Preliminary Report # 2

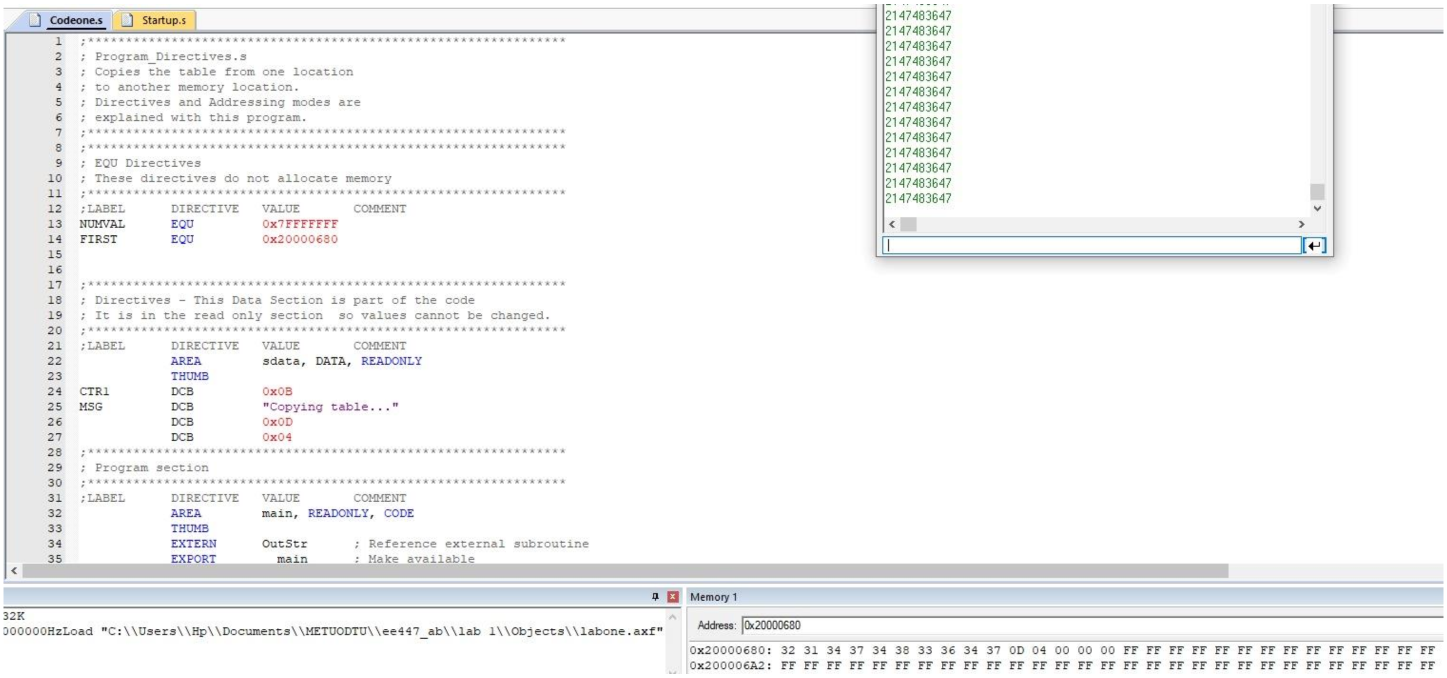
Question 1.

The following pseudo code governs the overall aspect of the code written in question 1:

```
CONST NUMVAL = 0x120
CONST FIRST = 0x20000680
DATA_SECTION
  CTR1 ← 0x0B
  MSG ← "Copying table..."
  MSG_ENDLINE ← 0x0D
  MSG_END ← 0x04

MAIN_SECTION
  R9 ← 10
  R8 ← 0
  R4 ← NUMVAL

  WHILE (TRUE)
    R5 = R4 / R9
    R6 = R5 * R9
    R7 = R4 - R6

    IF (R7 == 0) THEN
      PUSH R7
      R8 ← R8 + 1
      R4 ← R5
    ELSE
      PUSH R5
      R8 ← R8 + 1

    IF (R8 == 0) THEN
      BREAK

  R8 ← R8 - 1
  R2 ← FIRST

  WHILE (R8 > 0)
    R1 ← POP()
    R1 ← R1 + 0x30
    [R2] → R1
    R2 ← R2 + 1
    R8 ← R8 - 1

  [R2] → MSG_ENDLINE
  R2 ← R2 + 1
  R1 ← MSG_END
  R1 ← R1 + 0x30
  [R2] → R1
  R2 ← FIRST

  R0 ← R2
  CALL OutStr
  CONTINUE
```

Preliminary Report # 2

The following picture indicates the relevant outputs as shown below, the address indicated by 0x2000.0680 is pointed by R0 and in the termite emulator the max number 0x7FFF.FFFF in decimal can be observed in the display.



```
;***************************************************************
; 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
NUMVAL       EQU         0x7FFFFFFF
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
;***************************************************************
; Program section
;***************************************************************
;LABEL       DIRECTIVE   VALUE        COMMENT
             AREA        main, READONLY, CODE
             THUMB
             EXTERN      OutStr       ; Reference external subroutine
             EXPORT        main       ; Make available
```

```
2147483647
2147483647
2147483647
2147483647
2147483647
2147483647
2147483647
2147483647
2147483647
2147483647
2147483647
2147483647
2147483647
```

```
32K
000000HzLoad "C:\\Users\\Hp\\Documents\\METUODTU\\ee447_ab\\lab 1\\Objects\\labone.axf"
```

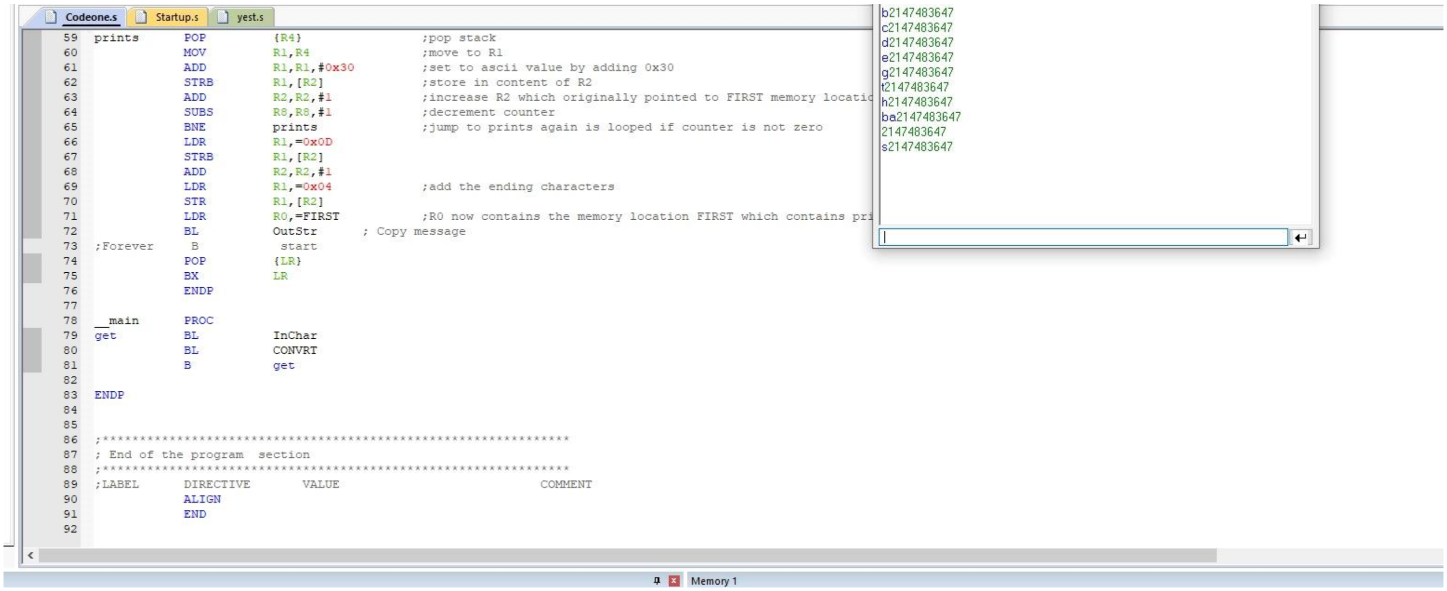
Address: 0x20000680

```
0x20000680: 32 31 34 37 34 38 33 36 34 37 0D 04 00 00 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x200006A2: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

Fig 1. Output Results for Q1

Question 2.

In this program both the sub routines of CONVRT and InChar have been implemented as shown and the emulator output has been observed as follows as shown in the diagram.



```
prints       POP         {R4}                   ;pop stack
             MOV         R1,R4                  ;move to R1
             ADD         R1,R1,#0x30            ;set to ascii value by adding 0x30
             STRB        R1,[R2]                ;store in content of R2
             ADD         R2,R2,#1               ;increase R2 which originally pointed to FIRST memory locatic
             SUBS        R8,R8,#1               ;decrement counter
             BNE         prints                 ;jump to prints again is looped if counter is not zero
             LDR         R1,=0x0D
             STRB        R1,[R2]
             ADD         R2,R2,#1
             LDR         R1,=0x04               ;add the ending characters
             STR         R1,[R2]
             LDR         R0,=FIRST              ;R0 now contains the memory location FIRST which contains pri
             BL          OutStr       ; Copy message
;Forever      B           start
             POP         {LR}
             BX          LR
             ENDP

__main       PROC
get          BL          InChar
             BL          CONVRT
             B           get

ENDP


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

```
b2147483647
c2147483647
d2147483647
e2147483647
g2147483647
t2147483647
h2147483647
ba2147483647
2147483647
s2147483647
```

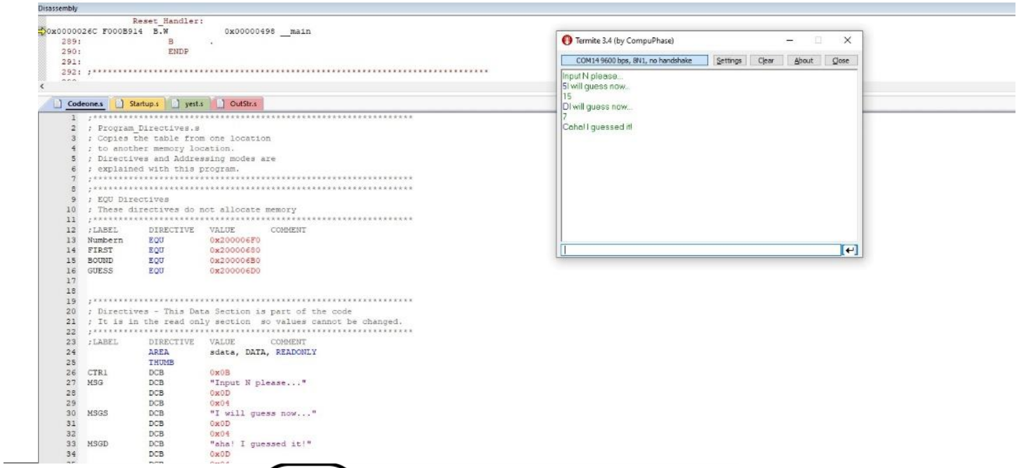
Fig 2. Output Results for Q2

Preliminary Report # 2

Question 3.

The following flow chart shows the path flow of data while using the subroutine functions of InChar OutChar, CONVRT and UPBND incorporated in the diagram:

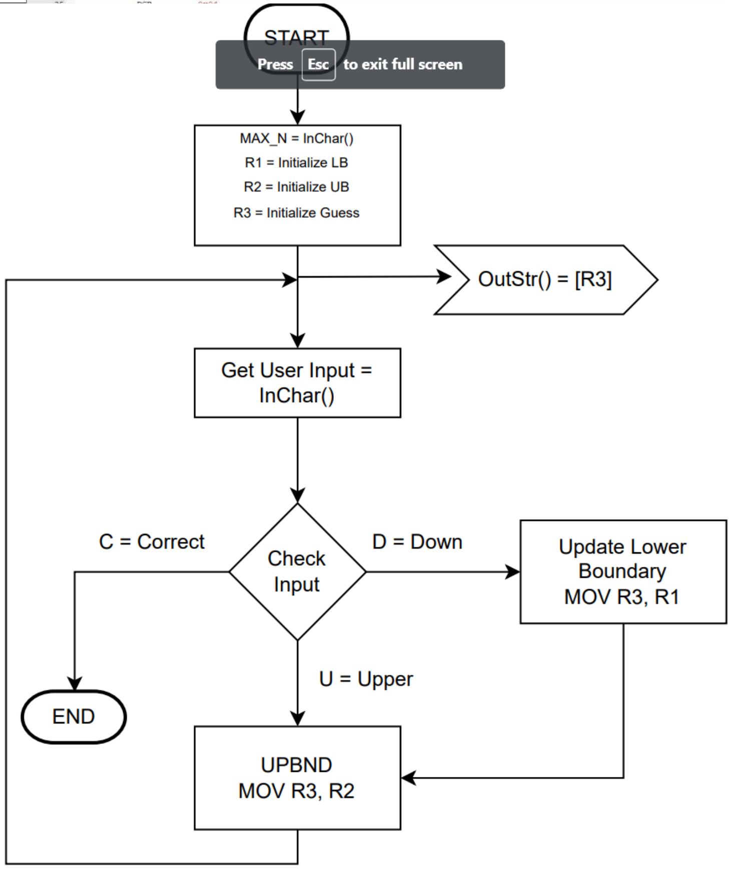
The flowchart illustrates a number-guessing program that starts by prompting the user to input 'n', representing the maximum range for the number to be guessed (MAX_N). It initializes the lower (R1) and upper (R2) boundaries and the current guess (R3). The program enters a loop where it displays the current guess, awaits user feedback ('C' for correct, 'U' for up, 'D' for down), and refines the boundaries accordingly. If the user input is 'C', indicating a correct guess, the program ends. If 'U' is received, the upper boundary is adjusted, and if 'D' is received, the lower boundary is adapted. This loop continues until the correct number is identified.



```
                  Reset_Handler:
0x0000026C F000B914  B.W          0x00000498 __main
                        B       .
                        ENDP

      ;*************************************************************************
```

```
;*****************************************************************
; 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
Numbern     EQU         0x200006F0
FIRST       EQU         0x20000680
BOUND       EQU         0x200006B0
GUESS       EQU         0x200006D0


;*****************************************************************
; 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         "Input N please..."
            DCB         0x0D
            DCB         0x04
MSGS        DCB         "I will guess now..."
            DCB         0x0D
            DCB         0x04
MSGD        DCB         "aha! I guessed it!"
            DCB         0x0D
```

```
Input N please...
5I will guess now...
15
DI will guess now...
7
Caha! I guessed it!
```



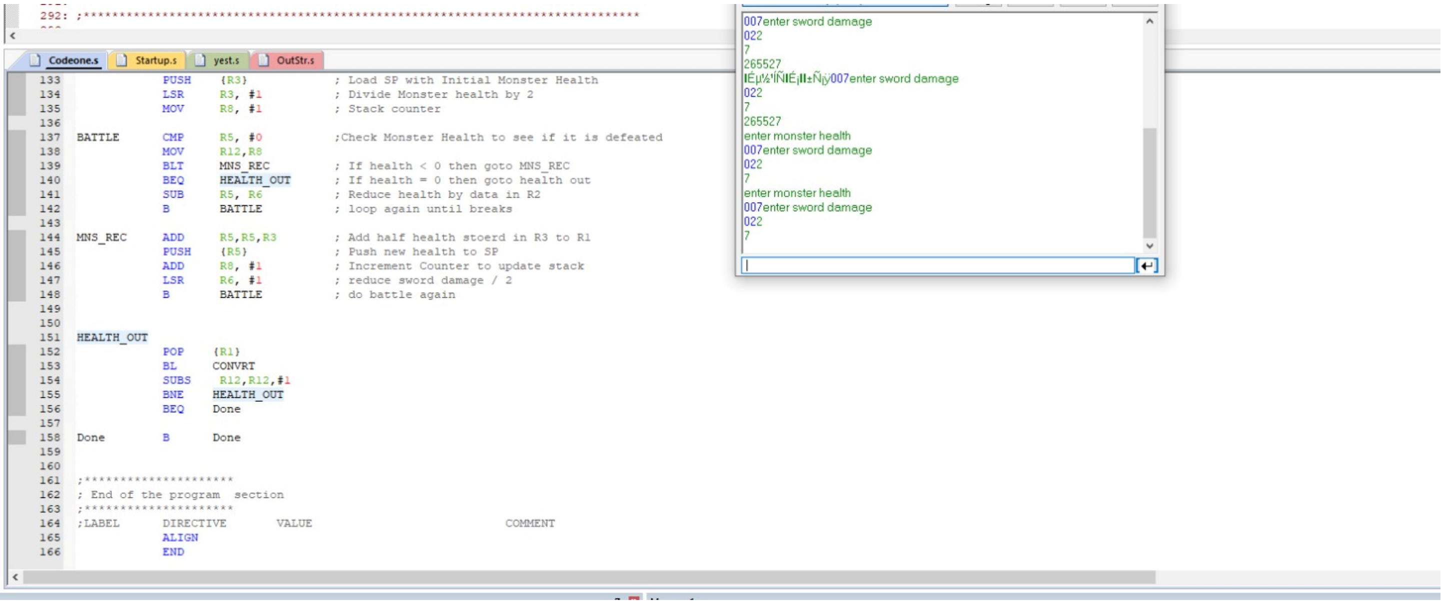
Preliminary Report # 2

Question 4.

This pseudo code program gives us the control algorithm for the monster game:
Start

```
Input: Monster's health, Sword damage

if Monster's health <= 0 then
    Monster is defeated
    Print Monster's health (after-heal health)
    Return
else
    Calculate new Monster's health after the strike

    if Monster's health is now negative then
        Calculate healing amount as half of the initial health
        Add the healing amount to Monster's health
        Halve Sword damage
    end if

    Push initial health onto the stack
    Recursively call the function with updated health and damage
    Pop initial health from the stack

    Print Monster's health (after-heal health)
    Print Initial health
end if

End
```

The following diagram shows the respective outputs and their input conditions.



```
292: ;*****************************************************************************

                PUSH    {R3}            ; Load SP with Initial Monster Health
                LSR     R3, #1          ; Divide Monster health by 2
                MOV     R8, #1          ; Stack counter

BATTLE          CMP     R5, #0          ;Check Monster Health to see if it is defeated
                MOV     R12,R8
                BLT     MNS_REC         ; If health < 0 then goto MNS_REC
                BEQ     HEALTH_OUT      ; If health = 0 then goto health out
                SUB     R5, R6          ; Reduce health by data in R2
                B       BATTLE          ; loop again until breaks

MNS_REC         ADD     R5,R5,R3        ; Add half health stoerd in R3 to R1
                PUSH    {R5}            ; Push new health to SP
                ADD     R8, #1          ; Increment Counter to update stack
                LSR     R6, #1          ; reduce sword damage / 2
                B       BATTLE          ; do battle again


HEALTH_OUT
                POP     {R1}
                BL      CONVRT
                SUBS     R12,R12,#1
                BNE     HEALTH_OUT
                BEQ     Done

Done            B       Done


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

```
007enter sword damage
022
7
265527
IÉµ'½'ÍÑIÉ¡Il±Ñ¡ÿ007enter sword damage
022
7
265527
enter monster health
007enter sword damage
022
7
enter monster health
007enter sword damage
022
7
```