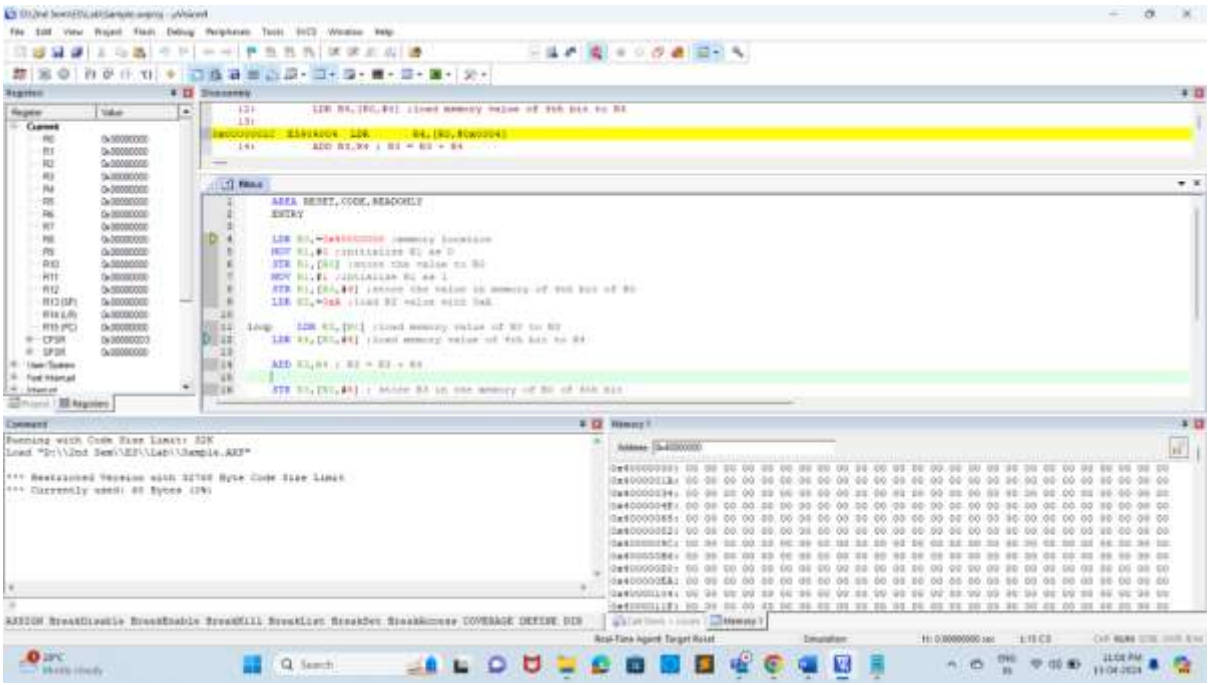
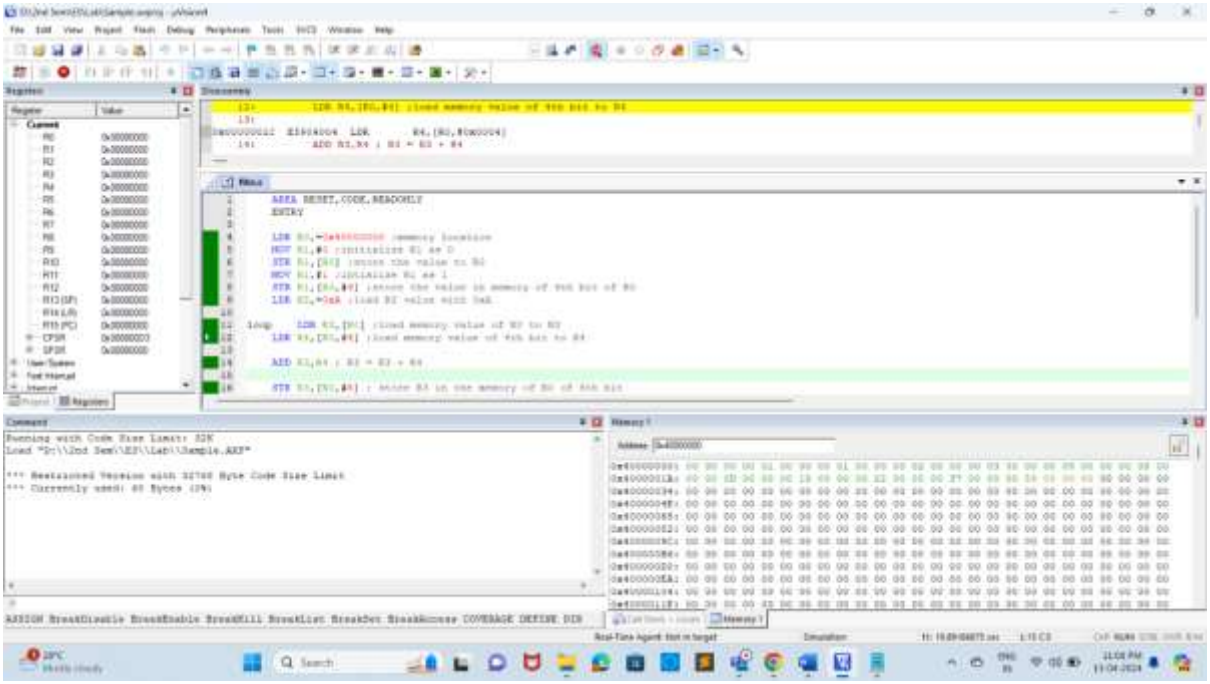
1. **Write a ASR to print Fibonacci series upto N numbers.**

```
        AREA RESET,CODE,READONLY
        ENTRY

        LDR R0,=0x40000000 ;memory location
        MOV R1,#0 ;initialize R1 as 0
        STR R1,[R0] ;store the value to R0
        MOV R1,#1 ;intialize R1 as 1
        STR R1,[R0,#4] ;store the value in memory of 4th bit of R0
        LDR R2,=0xA ;load R2 value with 0xA

loop    LDR R3,[R0] ;load memory value of R0 to R3
        LDR R4,[R0,#4] ;load memory value of 4th bit to R4

        ADD R3,R4 ; R3 = R3 + R4

        STR R3,[R0,#8] ; store R3 in the memory of R0 of 8th bit

        ADD R0,#4 ; R0 = R0 + 4
        SUB R2,#1 ; R2 = R2 - 1
        CMP R2,#0 ; Compare R2 = 0
        BNE loop ;Branch not equal to loop

Stop B Stop

        END
```

Before Execution:



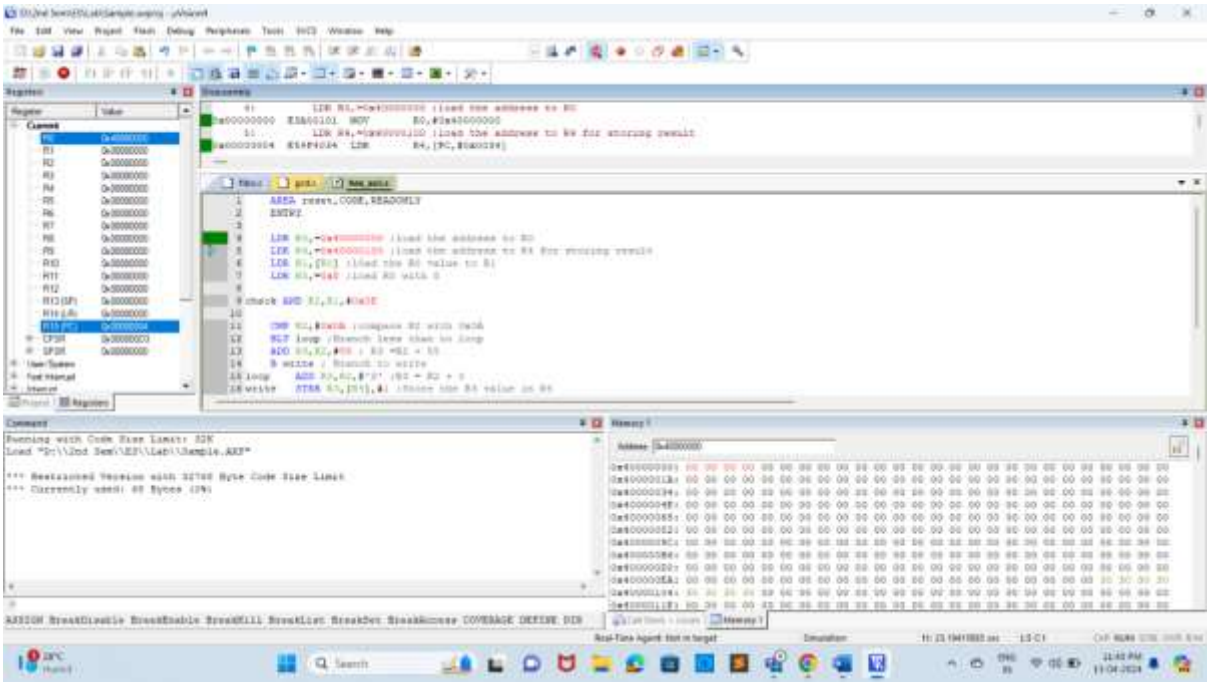
After Execution:



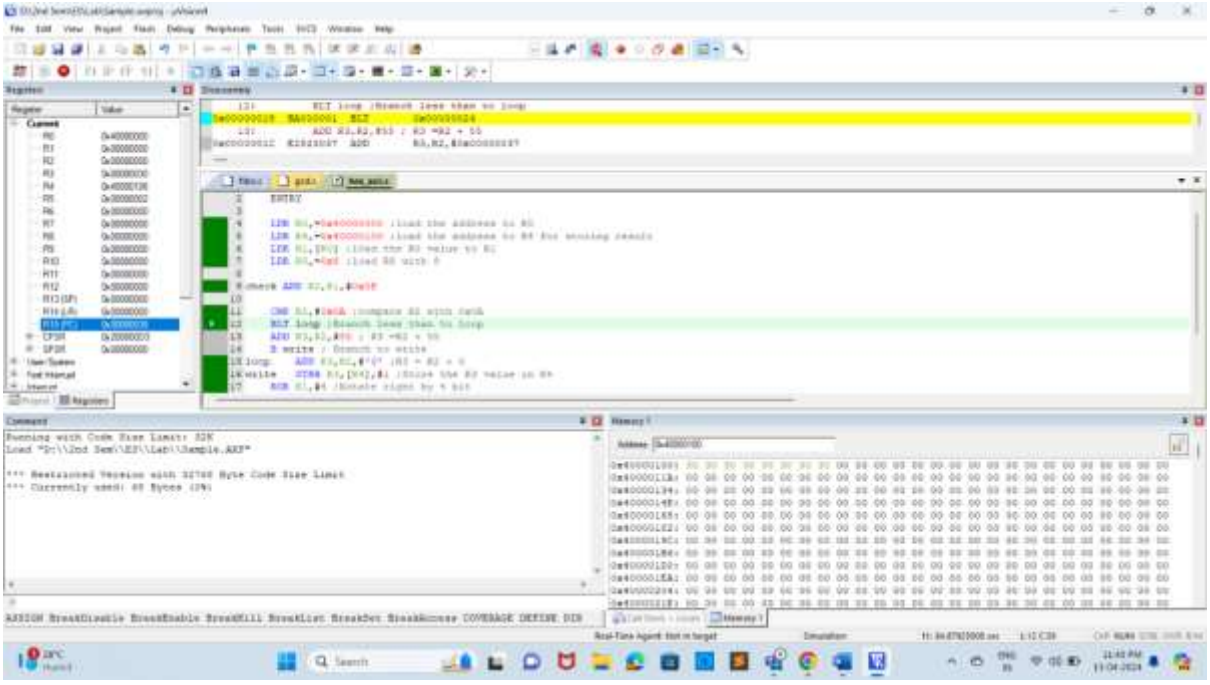
2. Write a ASR program to convert a given number from Hex to ASCII code.

```
        AREA reset,CODE,READONLY
        ENTRY

        LDR R0,=0x40000000 ;load the address to R0
        LDR R4,=0x40000100 ;load the address to R4 for storing result
        LDR R1,[R0] ;l0ad the R0 value to R1
        LDR R5,=0x8 ;load R5 with 8

check AND R2,R1,#0x0f

        CMP R2,#0x0A ;compare R2 with 0x0A
        BLT loop ;Branch less than to loop
        ADD R3,R2,#55 ; R3 =R2 + 55
        B write ; Branch to write
loop    ADD R3,R2,#'0' ;R3 = R2 + 0
write   STRB R3,[R4],#1 ;Store the R3 value in R4
        ROR R1,#4 ;Rotate right by 4 bit
        SUBS R5,#1 ;subtract R5 by 1
        CMP R5,#0 ;Compare R5 with 0
        BNE check ;Branch not equal to check

Stop B Stop
        END
```

Before Execution:



After Execution:

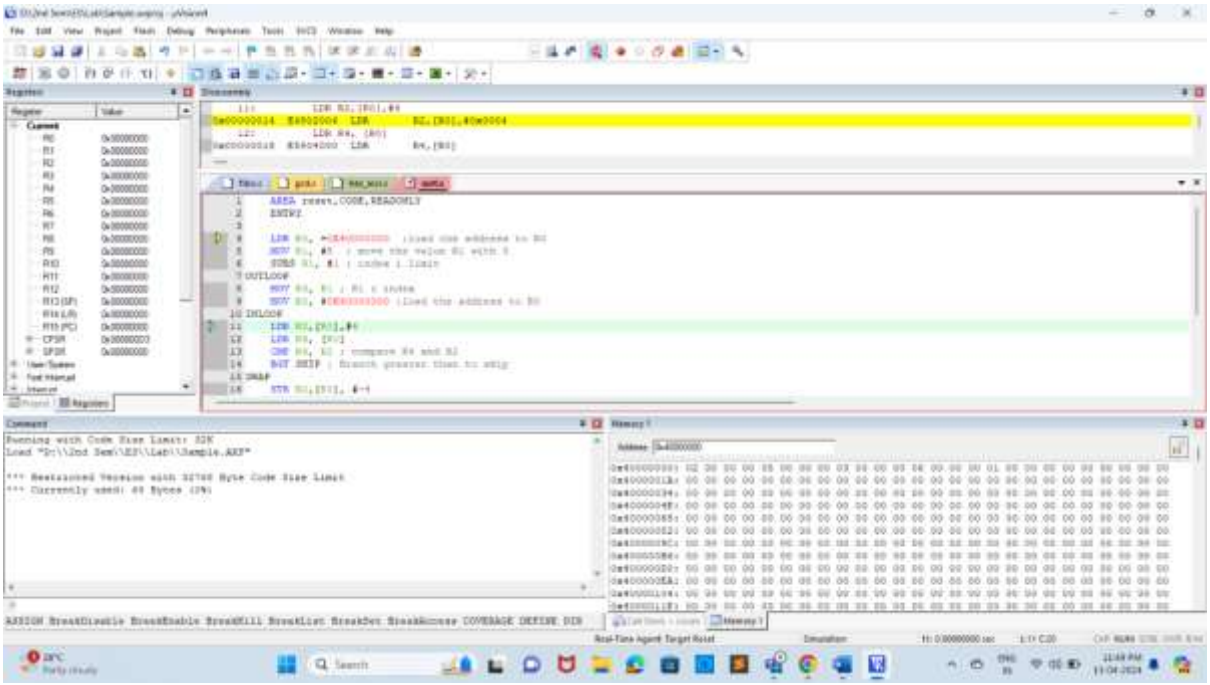


3. **Sort N given numbers using bubble sorting technique.**

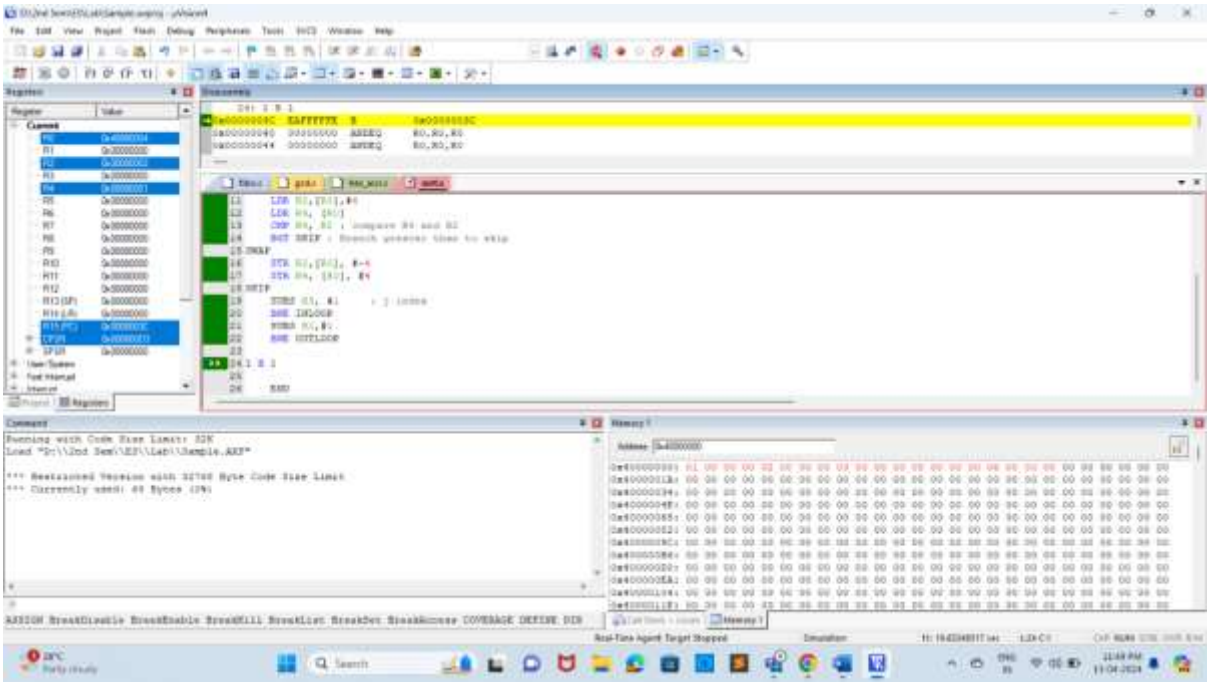
```
        AREA reset,CODE,READONLY
        ENTRY

        LDR R0, =0X40000000  ;load the address to R0
        MOV R1, #5   ; move the value R1 with 5
        SUBS R1, #1 ; index i limit
OUTLOOP
```

```
        MOV R3, R1 ; R1 i index
        MOV R0, #0X40000000 ;load the address to R0
INLOOP
        LDR R2,[R0],#4
        LDR R4, [R0]
        CMP R4, R2 ; compare R4 and R2
        BGT SKIP ; Branch greater than to skip
SWAP
        STR R2,[R0], #-4
        STR R4, [R0], #4
SKIP
        SUBS R3, #1  ; j index
        BNE INLOOP
        SUBS R1,#1
        BNE OUTLOOP

Stop B Stop

        END
```

Before Execution:



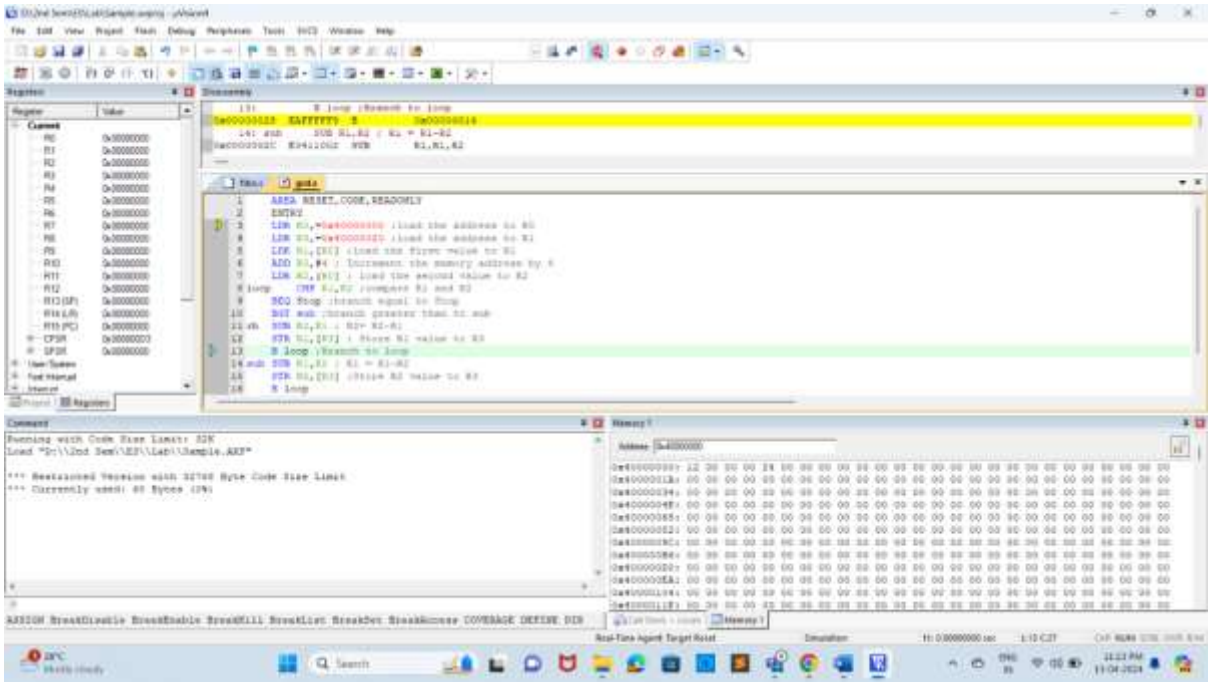
After Execution:



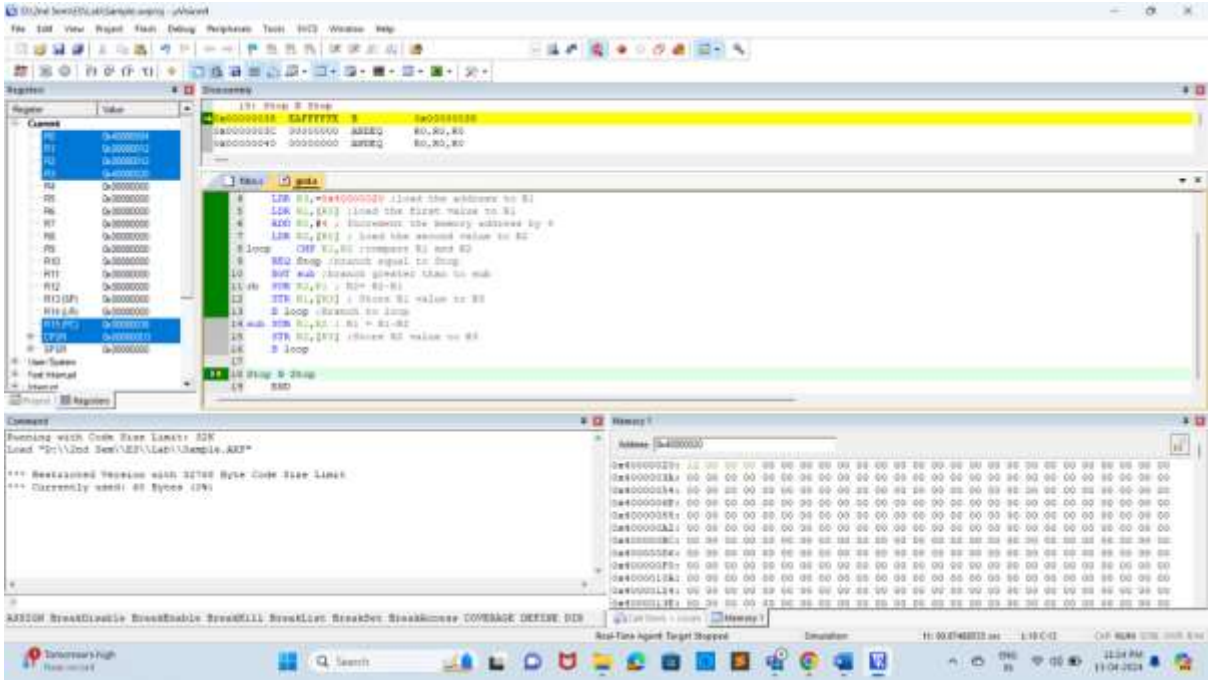
4. **Find GCD of two number.**

```
        AREA RESET,CODE,READONLY
        ENTRY

        LDR R0,=0x40000000 ;load the address to R0
        LDR R3,=0x40000020 ;load the address to R1

        LDR R1,[R0] ;load the first value to R1
        ADD R0,#4 ; Increment the memory address by 4
        LDR R2,[R0] ; load the second value to R2

loop    CMP R1,R2 ;compare R1 and R2
        BEQ Stop ;branch equal to Stop
        BGT sub ;branch greater than to sub

rb      SUB R2,R1 ; R2= R2-R1
        STR R1,[R3] ; Store R1 value to R3
        B loop ;Branch to loop

sub     SUB R1,R2 ; R1 = R1-R2
        STR R2,[R3] ;Store R2 value to R3
        B loop

Stop B Stop

        END
```

Before Execution:



After Execution:

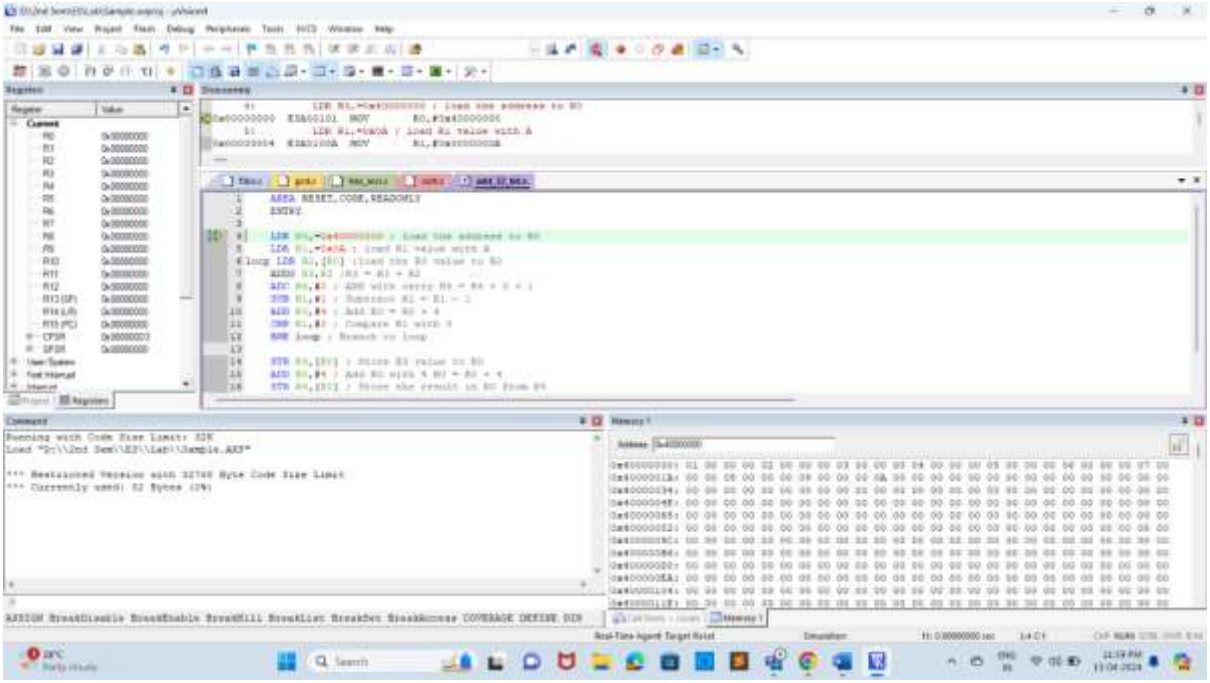


5. **Add ten 32-bit numbers with carry.**

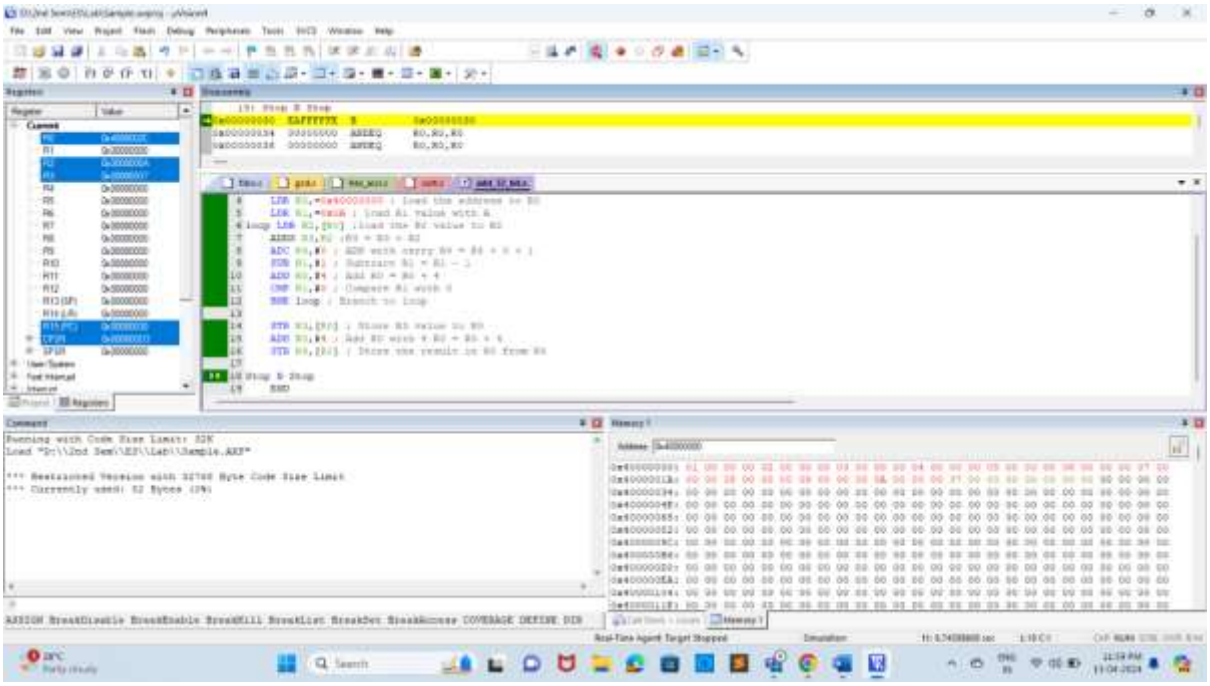
```
        AREA RESET,CODE,READONLY
        ENTRY

        LDR R0,=0x40000000 ; load the address to R0
        LDR R1,=0x0A ; load R1 value with A

Loop    LDR R2,[R0] ;load the R0 value to R2
```

```
        ADDS R3,R2 ;R3 = R3 + R2
        ADC R4,#0 ; ADD with carry R4 = R4 + 0 + 1
        SUB R1,#1 ; Subtract R1 = R1 - 1
        ADD R0,#4 ; Add R0 = R0 + 4
        CMP R1,#0 ; Compare R1 with 0
        BNE loop ; Branch to loop

        STR R3,[R0] ; Store R3 value to R0
        ADD R0,#4 ; Add R0 with 4 R0 = R0 + 4
        STR R4,[R0] ; Store the result in R0 from R4

Stop B Stop
        END
```

Before Execution:



After Execution:



6. Develop Free RTOS code for accessing a critical resource using semaphores.

```
#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"

// Define a semaphore
SemaphoreHandle_t xSemaphore;

// Define your critical resource (for example, a shared variable)
volatile int shared_variable = 0;

// Task that accesses the critical resource
void TaskAccessResource(void *pvParameters) {
    while (1) {
        // Wait until the semaphore is available
        if (xSemaphoreTake(xSemaphore, portMAX_DELAY) == pdTRUE) {
            // Access the critical resource
            shared_variable++;
            // Release the semaphore
            xSemaphoreGive(xSemaphore);
        }
        // Task delays for a while to allow other tasks to run
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}

// Task that consumes the critical resource
void TaskConsumeResource(void *pvParameters) {
    while (1) {
        // Wait until the semaphore is available
```

```
        if (xSemaphoreTake(xSemaphore, portMAX_DELAY) == pdTRUE) {
            // Access the critical resource
            printf("Shared variable value: %d\n", shared_variable);
            // Release the semaphore
            xSemaphoreGive(xSemaphore);
        }
        // Task delays for a while to allow other tasks to run
        vTaskDelay(pdMS_TO_TICKS(2000));
    }
}

int main(void) {
    // Create the semaphore
    xSemaphore = xSemaphoreCreateBinary();

    // Check if the semaphore was created successfully
    if (xSemaphore != NULL) {
        // Create TaskAccessResource
        xTaskCreate(TaskAccessResource, "AccessTask",
configMINIMAL_STACK_SIZE, NULL, 1, NULL);
        // Create TaskConsumeResource
        xTaskCreate(TaskConsumeResource, "ConsumeTask",
configMINIMAL_STACK_SIZE, NULL, 1, NULL);

        // Start the scheduler
        vTaskStartScheduler();
    }

    // Program should never reach here
    return 0;
}
```

Output:

```
Shared variable value: 1
Shared variable value: 2
Shared variable value: 3
```