MIDDLE EAST TECHNICAL UNIVERSITY

ELECTRICAL-ELECTRONICS ENGINEERING DEPARTMENT

EE447 EXPERIMENT 1 PRELIMINARY WORK

Student Names: Arda ÜNVER - Mustafa YILMAZ

Student ID: 2444081 - 2305746

Submission Date: 30.10.2023

QUESTION 1

(18%) Write a subroutine, **CONVRT**, that converts an m-digit decimal number represented by n **bits** ($n < 32$) in register R4 into such a format that the ASCII codes of the digits of its decimal equivalent would listed in the memory starting from the location address of which is stored in register R5. When printed using OutStr, the printed number is to contain no leading 0s, that is, exactly m digits should be printed for an m-digit decimal number. Before writing the subroutine, the corresponding pseudo-code or flow chart is to be generated.

Some exemplar printings *(righthand side)* for the corresponding register contents *(lefthand side)* are provided below:

```
R4: 0x7FFFFFFF --- 2147483647 (max. value possible)
R4: 0x0000000A --- 10
R4: 0x00000000 --- 0
```

In the first question, we are asked to convert a base 16 number to its base 10 equivalent. After the conversion, we need to display it on the screen via Termite. Our method for this is to constantly divide the number by 10 and save the remaining number in memory. After recording the number, move it one digit to continue dividing by 10 (kind of like dividing by powers of 10). When it comes to a number less than 10, it is necessary to put that number into memory and finish the translation process. After this stage, we start to retrieve the numbers we wrote into memory one by one. We need to reverse the numbers we got. Because of the Little-Endian Format. We also convert the numbers we receive into ASCII format so that we can display them via Termite. Finally, we use OutStr to display the base 10 equivalent of our number on the screen. Assembly code and output examples can be seen from figures below.

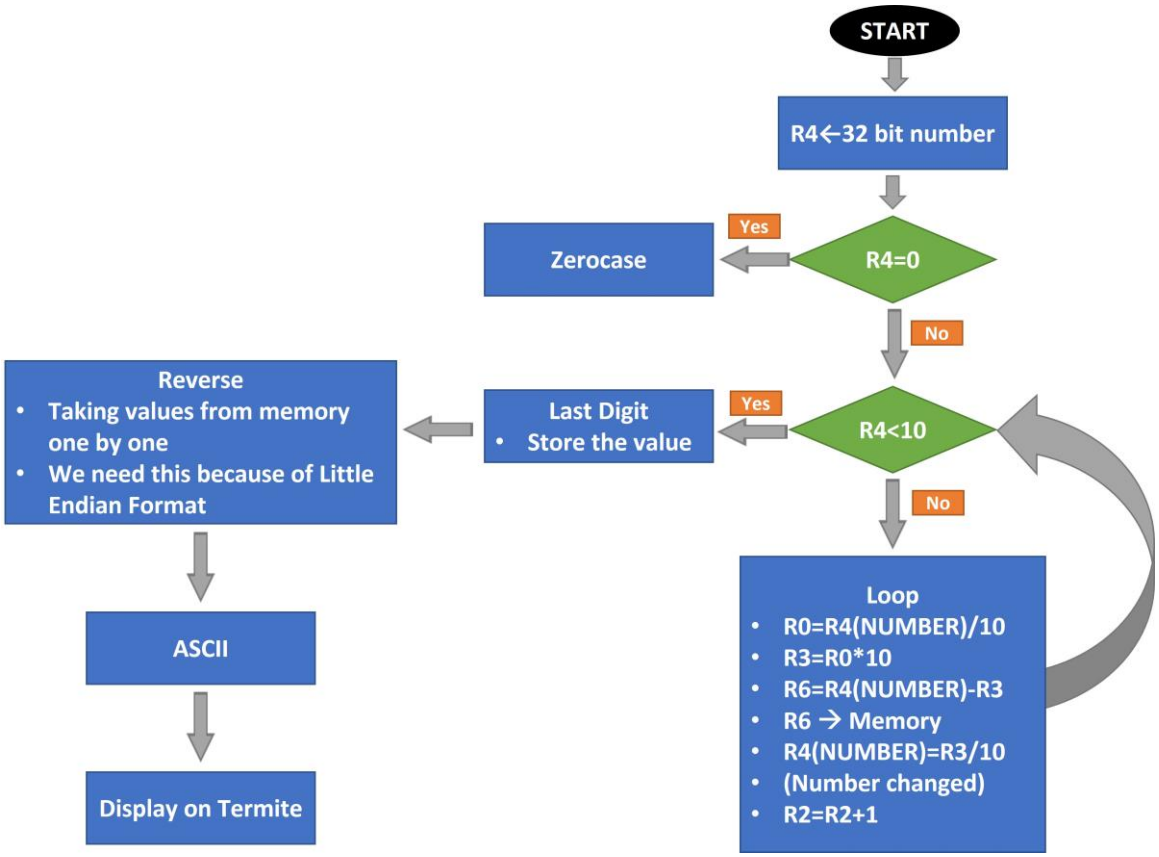

Figure 1. Flow chart of Q1

CONVRT.s | main.s

```
1  ;SYMBOL      DIRECTIVE   VALUE           COMMENT
2  NUMBER       EQU         0x00000000
3  NUM          EQU         0x20000400
4  OFFSET       EQU         0x22
5  ;*****************************************************************
6  ;LABEL       DIRECTIVE   VALUE           COMMENT
7               AREA        main, READONLY, CODE
8               THUMB
9               EXPORT      CONVRT
10
11              ; Try 0x0000000A, 0x0000001A,, 0x00000ABC 0x00000081, 0x7FFFFFFF
12
13
14 CONVRT       PROC
15              CMP         R4,#0           ; Check if number is equal to 0
16              BEQ         zerocase
17              MOV         R2,#1           ; Number of digits
18              MOV         R10,#10         ;Load 10 for Base-10
19
20 ; "loop" converts the HEX number to decimal
21 ; and stores the number digit by digit
22 loop         CMP         R4,#10
23              BMI         lastdigit
24              SDIV        R0,R4,R10       ; Signed divison (R0 = R4(number)/10)
25                                          ; EX: 2 = 26 / 10 stored in R0
26              MUL         R3,R0,R10       ; EX: 20 = 2 * 10 stored in R7
27              SUB         R6,R4,R3        ; EX: 6 = 26 - 20 stored in R7
28              STRB        R6,[R5], #1     ; digit (6) stored in memory and increment the pointer
29              SDIV        R4,R3,R10       ; Signed divison (R0 = R4(number)/10)
30              ADD         R2,#1           ; Increment the number of digits
31              B           loop
32
33 zerocase     MOV         R3,R4
34              ADD         R3,#48
35              STRB        R3,[R5]
36              ADD         R11,#1
37              B           fin
38
39
40 lastdigit    STRB        R4,[R5]         ; Store the last digit
41              LDR         R10,=NUM
42              ADD         R10,#OFFSET
43              MOV         R11,R2          ; Preserve the number of digits
44
45 ; "reverse" reverses the digits so that it is in the correct order
46 reverse      CMP         R2,#0
47              BEQ         prep
48              LDR         R3,[R5]         ; Convert the table to ASCI values
49              STRB        R3,[R10], #1        ; Store the ASCII values and increment the pointer
50              MOV         R3,#0
51              SUB         R5,#1
52              SUB         R2,#1
53              B           reverse
54
55 ; "prep" prepares the register for the "ascii" loop
56 prep         ADD         R5,#1
57              MOV         R2, R11
58
59 ; "ascii" adds 48 to each digit so that the digits are in ASCII form
60 ; and stores the digits in the correct memory address
61 ascii        CMP         R2,#0
62              BEQ         fin
63              LDRB        R3, [R5,#OFFSET]
64              ADD         R3,#48
65              STRB        R3,[R5], #1         ; Store the digit in the correct address
66              SUB         R2,#1
67              B           ascii
68
69 fin          LDR         R5,=NUM     ; R5 points to the address which stores the digits of the number in ASCII form
70              ADD         R6, R5, R11
71              LDR         R11,=0x0D
72              STRB        R11,[R6], #1
73              LDR         R11,=0x04
74              STRB        R11,[R6]
75              LDR         R5,=NUM
76              LDR         R0,=NUM
77
78              BX          LR
79
80              ENDP
81
82              ALIGN
83              END
```

Figure 2. CONVRT Subroutine Assembly Code

CONVRT.s | main.s

```
1  ;SYMBOL      DIRECTIVE   VALUE           COMMENT
2  NUMBER       EQU         0x00000081
3  NUM          EQU         0x20000400
4  OFFSET       EQU         0x22
5  ;*****************************************************************
6  ;LABEL       DIRECTIVE   VALUE           COMMENT
7                AREA        main, READONLY, CODE
8                THUMB
9                DCB         0X0D
10               DCB         0X04
11               EXTERN      InChar
12               EXTERN      CONVRT
13               EXTERN      OutStr
14               EXPORT      __main
15
16 __main
17               ; Number is stored in memory location NUM as a key is pressed
18 start         LDR         R0,=0x0
19               LDR         R2,=0x0
20               LDR         R4,=NUMBER
21               LDR         R5,=NUM      ; initialize a pointer
22               BL          InChar
23               STR         R4,[R5]
24               MOV         R10,#0xA         ; Base-10
25
26               BL          CONVRT
27
28               BL          OutStr
29
30 done          B           start        ; infinite loop
31
32               ALIGN
33               END
```

Figure 3. Main Code of Q1

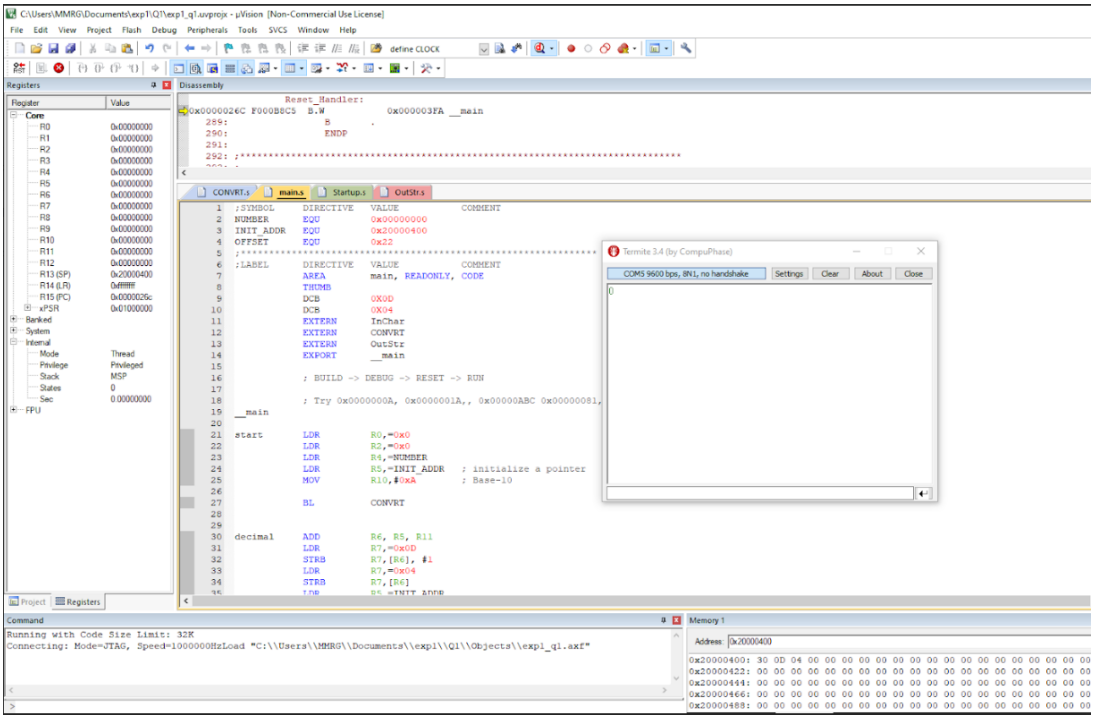

Figure 4. CONVRT Subroutine Example 0x00000000

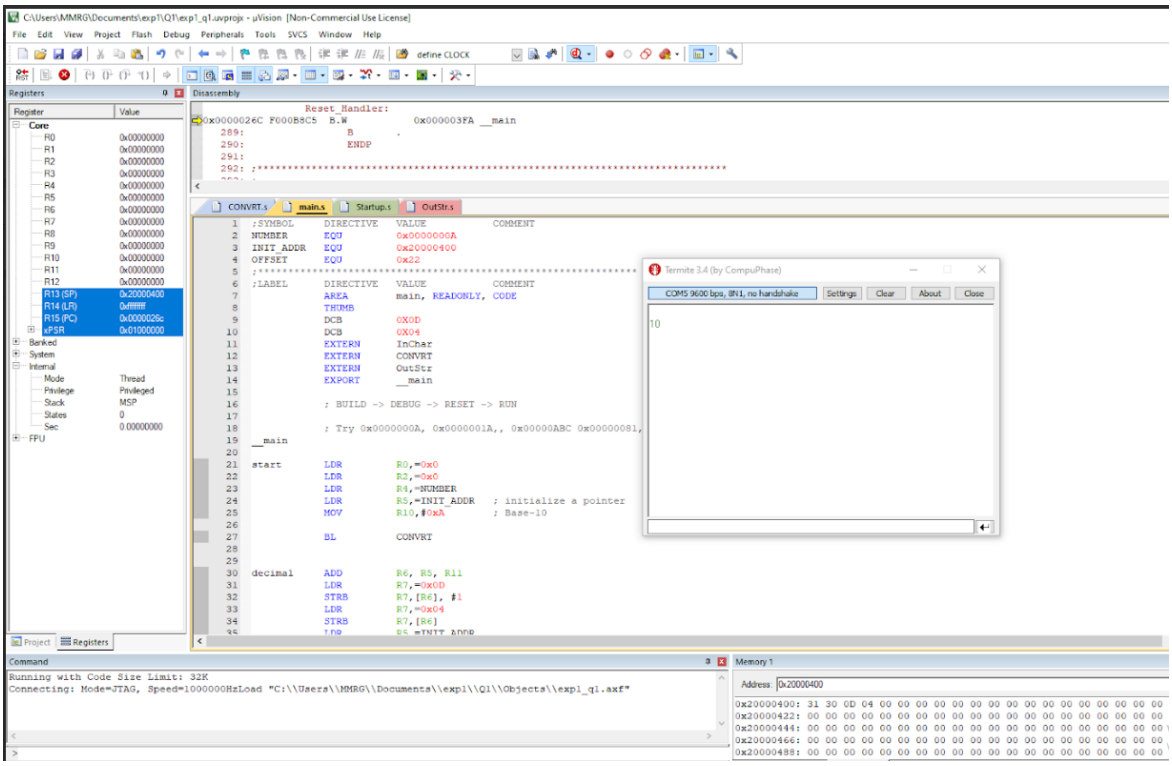

Figure 5. CONVRT Subroutine Example 0x0000000A

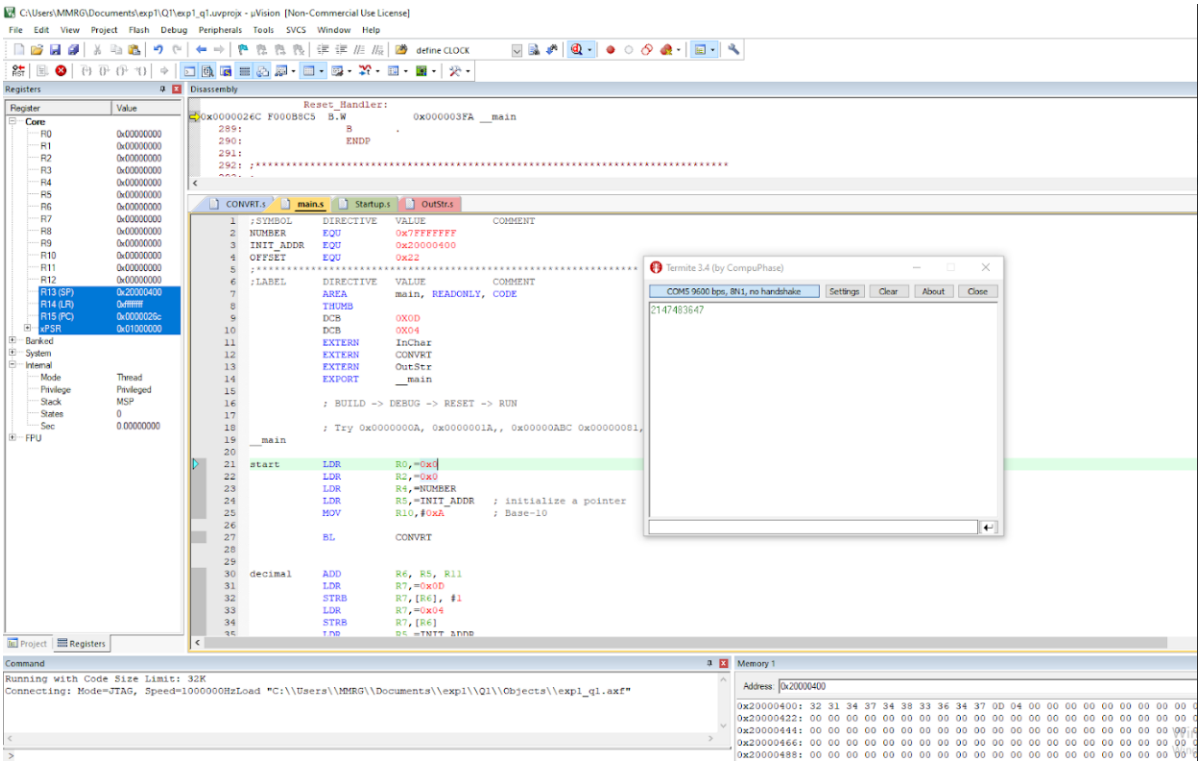

Figure 6. CONVRT Subroutine Example 0x7FFFFFFF

QUESTION 2

(7%) Write a program that, in an infinite loop, waits for a user prompt (any key to be pressed) and prints the decimal equivalent of the number stored in 4 bytes starting from the memory location **NUM**. Note that you may define **NUM** by using proper assembly directives. In this part, you are expected to use the subroutine you are written in Part-1. Explain which arguments should be passed and how.

In the second question, we are asked to write the number converted to base 10 via the subroutine in the first question, regardless of the input received from the user. While doing this, it is expected to be in an infinite loop and our number will be stored using NUM memory location. To achieve what was requested, we wrote and tried the code which you can see in the Figure 7. You can find examples in the figures below.

main.s | CONVRT.s

```
;SYMBOL      DIRECTIVE   VALUE           COMMENT
NUMBER       EQU         0x00000081
NUM          EQU         0x20000400
OFFSET       EQU         0x22
;*****************************************************************
;LABEL       DIRECTIVE   VALUE           COMMENT
             AREA        main, READONLY, CODE
             THUMB
             DCB         0X0D
             DCB         0X04
             EXTERN      InChar
             EXTERN      CONVRT
             EXTERN      OutStr
             EXPORT      __main

__main
             ; Number is stored in memory location NUM as a key is pressed
start        LDR         R0,=0x0
             LDR         R2,=0x0
             LDR         R4,=NUMBER
             LDR         R5,=NUM     ; initialize a pointer
             BL          InChar
             STR         R4,[R5]
             MOV         R10,#0xA        ; Base-10

             BL          CONVRT

             BL          OutStr

done         B           start       ; infinite loop

             ALIGN
             END
```

Figure 7. Main Assembly Code for Q2

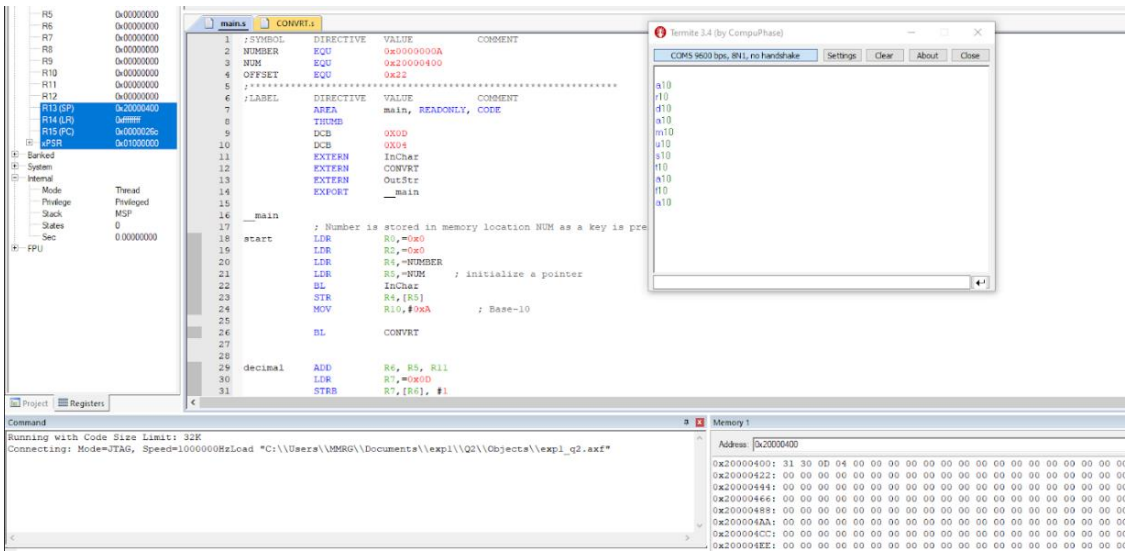

Figure 8. Q2 Example 0x0000000A

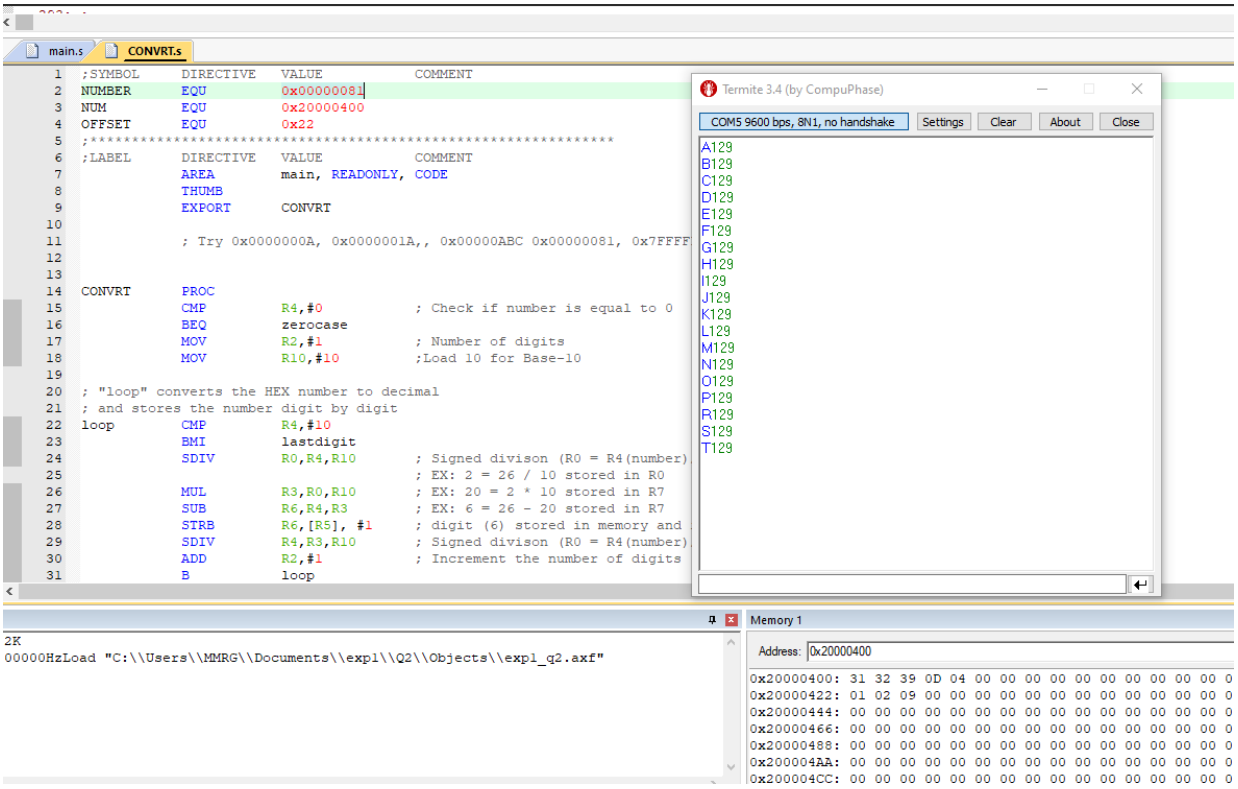

Figure 9. Q2 Example 0x00000081

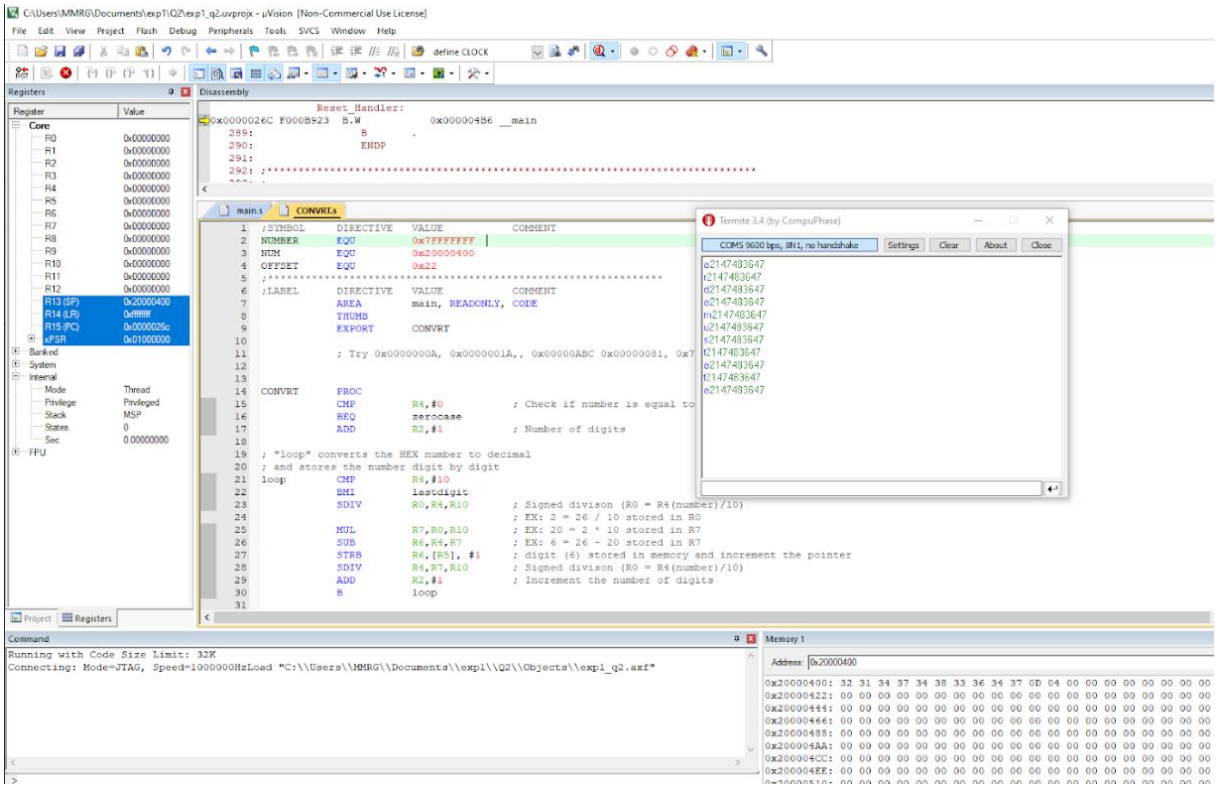

Figure 10. Q2 Example 0x7FFFFFFF

QUESTION 3

(35%) Write a program for decimal number guessing using binary search method. The number is to be an integer in the range $(0, 2^n)$, i.e. $0 < number < 2^n$, where $n < 32$ and n is determined by a user-input. Then, the guessing phase is to be handled through a simple interface where the processor outputs its current guess in decimal base and calculate the next according to the user inputs, **D** standing for down, **U** standing for up, or **C** standing for correct. To fulfill the requirements given above, include the subroutine **CONVRT** from the Part-1 in your main program as well as a new subroutine **UPBND** that updates the search boundaries after each guess. Prior to writing the code itself, draw a flowchart of the main algorithm leaving the subroutine parts as black boxes.

In the 3rd question, what we are asked to do is to perform a binary search and find the number predicted by this method. For the program, first the information about how many bits will be received is obtained from the user. Then, a prediction is made based on the number received from the user. Depending on the estimated number, the user directs it with the "U" or "D" commands. If "U" is used, it is understood that the number is greater than the estimate and the lower limit is made equal to the estimate. If "D" is used, it is understood that the number is less than the estimate and the upper limit is made equal to the estimate. These processes are repeated at each step according to the instructions. When the correct answer is reached, the user is expected to give the "C" command. You can see the assembly code and examples written for this purpose in the figures below.

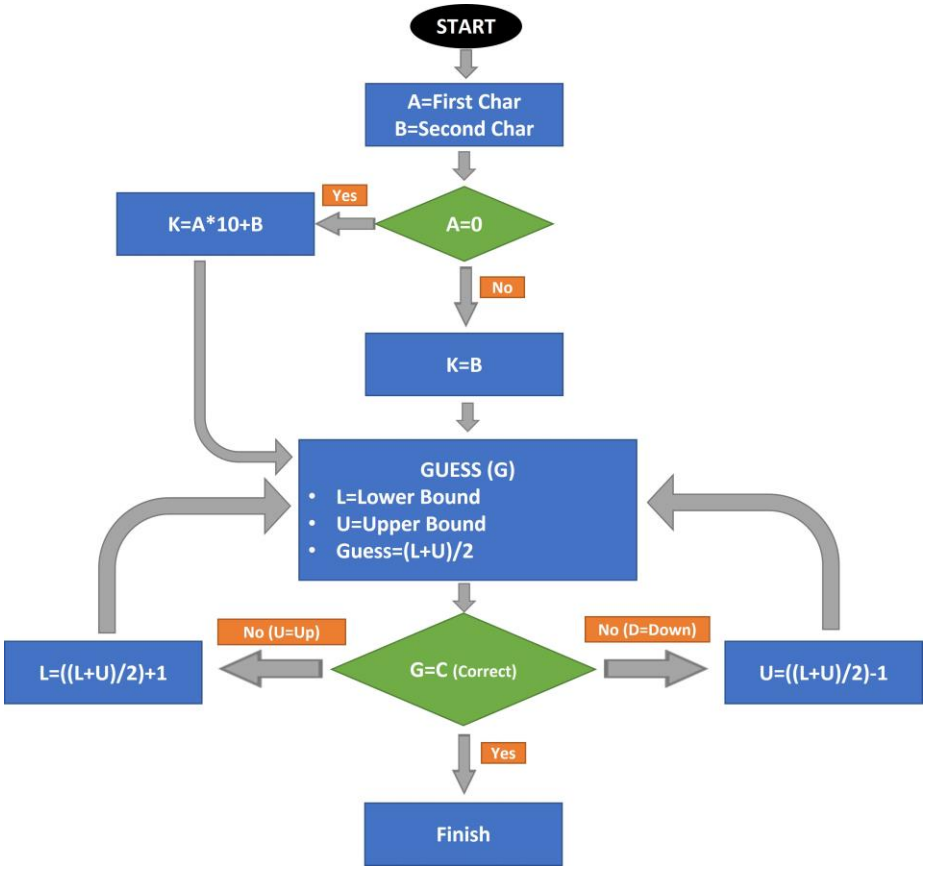

Figure 11. Flow chart of Q3

```
;SYMBOL     DIRECTIVE   VALUE           COMMENT
NUMBER      EQU         0x00000000
NUM         EQU         0x20000400
OFFSET      EQU         0x22
;*****************************************************************
;LABEL      DIRECTIVE   VALUE           COMMENT
            AREA        main, READONLY, CODE
            THUMB
            DCB         0X04
            DCB         0X0D
            EXTERN      CONVRT
            EXTERN      UPBND
            EXTERN      InChar
            EXTERN      OutStr
            EXPORT      __main
__main      PROC
start       LDR         R10,=0xA    ; Load 10 for Base-10
            BL          InChar      ; Get the first element of input (N) from the user
            SUB         R0,#0x30    ; Convert the ASCII value to HEX
            MOV         R11,R0      ; Store the value in R11

            BL          InChar      ; Get the second element of input (N) from the user
            SUB         R0,#0x30    ; Convert the ASCII value to HEX
            MOV         R12,R0      ; Store the value in R12

            MUL         R1,R11,R10  ; Multiply the "Tens" digit by 10
            ADD         R1,R12      ; Add the "Ones" digit
                                    ; R1 holds the value N (input from the user)


            SUB         R1,#1       ; If N = 8, 0x1 must be shifted left 7 times
            MOV         R9,#1
            LSL         R9,R1       ; First "Guess" Number
            MOV         R1,R9

            CMP         R1,#0       ; If N is zero increment the first guess
            ADDEQ       R9,#1

initbounds  MOV         R7,#0       ; First Lower Bound
            LSL         R8,R9,#1    ; First Upper Bound
            B           output



feedback    BL          InChar      ; Get input from the user (C or U or D)
            CMP         R0,#0x43    ; Check if the input is (C)orrect
            BEQ         done        ; Break out the loop

            BL          UPBND       ; Else if the input is either (U)p or (D)own, enter UPBND
                                    ; subroutine to change the "Guess" number
            B           output      ; "Guess" number will be printed as output

output      MOV         R4,R9       ; "Guess" number must be stored in R4 before CONVRT
            LDR         R5,=NUM     ; An address must be stored in R5 before UPBND
            BL          CONVRT      ; Convert the HEX number to decimal

            BL          OutStr
            B           feedback

done        B           done        ; End the code
            ALIGN
            ENDP
            END
```

Figure 12. Main Assembly Code for Q3

main.s | UPBND.s

```
1  ;SYMBOL      DIRECTIVE    VALUE           COMMENT
2  NUMBER       EQU          0x00000000
3  NUM          EQU          0x20000400
4
5               AREA         upbound, READONLY, CODE
6               THUMB
7               EXPORT       UPBND
8  UPBND        PROC
9
10              CMP          R0,#85      ; Check if the feedback is (U)p
11              BNE          down
12
13 ; Lower and upper bound are represented with R7 and R8, respectively
14
15 ; If the number is greater than the "Guess" Number (R9)
16 ; Lower bound must be updated
17              ADD          R7,R9,#1    ; New Lower Bound = 1 + Old Guess
18              ADD          R9,R8,R7    ; New Guess =
19              LSR          R9,#1       ; (New Lower Bound + Old Upper Bound)//2
20              B fin
21
22 down         CMP          R0,#68      ; Check if the feedback is (D)own
23              SUB          R8,R9,#1    ; New Upper Bound = Old Guess - 1
24              ADD          R9,R8,R7    ; New Guess =
25              LSR          R9,#1       ; (Old Lower Bound + New Upper Bound)//2
26
27 fin          BX           LR
28              ALIGN
29              ENDP
30              END
```

Figure 13. UPBND Subroutine Assembly Code

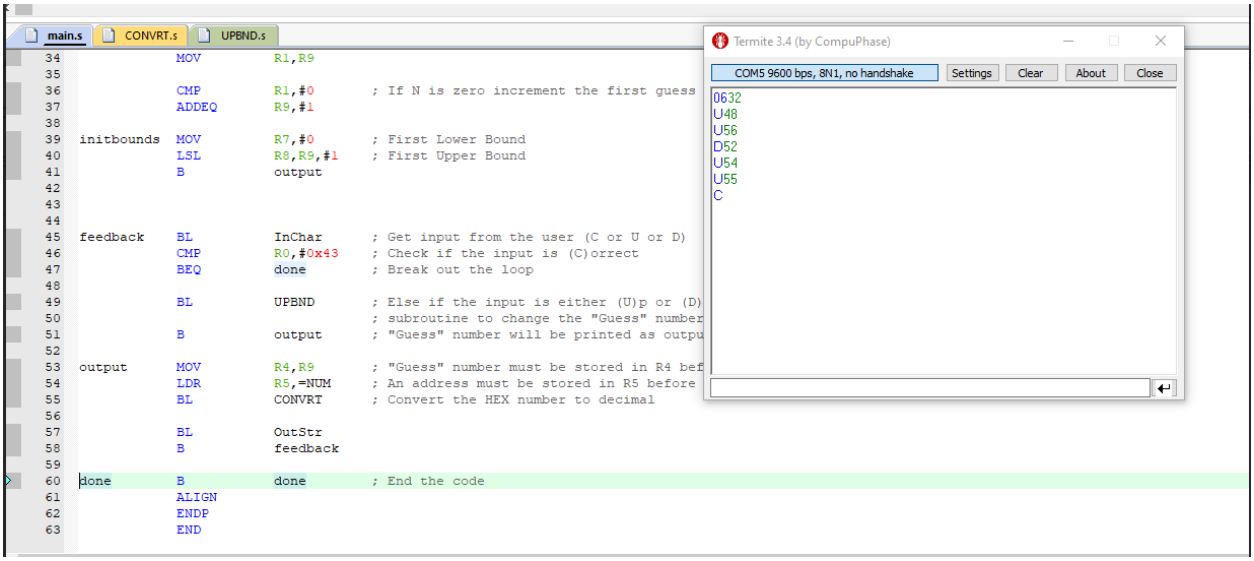

Figure 14. Q3 Example with number 55 and 6 bits

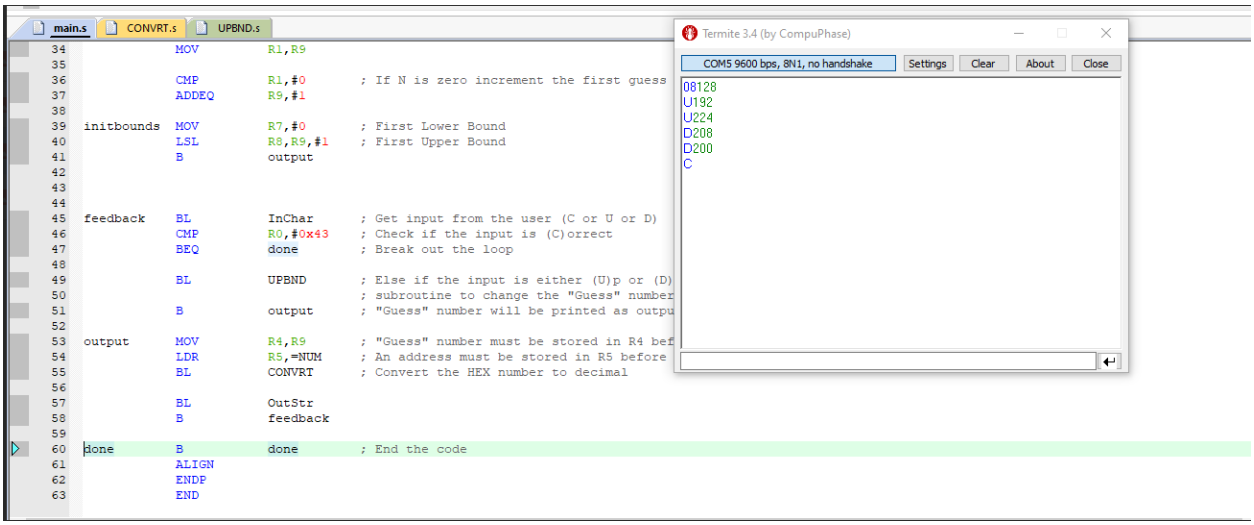

Figure 15. Q3 Example with number 200 and 8 bits

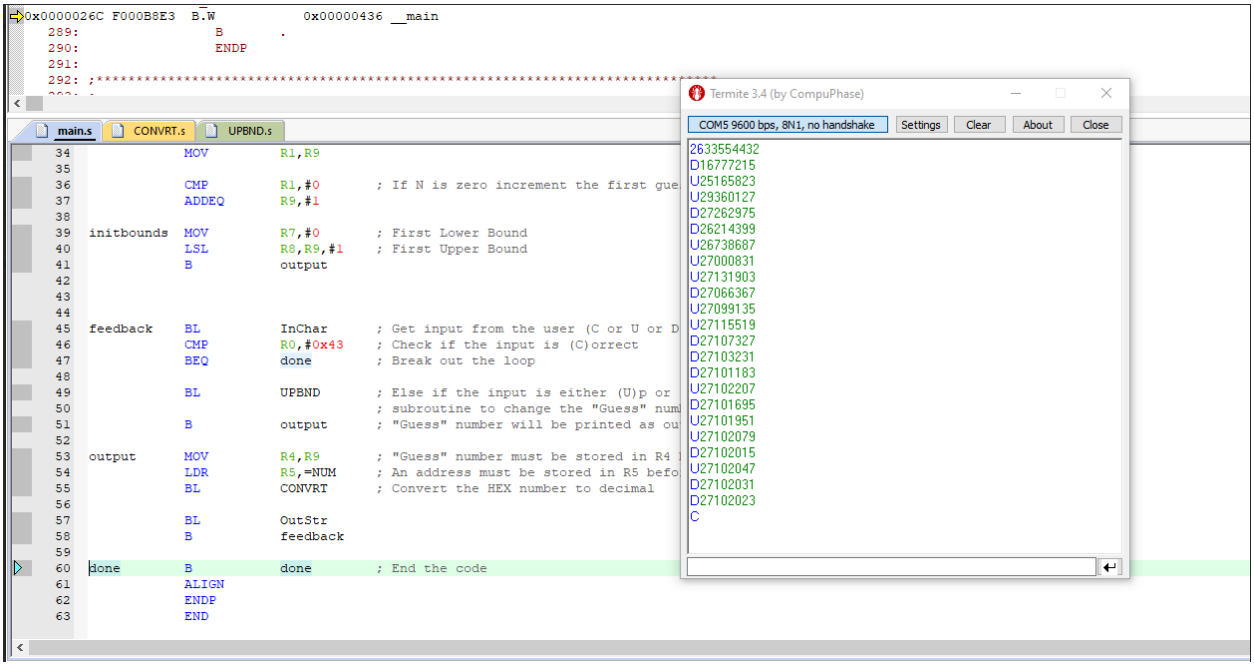

Figure 16. Q3 Example with date 27102023 and 26 bits

QUESTION 4

(40%) You are fighting a cursed monster with a cursed sword. The monster has a specific(3 digit) amount of health and your cursed sword deals a specific(2 digit) amount of damage(Both of which will be provided via *Termite* in decimal base. You can use leading zeros). If you manage to bring the health bar of the monster to zero, the monster dies. But if you bring the monsters health to a negative value, it will heal by half of its initial health and your swords damage will be halved(Use integer division). An example would go like this:

- The monster has 26 initial health and your sword has 7 initial damage.
- You bring the monsters health to -2.
- The monster heals by 26/2=13. Now has 11 health and your sword now deals 7/2=3 damage.
- You bring the monsters health to -1.
- The monster heals once again by 13. Now has 12 health and your sword now deals 3/2=1 damage.
- You bring the monsters health to 0 and it dies.

You will need to write a **recursive** assembly program that takes monster health and sword damage as input and executes the steps needed to bring its health to zero. After that you will need to print the initial and after-heal healths of the monster in reverse order via *Termite* in decimal base(Hint: Use the stack). As for the aforementioned example, the output would look like "12, 11, 26".

For this item, please explain how you solve the problem and draw a flowchart of your algorithm.

In the 4th question, we were asked to write a game with rules and display this game via termite. While displaying, the CONVRT subroutine that we wrote in the first question was used. The game is simply a game of killing a monster with a sword. The monster's health and the damage done by the sword vary depending on the situation. For this, we wrote our code via assembly. How the code works is explained in detail below.

The assembly subroutine, named "fight," is intended to imitate a combat situation in which the player battles a monster with the objective of bringing the monster's health down to zero. The procedure runs recursively until the monster's health reaches or drops below zero. It is constructed using Thumb2 instructions.

The CMP instruction is used to compare the monster's health, which is kept in register R4, to zero at the start of the subroutine. The code instantly branches to the "return" label, so ending the combat, if the monster's health is already at zero.

Before the real fight occurs, the subroutine saves the current context by using the STMDB instruction to push the monster's health (R4) and the link register (LR) onto the stack. It is common practice to keep the current state before calling a function.

The real fighting occurs within a loop named "attack." Within this loop, the player assaults the monster repeatedly by removing a number from R5 from the monster's health (R4). The code examines the monster's health after each assault to see if it is still more than zero. If this is the case, the loop is continued, allowing the player to continue assaulting. If the monster's health hits zero while the loop is running, the code jumps to the "return" label, thereby terminating the combat.

If the monster's health has not yet reached zero, it indicates that the creature can heal itself. In this scenario, the subroutine uses the ADD instruction and right-shifts (LSR) value to boost the monster's health by half of its starting health, which is represented by R2. To balance the fight, the player's sword's damage is halved using unsigned division (UDIV) on R5 and R6.

In R7, a counter is also utilized to keep track of how many times the monster has healed itself. Following the healing and damage computations, the subroutine calls "fight" with the updated monster's health, thereby resuming the combat scenario.

The subroutine utilizes the LDMIA instruction to pop values (R4 and LR) from the stack to restore the prior context before returning. Finally, the "return" label is reached, at which point the subroutine saves the last monster's health number in memory and returns to the caller context, which is indicated by shifting the value in LR to the program counter (PC).

You can find the assembly code and examples written for these operations in the figures below.

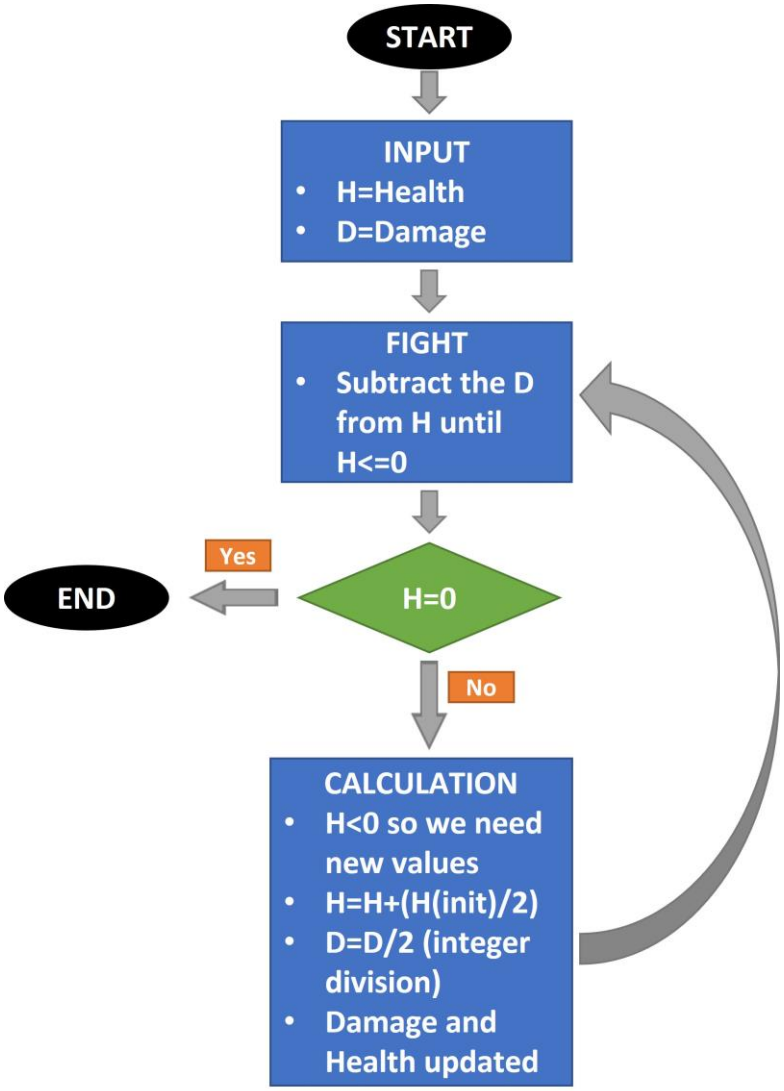

Figure 17. Flow chart of Q4

main.s*

```
;SYMBOL     DIRECTIVE   VALUE           COMMENT
NUMBER      EQU         0x00000000
ADDR        EQU         0x20000400      ; Address to store HEALTH and SWORD
ADDR2       EQU         0x20000424      ; Address to store health values after the fight is over
ADDR3       EQU         0x20000466      ; Address to where CONVRT take place
OFFSET      EQU         0x22            ; OFFSET is used in CONVRT
;*************************************************************
;LABEL      DIRECTIVE   VALUE           COMMENT
            AREA        main, READONLY, CODE
            THUMB
CTR1        DCB         0x0B ;
NEWLINE     DCB         0xA
            DCB         0x0D
            DCB         0x04
MSG1        DCB         "Enter Monster Health:"
            DCB         0x0D
            DCB         0x04
MSG2        DCB         "Enter Sword Strength:"
            DCB         0x0D
            DCB         0x04
MSG3        DCB         "FIGHT!"
            DCB         0x0D
            DCB         0x04
            EXTERN      CONVRT
            EXTERN      InChar
            EXTERN      OutStr
            EXPORT      __main

            ; HEALTH represents the health of the monster
            ; Max HEALTH value = 999 (3E7 in HEX)
            ; SWORD represents the damage of the sword
            ; Max SWORD damage  =  99 (63 in HEX)

__main      PROC
start       MOV         R0,#0
            LDR         R0,=MSG1
            LDR         R1,=ADDR    ; Address to store HEALTH and SWORD
            LDR         R10,=0xA    ; Load 10 for Base-10
            LDR         R11,=0x64   ; Load 100 for Base-100

            BL          OutStr      ; Print "Enter Monster Health:"
            MOV         R6,#3       ; Counter for the "health" loop


            ; "health" loop gets the monster health input from the user

health      CMP         R6,#0       ; Check the counter
            BEQ         hex1        ; Branch if Z is set
            BL          InChar      ; Get input (HEALTH) from the user
            SUB         R0,#0x30    ; Convert the ASCII value to HEX
            MUL         R0,R11      ; Multiply regarding the digit (hundreds, tens, ones)
            UDIV        R11,R10     ; Change the divisor for the next loop
            STR         R0,[R1],#4  ; Store the value in ADDR
            SUB         R6,#1       ; Decrement the counter
            MOV         R0,#0       ; Clear the register
            ADD         R4,#1       ; Count the digits (counter for "hex1" loop)
            B           health      ; Loop again

            ; "hex1" loop sums the stored values
            ; R3 holds the HEALTH value in HEX after the loop

hex1        CMP         R4,#0       ; Check the counter
            BEQ         prep1       ; Branch if Z is set
            SUB         R1,#4       ; Pointer decremented
            LDR         R2,[R1]     ; Digit is loaded
            ADD         R3,R2       ; Digit is added to the sum
            SUB         R4,#1       ; Decrement counter
            B           hex1        ; Loop again

            ; "prep1" loop prepares the registers for the "sword" loop

prep1       MOV         R11,#100    ; Load 100 for Base-100
            ADD         R1,#OFFSET  ; Pointer = ADDR + OFFSET
            MOV         R2,R3       ; Store the HEALTH value in R2
            MOV         R3,#0       ; Clear R3
            MOV         R6,#2       ; Set the counter (SWORD is 2 digits)
            LDR         R0,=NEWLINE
            BL          OutStr      ; Print newline
            MOV         R0,#0
            LDR         R0,=MSG2
            BL          OutStr      ; Print "Enter Sword Strength:"
            MOV         R0,#0       ; Clear R0

            ; "sword" loop gets the sword power input from the user

sword       CMP         R6,#0       ; Check the counter
            BEQ         hex2        ; Branch if Z is set
            BL          InChar      ; Get input (SWORD) from the user
            SUB         R0,#0x30    ; Convert the ASCII value to HEX
            MUL         R0,R10      ; Multiply regarding the digit (hundreds, tens, ones)
```

Figure 18. Main Assembly Code for Q4

```
91              UDIV        R10,R10     ; Change the divisor for the next loop
92              STR         R0,[R1],#4  ; Store the value in ADDR + OFFSET and increment pointer
93              SUB         R6,#1       ; Decrement counter
94              MOV         R0,#0       ; Clear R0
95              ADD         R4,#1       ; Count the digits for the "hex2" loop
96              B           sword       ; Loop again
97
98              ; "hex2" loop sums the stored values
99              ; R3 holds the SWORD value in HEX after the loop
100
101 hex2        CMP         R4,#0       ; Check the counter
102             BEQ         prep2       ; Branch if Z is set
103             SUB         R1,#4       ; Pointer decremented
104             LDR         R7,[R1]     ; Digit is loaded
105             ADD         R3,R7       ; Digit is added to the sum
106             SUB         R4,#1       ; Decrement counter
107             B           hex2        ; Loop again
108
109 prep2       LDR         R0,=NEWLINE
110             BL          OutStr      ; Print newline
111             MOV         R0,#0
112             LDR         R0,=MSG3    ; Print "FIGHT!"
113             BL          OutStr
114             CMP         R12,#1      ; If the flag is set, the fight is over
115             BEQ         prep3       ; Branch to prep3
116             MOV         R0,#0       ; Clear R0
117             MOV         R7,#0       ; Clear R7
118             MOV         R4,R2       ; Current HEALTH
119             MOV         R5,R3       ; Current SWORD
120             MOV         R12,#1      ; Set the flag so that you only fight the monster once
121             MOV         R6,#2       ; R6 will be used for division
122
123             ; R1 points to ADDR + OFFSET
124             ; R2 holds HEALTH, R3 holds SWORD
125             ; R4 holds Current HEALTH, R5 Current holds SWORD
126             ; R6 holds 2 for division
127             ; R10 and R11 hold 1 and 100, respectively
128             ; R12 is used as flag
129
130             ; "fight" is the recursive subroutine, where you fight the monster
131             ; Purpose is to bring the monster's health to zero
132             ; You keep attacking the monster, until it has zero or negative health
133             ; If its health is zero, monster is slaughtered
134             ; If its health is negative, monster is healed by the half of its initial health
135 fight       CMP         R4, #0              ; If monster has 0 health
136             BEQ         return              ; Return
137             STMDB       SP!, {R4,LR}        ; PUSH monster's health and LR to the stack
138 attack      SUB         R4,R5               ; Hit the monster with your sword
139             CMP         R4,#0               ; If its health is above 0
140             BGT         attack              ; Keep attacking
141             CMP         R4, #0              ; If its health is zero
142             BEQ         return              ; Return
143             ADD         R4,R2, LSR #1       ; Monster is healed
144             UDIV        R5,R6               ; Damage of the sword is halved
145             ADD         R7,#1               ; Count the times monster healed
146             BL          fight               ; Recursive call
147             LDMIA       SP!, {R4,LR}        ; POP from the stack
148
149             ; Recursive functions return
150 return      STR         R4,[R1],#2          ; Store the health values of the monster
151             MOV         PC, LR              ; PC <-- LR
152
153 prep3       ADD         R8, R7,#1           ; R8 holds the amount of the health values (count)
154             LDR         R0,=NEWLINE         ; Load the ascii value of newline
155             BL          OutStr              ; Print newline
156             MOV         R0,#0               ; Clear R0
157             LDR         R1,=ADDR2           ; Load the address where the health values are stored
158             MOV         R12,#0              ; Clear R12
159
160 print       CMP         R8,#0               ; Check if count is zero (no more health values)
161             BEQ         done                ; Branch to the end
162             LDR         R1,=ADDR2           ; Load the address where the health values are stored
163             ADD         R1,R12              ; Increment the pointer by 0,2,4,6,8...
164             LDRB        R4,[R1]             ; Load the health value
165             LDRB        R9,[R1,#1]          ; If health is below 255 (0xFF), this byte will be zero
166                                             ; If health is over 255,
167             LSL         R9,#8               ; this byte will be shifted left 8 times (or mult. by 256)
168             ADD         R4,R9               ; and added to the previous byte
169             LDR         R5,=ADDR3           ; Load the address where "CONVRT" will take place
170             BL          CONVRT              ; Convert the digits of the value in R4 to ASCII
171             LDR         R0,=ADDR3           ; Load the address of the converted digits
172             BL          OutStr              ; OutStr prints from the address stored in R0 until 0x0D and 0x04 bytes
173             SUB         R8,#1               ; Decrement the counter
174             ADD         R12,#2              ; Increment of the pointer depends on R12
175             B           print               ; Loop again
176
177 done        B           done                ; End the code
178             ALIGN
179             ENDP
180             END
```

Figure 19. Main Assembly Code for Q4

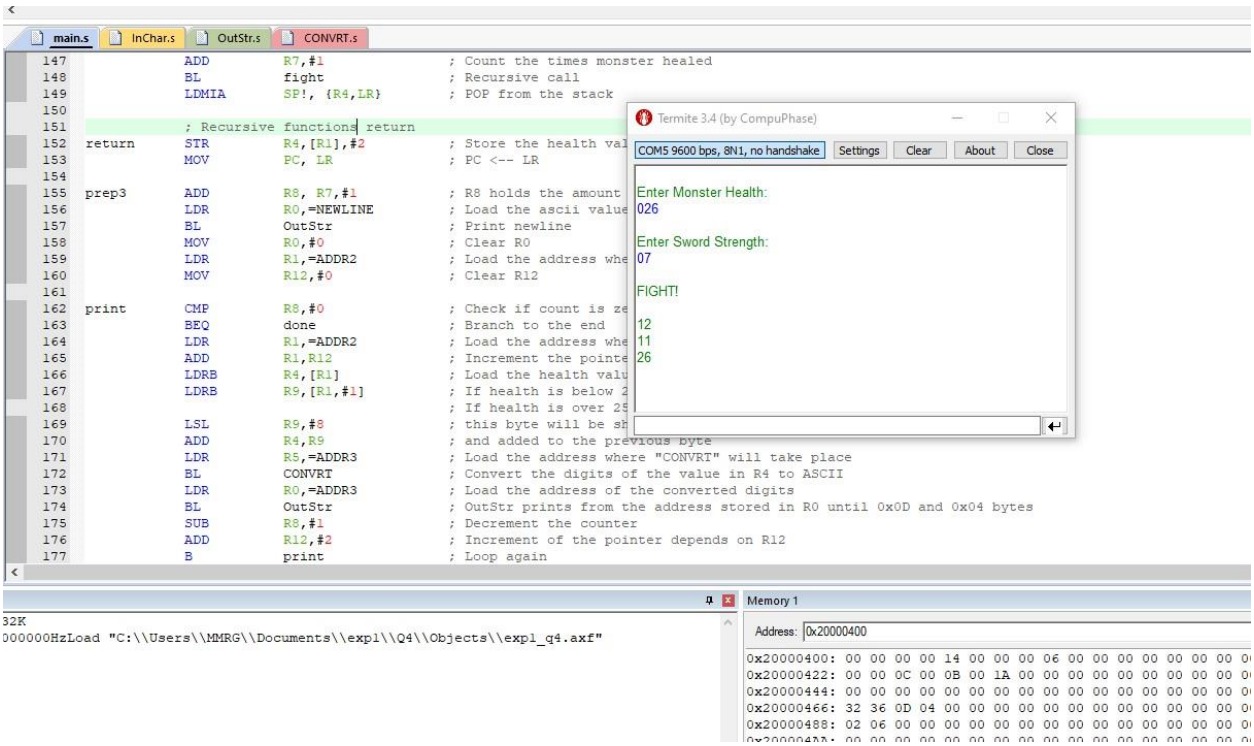

Figure 20. Q4 Example with 26 Health and 7 Damage

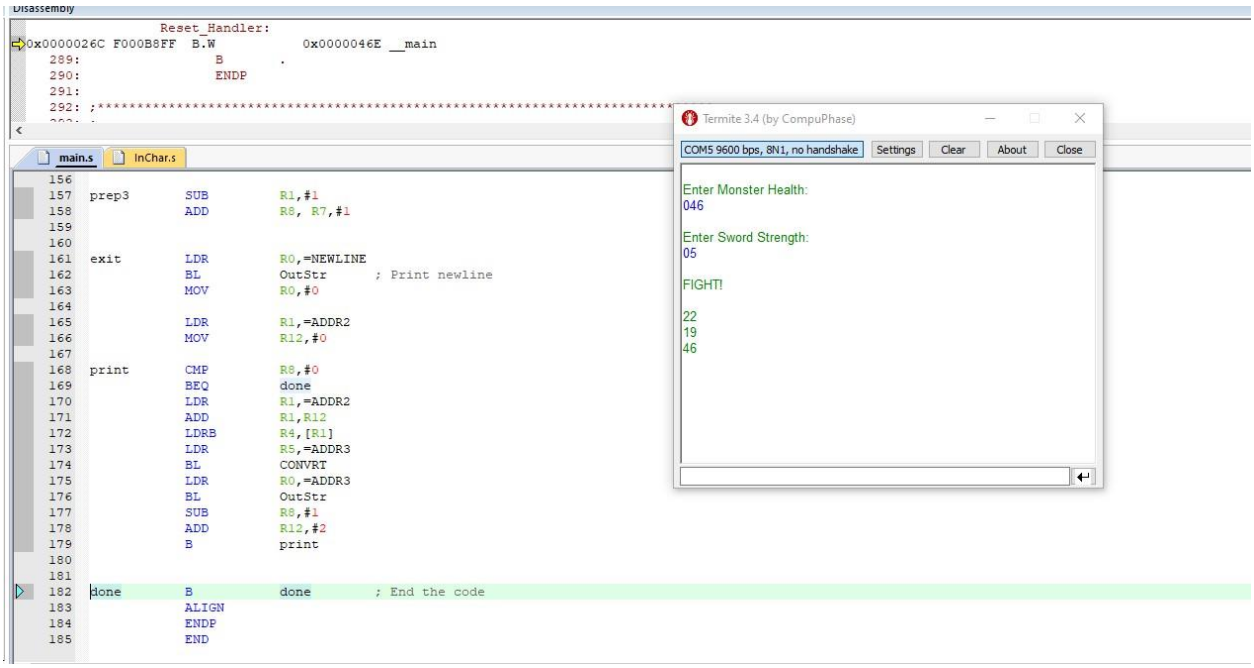

Figure 21. Q4 Example with 46 Health and 5 Damage

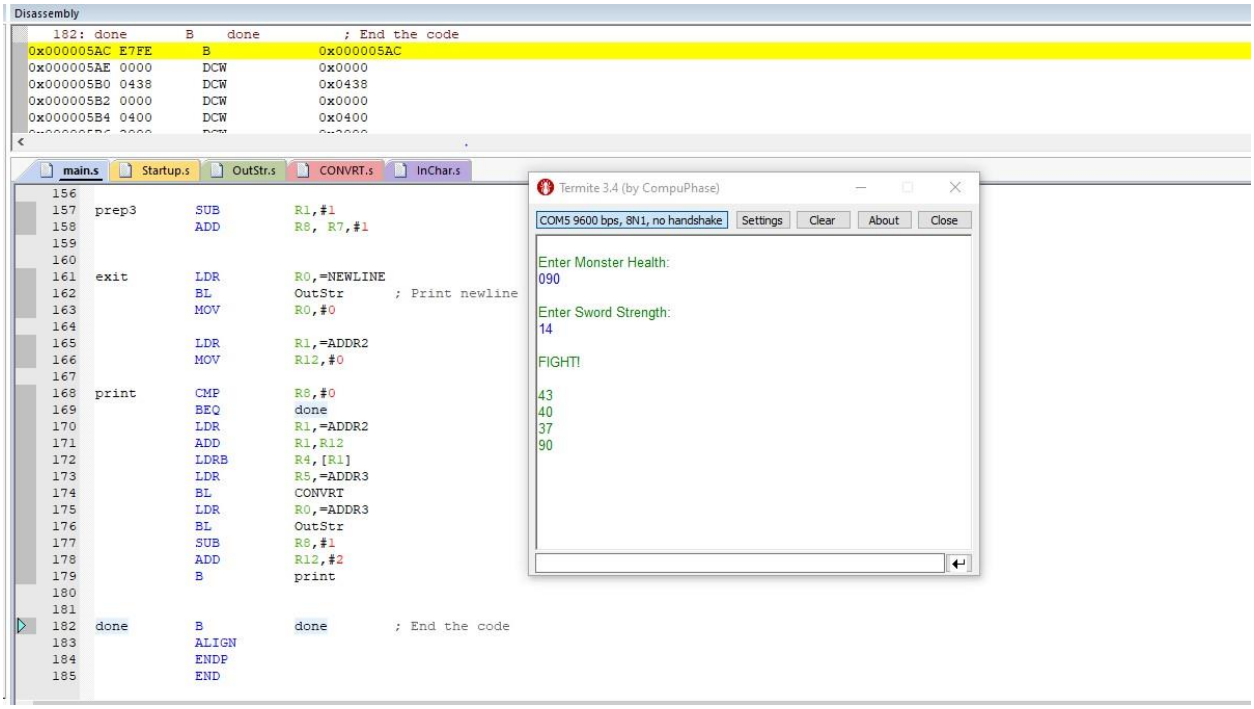

Figure 22. Q4 Example with 90 Health and 14 Damage

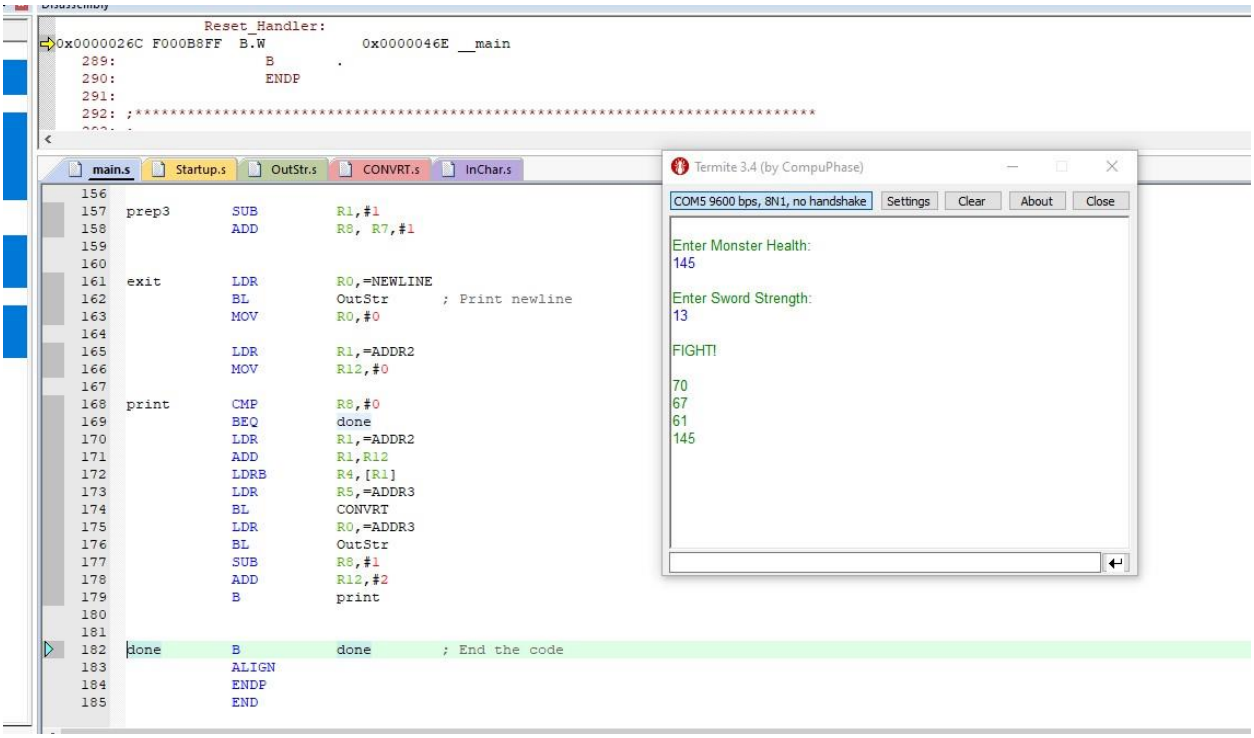

Figure 23. Q4 Example with 145 Health and 13 Damage

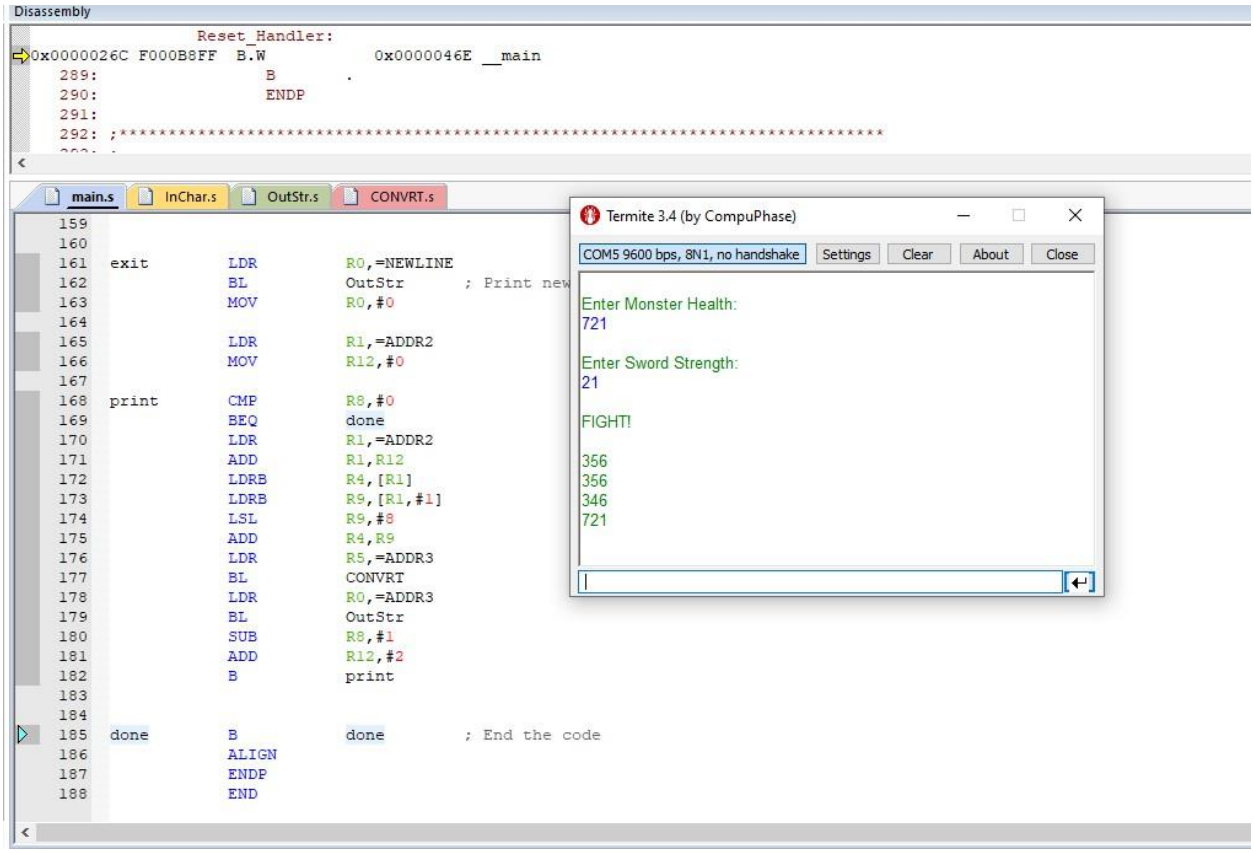

Figure 24. Q4 Example with 721 Health and 21 Damage

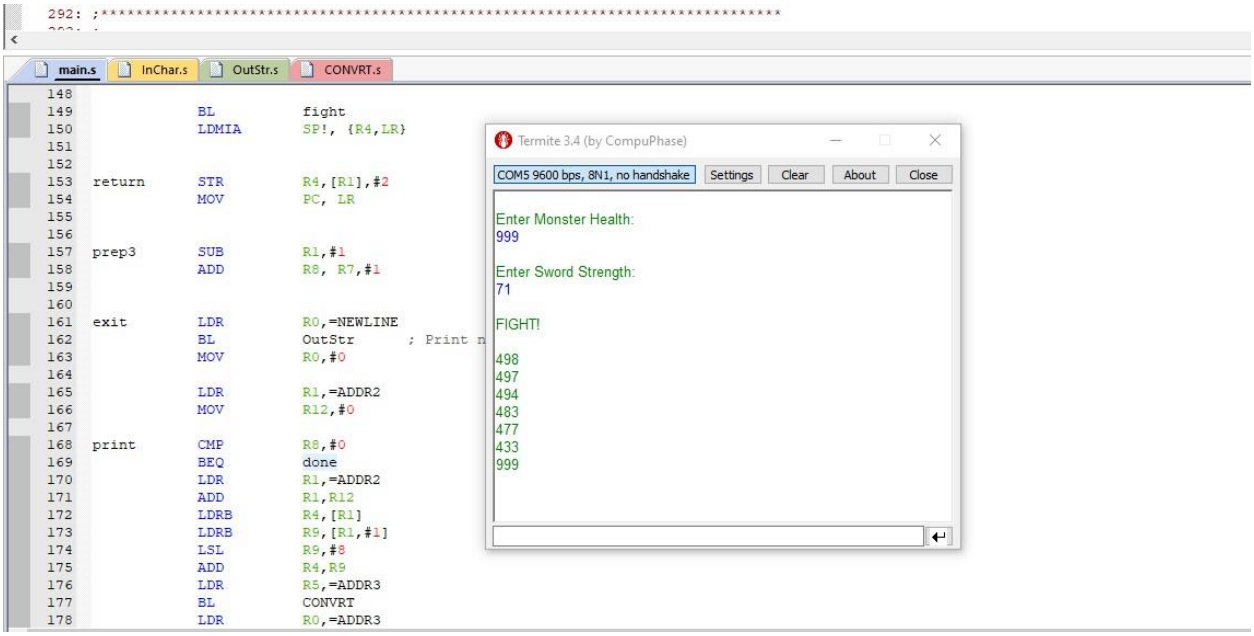

Figure 25. Q4 Example with 999 Health and 71 Damage