MIDDLE EAST TECHNICAL UNIVERSITY

ELECTRICAL-ELECTRONICS ENGINEERING DEPARTMENT

EE447 EXPERIMENT 0 PRELIMINARY WORK

Student Names: Arda ÜNVER - Mustafa YILMAZ

Student ID: 2444081 - 2305746

Submission Date: 15.10.2023

QUESTION 1

(15 %) Build, run and understand *Practice_Lab.s* Put a screenshot of Keil in your report, showing the contents of the modified memory locations in Memory Window in the debugger.

In this question, R1 acts as a pointer. Starting from the address 0x20000400, this pointer is incremented within the loops, loop1 and loop2. Moreover, R2 holds a constant value (in our case it is 0x20 which corresponds to 32 in decimal).

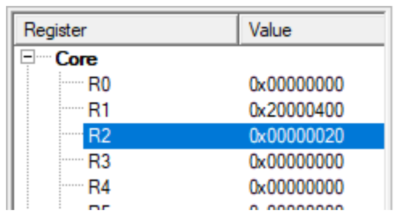

Figure 1. Values stored R0, R1, and R2 Registers

In loop1, the value stored in R0 is stored to the address pointed by R1. Pointer is incremented. The value in R2 is decremented by 1. If the value in R2 is equal to 0, then flag **Z (Zero)** is 1 (due to SUB**S**), and the loop ends with the **BNE** (Branch Not Equal) operation. If not, the loop continues (flag **Z (Zero)** is 0). Since the value in R0 is not incremented, the value 0 is written to a different address, **32** times. The status of the registers and the data written in memory after the execution of **loop1** can be seen from code in Figure 2.

Other loop, loop2, functions similarly. However, the value stored in R0 is incremented at each cycle. Thus, the numbers are written in an increasing order to the addresses. The status of the registers and the data written in memory after the execution of **loop2** can be observed from Figure 2.

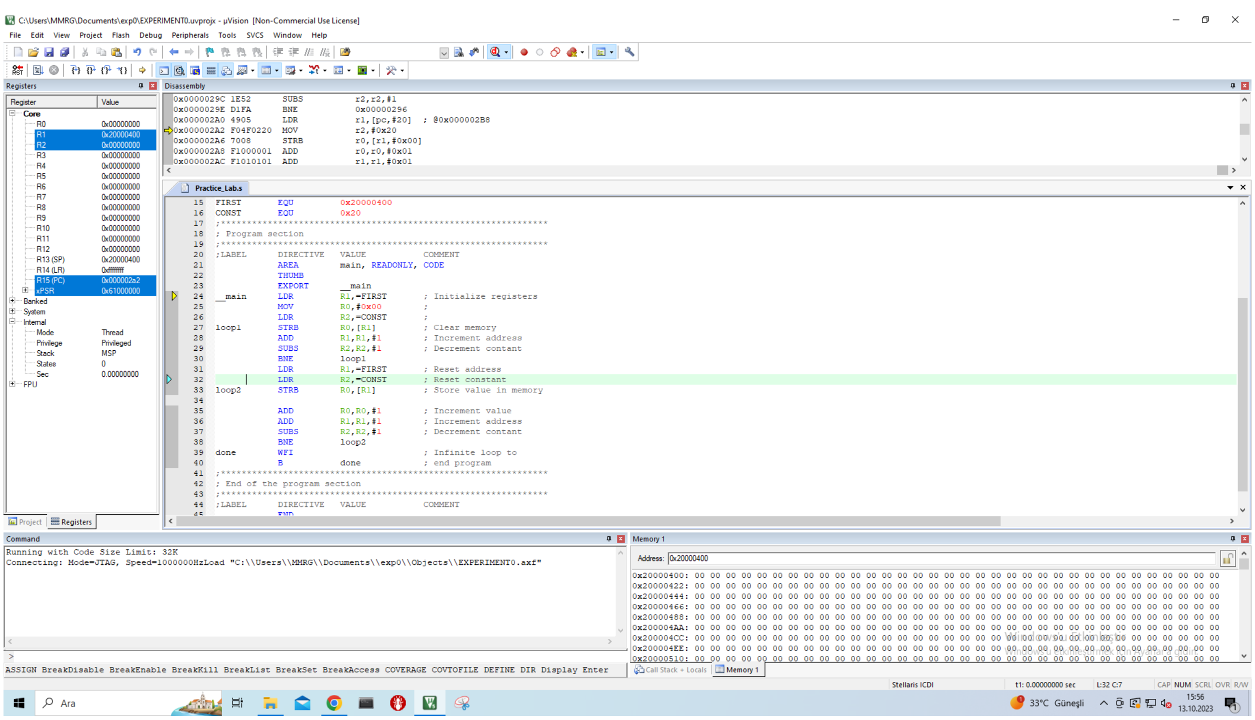

Figure 2. *Practice_Lab.s* assembly code

QUESTION 2

(15 %) Build, run and understand *Program_Directives.s*. You have to add *OutStr.s* to your project folder. Put a screenshot of Keil in your report, showing the contents of the modified memory locations in Memory Window in the debugger.

At the beginning, R1 is loaded with the value 0x20000400. Again, the purpose of this operation to use the value stored in R1 as a pointer. The HEX value 0x10 is stored in R2, which corresponds to 16 in decimal.

In loop1, the value in R0 is stored in the address stored in R1. R0 is incremented by 1. R1 points the next address (R1 is incremented). R2 is decremented by 1. The loop continues **16** times, until R2 is equal to zero, like the loops in Question 1. The status of the registers and the data written in memory after the execution of **loop1** can be observed from Figure 3 and 4.

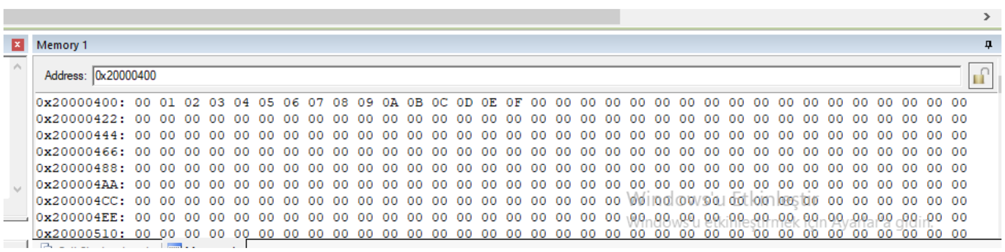

Figure 3. Memory after Loop 1

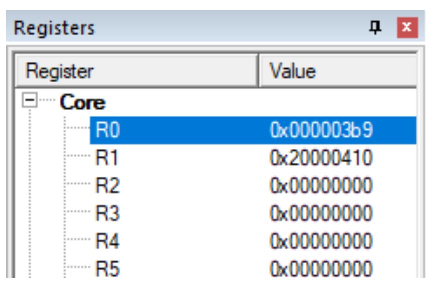

Figure 4. The Status of Registers after at Loop 1

When loop1 ends, R0 is loaded with the value corresponds to the message (MSG) in ASCII. R1 is loaded with the initial address value. OutStr prints the message **"Copying table"** (visible through Termite after configuring the port and baud rate).

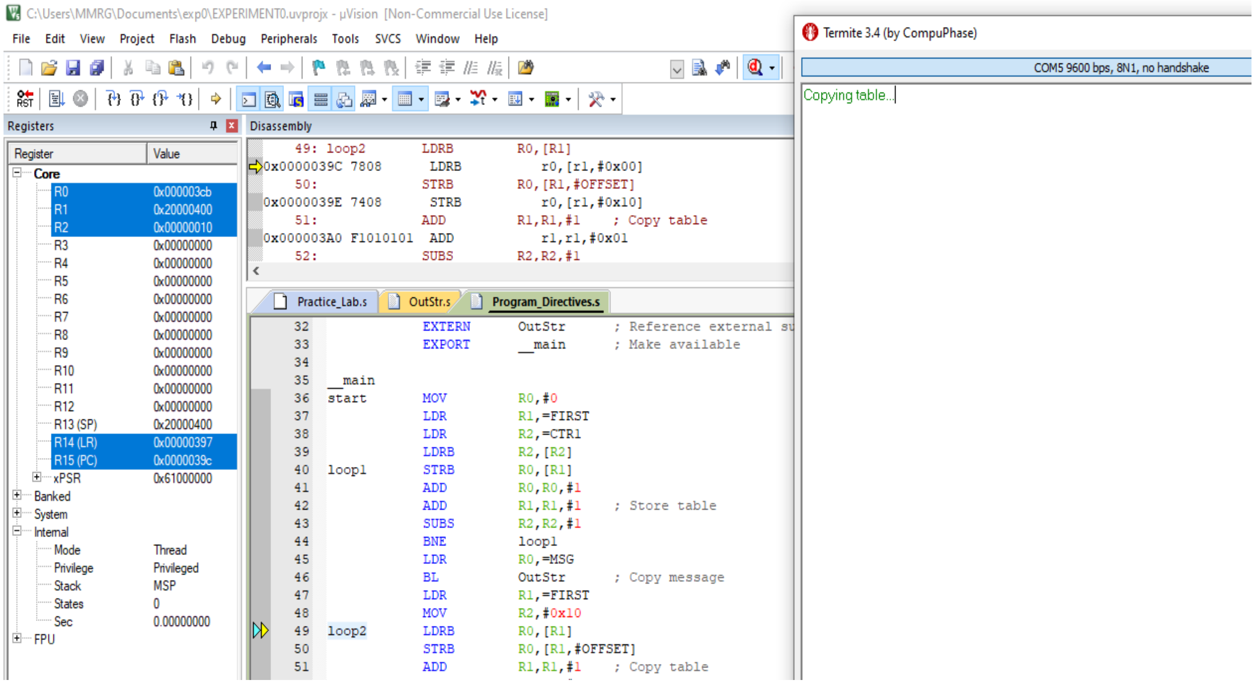

Figure 5. *Program_Directive.s* assembly code

In loop2, the process is similar. However, this time, the value from the initial address is stored in R0 and stored in memory with an offset value. In other words, the table is copied. This offset value is equal to 16 in HEX. Hence, the table will be copied with this exact offset value, starting from [R1] + OFFSET address which is **0x20000410**. At the end of loop2, R0 will hold the value 0F and R1 will hold the final address value (0x20000410), since the loop continued for **16** cycles. The status of the registers and the data written in memory after the execution of **loop2** can be observed Figure 6 and 7.

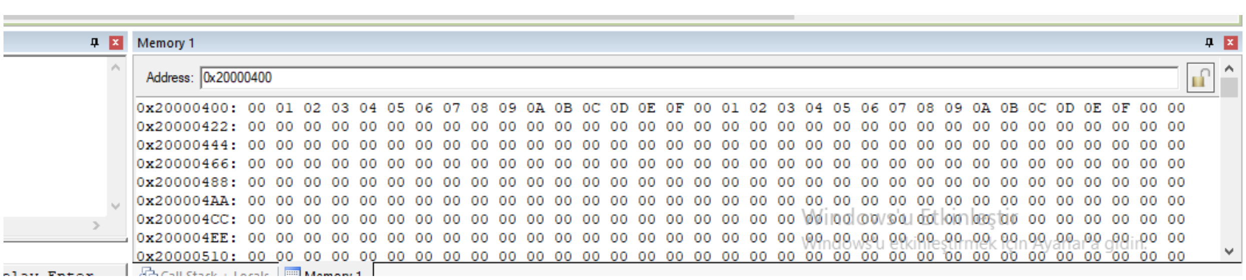

Figure 6. Memory after Loop 2

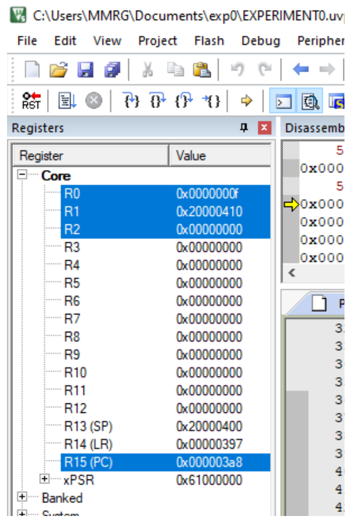

Figure 7. The Status of Registers after at Loop 2

QUESTION 3

(25 %) Make the following modifications on *Program_Directives.s*. This time you will create and copy a different table.

- The table is created starting at 0x2000.0680
- The table starts with 0x00 and goes like this:

0x2000.0680	0x00
0x2000.0681	0x00
0x2000.0682	0x01
0x2000.0683	0x01
0x2000.0684	0x02
0x2000.0685	0x02
...	...
0x2000.0—	0x0A
0x2000.0—	0x0A

Which means, the numbers are repeated twice (Until 0x0A).

- Copy the contents of this new table and paste to the end of itself. Note that the length of this table is different from that in *Program_Directives*.

Put a screenshot of Keil in your report, showing the contents of the modified memory locations in Memory Window in the debugger.

```
;LABEL      DIRECTIVE   VALUE       COMMENT
OFFSET      EQU         0x1
FIRST       EQU         0x20000680
;***************************************************************
; Directives - This Data Section is part of the code
; It is in the read only section  so values cannot be changed.
;***************************************************************
;LABEL      DIRECTIVE   VALUE       COMMENT
            AREA        sdata, DATA, READONLY
            THUMB
CTR1        DCB         0x0B
MSG         DCB         "Copying table..."
            DCB         0x0D
            DCB         0x04
;***************************************************************
```

```
__main
start       MOV         R0,#0
            LDR         R1,=FIRST
            LDR         R2,=CTR1
            LDRB        R2,[R2]
loop1       STRB        R0,[R1]
            ADD         R0,R0,#1
            ADD         R1,R1,#2     ; Store table
            SUBS        R2,R2,#1
            BNE         loop1
            LDR         R0,=MSG
            BL          OutStr       ; Copy message
            LDR         R1,=FIRST
            MOV         R2,#0x0B
loop2       LDRB        R0,[R1]
            STRB        R0,[R1, #OFFSET]
            ADD         R1,R1,#2     ; Copy table
            SUBS        R2,R2,#1
            BNE         loop2
            B           start
;***************************************************************
```

Figure 8. Modified *Program_Directive.s* assembly code

Program_Directives.s file is modified as it can be seen from the Figure 8. OFFSET value is changed to 0x1. Since the numbers will repeat just after and only once, the OFFSET value is equal to 1. FIRST value which points the initial address is changed to 0x20000680. CTR1 value is changed to 0x0B. This value determines the loop amount. Since the value written in memory is incremented by 1 at each cycle, the final value that is written will be CTR1-1. In other words, when CTR1 is set to 0x0B, the final value written will be 0x0A. In loop1 and loop2, ADD R1, R1, #2 is written since there must be one empty address left, so that the repetitive values can be written in between the previously written ones. Before loop2, 0x0B value is stored in R2 instead of 0x10. The status of registers and the data written in memory after the execution of **loop1** can be observed Figure 9 and 10.

```
Memory 1
Address: 0x20000680
0x20000680: 00 00 01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 00 09 00 0A 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200006A2: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200006C4: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200006E6: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000708: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000072A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000074C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000076E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000790: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 9. Memory after Loop 1

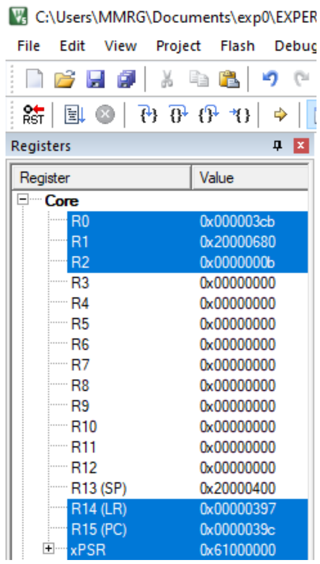

Figure 10. The Status of Registers after at Loop 1

While the code is in loop2, empty addresses is filled with the values stored before them as it can be seen from Figure 11.

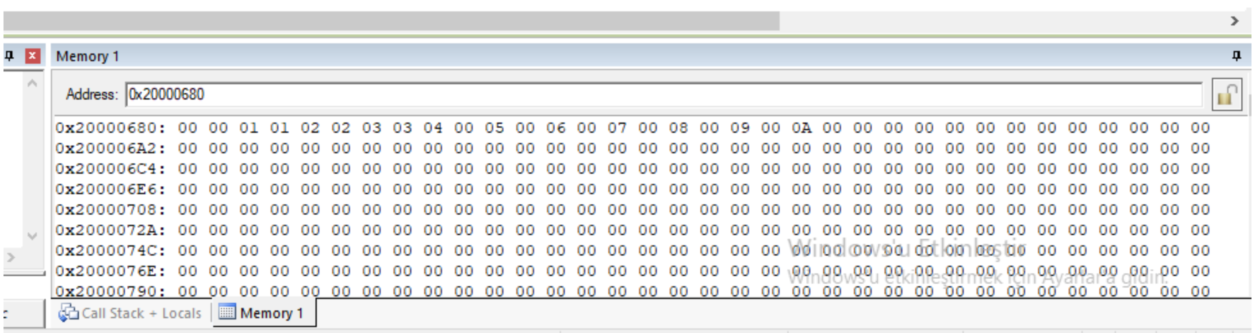

Figure 11. Memory at Loop 2

The status of registers and the data written in memory after the execution of **loop2** can be seen from Figure 12 and 13.

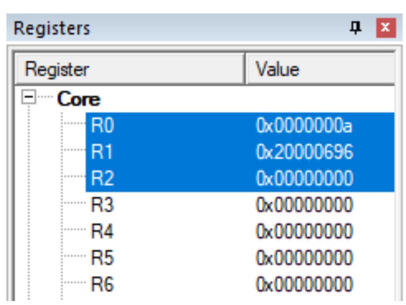

Figure 12. The Status of Registers after at Loop 2

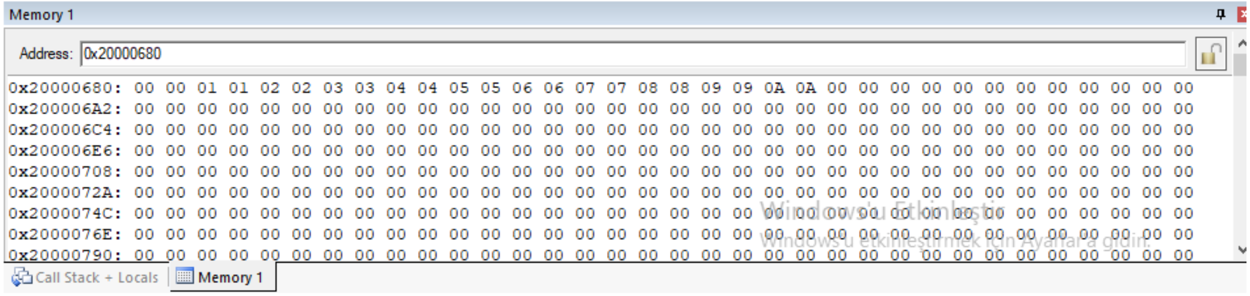

Figure 13. Memory after Loop 2

R0 holds the last value that it has written to the memory. R1 points the [last written address] + 1. R2 holds zero since loop2 ended. Memory is as suggested in the question.

QUESTION 4

(15 %) Write the program given in 1.10. You will have to add *InChar.s*, *OutChar.s* to your project folder.

```
get     BL      InChar
        CMP     R0,#0x20
        BEQ     done
        BL      OutChar
        B       get
done    B       done
```

This program echoes every key you hit using OutChar until you hit SPACEBAR.

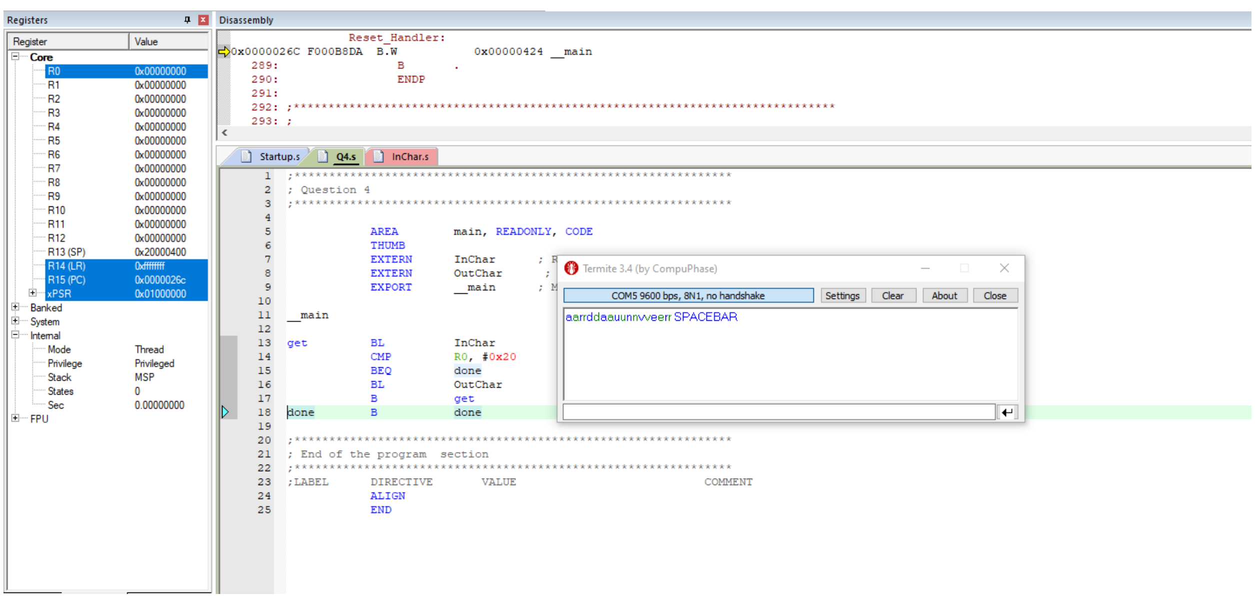

Figure 14. Assembly code to use *InChar.s OutChar.s*

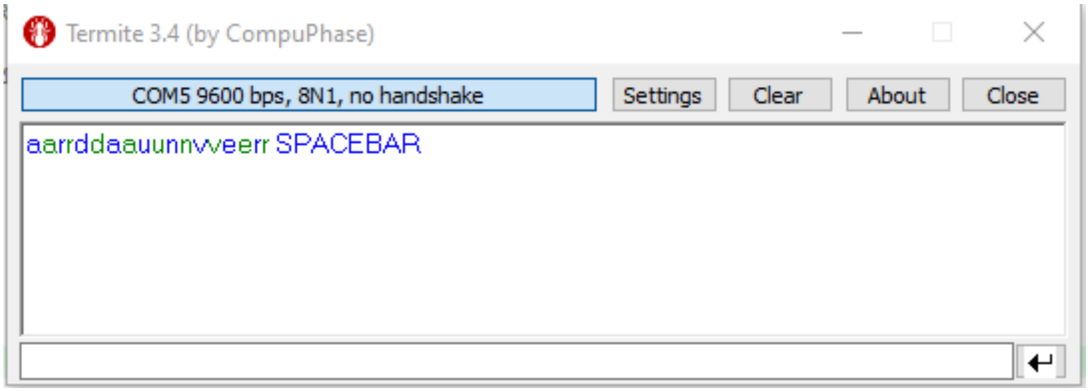

Figure 15. Resultant termite output

As seen from Figure 14 and 15, giving an input of 0x20 (which corresponds to a spacebar input in ASCII) led to the termination of the echo process. I typed my first and last name and then gave a spacebar input. Then I wrote SPACEBAR, **program did not echo the letters**.

QUESTION 5

(15 %) Build, run and understand *Program_Directives.c*. You have to add *OutStr.s* to your project. Put a screenshot of Keil in your report, showing the contents of the modified memory locations in Memory Window in the debugger.

Program_Directives.c consists of two main parts. The "setup" part where the variables are declared and the "loop" part which continues to loop until it is manually stopped. The image below shows the Termite outputs. As in Question 2 where the Assembly code is utilized, this program achieves the same exact task in C programming language. Copying table with an offset. However, I could not find the written addresses. One idea was checking the addresses that is stored in the registers. But it did not work. C compiler apparently finds a suitable address location which the data can be written to.

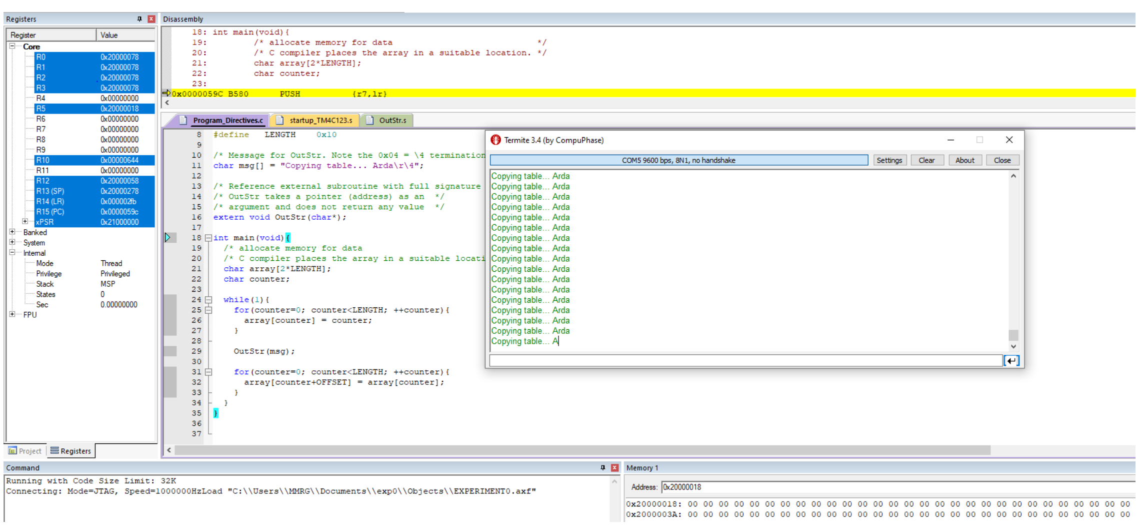

Figure 16. C program to use *Program_Directives.c with adding OutStr.s*

QUESTION 6

(15 %) Rewrite the program given in 1.10 in C language. You will have to add *InChar.s*, *OutChar.s* to your project.

As we did in Question 4, Q6.c program achieves the same task as it can be seen from below. Program stays in the while (1) loop until the spacebar input is provided. In the while loop, each character input is echoed through Termite. It can be seen from Figure 17.

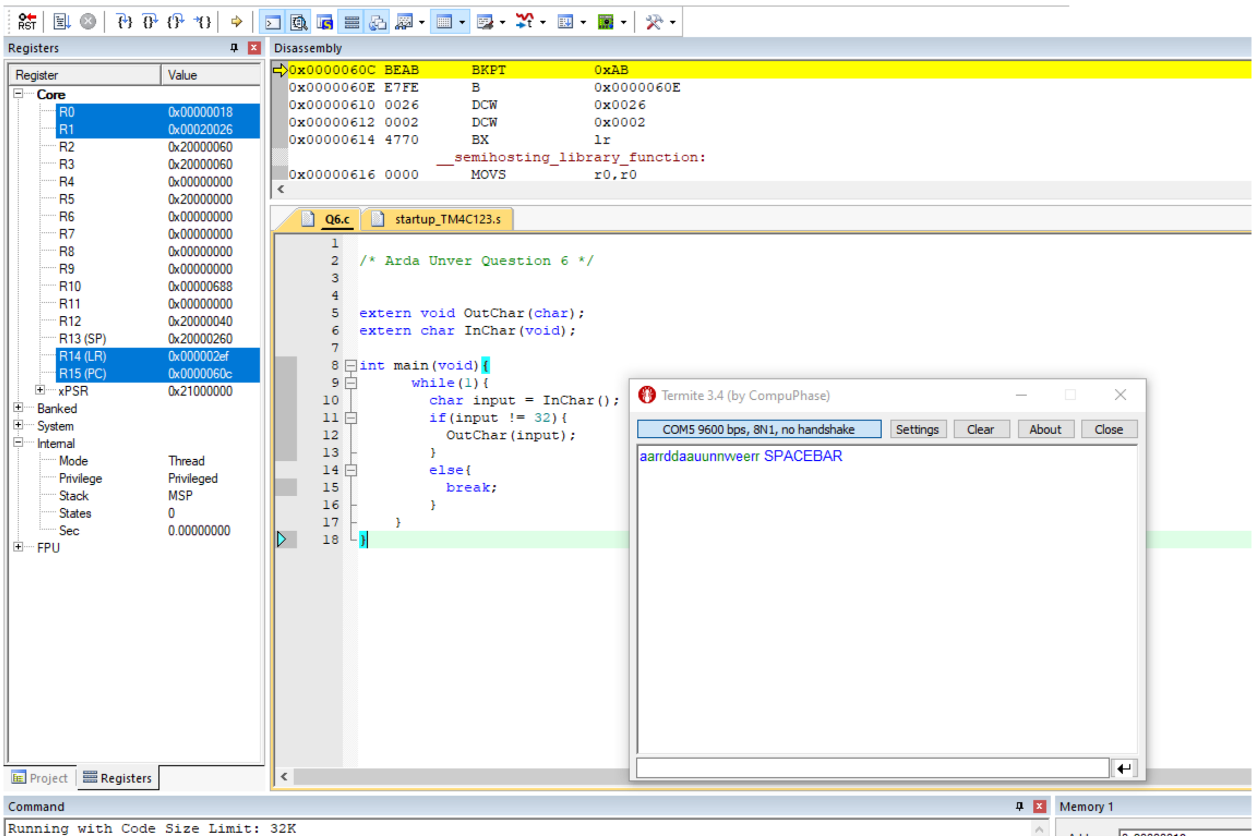

Figure 17. C program and Termite Output