ESZG512 (S1-25)

Lab Assignment - 1

Embedded System Design Lab

PALAASH ATRI
(2025NS01017)

Lab Assignment-1

Q.1 Assembly Language Programs (ALP) for an ARM7TDMI processor to implement following IF-ELSE statement are given below:

```
if ( a < b )
{
 x = 5
 y = c + d
}
else {
 y = c – d
}
```

Simulate the given Code-1 and Code-2 using Keil uVision5 software for ARM7TDMI processor or LPC2378 microcontroller and answer the following questions.

Simulation:

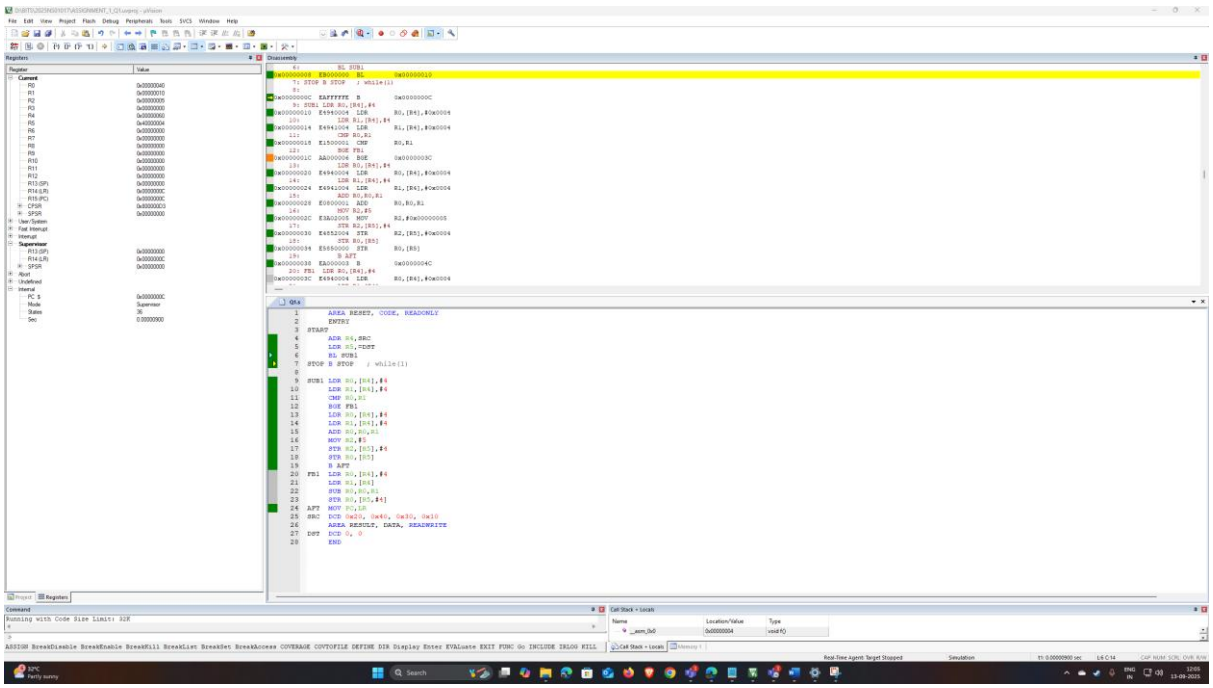
Step 1: Execution of Code 1

```
      AREA RESET, CODE, READONLY
      ENTRY
START
      ADR R4,SRC
      LDR R5,=DST
      BL SUB1
STOP B STOP   ; while(1)

SUB1 LDR R0,[R4],#4
      LDR R1,[R4],#4
      CMP R0,R1
      BGE FB1
      LDR R0,[R4],#4
```

```
        LDR R1,[R4],#4
        ADD R0,R0,R1
        MOV R2,#5
        STR R2,[R5],#4
        STR R0,[R5]
        B AFT
FB1   LDR R0,[R4],#4
        LDR R1,[R4]
        SUB R0,R0,R1
        STR R0,[R5,#4]
AFT   MOV PC,LR
SRC   DCD 0x20, 0x40, 0x30, 0x10
        AREA RESULT, DATA, READWRITE
DST   DCD 0, 0
        END
```

Output:



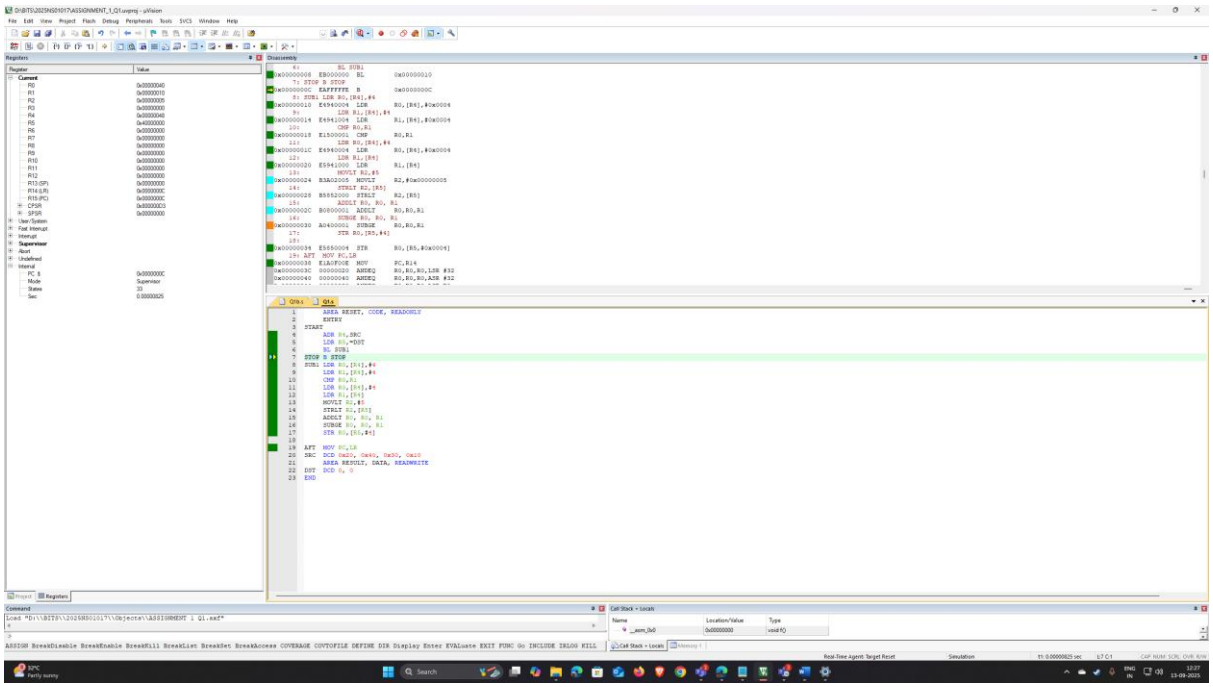
Execution of Code 1

Step 2: Execution of Code 2

```
      AREA RESET, CODE, READONLY
      ENTRY
START
      ADR R4,SRC
      LDR R5,=DST
      BL SUB1
STOP B STOP
SUB1 LDR R0,[R4],#4
      LDR R1,[R4],#4
      CMP R0,R1
      LDR R0,[R4],#4
      LDR R1,[R4]
      MOVLT R2,#5
      STRLT R2,[R5]
      ADDLT R0, R0, R1
      SUBGE R0, R0, R1
      STR R0,[R5,#4]

AFT  MOV PC,LR
SRC  DCD 0x20, 0x40, 0x30, 0x10
      AREA RESULT, DATA, READWRITE
DST  DCD 0, 0
END
```

Output:

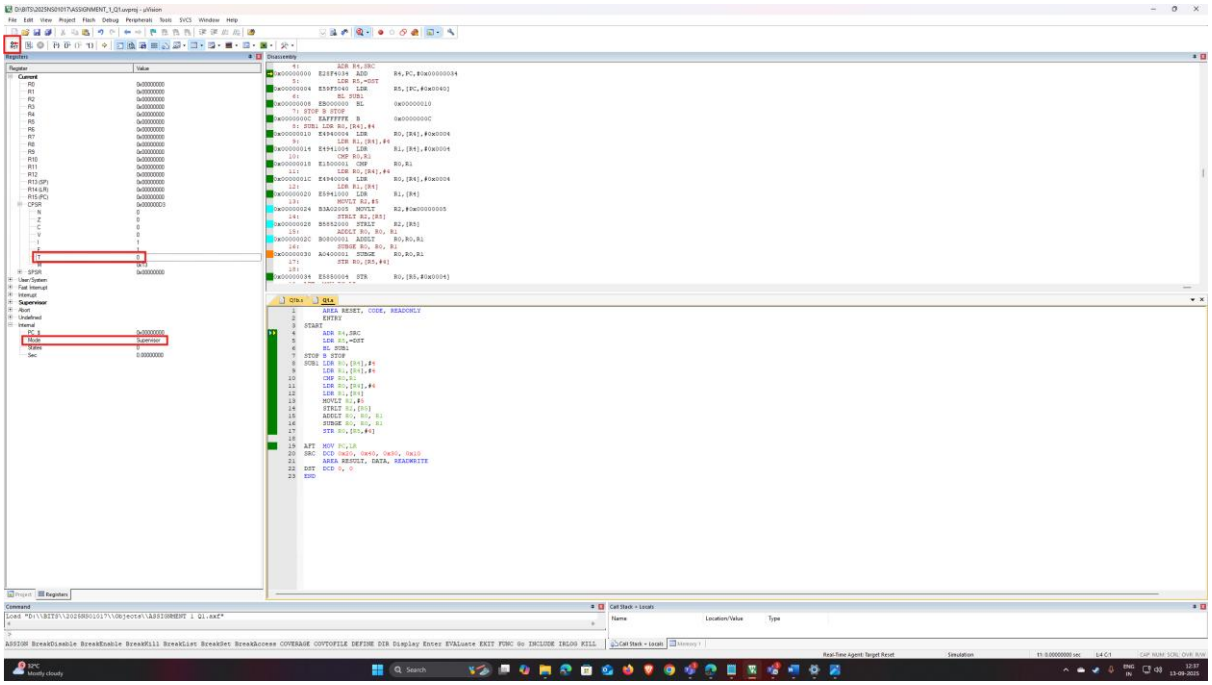


Execution of Code 2

a) On reset what is the ARM7TDMI processor's state and mode of operation?

Processor State: ARM (shown by CPSR > T = 0 in the screenshot below)

Mode of Operation: Supervisor (shown in screenshot below)



Processor State and Mode of Operation after Reset

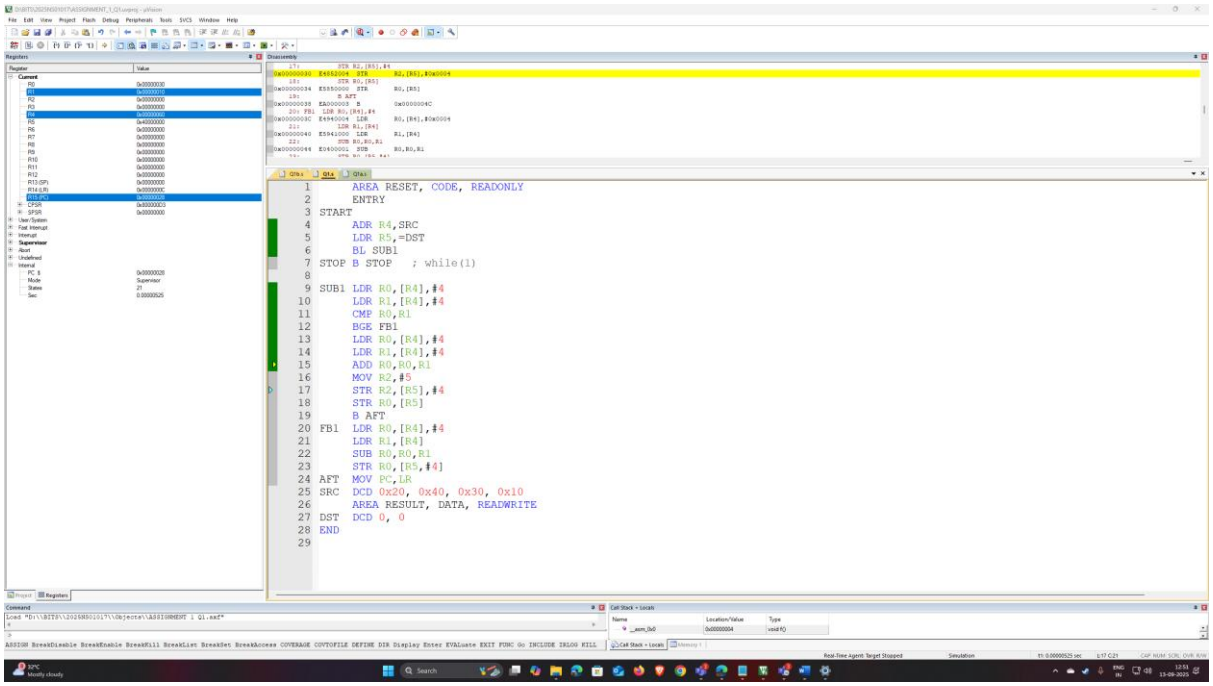
b) How many states are taken for the execution of an Arithmetic instruction, Load and Store instruction respectively (For Code-1)?

- States taken are shown in table below:

Instruction Type	Number of states taken
Arithmetic	1
Load	3 (ADR: 1, ADR+LOAD: 4, therefore LOAD: 3)
Store	2

Simulation Results:

1. Arithmetic
    a. Before Execution

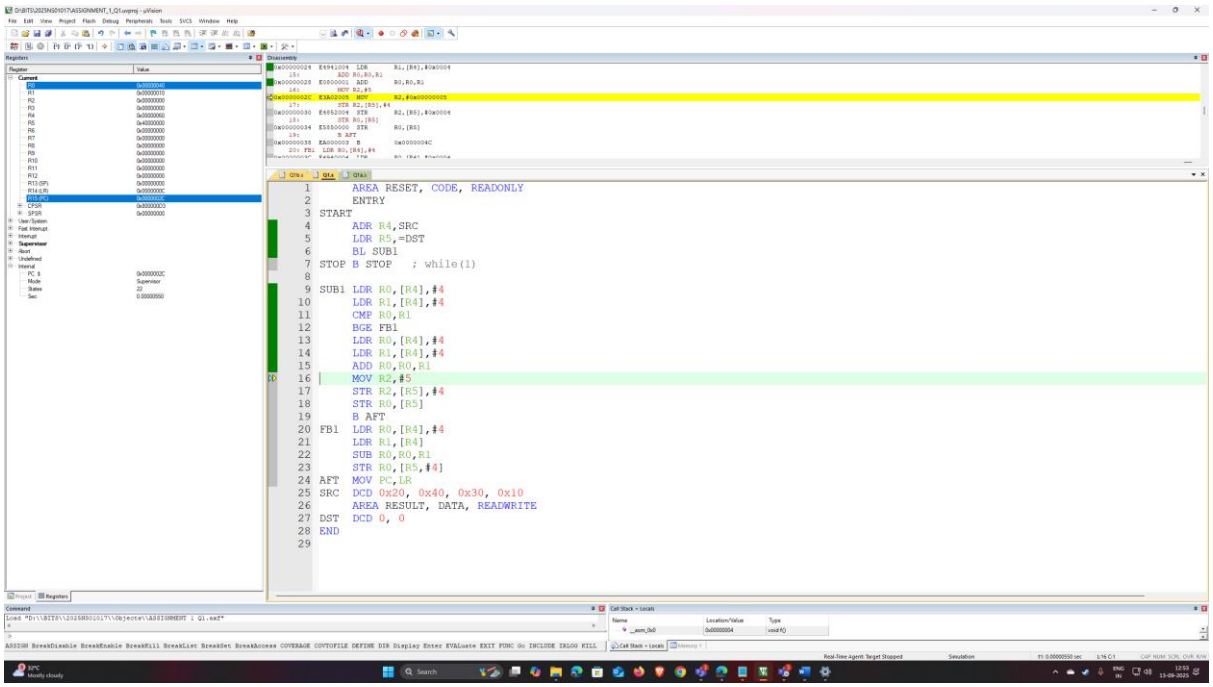


```
    AREA RESET, CODE, READONLY
    ENTRY
START
    ADR R4,SRC
    LDR R5,=DST
    BL SUB1
STOP B STOP   ; while(1)

SUB1 LDR R0,[R4],#4
    LDR R1,[R4],#4
    CMP R0,R1
    BGE FB1
    LDR R0,[R4],#4
    LDR R1,[R4],#4
    ADD R0,R0,R1
    MOV R2,#5
    STR R2,[R5],#4
    STR R0,[R5]
    B AFT
FB1 LDR R0,[R4],#4
    LDR R1,[R4]
    SUB R0,R0,R1
    STR R0,[R5,#4]
AFT MOV PC,LR
SRC DCD 0x20, 0x40, 0x30, 0x10
    AREA RESULT, DATA, READWRITE
DST DCD 0, 0
END
```

Notice: States = 21 after execution of LDR

    b. After Execution

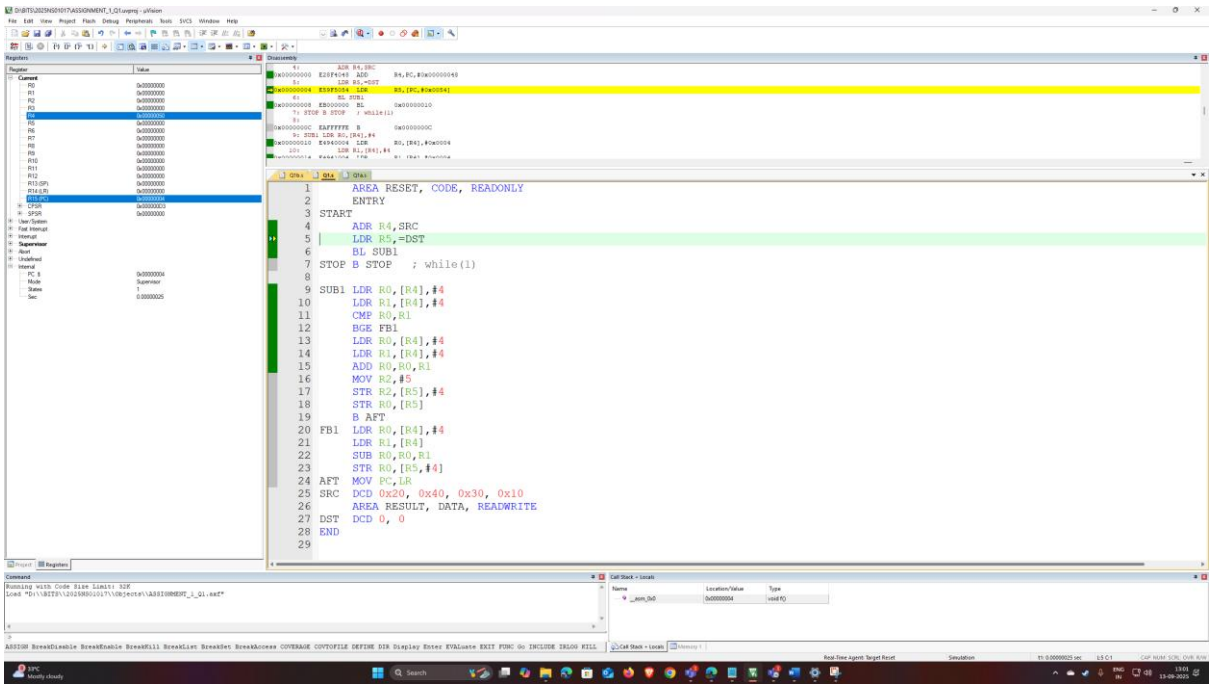


```
    AREA RESET, CODE, READONLY
    ENTRY
START
    ADR R4,SRC
    LDR R5,=DST
    BL SUB1
STOP B STOP   ; while(1)

SUB1 LDR R0,[R4],#4
    LDR R1,[R4],#4
    CMP R0,R1
    BGE FB1
    LDR R0,[R4],#4
    LDR R1,[R4],#4
    ADD R0,R0,R1
    MOV R2,#5
    STR R2,[R5],#4
    STR R0,[R5]
    B AFT
FB1 LDR R0,[R4],#4
    LDR R1,[R4]
    SUB R0,R0,R1
    STR R0,[R5,#4]
AFT MOV PC,LR
SRC DCD 0x20, 0x40, 0x30, 0x10
    AREA RESULT, DATA, READWRITE
DST DCD 0, 0
END
```

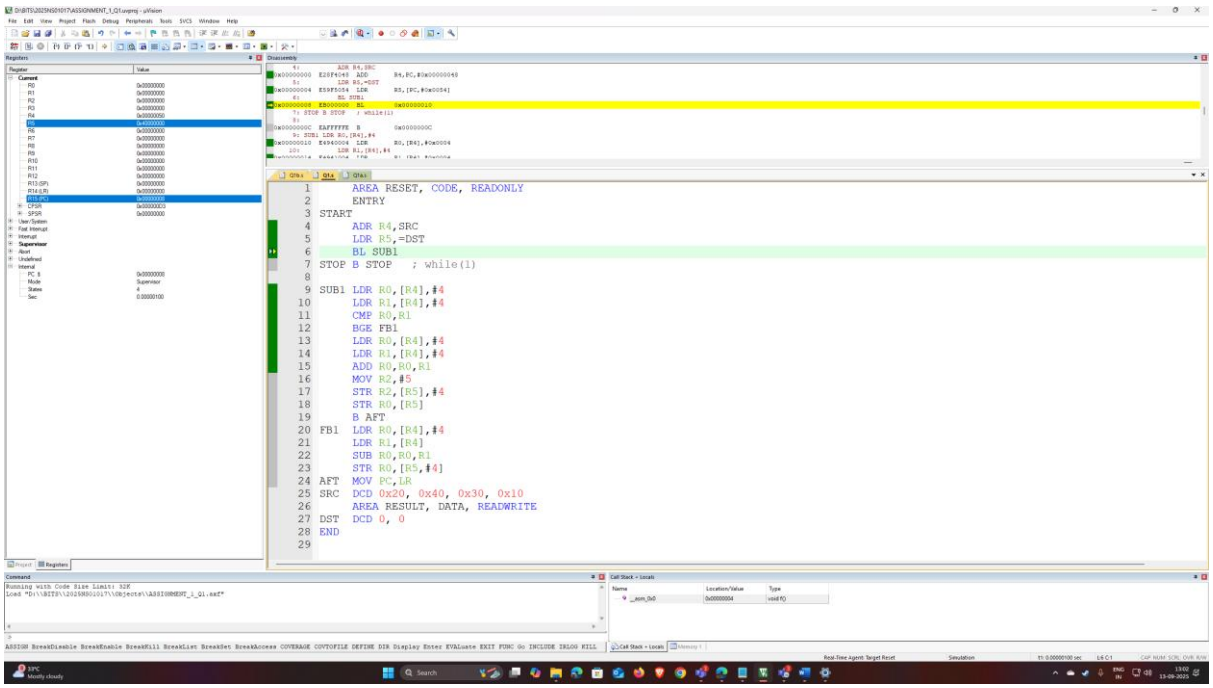
Notice: States = 22 after execution of ADD (diff = 1)

2. LOAD instruction
    a. Before Execution



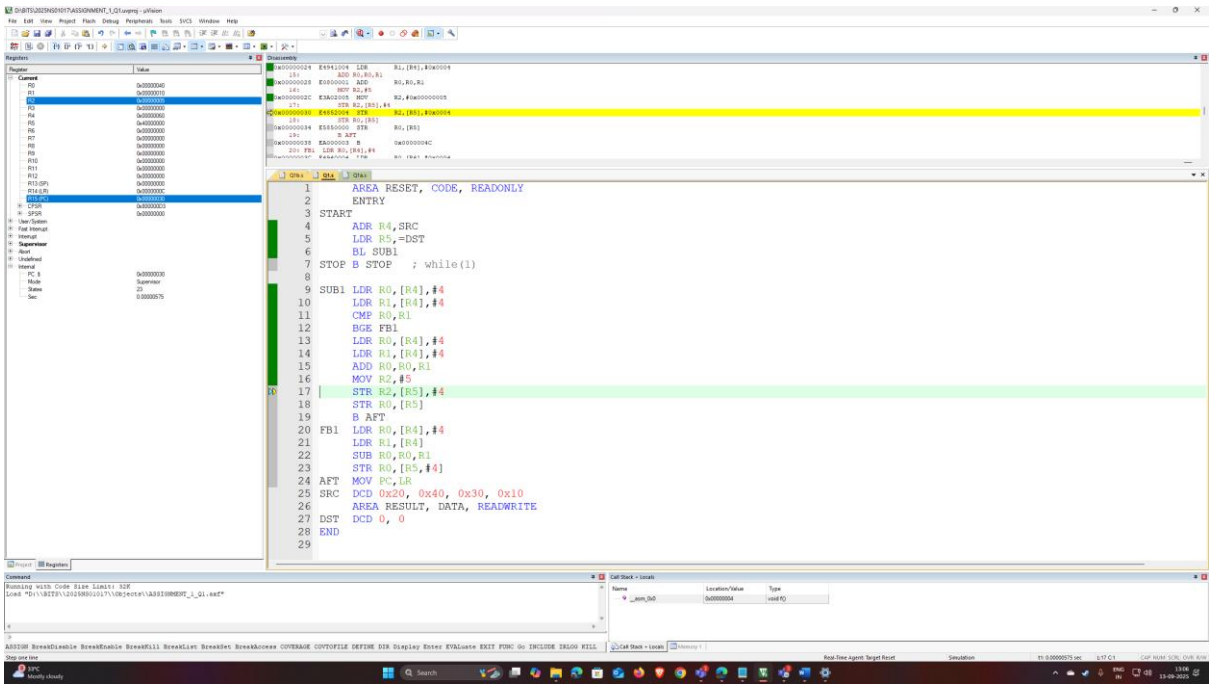
Notice: States = 1 after execution of ADR

    b. After Execution



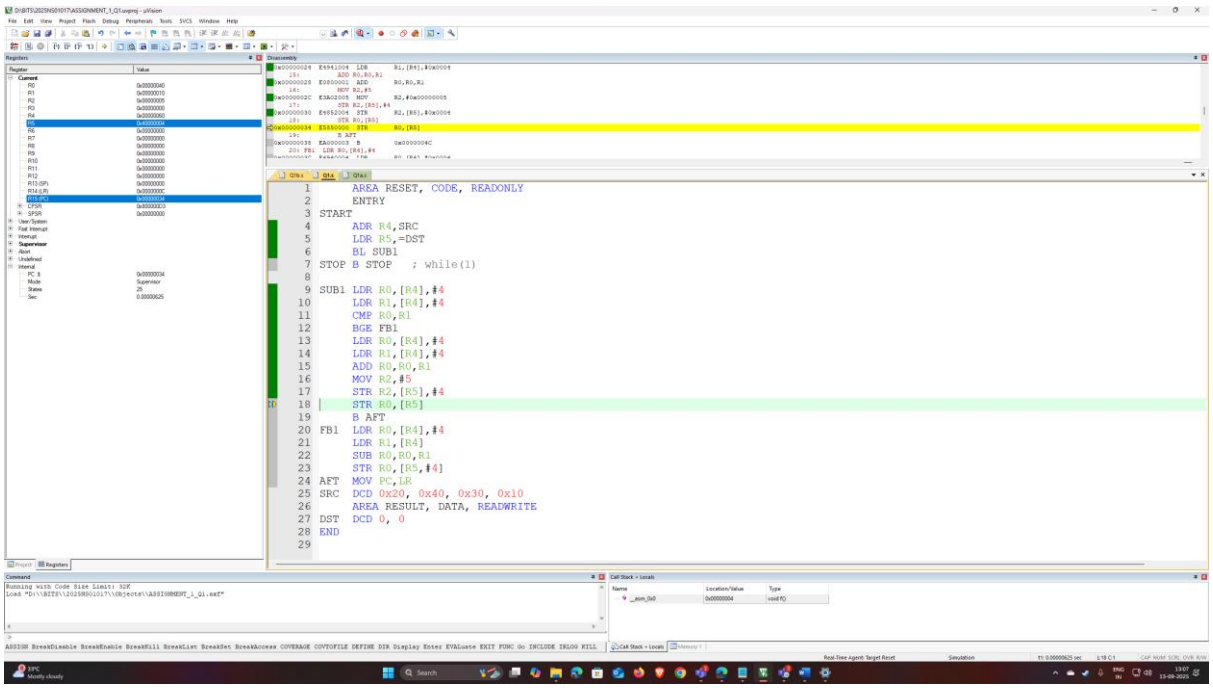
Notice: States = 4 after execution of LOAD (diff = 3)

3. STORE instruction
    a. Before Execution



Notice: States = 23 after execution of MOV

    b. After Execution



Notice: States = 25 after execution of STR (diff = 2)

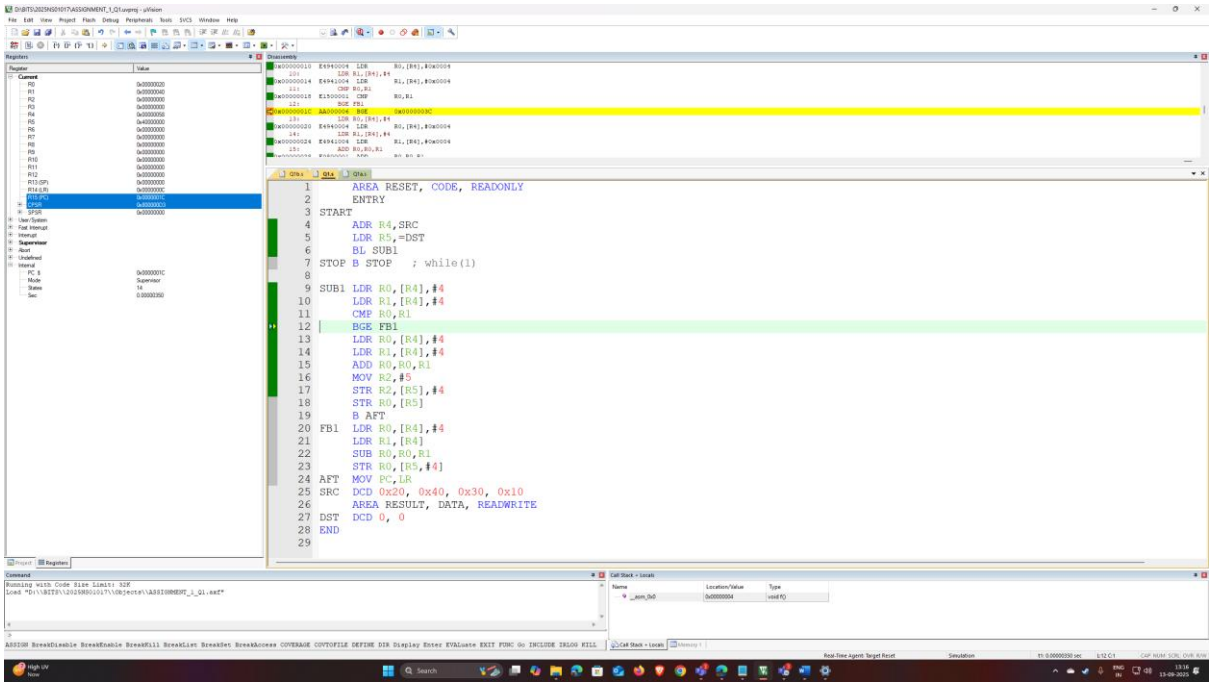
c) Are the number of states taken for completion same for BGE instruction if the branch – (1) is taken (2) not taken? Please give the states are taken for each. (For Code-1)

➢ The number of states for the BGE (Branch if Greater or Equal) instruction depends on whether the branch is taken. This is due to the processor's instruction pipeline.

- Branch Not Taken: If the condition is false and the branch is not taken, the processor simply moves to the next instruction in sequence. **This takes 1 clock cycle**.
- Branch Taken: If the condition is true, the processor must discard the instructions it has already fetched and loaded into its pipeline and then fetch the new instruction from the branch target address. This process, known as a "pipeline flush," incurs a penalty**. A taken branch takes 3 clock cycles**.

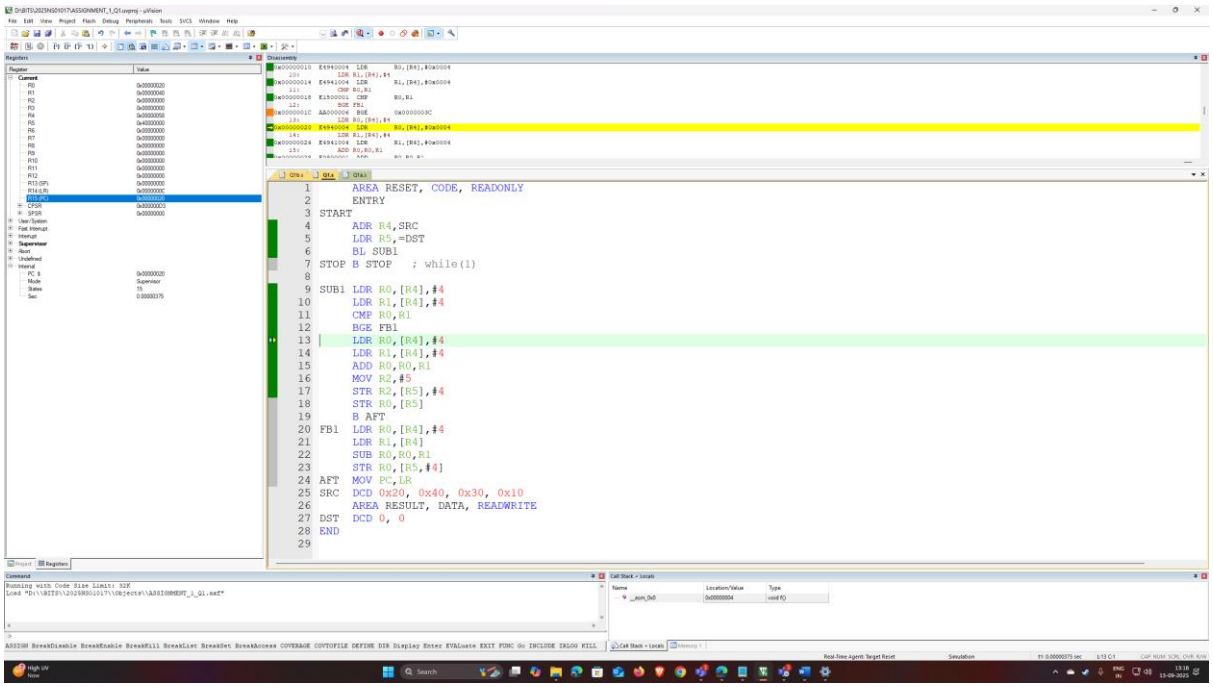
Verification:

1. Branch not taken
    a. Before BGE instruction



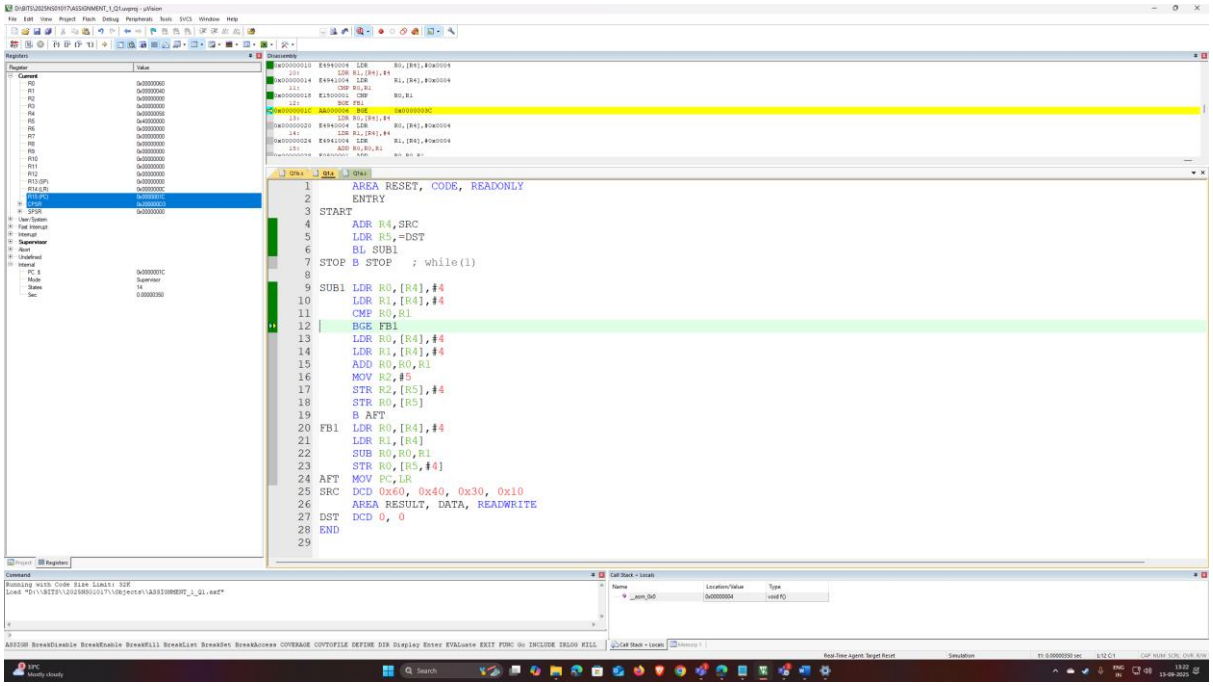
States = 14

    b. After BGE instruction



States = 15

2. Branch Taken
    a. Before BGE instruction

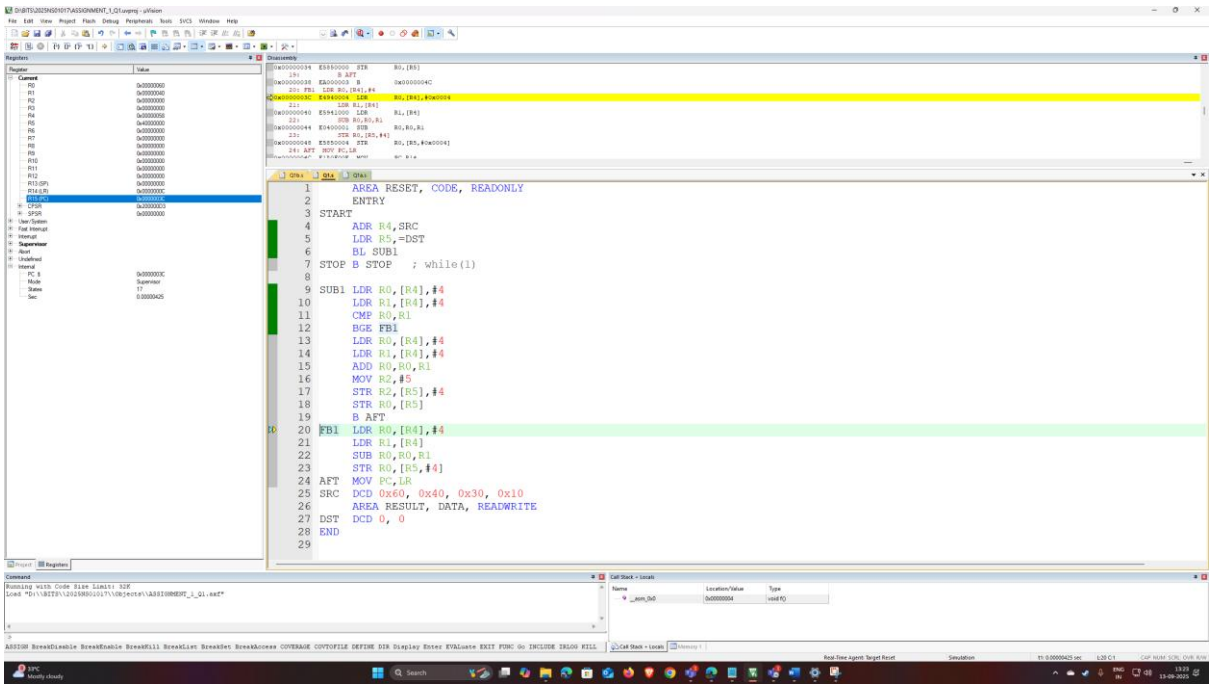


```
        AREA RESET, CODE, READONLY
        ENTRY
START
        ADR R4,SRC
        LDR R5,=DST
        BL SUB1
STOP B STOP   ; while(1)

SUB1 LDR R0,[R4],#4
        LDR R1,[R4],#4
        CMP R0,R1
        BGE FB1
        LDR R0,[R4],#4
        LDR R1,[R4],#4
        ADD R0,R0,R1
        MOV R2,#5
        STR R2,[R5],#4
        STR R0,[R5]
        B AFT
FB1  LDR R0,[R4],#4
        LDR R1,[R4]
        SUB R0,R0,R1
        STR R0,[R5,#4]
AFT  MOV PC,LR
SRC  DCD 0x60, 0x40, 0x30, 0x10
        AREA RESULT, DATA, READWRITE
DST  DCD 0, 0
END
```

States = 14

    b. After BGE instruction



```
        AREA RESET, CODE, READONLY
        ENTRY
START
        ADR R4,SRC
        LDR R5,=DST
        BL SUB1
STOP B STOP   ; while(1)

SUB1 LDR R0,[R4],#4
        LDR R1,[R4],#4
        CMP R0,R1
        BGE FB1
        LDR R0,[R4],#4
        LDR R1,[R4],#4
        ADD R0,R0,R1
        MOV R2,#5
        STR R2,[R5],#4
        STR R0,[R5]
        B AFT
FB1  LDR R0,[R4],#4
        LDR R1,[R4]
        SUB R0,R0,R1
        STR R0,[R5,#4]
AFT  MOV PC,LR
SRC  DCD 0x60, 0x40, 0x30, 0x10
        AREA RESULT, DATA, READWRITE
DST  DCD 0, 0
END
```

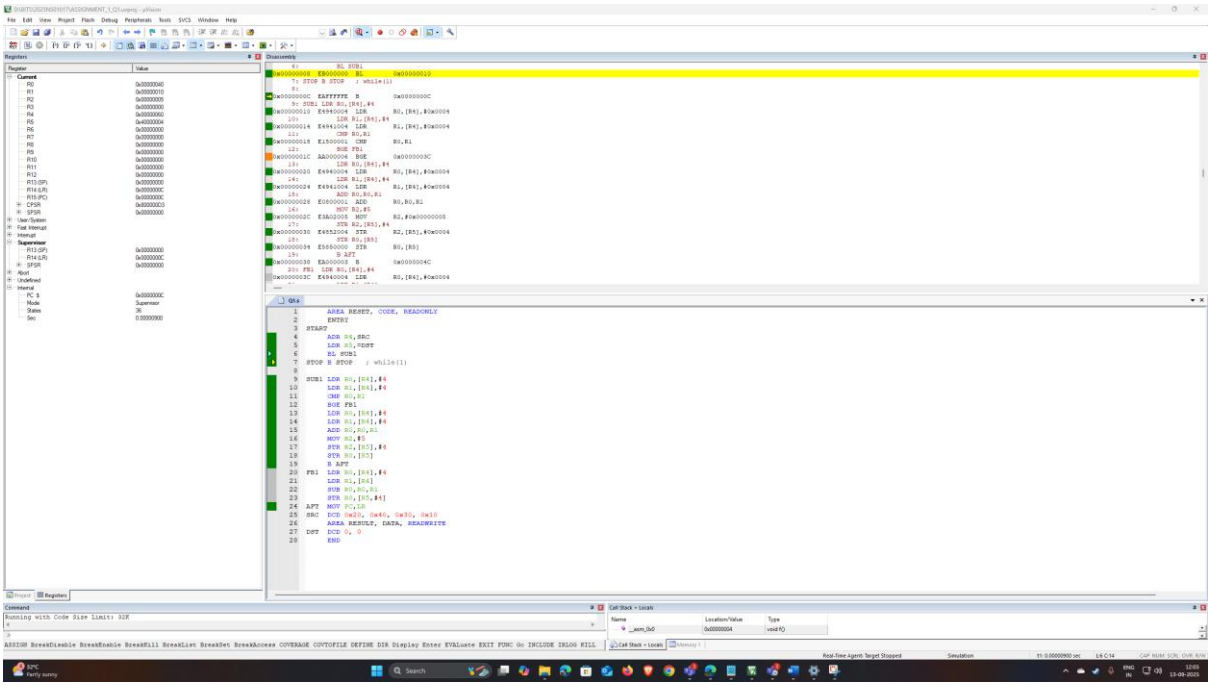
States = 17

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

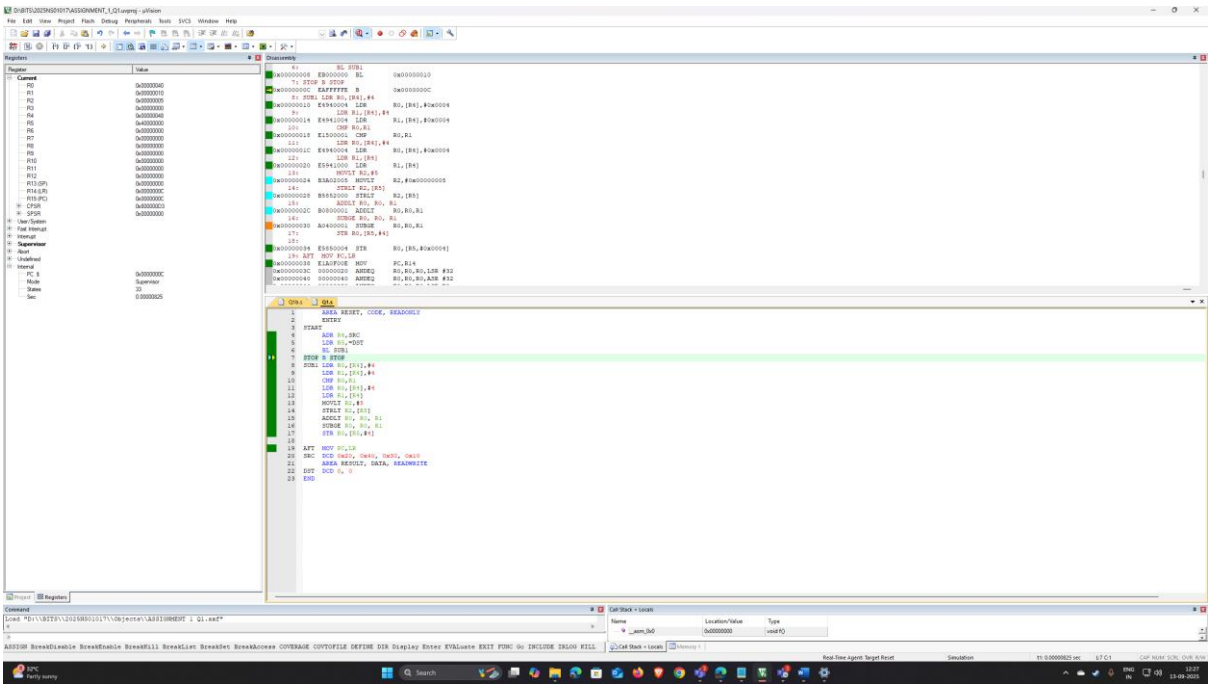
Condition	Code-1 States	Code-2 States
a < b	36	33
a > b	32	32
a = b	32	32

Verification:

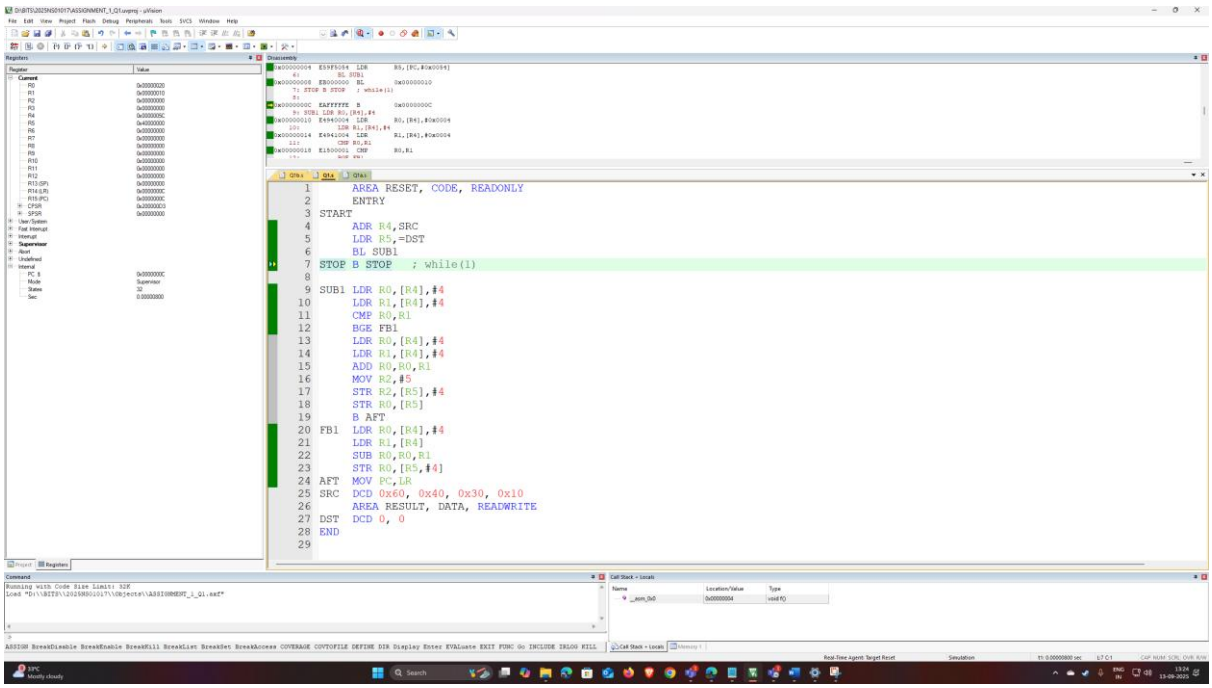
1. Condition: a<b
   a. Code-1



   b. Code-2



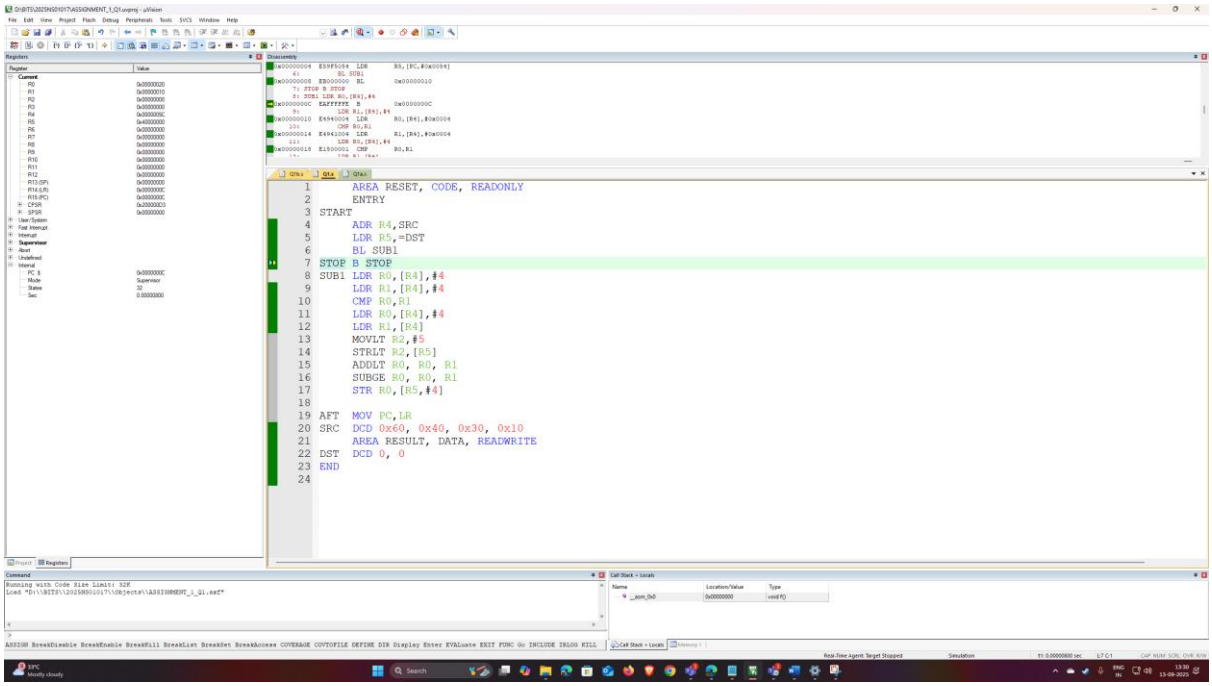
2. Condition: a>b
    a. Code-1



```
        AREA RESET, CODE, READONLY
        ENTRY
START
        ADR R4,SRC
        LDR R5,=DST
        BL SUB1
STOP B STOP   ; while(1)

SUB1 LDR R0,[R4],#4
     LDR R1,[R4],#4
     CMP R0,R1
     BGE FB1
     LDR R0,[R4],#4
     LDR R1,[R4],#4
     ADD R0,R0,R1
     MOV R2,#5
     STR R2,[R5],#4
     STR R0,[R5]
     B AFT
FB1  LDR R0,[R4],#4
     LDR R1,[R4]
     SUB R0,R0,R1
     STR R0,[R5,#4]
AFT  MOV PC,LR
SRC  DCD 0x60, 0x40, 0x30, 0x10
     AREA RESULT, DATA, READWRITE
DST  DCD 0, 0
END
```

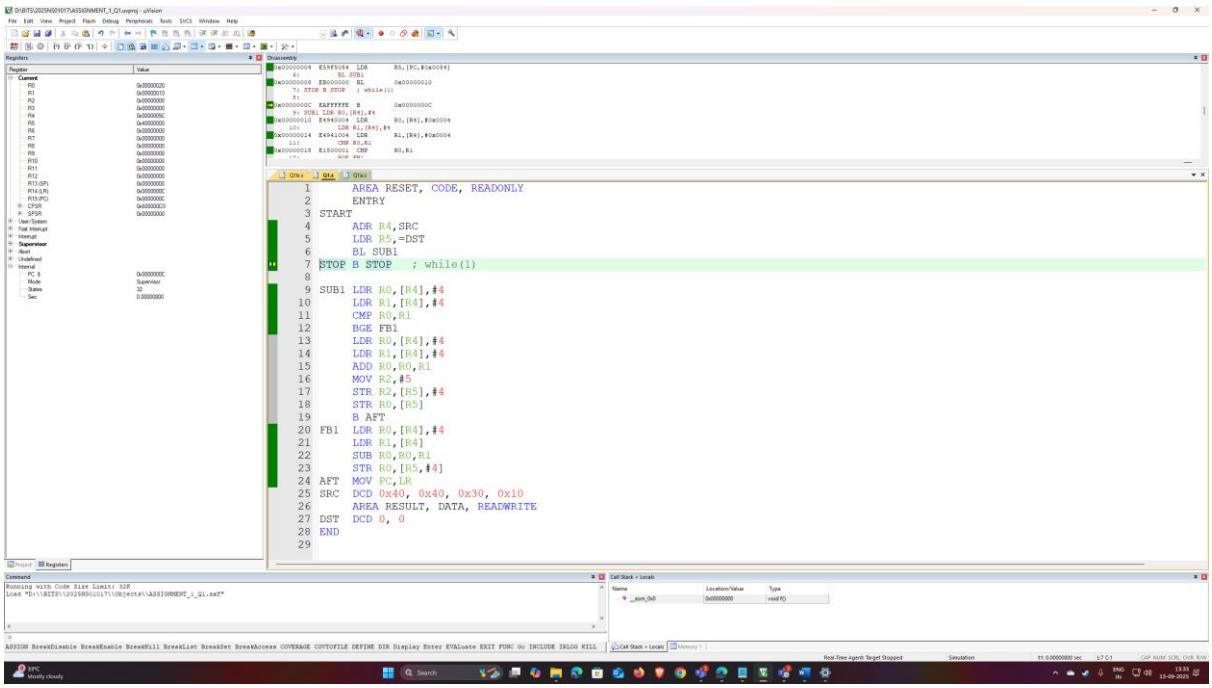
    b. Code-2



```
        AREA RESET, CODE, READONLY
        ENTRY
START
        ADR R4,SRC
        LDR R5,=DST
        BL SUB1
STOP B STOP
SUB1 LDR R0,[R4],#4
     LDR R1,[R4],#4
     CMP R0,R1
     LDR R0,[R4],#4
     LDR R1,[R4]
     MOVLT R2,#5
     STRLT R2,[R5]
     ADDLT R0, R0, R1
     SUBGE R0, R0, R1
     STR R0,[R5,#4]

AFT  MOV PC,LR
SRC  DCD 0x60, 0x40, 0x30, 0x10
     AREA RESULT, DATA, READWRITE
DST  DCD 0, 0
END
```

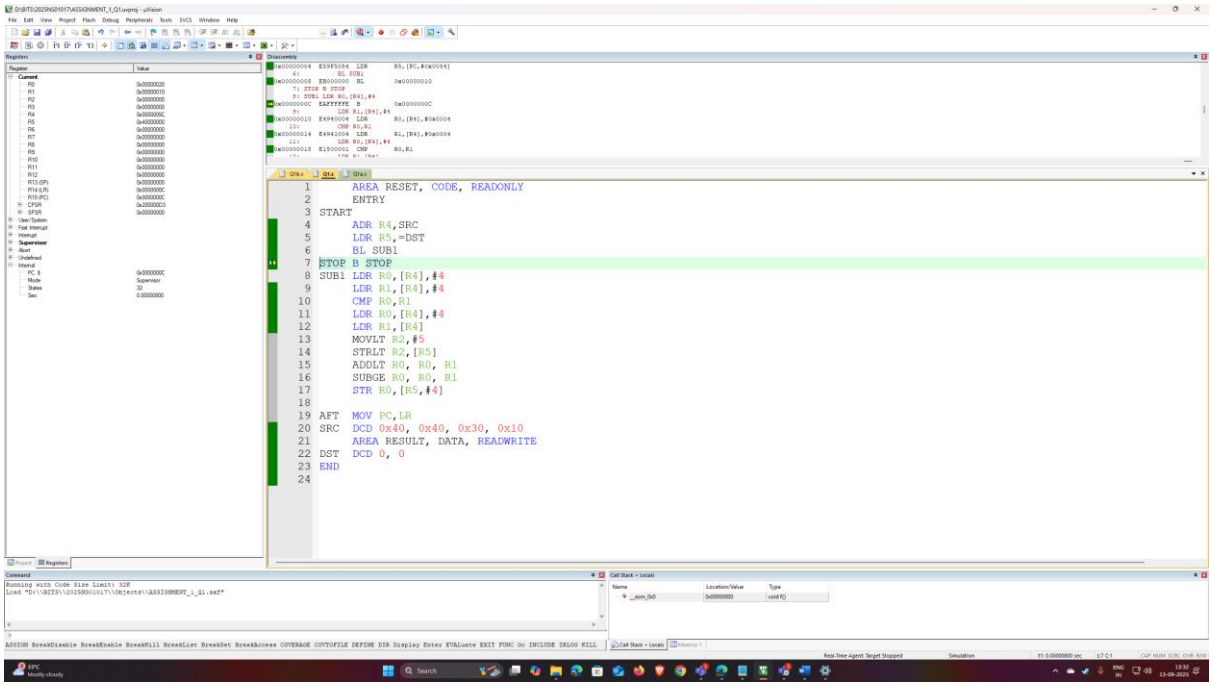
3. Condition: a=b
    a. Code-1



```
        AREA RESET, CODE, READONLY
        ENTRY
START
        ADR R4,SRC
        LDR R5,=DST
        BL SUB1
STOP B STOP   ; while(1)

SUB1 LDR R0,[R4],#4
        LDR R1,[R4],#4
        CMP R0,R1
        BGE FB1
        LDR R0,[R4],#4
        LDR R1,[R4],#4
        ADD R0,R0,R1
        MOV R2,#5
        STR R2,[R5],#4
        STR R0,[R5]
        B AFT
FB1  LDR R0,[R4],#4
        LDR R1,[R4]
        SUB R0,R0,R1
        STR R0,[R5,#4]
AFT  MOV PC,LR
SRC  DCD 0x40, 0x40, 0x30, 0x10
        AREA RESULT, DATA, READWRITE
DST  DCD 0, 0
END
```

    b. Code-2



```
        AREA RESET, CODE, READONLY
        ENTRY
START
        ADR R4,SRC
        LDR R5,=DST
        BL SUB1
STOP B STOP
SUB1 LDR R0,[R4],#4
        LDR R1,[R4],#4
        CMP R0,R1
        LDR R0,[R4],#4
        LDR R1,[R4]
        MOVLT R2,#5
        STRLT R2,[R5]
        ADDLT R0, R0, R1
        SUBGE R0, R0, R1
        STR R0,[R5,#4]

AFT  MOV PC,LR
SRC  DCD 0x40, 0x40, 0x30, 0x10
        AREA RESULT, DATA, READWRITE
DST  DCD 0, 0
END
```

Q.2 Write an assembly language program for ARM7TDMI 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. Show the flowchart or explain the algorithm for the same. Verify your result by performing manual calculation.

Give suitable screen shots of the KEIL IDE-in debug mode to demonstrate the desired outputs. Ensure that the screenshot captures system date and time.

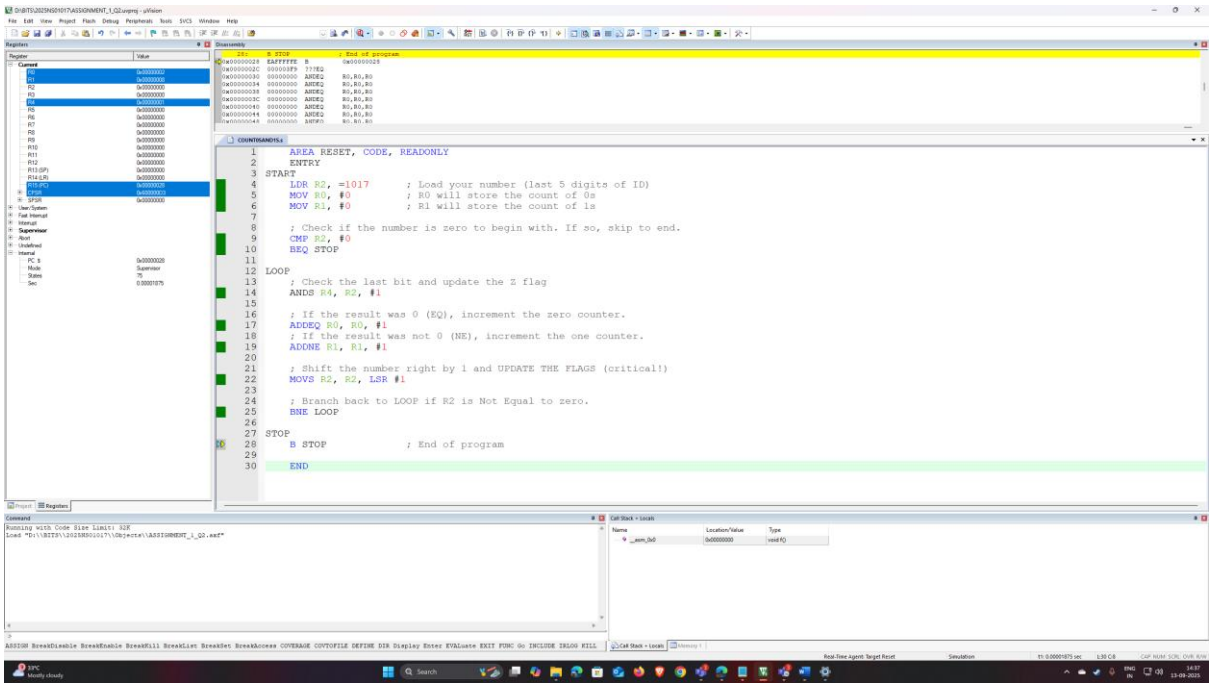
Solution:

- BITS ID: 2025NS01017
- Last 5 digits: 01017
- Resultant Decimal Number: 1017
- Conversion of decimal to binary: 1111111001
- Conversion of decimal to hexadecimal: 0x000003F9
- Number of Zeroes in binary representation: 2
- Number of Ones in binary representation: 8

Algorithm:

1. **Start.**
2. **Initialize:**
    - Load the target number (e.g., 1017) into a register N.
    - Initialize Zeros_Count to 0.
    - Initialize Ones_Count to 0.
3. **Check Edge Case:**
    - Is the number N equal to 0?
    - If **YES**, go directly to **End**.
    - If **NO**, proceed to the loop.
4. **Loop Start:**
5. **Check the Last Bit:**
    - Perform a bitwise AND between N and 1.
6. **Decision:**
    - Was the result of the check 0?
    - If **YES**, increment Zeros_Count.
    - If **NO**, increment Ones_Count.
7. **Update Number:**
    - Logically shift the number N one position to the right.
8. **Loop Condition:**
    - Is the number N now not equal to 0?
    - If **YES**, go back to **Loop Start** (Step 4).
    - If **NO**, proceed to **End**.
9. **End.**

Output:



```
    AREA RESET, CODE, READONLY
    ENTRY
START
    LDR R2, =1017       ; Load your number (last 5 digits of ID)
    MOV R0, #0          ; R0 will store the count of 0s
    MOV R1, #0          ; R1 will store the count of 1s

    ; Check if the number is zero to begin with. If so, skip to end.
    CMP R2, #0
    BEQ STOP

LOOP
    ; Check the last bit and update the Z flag
    ANDS R4, R2, #1

    ; If the result was 0 (EQ), increment the zero counter.
    ADDEQ R0, R0, #1
    ; If the result was not 0 (NE), increment the one counter.
    ADDNE R1, R1, #1

    ; Shift the number right by 1 and UPDATE THE FLAGS (critical!)
    MOVS R2, R2, LSR #1

    ; Branch back to LOOP if R2 is Not Equal to zero.
    BNE LOOP

STOP
    B STOP              ; End of program

    END
```

R0: Number of 0s in binary representation, R1: Number of 1s in binary representation

Code:

```
    AREA RESET, CODE, READONLY
    ENTRY
START
    LDR R2, =1017      ; Load your number (last 5 digits of ID)
    MOV R0, #0         ; R0 will store the count of 0s
    MOV R1, #0         ; R1 will store the count of 1s
    ; Check if the number is zero to begin with. If so, skip to
end.
    CMP R2, #0
    BEQ STOP
LOOP
    ; Check the last bit and update the Z flag
    ANDS R4, R2, #1
    ; If the result was 0 (EQ), increment the zero counter.
    ADDEQ R0, R0, #1
    ; If the result was not 0 (NE), increment the one counter.
    ADDNE R1, R1, #1
    ; Shift the number right by 1 and UPDATE THE FLAGS
(critical!)
    MOVS R2, R2, LSR #1

    ; Branch back to LOOP if R2 is Not Equal to zero.
    BNE LOOP
STOP
    B STOP             ; End of program
    END
```

Q3. Write an assembly language program for ARM Cortex M3/4 to handle Supervisor Call (SVC) exception. The program should meet the following requirements:

1. The SVC should be called from an application task running at Thread unprivileged mode. [SVC number must be your BITS ID either last 3 digits or last 2 digits, depending on whether it's less than (or equal) or greater than to 255 (number must be given in decimal number format)]

2. Two parameters [each parameter is last 5 digits of your BITS ID (number must be given in decimal number format)] should be passed to the SVC handler.

3. The SVC number should be determined dynamically by examining the actual SVC instruction in memory. [SVC Number Extraction]

4. The SVC exception handler should set up a stack pointer in SVC mode before handling the exception.

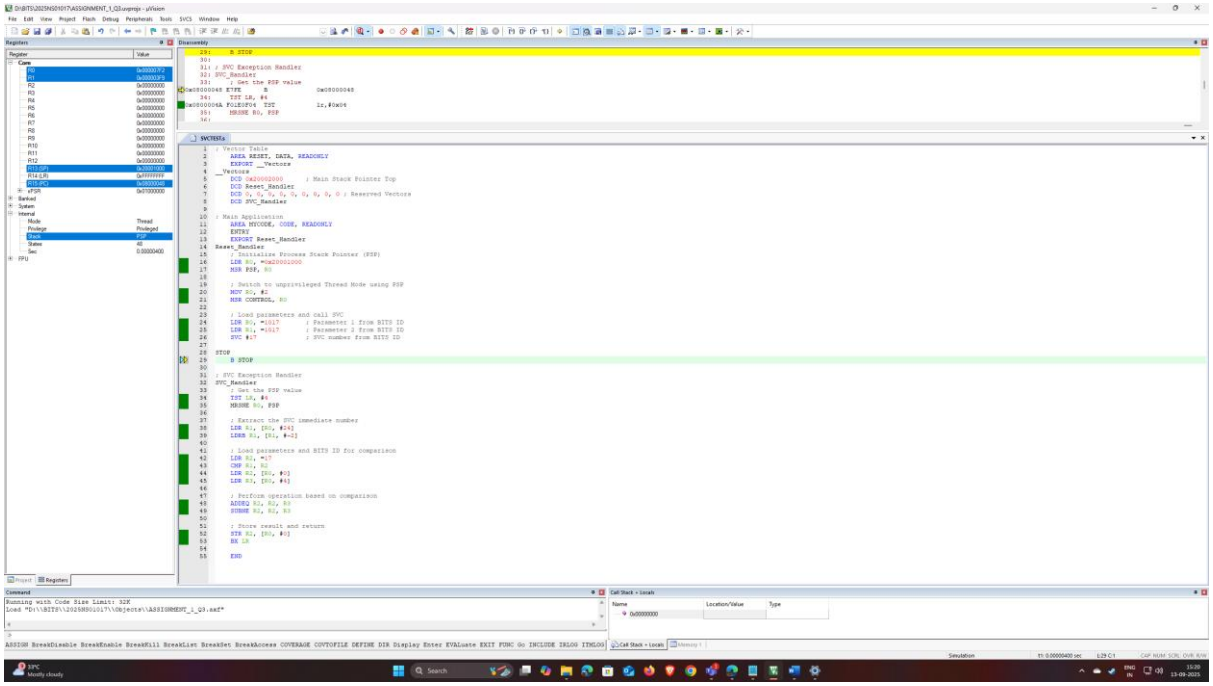
5. If the SVC number matches your BITS ID (either last 3 digits or last 2 digits, depending on whether it's less than or equal to 255), the handler should perform addition of the two passed parameters.

6. If the SVC number does not match your BITS ID, the handler should perform subtraction of the two passed parameters. Ensure that the program properly handles stack usage and resumes the application task after performing the required operation.

Give suitable screen shots of the KEIL IDE-in debug mode to demonstrate the desired outputs. Ensure that the screenshot captures system date and time.

Solution:

- BITS ID: 2025NS01017
- Last 5 digits: 01017
- Resultant Decimal number: 1017
- Resultant SVC Number: 17
- Resultant addition in case of SVC match: 1017 + 1017 = 2034 (0x7F2 in hexadecimal)
- Expected Result: R0 should contain the value 0x752 (which is 2034 in hexadecimal)



R0: 0x7F2 (which is 2034 in hexadecimal, sum of 1017 and 1017)

Code:

```
; Vector Table
    AREA RESET, DATA, READONLY
    EXPORT __Vectors
__Vectors
    DCD 0x20002000      ; Main Stack Pointer Top
    DCD Reset_Handler
    DCD 0, 0, 0, 0, 0, 0, 0, 0, 0 ; Reserved Vectors
    DCD SVC_Handler

; Main Application
    AREA MYCODE, CODE, READONLY
    ENTRY
    EXPORT Reset_Handler
Reset_Handler
    ; Initialize Process Stack Pointer (PSP)
    LDR R0, =0x20001000
    MSR PSP, R0

    ; Switch to unprivileged Thread Mode using PSP
    MOV R0, #2
    MSR CONTROL, R0

    ; Load parameters and call SVC
    LDR R0, =1017       ; Parameter 1 from BITS ID
    LDR R1, =1017       ; Parameter 2 from BITS ID
    SVC #17             ; SVC number from BITS ID

STOP
```

```
    B STOP

; SVC Exception Handler
SVC_Handler
    ; Get the PSP value
    TST LR, #4
    MRSNE R0, PSP

    ; Extract the SVC immediate number
    LDR R1, [R0, #24]
    LDRB R1, [R1, #-2]

    ; Load parameters and BITS ID for comparison
    LDR R2, =17
    CMP R1, R2
    LDR R2, [R0, #0]
    LDR R3, [R0, #4]

    ; Perform operation based on comparison
    ADDEQ R2, R2, R3
    SUBNE R2, R2, R3

    ; Store result and return
    STR R2, [R0, #0]
    BX LR

    END
```