Preliminary work

EE 447: Lab #0

Introduction to Microprocessors Laboratory
with Assembly Programming

Berkay İPEK

2304814 – Sec.2

Question 1)

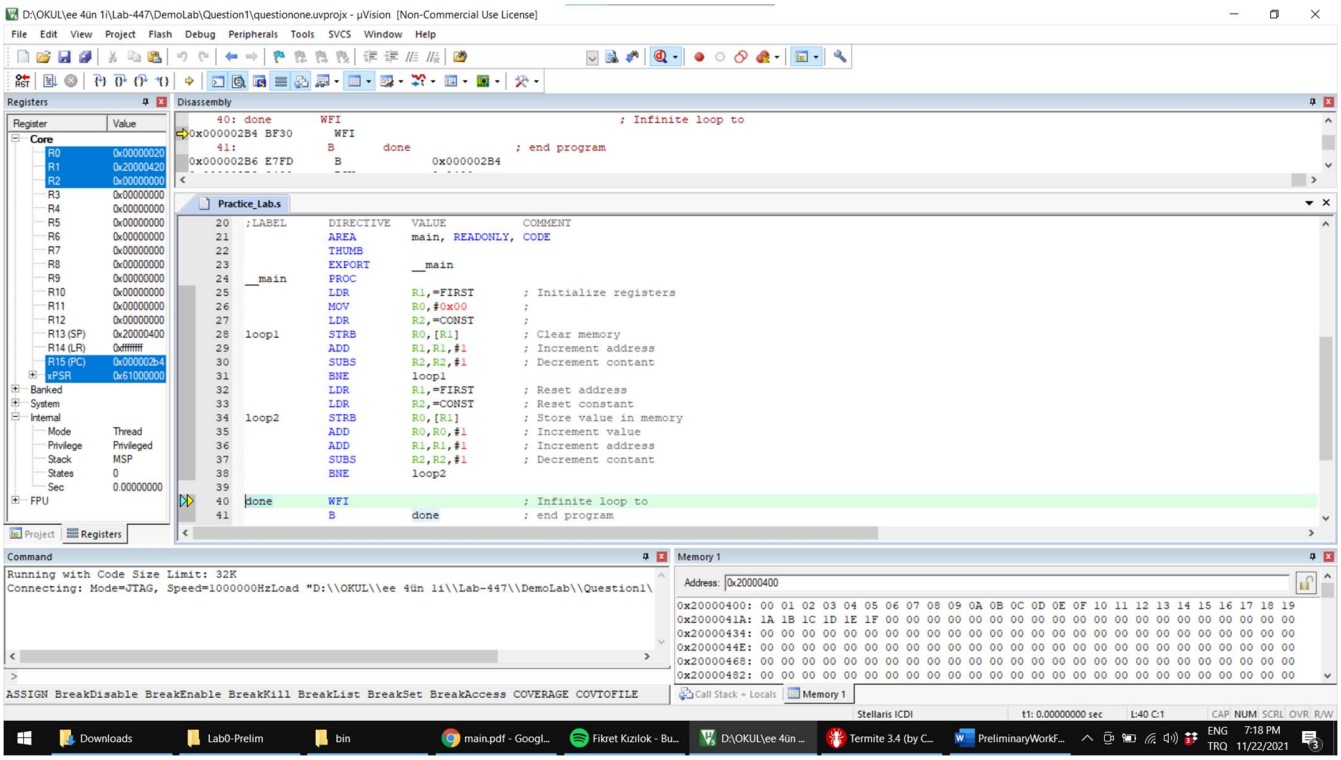

Figure 1: Debugging session for PractiseLab.s file

What this assembly code does is that firstly clearing memory between the address of R1 ([R1]) and the address of R1 + 0x20 ([R1,#0x20]) then placing the hexadecimal numbers in the order of increasing by one (see bottom part, Memory 1, of the figure 1.).

Question 2)

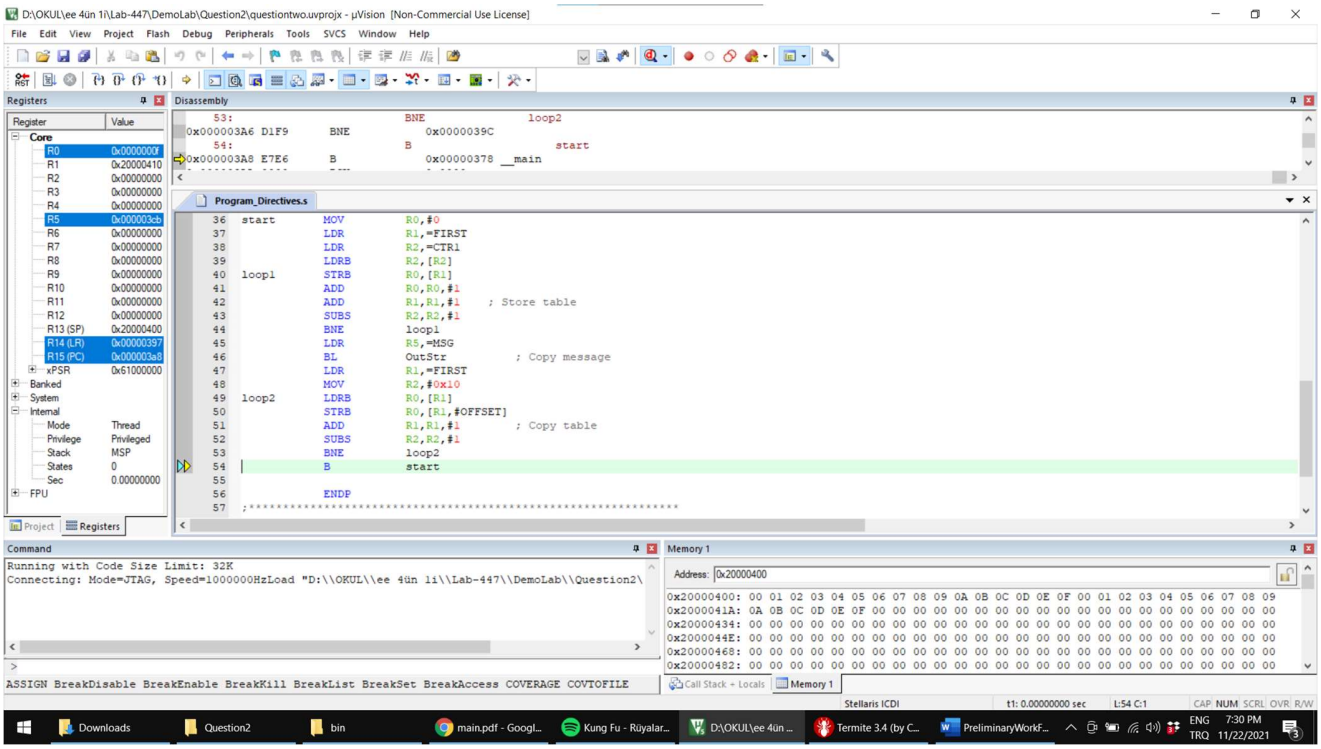

Figure 2: Debugging session for Program_Directives.s file

Explaining of this script can be done by introducing two different for loops.

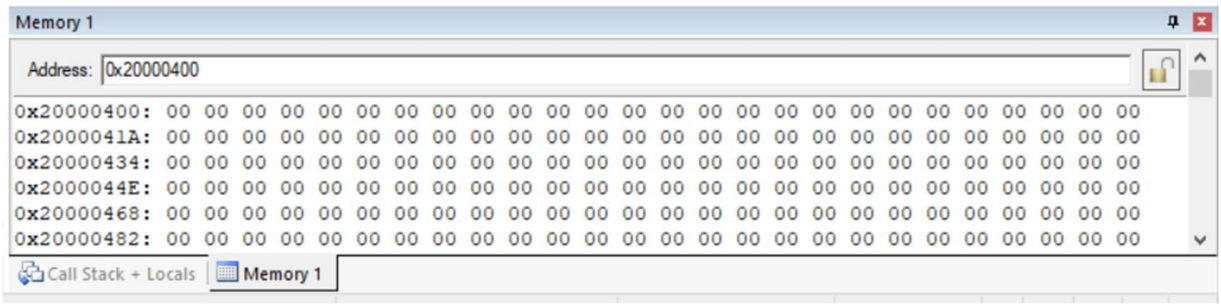

Figure 3: Memory part that shows the address of 0x2000400 before running

```
__main      PROC
start       MOV         R0,#0
            LDR         R1,=FIRST
            LDR         R2,=CTR1
            LDRB        R2,[R2]
loop1       STRB        R0,[R1]
            ADD         R0,R0,#1
            ADD         R1,R1,#1     ; Store table
            SUBS        R2,R2,#1
            BNE         loop1
```

Figure 4: The first loop in Program_Directives.s

After running this part, memory at the address of R1 will be changed. It will write by starting value as 00. In the next iteration, due to changes in R1 and R0, 01 will be written into 0x2000401. Since it is a for loop with the iteration value of R2, it will continue until 0F is written. This is the reason why the length of table is 0x10.

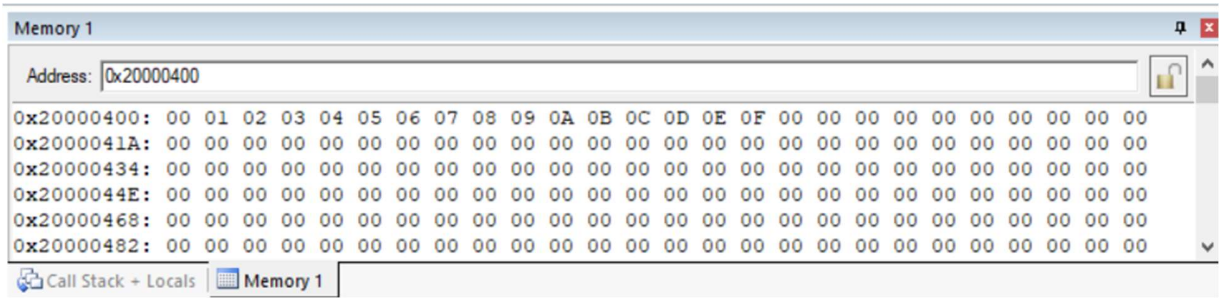

Figure 5: Memory part that shows the address of 0x2000400 after the first loop

The 46th line of the code contains external function, which is outstr. Thanks to this function, termite will look like this:

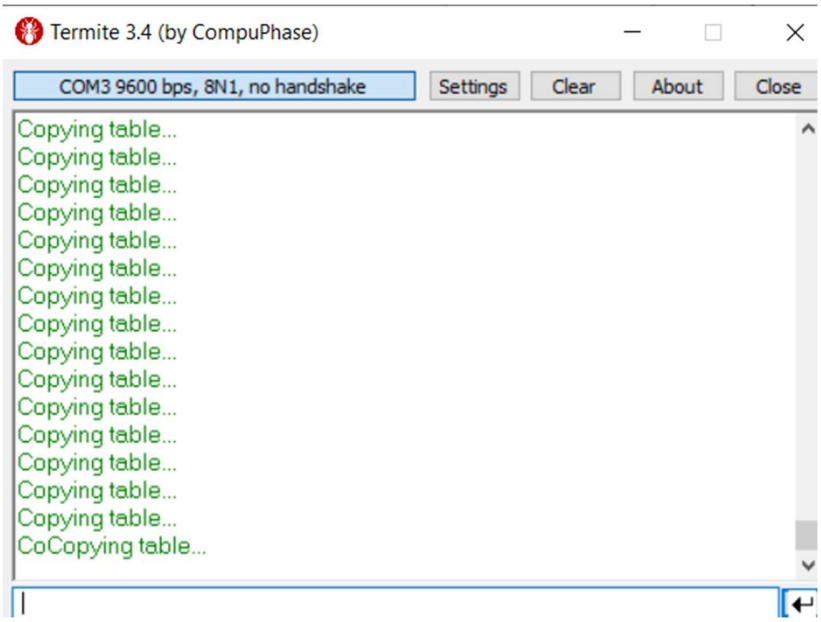

Figure 6: User interface of Termite after the first loop

```
                LDR         R1,=FIRST
                MOV         R2,#0x10
loop2           LDRB        R0,[R1]
                STRB        R0,[R1,#OFFSET]
                ADD         R1,R1,#1          ; Copy table
                SUBS        R2,R2,#1
                BNE         loop2
```

Figure 7: The second loop in Program_Directives.s

What the second loop does is that firstly loading the value at the address of R1 into R0 and then this value is going to be written on the address of R1+OFFSET. Since OFFSET is 0x10 (and also the length of table is 0x10 as explained in the first loop.), the result is looking like in figure 8.

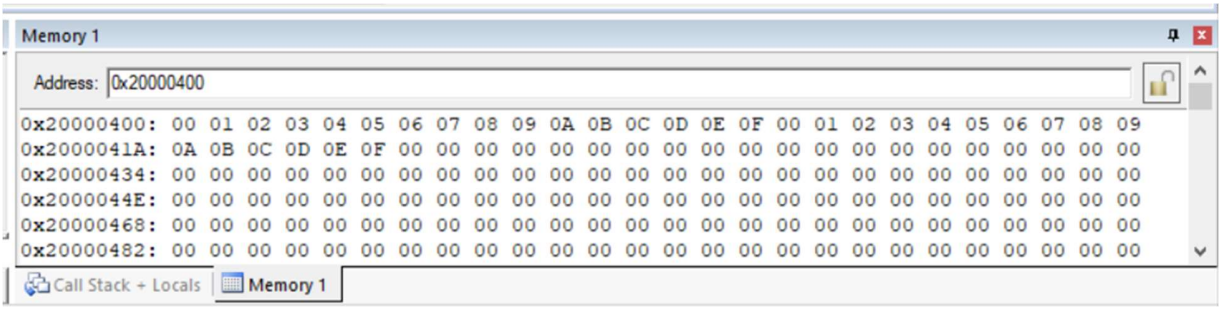

Figure 8: Memory part that shows the address of 0x2000400 after the second loop

Question 3)

```
35  __main      PROC
36  start       MOV         R0,#0x01
37              LDR         R1,=FIRST
38              LDR         R2,=CTR1
39              LDRB        R2,[R2]
40  loop1       STRB        R0,[R1]
41              STRB        R0,[R1,#1]!
42              ADD         R0,R0,#1
43              ADD         R1,R1,#1     ; Store table
44              SUBS        R2,R2,#1
45              BNE         loop1
46              LDR         R5,=MSG
47              BL          OutStr           ; Copy message
48              LDR         R1,=FIRST
49              MOV         R2,#0x20
50  loop2       LDRB        R0,[R1]
51              STRB        R0,[R1,#OFFSET]
52              ADD         R1,R1,#1         ; Copy table
53              SUBS        R2,R2,#1
54              BNE         loop2
55              B           start
```

Figure 9: Modified Program_Directives.s with using OFFSET

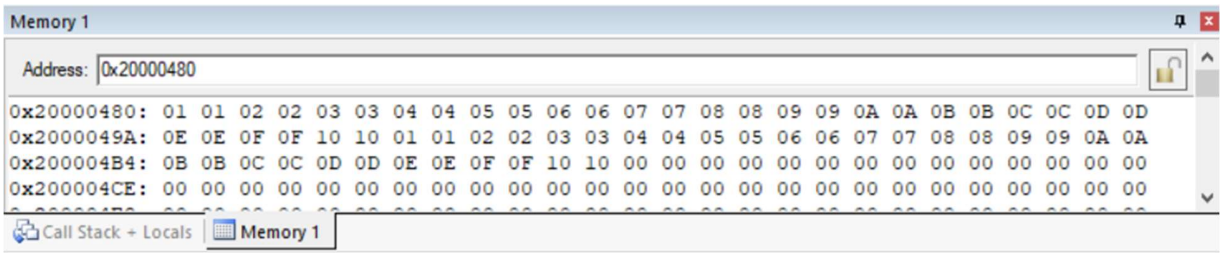

Figure 10: Memory part that shows the address of 0x2000480 after the second loop

Program_Directives_modified.s

```
34
35  __main      PROC
36  start       MOV         R0,#0x01
37              LDR         R1,=FIRST
38              LDR         R2,=CTR1
39              LDRB        R2,[R2]
40  loop1       STRB        R0,[R1]
41              STRB        R0,[R1,#1]!
42              ADD         R0,R0,#1
43              ADD         R1,R1,#1     ; Store table
44              SUBS        R2,R2,#1
45              BNE         loop1
46              LDR         R5,=MSG
47              BL          OutStr           ; Copy message
48              LDR         R3,=FIRST
49              MOV         R2,#0x20
50  loop2       LDRB        R0,[R3]
51              STRB        R0,[R1]
52              ADD         R1,R1,#1         ; Copy table
53              ADD         R3,R3,#1
54              SUBS        R2,R2,#1
55              BNE         loop2
56              B           start
```

Figure 11: Modified Program_Directives.s without using OFFSET

As stated in figure 11, I solved the problem by using two different registers to hold the address and the value. At the line of 48, R3 is the register that will hold the value at that specific address, [R3]. Also, I added increment statement on R3 in the loop2 so that copying can be done properly. As shown in figure 12, memory is in the desired state after running this script, which means modification on Program_Directives.s file is worked.

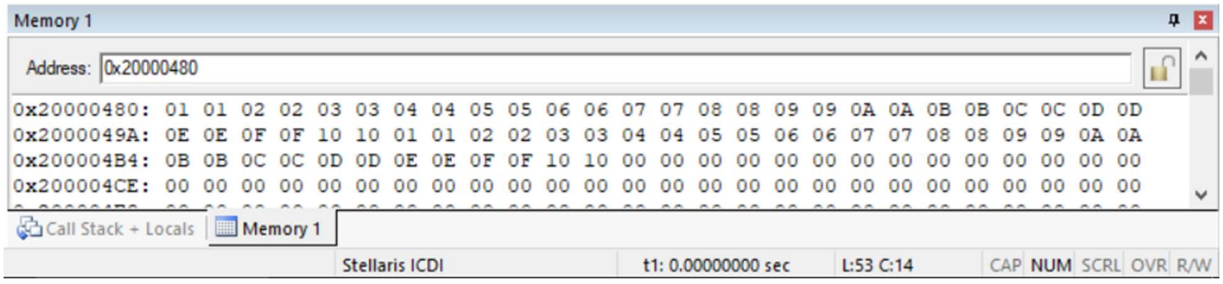

Figure 12: Memory part that shows the address of 0x2000480 after the second loop

Question 4)

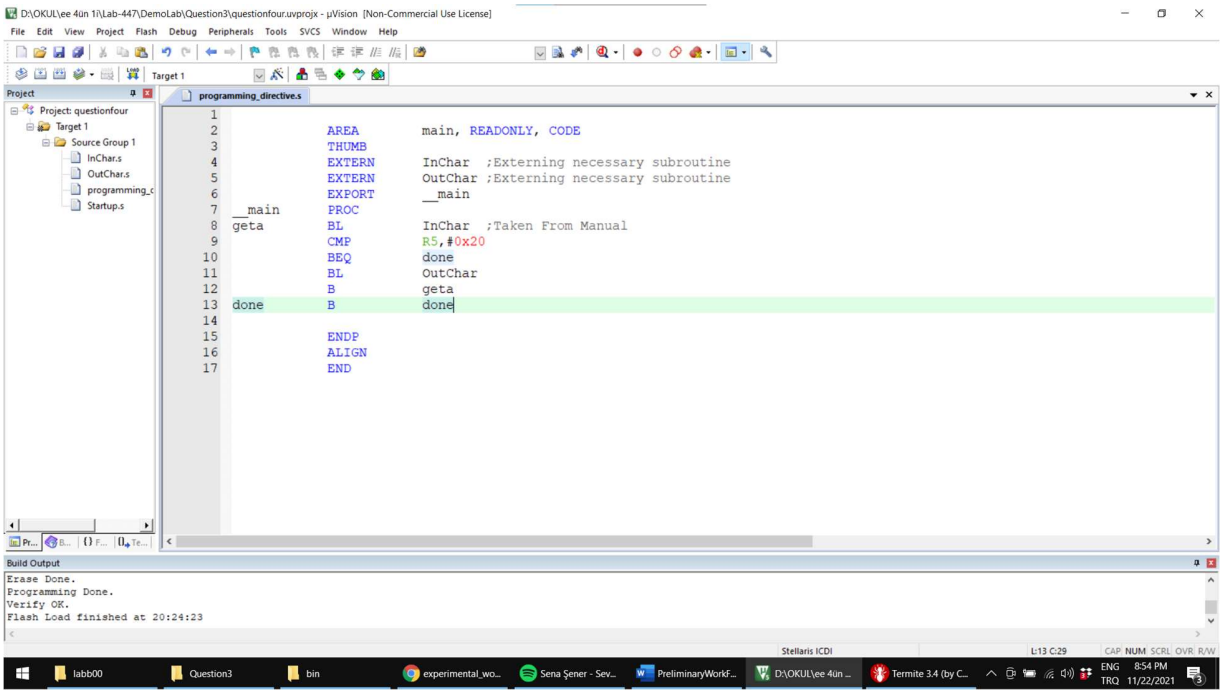

Figure 13: The code for Monitor Utility Subroutines

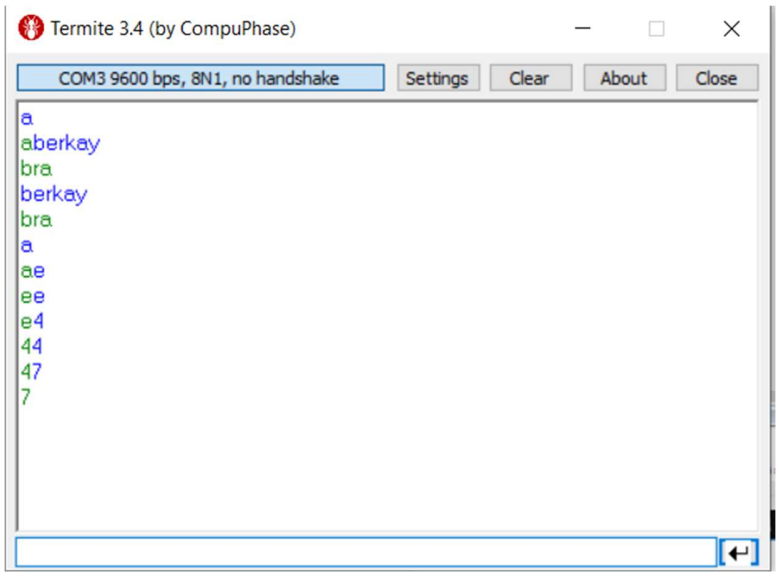

Figure 14: Results for the code in figure 14

```
;*****************************************************************
; Program_Directives.s
; Copies the table from one location
; to another memory location.
; Directives and Addressing modes are
; explained with this program.
;*****************************************************************
;*****************************************************************
; EQU Directives
; These directives do not allocate memory
;*****************************************************************
;LABEL      DIRECTIVE   VALUE       COMMENT
OFFSET      EQU         0x20
FIRST       EQU         0x20000480
;*****************************************************************
; 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         0x10
MSG         DCB         "Copying table..."
            DCB         0x0D
            DCB         0x04
;*****************************************************************
; Program section
;*****************************************************************
;LABEL      DIRECTIVE   VALUE       COMMENT
            AREA        main, READONLY, CODE
            THUMB
            EXTERN      OutStr  ; Reference external subroutine
            EXPORT      __main  ; Make available

__main      PROC
start       MOV         R0,#0x01
            LDR         R1,=FIRST
            LDR         R2,=CTR1
            LDRB        R2,[R2]
loop1       STRB        R0,[R1]
            STRB        R0,[R1,#1]!
            ADD         R0,R0,#1
            ADD         R1,R1,#1    ; Store table
            SUBS        R2,R2,#1
            BNE         loop1
            LDR         R5,=MSG
            BL          OutStr          ; Copy message
            LDR         R3,=FIRST
            MOV         R2,#0x20
loop2       LDRB        R0,[R3]
            STRB        R0,[R1]
            ADD         R1,R1,#1        ; Copy table
            ADD         R3,R3,#1
            SUBS        R2,R2,#1
            BNE         loop2
            B           start

            ENDP
;*****************************************************************
; End of the program  section
;*****************************************************************
;LABEL      DIRECTIVE       VALUE                           COMMENT
            ALIGN
            END

```