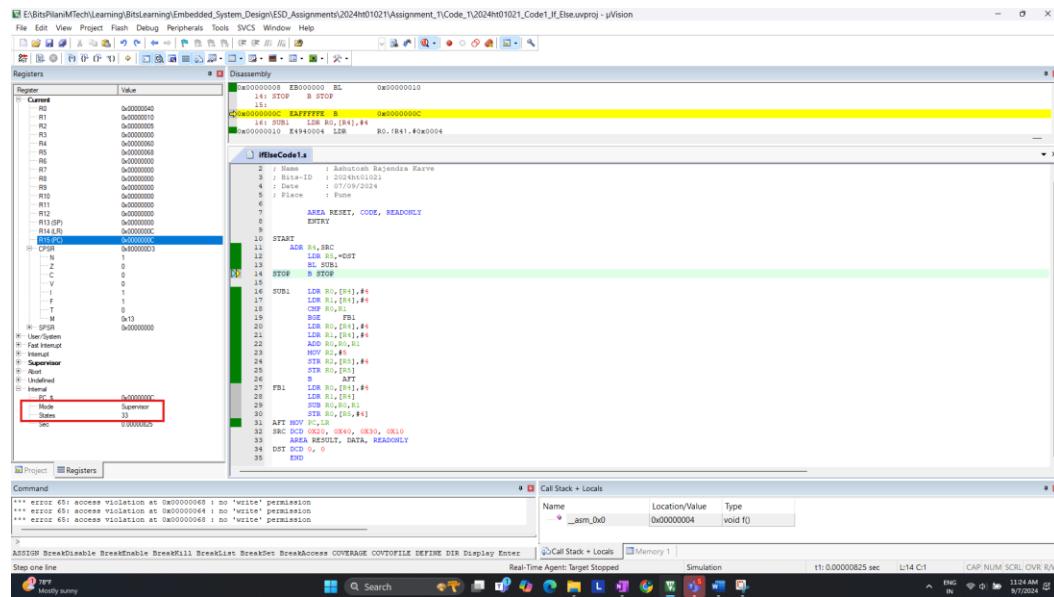
(Figure 1.4)

Answer:

- Looking at **CPSR**. You will see that the processor is in **Supervisor Mode** after reset and **0 states**. Check figure 1.4
- It takes a total of **33 states**. Check figure 1.3

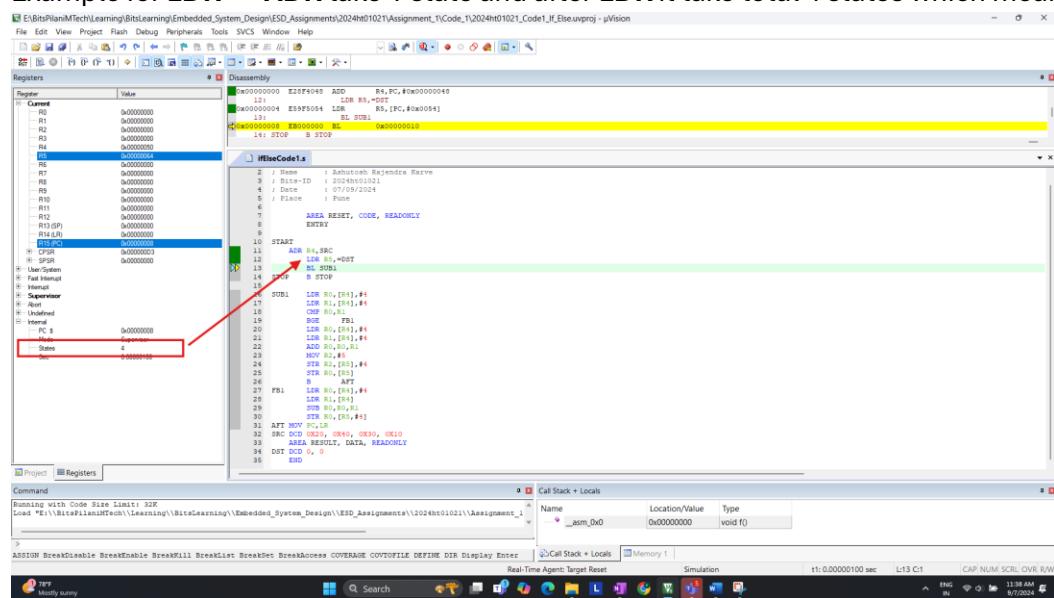
++++++END of Q.1 a)+++++

b. How many states are taken for the execution of an Arithmetic instruction, Load and Store instruction respectively (For Code-1



(Figure 1.5)

Example for **LDR >> ADR** take 1 state and after **LDR** it take total 4 states which means $4 - 1 = 3$



(Figure 1.6)

Answer:

- **Observing:** Load (**LDR**) , Store (**STR**) & Arithmetic (**ADD**)
 - **LDR** instruction takes **3 Cycles & 15 Total Cycles** for **LDR**
 - **STR** instruction takes **2 Cycles & 4 Total Cycles** for **STR**
 - **ADD** instruction takes **1 Cycles & 1 Total Cycles** for **ADD**
- Total Cycle takes place is **33 Cycles**. We can observe in the above screen shot.

++++++END of Q.1 b)+++++

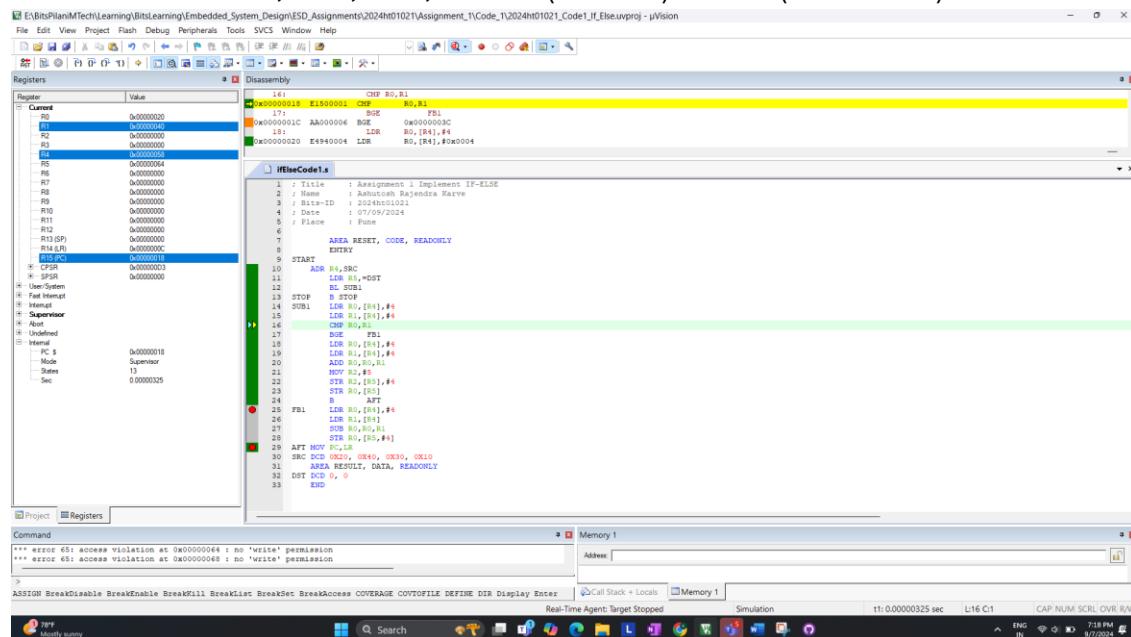
c. Are the number of states taken for completion the same for BGE instruction if the branch (1) is taken(2) not taken? Please give the states are taken for each. (For Code – 1)

We Need to determine the number of states for the BGE instruction when:

1. Branch is taken
2. Branch is not taken

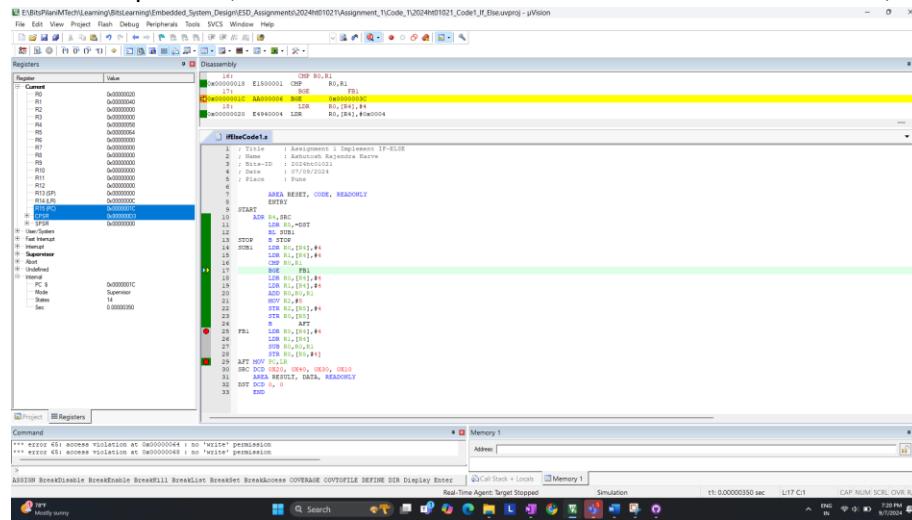
1. Branch Is Taken >>

code: SRC DCD 0x20, 0x40, 0x30, 0x10 (R0 < R1) (0x20 < 0x40)



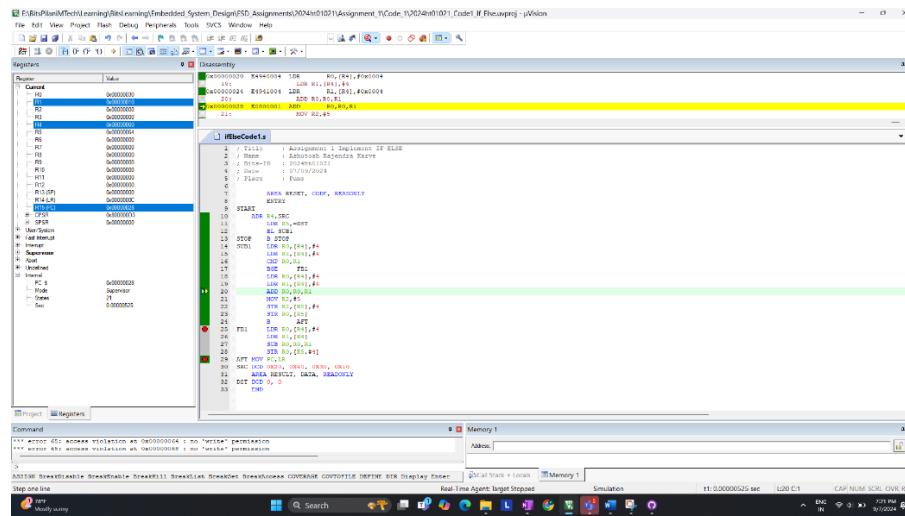
(Figure 1.7)

After comparison, for now R0 is loaded with 20 & R1 is Loaded with 40; States 13



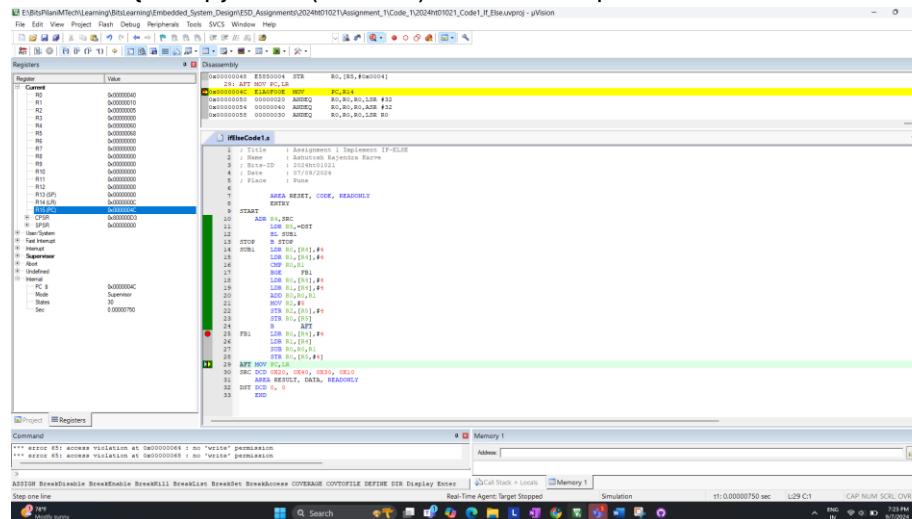
(Figure 1.8)

Now R0 and R1 is loaded with 30 & 10 after CMP we are in same block



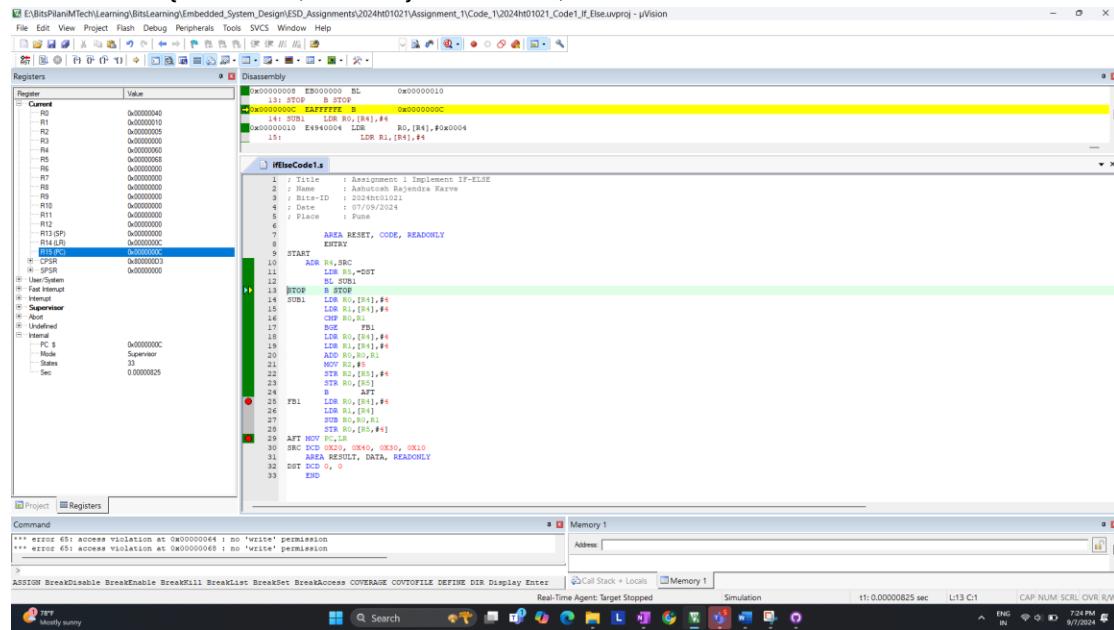
(Figure 1.9)

We are in {if loop} since ($R0 < R1$) new values update for R0 is 40 and R2 is 5



(Figure 2.0)

which is our {If Block X , Y Value} If is taken ; State : 30



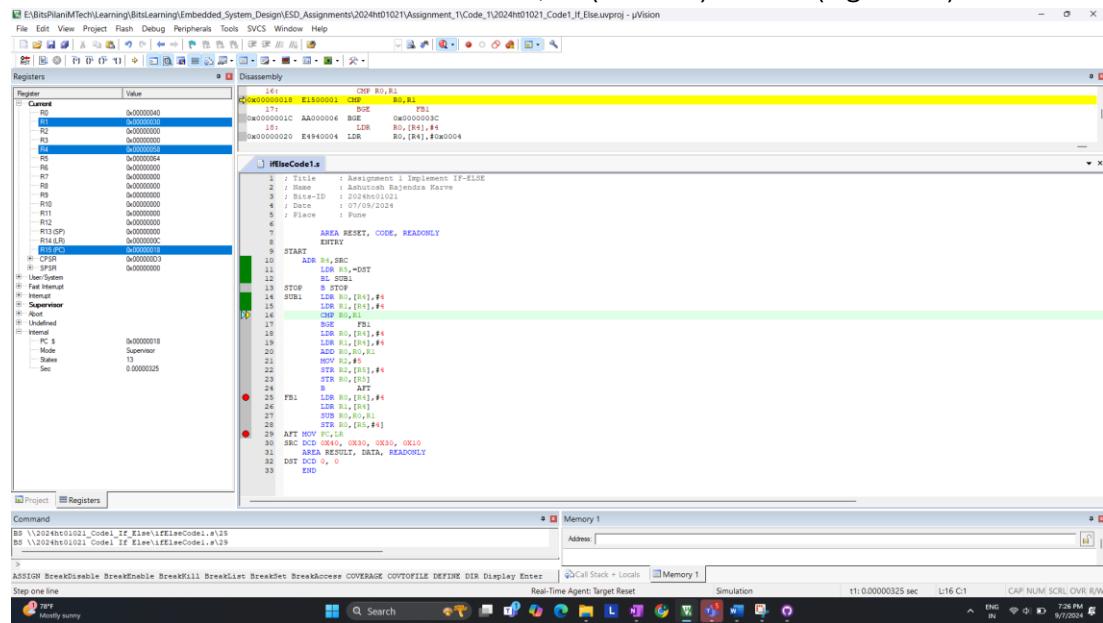
(Figure 2.1)

Answer: Final State with Branch is Taken is 33

2. Branch Is Not Taken:

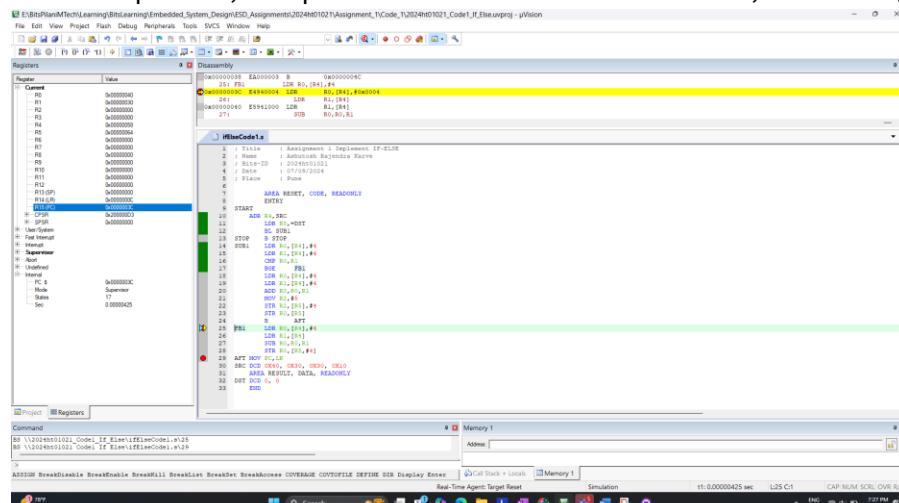
I. code: SRC DCD 0x40, 0x30, 0x30, 0x10 (R0 > R1)

For now value of R0 & R1 is loaded with 40,30 (40 > 30) check (Figure 2.2)



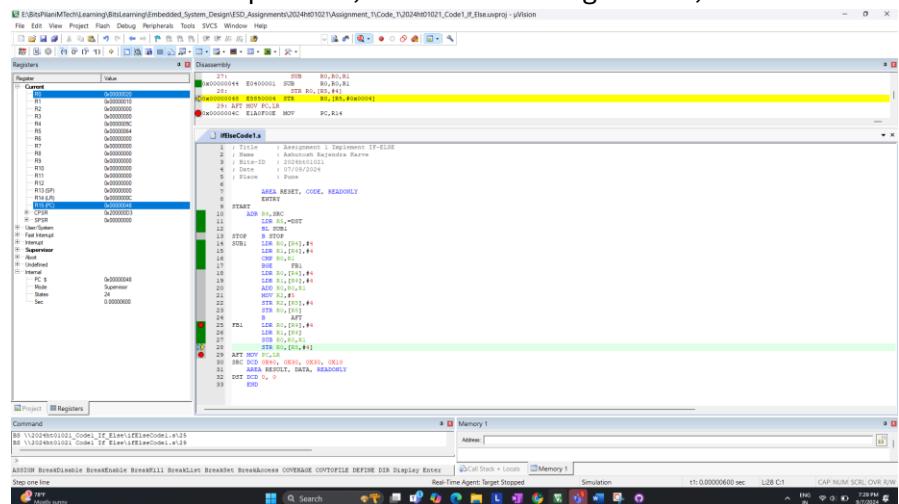
(Figure 2.2)

After comparison, it skip the IF-BLOCK enters ELSE-BLOCK; States 17 (check fig 2.3 line 25)

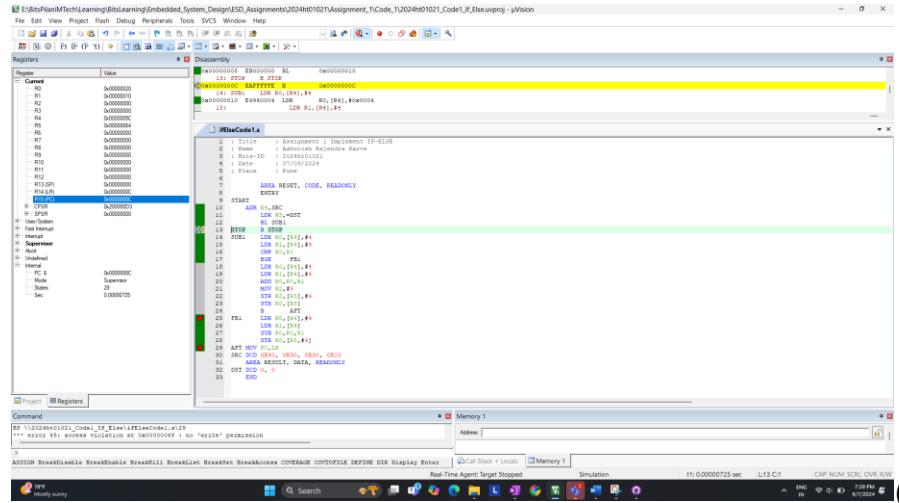


(Figure 2.3)

After Subtraction operation; Value of R0 Changes to 20; State 24



(Figure 2.4)

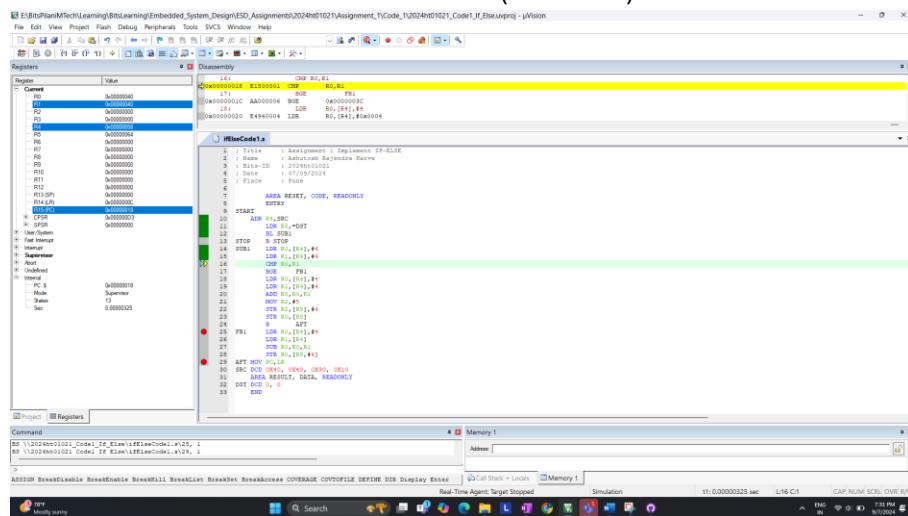


(Figure 2.5)

Answer: Final State with Branch is not Taken is 29. Check Figure 2.5

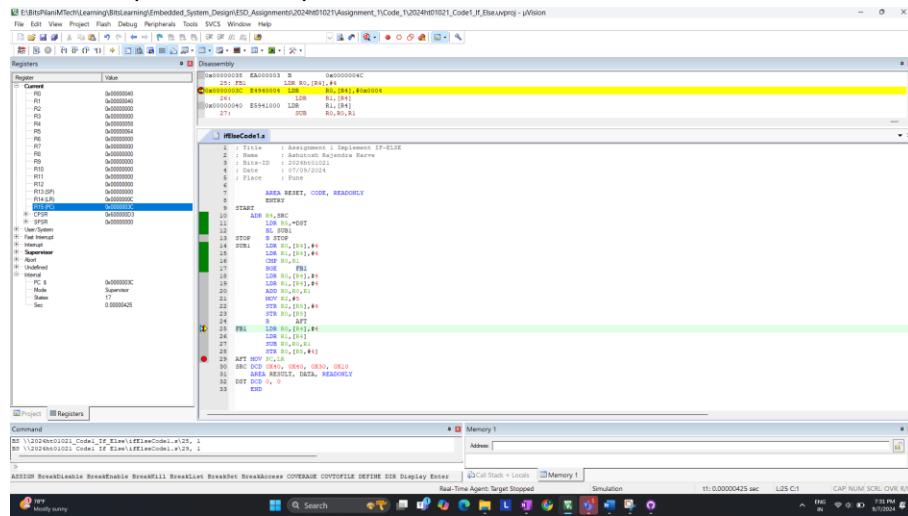
II. code: SRC DCD 0x40, 0x40, 0x30, 0x10 (R0 == R1)

For now R0 & R1 is loaded with 40 & 40 ($40 == 40$)

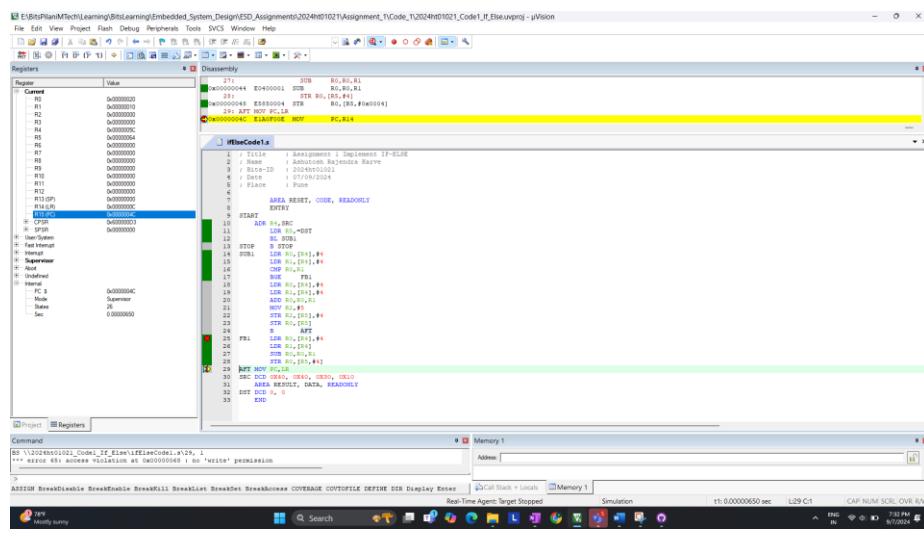


(Figure 2.6)

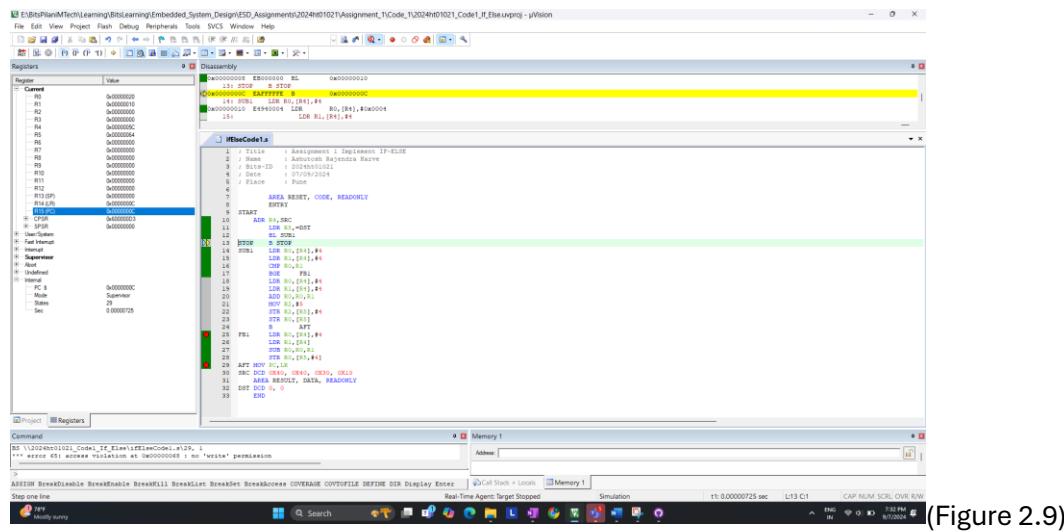
After comparison; it skip the IF-BLOCK enters ELSE-BLOCK; States 17



(Figure 2.7)



(Figure 2.8)



(Figure 2.9)

Final State with Branch is not Taken is 29

Answer:

- Branch Taken: The instruction takes 33 cycles when the branch is taken.
- Branch not Taken: The instruction takes 29 cycles when the branch is not taken.
- Number of states for completion is **not same** for BGE instruction

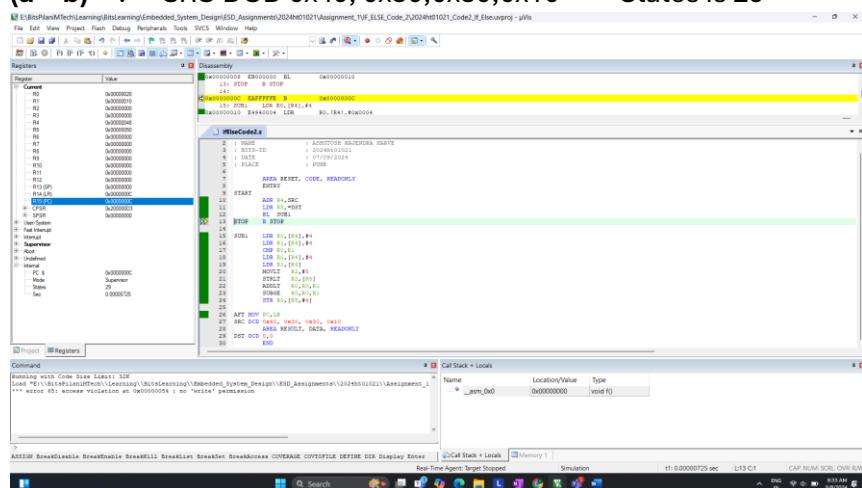
++++++END of Q.1 c)++++++

d. Measure the performance of code-1 and code-2 for the following conditions

Condition	Code-1-States	Code-2-States
a < b	33	30
a > b	29	29
a = b	29	29

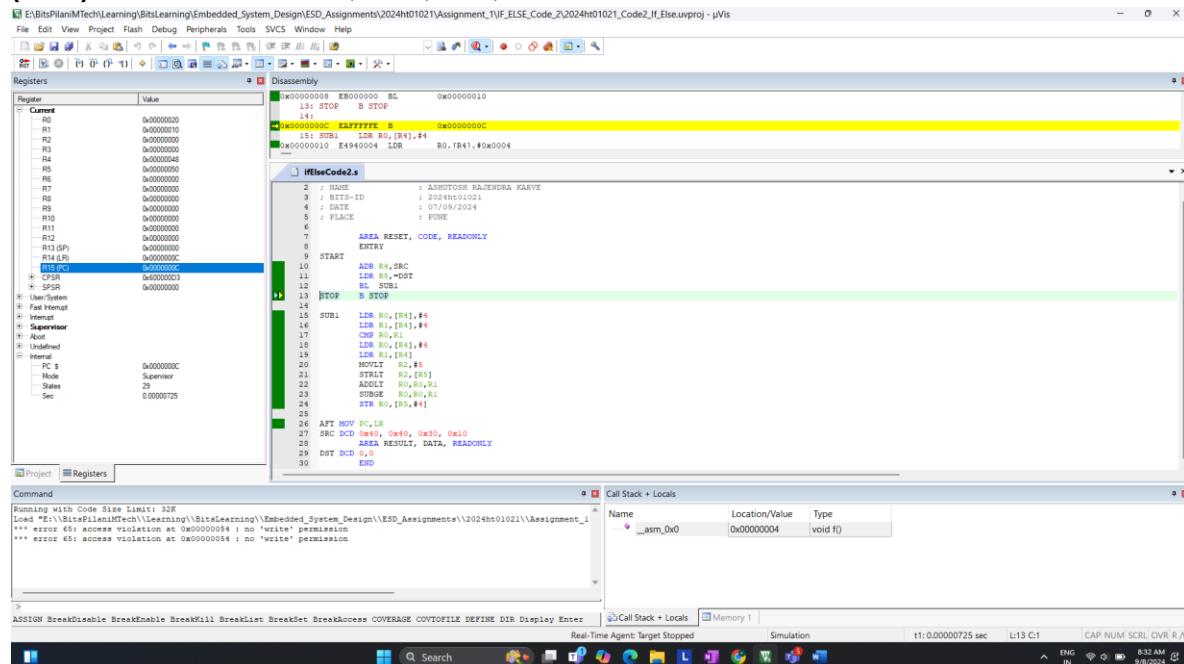
Screen Shots of Code-2-States

(a > b) : SRC DCD 0x40, 0x30,0x30,0x10 States is 29



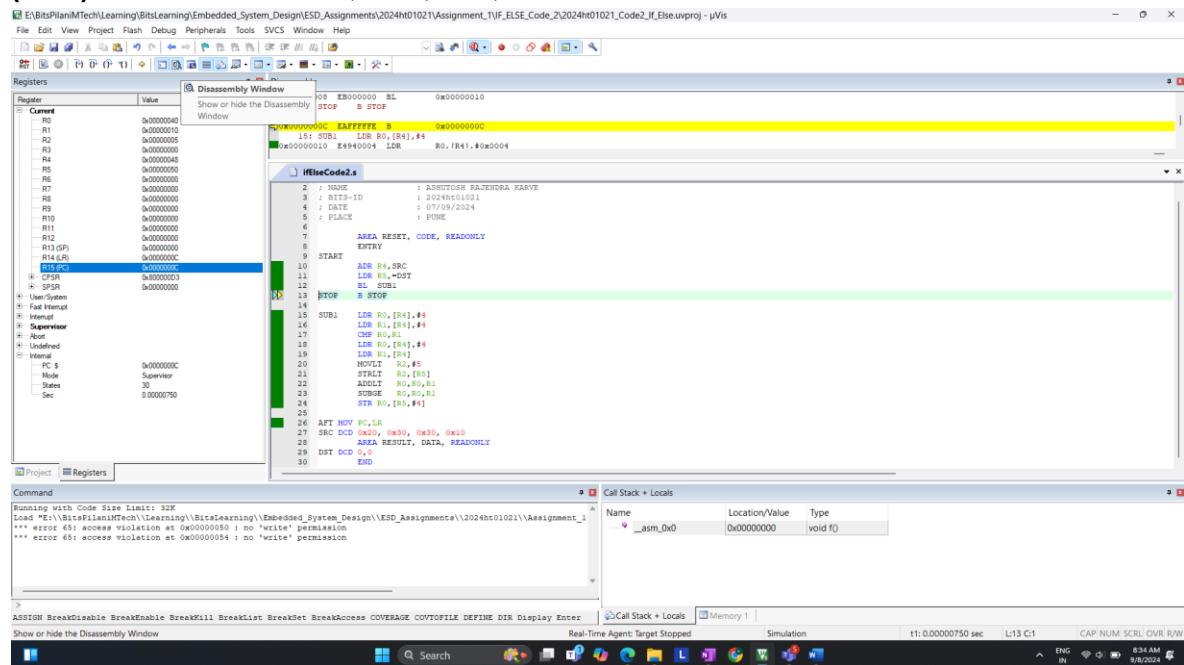
(Figure 3.0)

(a = b) : SRC DCD 0x40, 0x40,0x30,0x10 States is 29



(Figure 3.1)

(a < b) : SRC DCD 0x20, 0x30,0x30,0x10 States is 30



(Figure 3.2)

+++++END of Q.1 d)+++++

Q.2. Write an assembly language program for LPC2378 to count the number of 0's and 1's in the last 5 digits of your BITS ID (number must be given in decimal number format). Store the number of 0's in register R0 and the number of 1's in register R1. Verify your result by performing manual calculations.

STEP 1 : Code compilation with errors.

```

1 ; TITLE : Assignment_1 count the number of 0's and 1's in the last 5 digits of your BITS ID
2 ; NAME : ASHUTOSH RAJENDRA KAVNE
3 ; AUTHOR-ID : 2024ht01021
4 ; DATE : 07/09/2024
5 ; PLACE : PUNE
6
7 AREA RESET, CODE, READONLY
8
9 ENTRY
10 START
11 LDR R4, =SRC           ; Load the address of the BITS ID (01021)
12 LDR R5, =R5             ; Load the value (01021) into R5
13 LDR R6, =DST           ; Load the address of DST to store results
14 BL SUB1               ; Call the SUB1 function to count 0's and 1's
15 STOP      B STOP
16
17 SUB1
18    MOV R0, #0           ; Initialize R0 to store the count of 0's
19    MOV R1, #0           ; Initialize R1 to store the count of 1's
20    MOV R3, #10          ; Set loop counter for 10 bits (last 5 digits of the BITS ID)
21
22 LOOP
23    TST R3, #1            ; Test if the least significant bit is 1
24    BEQ COUNT_ZERO        ; If the bit is 0, branch to COUNT_ZERO
25    ADD R1, R1, #1         ; Increment R1 (count of 1's)
26    B NEXT_BIT            ; Jump to NEXT_BIT
27
28 COUNT_ZERO
29    ADD R0, R0, #1         ; Increment R0 (count of 0's)
30
31 NEXT_BIT
32    LSR R5, R5, #1         ; Logical shift right to check the next bit
33    SUBS R5, R5, #1        ; Decrement the bit counter
34    BNE LOOP              ; If R3 is not zero, repeat the loop
35
36    STR R0, [R4]           ; Store the count of 0's in DST[0]
37    STR R1, [R6, #4]        ; Store the count of 1's in DST[1]
38
39    BX LR                ; Return from the function
40
41 SRC    DCD 1021          ; Define the BITS ID value (01021) in decimal
42 DST    AREA RESULT, DATA, READWRITE
43 EQU
44

```

Build Output

```

.\obj\ConvertToBinary.exe(0): warning: 1614: No section matches pattern *(InRootS2Sections).
Project file size: Codesize 80-datas 12-dates 12-dates.
Finished: 0 Information, 1 warning and 0 error messages.
..\obj\ConvertToBinary.exe" - 0 Error(s), 1 Warning(s).
Build time Elapsed: 00:00:00

```

Count of 0's & 1's in memory after the execution; Check R0 & R1

Registers

Register	Value
C0	0x00000000
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000004
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13(BP)	0x00000000
R14(SP)	0x00000000
R15	0x00000000
R16	0x00000000
R17	0x00000000
R18	0x00000000
R19	0x00000000
R20	0x00000000
R21	0x00000000
R22	0x00000000
R23	0x00000000
R24	0x00000000
R25	0x00000000
R26	0x00000000
R27	0x00000000
R28	0x00000000
R29	0x00000000
R30	0x00000000
R31	0x00000000
SPSR	0x00000003

Call Stack + Locals

Name	Location/Value	Type
_asm_0x0	0x00000000	void f()

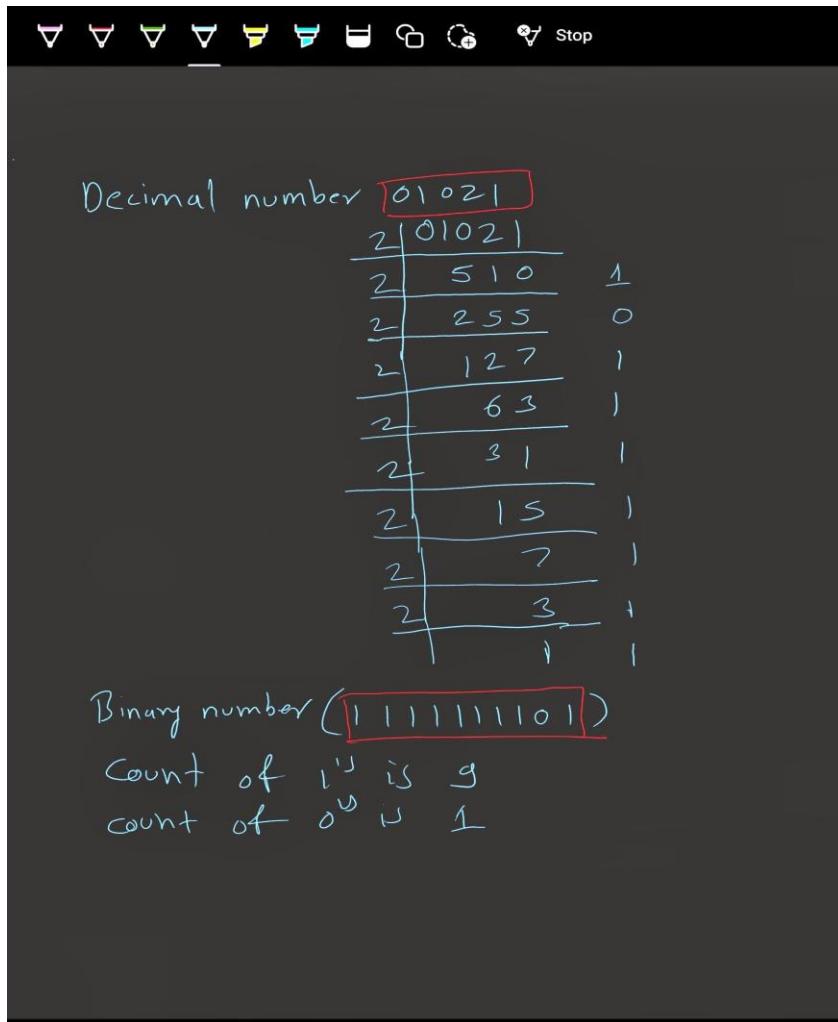
Verification:

1. Check the register value is updated

- R0: This register will contain the count of 0's
- R1: This register will contain the count of 1's

2. Manual Calculation

- BITS ID: **01021**
- Binary equivalent of the last 5 digits (1021): **1111111101**
- Count of 1's: 9
- Count of 0's: 1



++++++END of Q.2)++++++

Q.3. Write an assembly language program for ARM Cortex M3/4 to handle Supervisor Call (SVC) exception.

Requirements:

1. The SVC should be called from an application task running at **Thread unprivileged mode**. The SVC number is derived from the last two digits of my BITS ID, i.e., **21**.
2. Two parameters, representing the last five digits of my BITS ID (**01021**), will be passed to the **SVC handler**.
3. The SVC number will be dynamically determined by examining the SVC instruction in the memory.
4. The SVC exception handler should set up a stack pointer in SVC mode before handling the exception.
5. If the SVC number matches my BITS ID (21), the handler will perform the addition of the two parameters.
6. Otherwise, it will perform subtraction.

Assembly Code: Screen shot without errors

The screenshot shows the µVision IDE interface with the project 'SVC_Condition' open. The assembly code for 'SVCcompare.s' is displayed in the main window. The code implements a Reset Handler that performs an SVC call with number 21, pushing parameters onto the stack, and then branches to the SVC_handler. The assembly code is as follows:

```
1 ; TITLE : Assignment Solution : Q.3
2 ; NAME : AARUSH RAJENDRA KARVE
3 ; BITS_ID : 3024ht01021
4 ; DATE : 08/05/2024
5 ; PLACE : PUNE
6
7     AREA RESET, CODE, READONLY
8     EXPORT __Vectors
9     __Vectors
10    DCD 0x20001000, Reset_Handler+1,0,0,0,0,0,0,0,SVC_handler+1,0,0,0,0,0,0,0,0
11
12    AREA mycode, CODE, READONLY
13    ENTRY
14    EXPORT Reset_Handler
15
16 Reset_Handler
17     LDR R1, =0x20000200
18     MSR PSP, R1
19
20     MOV R0, #3
21     MSR CONTROL, R0
22     LDR R7, =$RC
23     LDR R1, [R7]          ; Load first parameter (I021)
24     LDR R2, [R7, #4]      ; Load second parameter (I021)
25     LDR R6, =DST
26
27     SVC 21              ; SVC call with number 21
28 STOP  B STOP
29
30
31 SVC_handler
32     PUSH {lr}
33     TST LR, #4
34     ITE EQ
35     MRSEQ R0, MSP
```

The bottom left pane shows the build output, which includes the warning about no section matches pattern and the build time elapsed.

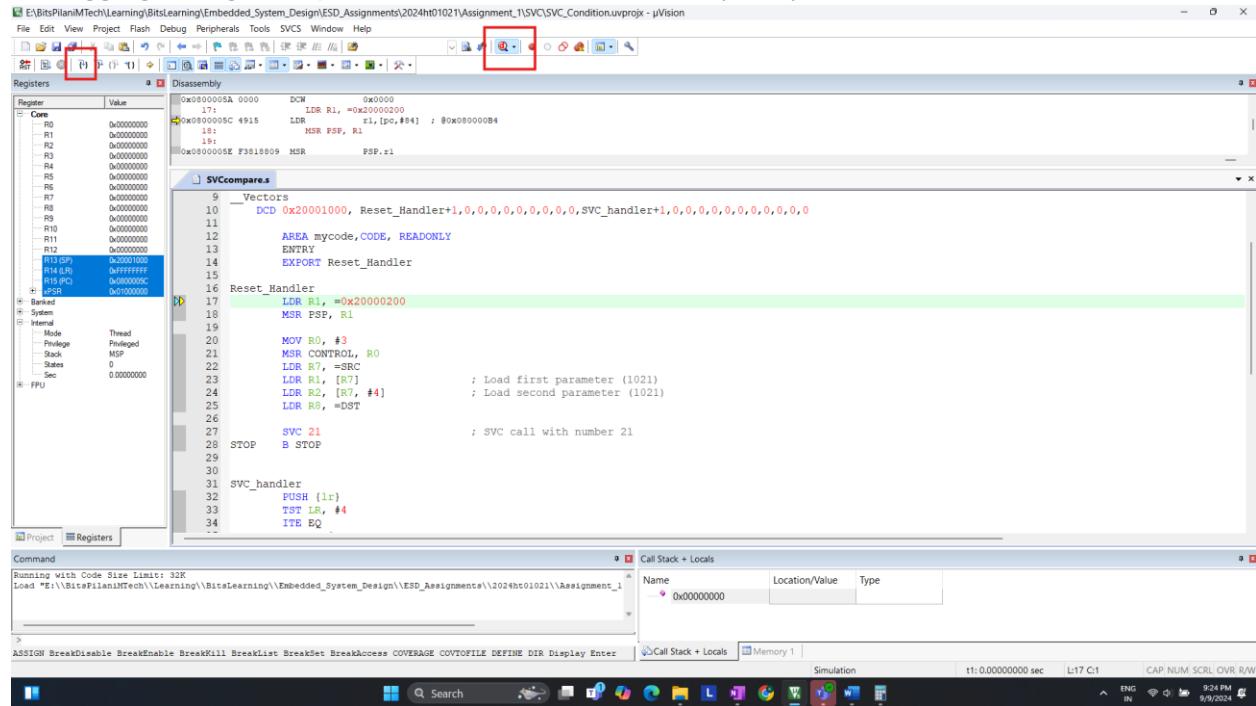
```
.\Objects\SVC_Condition.sct(8): warning: Ld314W: No section matches pattern *(InRoot@2$Sections).
Program Size: Code=192 RO-data=0 RW-data=0 ZI-data=0
Finished 0 information, 1 warning and 0 error messages.
.\Objects\SVC_Condition.axf" - 0 Error(s), 2 Warning(s).
Build Time Elapsed: 00:00:00
```

The screenshot shows the pVision software interface with the following details:

- Title Bar:** E:\BitsPilani\Tech\Learning\BITSLearning\Embedded_System_Design\ESD_Assignments\2024ht01021\Assignment_1\SVC_SVC_Condition.uvproj - pVision
- Menu Bar:** File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, Help
- Toolbar:** Includes icons for Open, Save, Build, Run, Stop, and others.
- Project Explorer:** Shows "Project SVC_Condition" with "Target 1" selected. Under Target 1, there is a "Source Group 1" containing "SVCcompares.s".
- Code Editor:** Displays the assembly code for the SVC handler. The code handles different SVC numbers (1, 2, 3) by performing addition or subtraction on registers R0, R1, and R2. It also handles a default case for other SVC numbers.
- Build Output:** Shows the build results for the project.
- Status Bar:** Shows simulation status (L63 C1), memory usage (CAP NUM SCR LVR RAW), and system information (ENG IN, 9/9/2023).

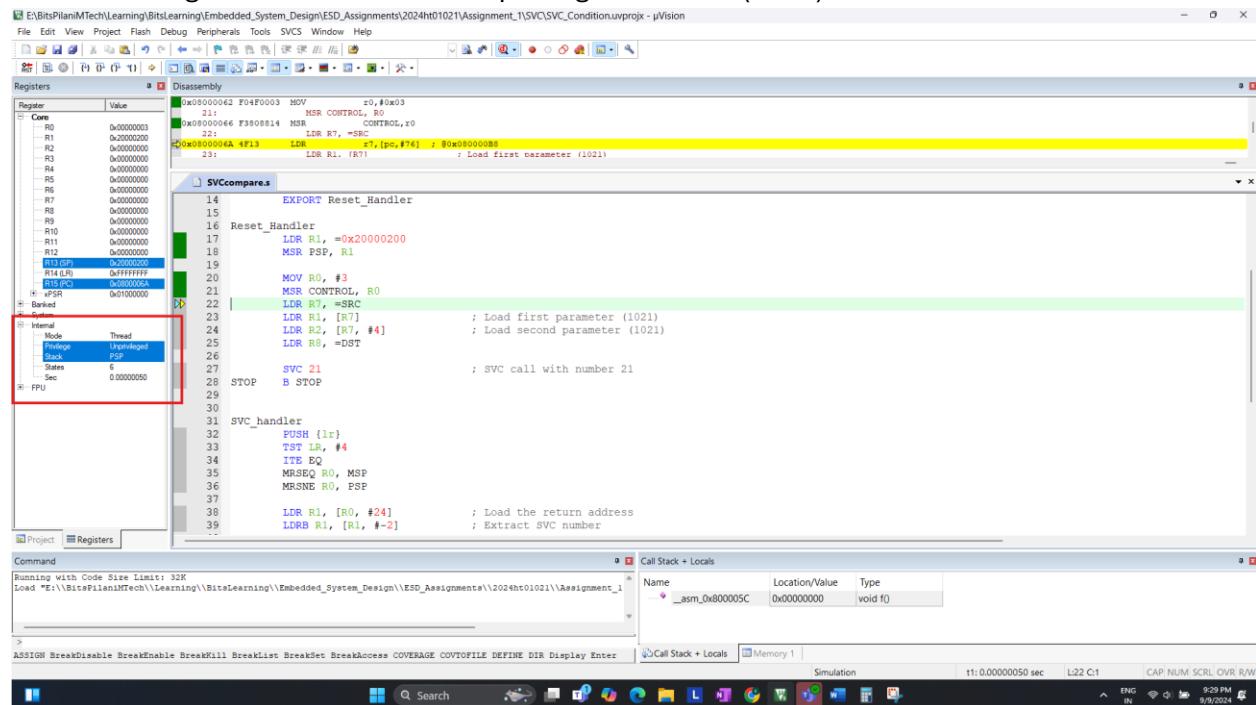
```
29
30
31 SVC_handler
32     PUSH {lr}
33     TST LR, #4
34     ITE EQ
35     MRSQ R0, MSP
36     MRSNE R0, PSP
37
38     LDR R1, [R0, #24]          ; Load the return address
39     LDRB R1, [R1, #-2]         ; Extract SVC number
40
41     CMP R1, #21               ; Compare SVC number with 21
42     BNE Subtract             ; If not 21, go to Subtract
43
44     ; If SVC number matches, perform addition
45     ADD R0, R1, R2            ; R0 = R1 + R2
46     STR R0, [R8]               ; Store result in DST
47     B EndSVC                 ; End of SVC handler
48
49 Subtract
50     ; If SVC number does not match, perform subtraction
51     SUB R0, R1, R2            ; R0 = R1 - R2
52     STR R0, [R8]               ; Store result in DST
53
54 EndSVC
55     POP {pc}
56
57 ; Data Section
58 SRC DCD 1021, 1021          ; Parameters (last 5 digits of BITS ID)
59 DST DCD 0                     ; Destination to store the result
60
61     AREA RES, DATA, READWRITE
62     END
63 |
```

Debugging: Using of Step one line to show execution >> (FIG 1)



(FIGURE 1)

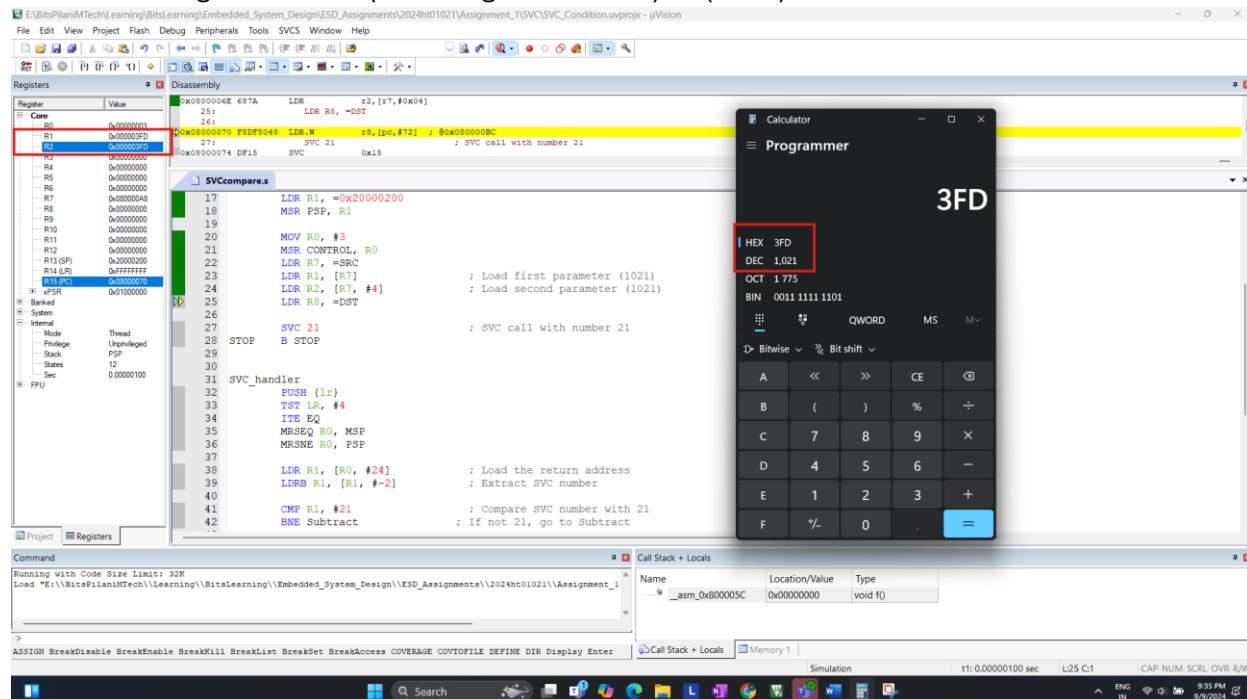
STEP 1: Setting Processor to Thread Unprivileged Mode >> (FIG 2)



(FIGURE 2)

We need to ensure that the processor is running in **Thread unprivileged mode**. This is achieved by writing to the **control** register to change the mode

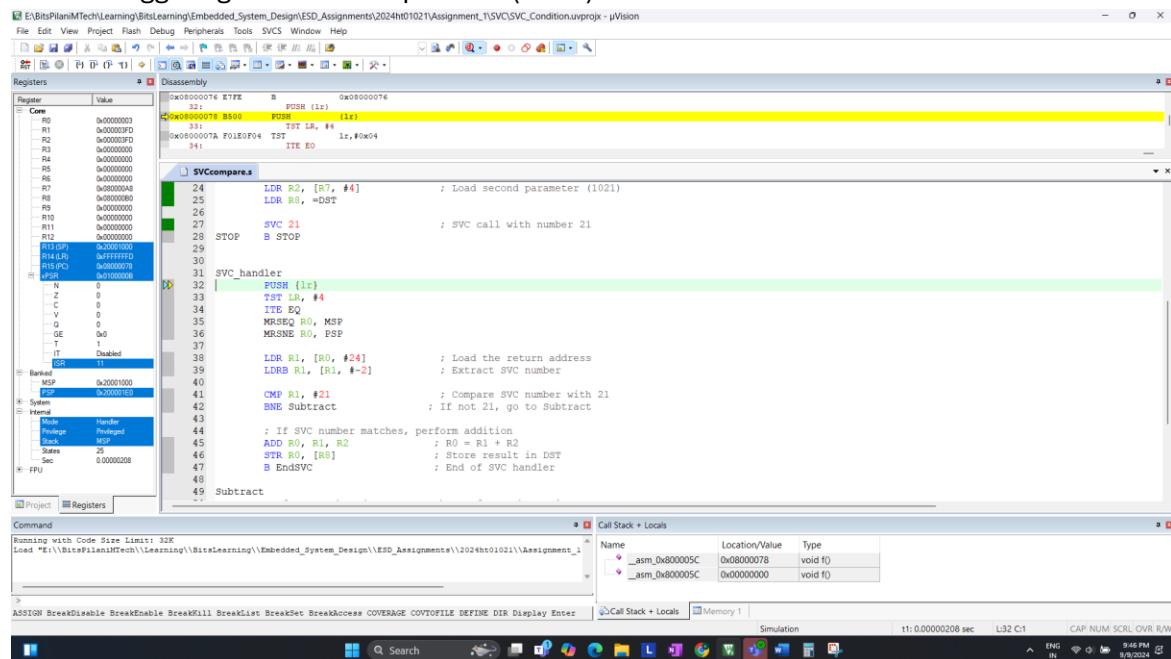
STEP 2: Loading Parameters (Last 5 digits of BITS ID) >> (FIG 3)



(FIGURE 3)

We load two parameters from memory, both of which are derived from the last 5 digits of my BITS ID (**1021**). These will be used in the SVC handler.

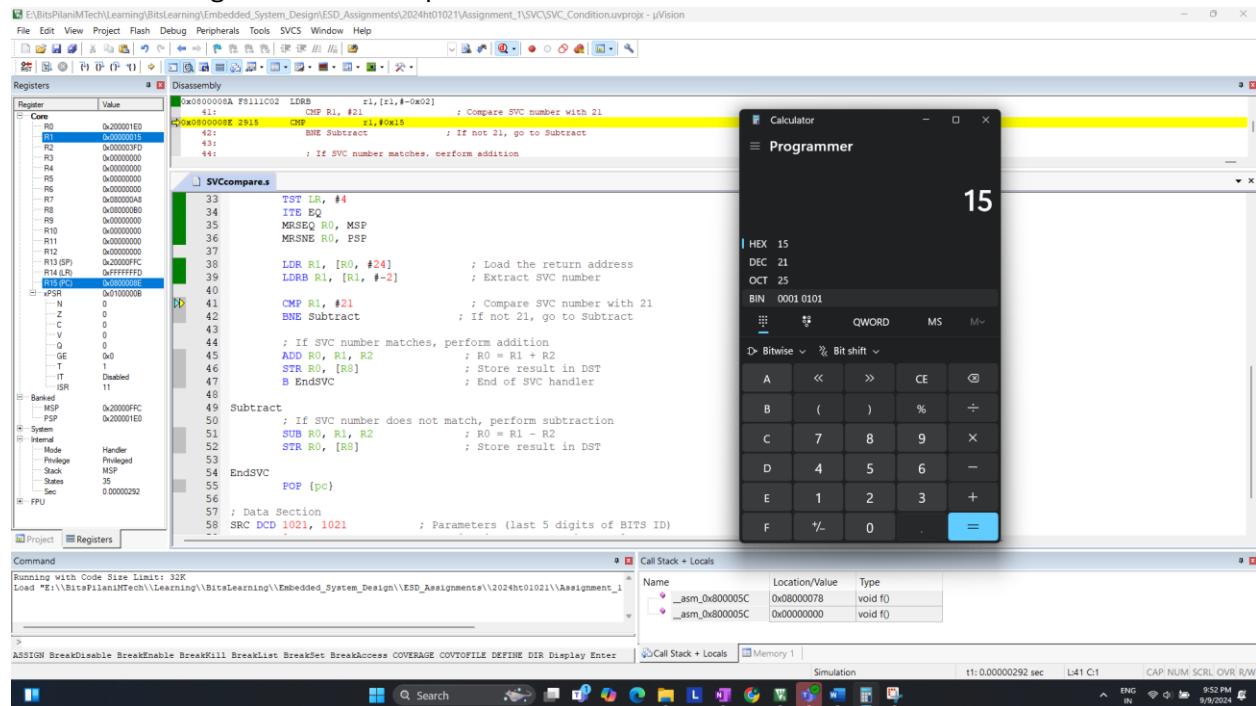
STEP 3: Triggering the SVC Exception >> (FIG 4)



(FIGURE 4)

We trigger the **SVC** exception using **SVC instruction**, passing **SVC Number**

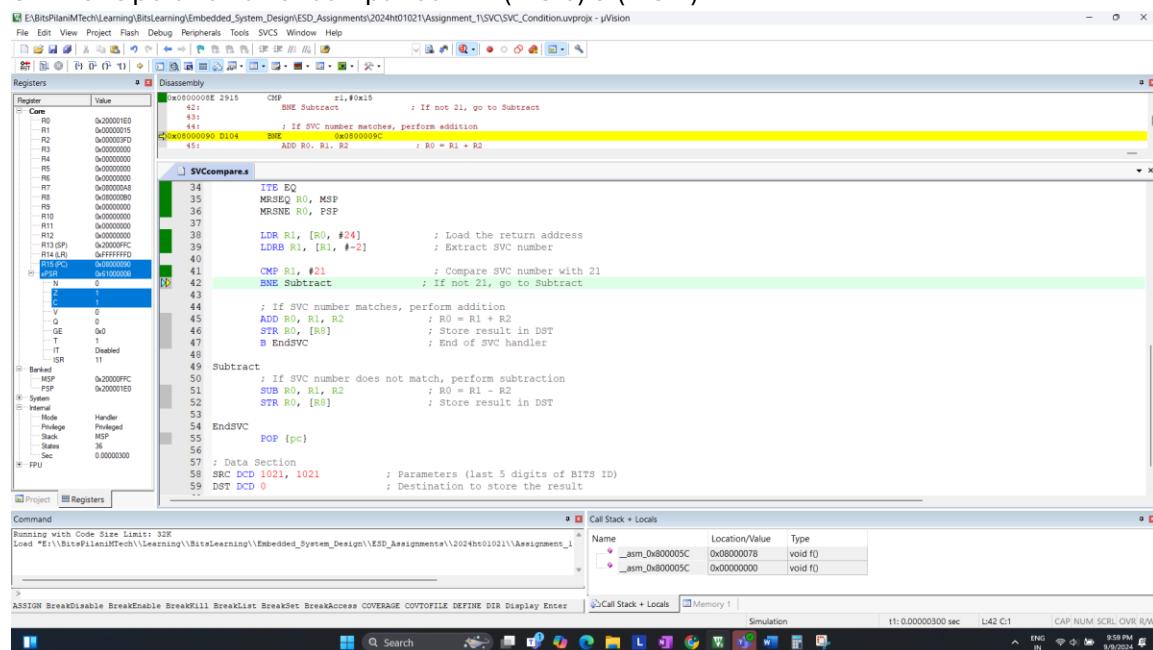
STEP 4: Handling SVC in the Exception Handler



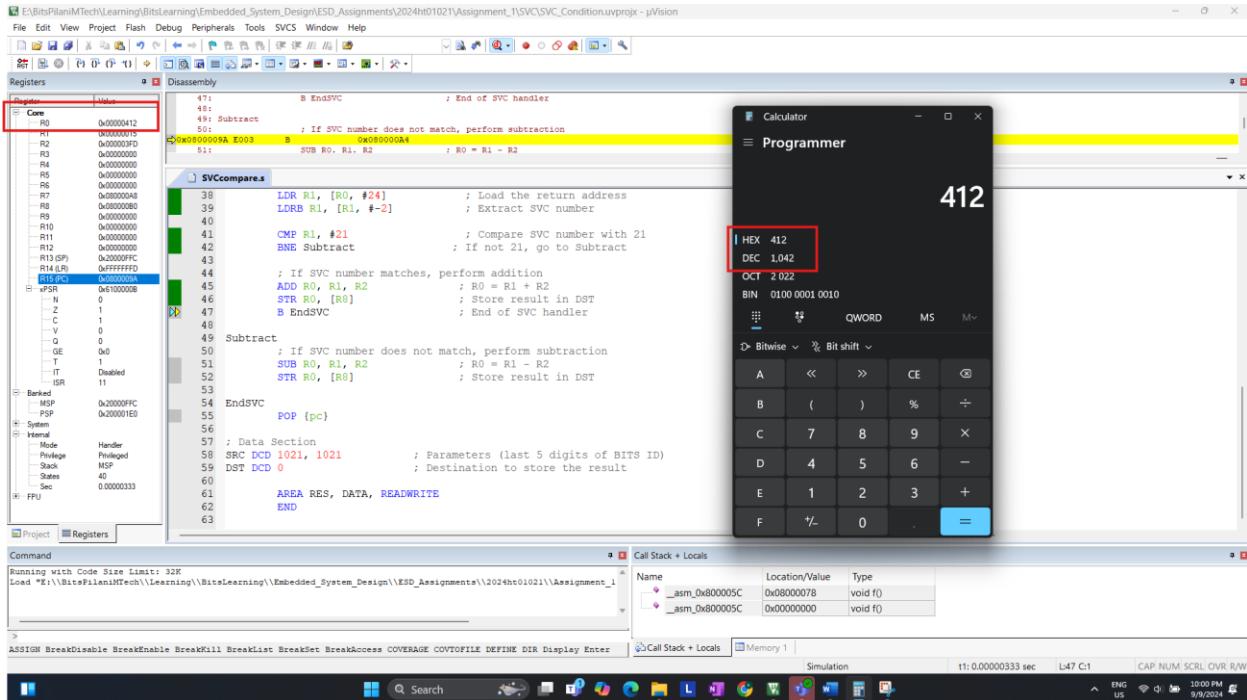
(FIGURE 5)

Inside the SVC handler, the program checks whether the SVC number matches 21. We can see R1 is loaded with 21 which will match and perform the **Addition** operation.

STEP 5: Operation after comparison >> (FIG 6) & (FIG 7)



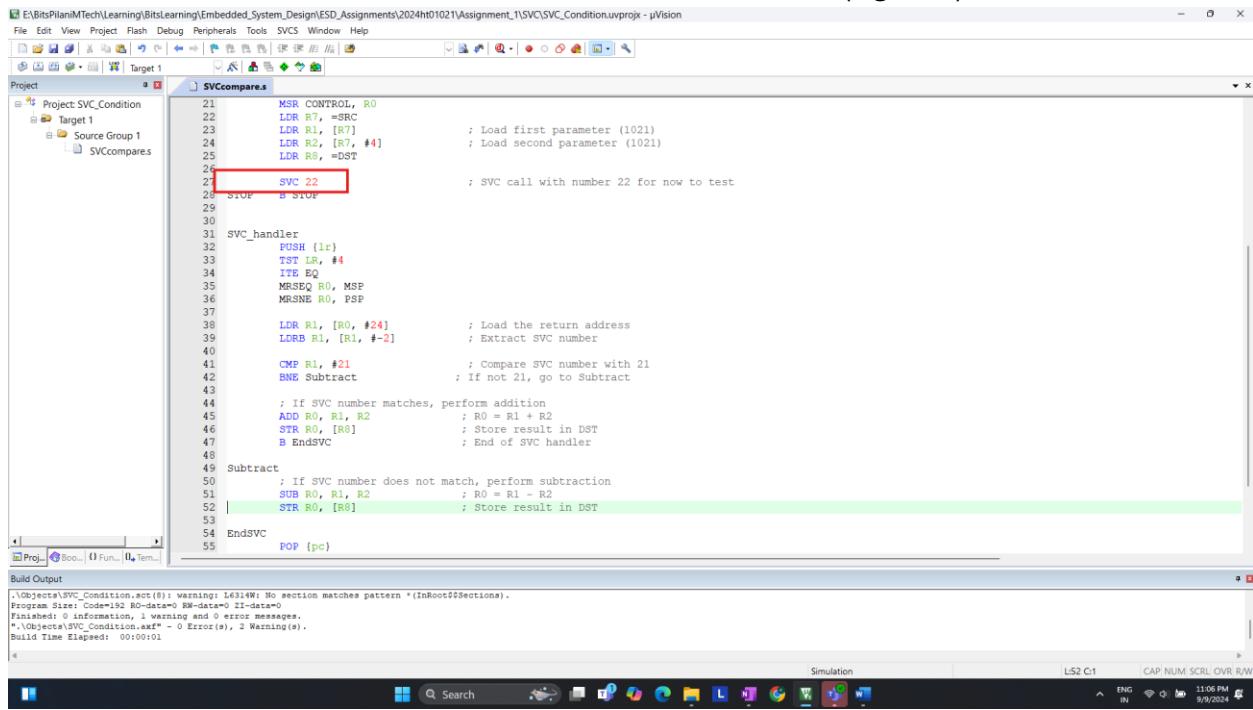
(FIGURE 6)



(FIGURE 7)

After executing the logic to extract the SVC number, SVC number (21) is compared, and the addition is performed because the SVC number matches.

ADDITION STEP TO VERIFY STEP 6: Test with different SVC number (Fig 8 & 9)



(FIGURE 8)

```

Registers          Disassembly
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
E:\BitsPilani\MTech\Learning\BitsLearning\Embedded_System_Design\ESD_Assignments\2024ht01021\Assignment_1\SVC\SVC_Condition.uvproj - µVision
R0 0xFFFFEC18 R1 0x00000016 R2 0x0000001D R3 0x000000FD R4 0x00000000 R5 0x00000000
R6 0x00000000 R7 0x00000048 R8 0x00000000 R9 0x00000000 R10 0x00000000 R11 0x00000000
R12 0x00000000 R13 (SP) 0x200000FC R14 (LR) 0xFFFFFFF7 R15 (PC) 0x000000A0
PSR 0x00000000
Banked System Internal
Mode Handler Privileged Stack MSP StackSize 40 Sec 0.00000000 FPU
Project Registers Command
Running with Code Size Limit: 32K Load E:\BitsPilani\MTech\Learning\BitsLearning\Embedded_System_Design\ESD_Assignments\2024ht01021\Assignment_1\SVC\SVC_Condition.uvproj - µVision
SVCCompare.s
30:    SUB R0, [R1], R2           ; Store result in DST
31:    STR R0, [R8]               ; Store result in DST
32:    B EndSVC
33:    LDR R1, [R0, #24]          ; Load the return address
34:    LDRB R1, [R1, #-2]          ; Extract SVC number
35:    CMP R1, #21                ; Compare SVC number with 21
36:    BNE Subtract              ; If not 21, go to Subtract
37:    ADD R0, R1, R2             ; R0 = R1 + R2
38:    STR R0, [R8]               ; Store result in DST
39:    B EndSVC
40:    CMP R1, #21                ; If SVC number matches, perform addition
41:    BNE Subtract              ; If not 21, go to Subtract
42:    BNE Subtract              ; If not 21, go to Subtract
43:    ADD R0, R1, R2             ; R0 = R1 + R2
44:    STR R0, [R8]               ; Store result in DST
45:    B EndSVC
46:    CMP R1, #21                ; If SVC number matches, perform subtraction
47:    BNE Subtract              ; If not 21, go to Subtract
48:    Subtract                 ; SVC number does not match, perform subtraction
49:    SUB R0, R1, R2             ; R0 = R1 - R2
50:    STR R0, [R8]               ; Store result in DST
51:    B EndSVC
52:    POP {pc}
53:    LDR R0, [R8]               ; Data Section
54:    ADD R0, R1, R2             ; Parameters (last 5 digits of BITS ID)
55:    STR R0, [R8]               ; Destination to store the result
56:    B EndSVC
57:    AREA RES, DATA, READWRITE
58:    END
59: DCD 1021, 1021
60: DCD 0
61: END
62: END
63:
Call Stack + Locals
Name Location/Value Type
asm_0x0800005C 0x08000078 void f0
asm_0x0800005C 0x00000000
Memory 1 Simulation t: 0.00000333 sec L52 C1 CAP NUM SCR OVR R/W
ENG IN 11:07 PM 9/8/2024

```

(FIGURE 9)

In Figure 9 we can observe that it is in subtract block, so logic is executed correctly.

STEP 7: Viewing Final Result

```

Registers          Disassembly
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
E:\BitsPilani\MTech\Learning\BitsLearning\Embedded_System_Design\ESD_Assignments\2024ht01021\Assignment_1\SVC\SVC_Condition.uvproj - µVision
R0 0x00000003 R1 0x0000000D R2 0x00000000 R3 0x00000000 R4 0x00000000
R5 0x00000000 R6 0x00000000 R7 0x00000048 R8 0x00000000 R9 0x00000000
R10 0x00000000 R11 0x00000000 R12 0x00000000 R13 (SP) 0x200000F0
R14 (LR) 0xFFFFFFFF R15 (PC) 0x000000A0
PSR 0x00000000
Banked System Internal
Mode Handler Unprivileged Stack MSP StackSize 20 Sec 55 FPU
Project Registers Command
Running with Code Size Limit: 32K Load E:\BitsPilani\MTech\Learning\BitsLearning\Embedded_System_Design\ESD_Assignments\2024ht01021\Assignment_1\SVC\SVC_Condition.uvproj - µVision
SVCCompare.s
28: STOP B STOP
29:
30: 31: SVC_handler
32: PUSH {lr}
33: LDR R1, #0x200000200
34: MSR PSP, R1
35: MOV R0, #3
36: MSR CONTROL, R0
37: LDR R7, =SRC
38: LDR R1, [R7]
39: LDR R2, [R7, #4]
40: LDR R6, =DST
41: LDR R5, =R0
42: SVC 21
43: B STOP
44: STOP B STOP
45:
46: SVC_handler
47: PUSH {lr}
48: TST LR, #4
49: ITE EQ
50: MSR EQ, MSP
51: MSRNE R0, PSP
52:
53: LDR R1, [R0, #24]
54: LDRB R1, [R1, #-2]
55: CMP R1, #21
56: B EndSVC
57: LDR R0, [R8] ; Load the return address
58: ADD R0, R1, R2 ; Extract SVC number
59: STR R0, [R8] ; Store result in DST
60: B EndSVC
61: AREA RES, DATA, READWRITE
62: END
63:
Call Stack + Locals
Name Location/Value Type
0x00000000
Memory 1 Simulation t: 0.00000458 sec L28 C1 CAP NUM SCR OVR R/W
ENG IN 11:33 PM 9/8/2024

```

(FIGURE 10)

After completion the execution of the SVC handler, Final result stored in memory

+++++END of Q.3+++++