Spring 2022, Homework 3 (20 points in total)

Q1. (2pts) Multiplication with shift instructions

```
MOV  R0,  #100
MUL  R2,  R1,  R0
```

The above code computes R2 = R1 * 100. Carry out the same computation, (i.e., a multiplication by 100) with another code, using only LSL and ADD instructions. **Don't use any loops, (i.e., no use of branches).** The result should be placed into R2 like the original code. You may use additional registers such as R3 and R4.
Note that you cannot run the provided code in VisUAL, because MUL is not a supported instruction in VisUAL.

```
MOV    R0, #100
MOV    R1, #N              ; N will be any integer

ADD    R2, R1, R1, LSL #6  ; R2 = R2 + (R1 · 64)
ADD    R3, R1, R1, LSL #4  ; R2 = R2 + (R1 · 16)
ADD    R2, R2, R3
ADD    R2, R2, R3
ADD    R2, R2, R1
```

Q2. (6 pts) Memory map

Let’s assume that you’re using Keil uVersion. Fill out the blanks of the memory map (address 0x00000004 to address 0x00000014) when running the following assembly program.

```
                THUMB
StackSize       EQU     0x00000100

                AREA    STACK, NOINIT, READWRITE, ALIGN=3
MyStackMem      SPACE   StackSize

                AREA    RESET, READONLY
                EXPORT  __Vectors
__Vectors
                DCD     MyStackMem + StackSize
                DCD     Reset_Handler

                AREA    MYDATA, DATA, READWRITE
dst             SPACE   8

                AREA    MYDCODE, CODE, READONLY
src0            DCB     "UWB", 0
src1            DCB     "CSS", 0

                ALIGN
                ENTRY
                EXPORT  Reset_Handler
Reset_Handler
                LDR     R0, =src0
                LDR     R1, =src1
                LDR     R2, =dst
loop1
                LDRB    R3, [R0], #1
                CBZ     R3, next
                STRB    R3, [R2], #1
                B       loop1
next
                MOV     R3, '.'
                STRB    R3, [R2], #1
loop2
                LDRB    R3, [R1], #1
                CBZ     R3, end_prog
                STRB    R3, [R2], #1
                B       loop2
end_prog
                B       end_prog
                END
```

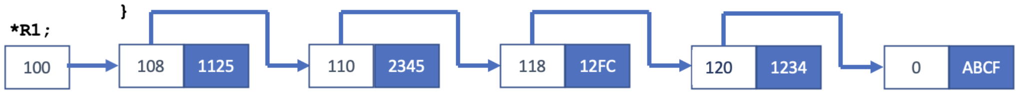
Address	Contents
	DRAM
0x60000000	Peripherals
0x40000000	SRAM
0x20000008	…
0x20000000	…
	…
0x00000014	"."
0x00000012	"\ S"
0x00000010	"S C"
0x0000000C	"."
0x00000008	"\ B"
0x00000004	"W U"
0x00000000	20000108

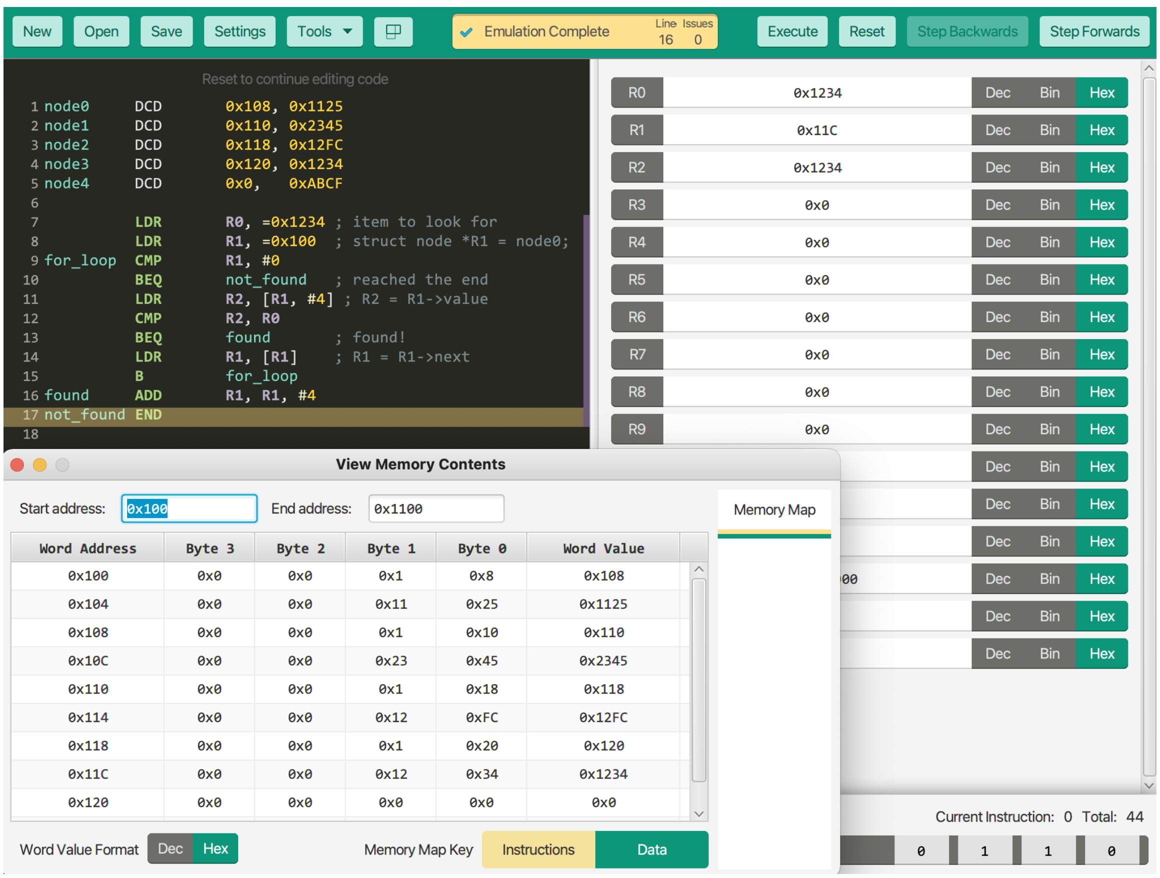
Q3. (12 pts) Pointer operations

On slide deck 6.ARM-InstrMem, we studied how to traverse a linked list using pre-indexed/register offset addressing. The following code intends to travers a linked list in search for a given value in R0 and returns the address of this value into R1 (but not the address of the node). If the value was not found, it returns 0 in R1, (i.e., a null address).

```
struct node {
   struct node *next;
   int value;
}
```

struct node *R1;

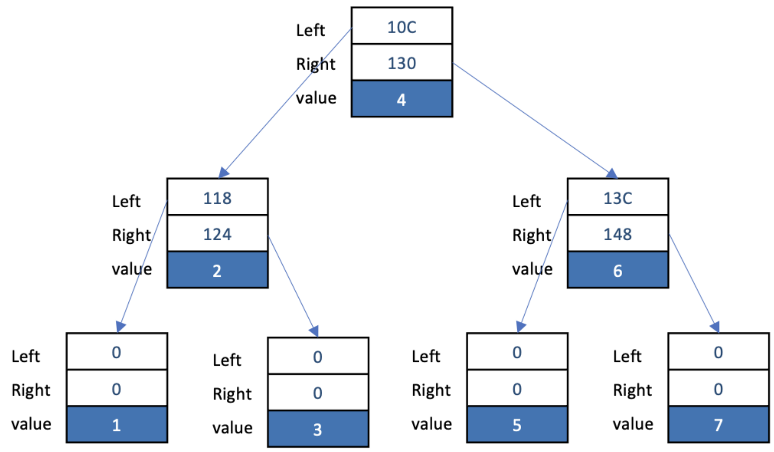




How about traversing a binary search tree?

R0 maintains a value to search. R1 first points to the tree root and is used to access each tree node. You can access its left pointer, right pointer, and value with

```
struct node {
    struct node *left; // R1
    struct node *right;// R1 + 4
    int value;         // R1 + 8
}
```



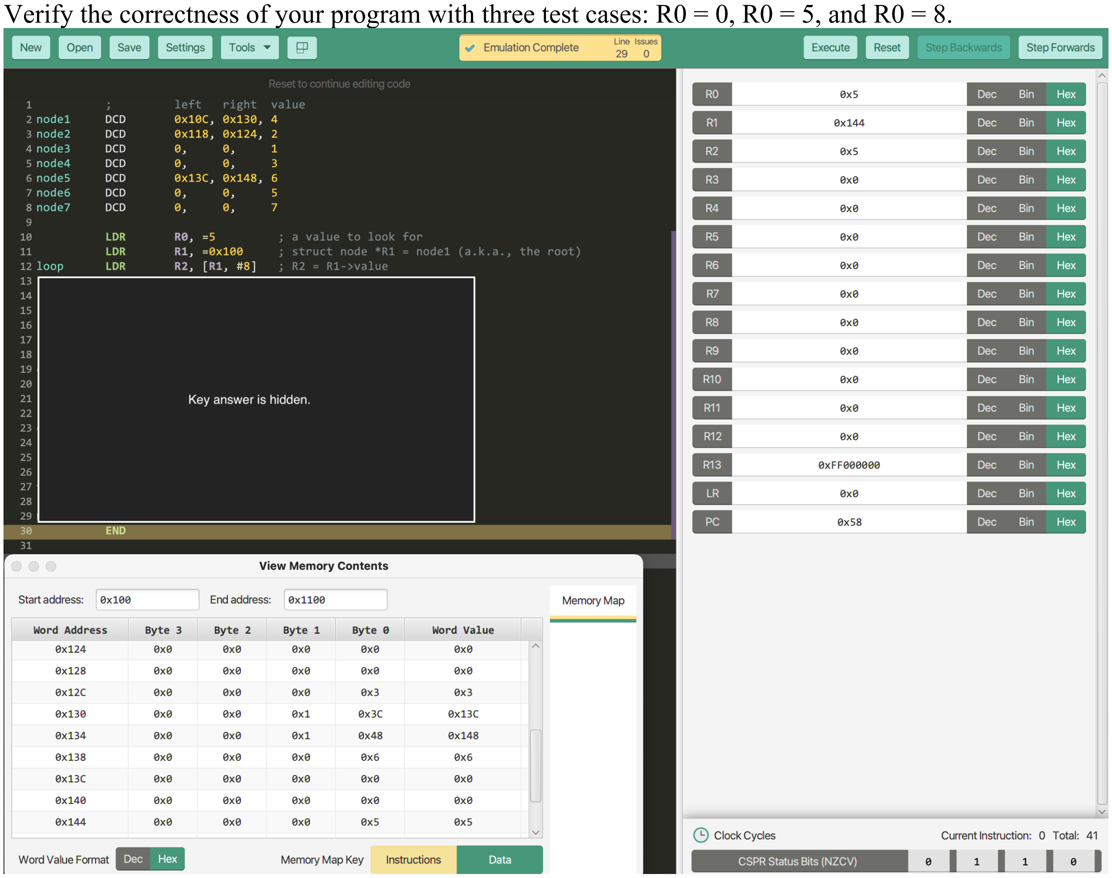
Using VisUAL, write a binary-tree search program. Your program searches for a value given in R0 and returns the address of this value into R1 (but not the address of the node). If the value was not found, it returns 0 in R1 (i.e., a null address).

Initialize a tree with the following code:

```
        ;       left    right   value
node1   DCD     0x10C,  0x130,  4
node2   DCD     0x118,  0x124,  2
node3   DCD     0,      0,      1
node4   DCD     0,      0,      3
node5   DCD     0x13C,  0x148,  6
node6   DCD     0,      0,      5
node7   DCD     0,      0,      7
```

You may assume that the tree root is located at memory address 0x100.

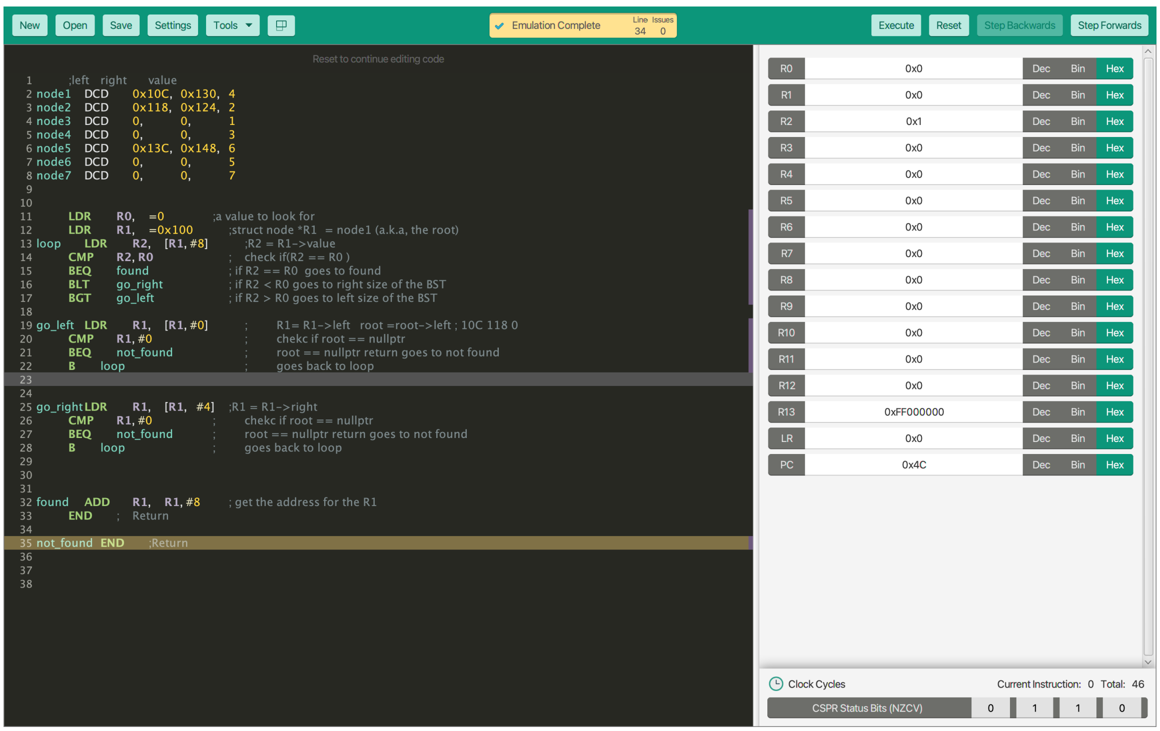
Verify the correctness of your program with three test cases: R0 = 0, R0 = 5, and R0 = 8.



What to submit: source code, screen shorts, and short explanations

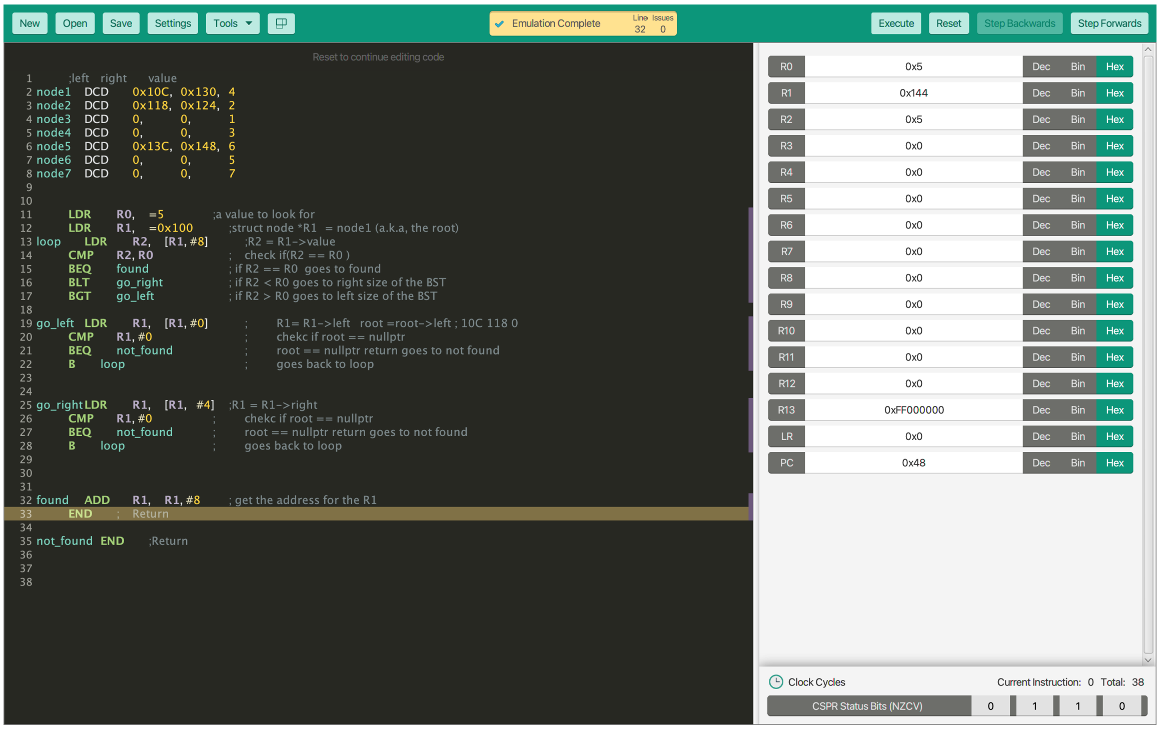
1. (6 pts) Your source code named hw3q3.s: **You need to add comment to each line of your code, otherwise, you get 0 for the coding part!**
2. (6 pts) In the same file recording your answers to Q1 and Q2, add screenshots and explanations for the following three test cases
   a. Test case 1 (where R1 = 0)'s screenshot of registers (R1 – R13, LR, and PC) and a short explanation: 2pt
   b. Test case 2 (where R1 = 5)'s screenshot of registers (R1 – R13, LR, and PC) and a short explanation: 2pt
   c. Test case 3 (where R1 = 8)'s screenshot of registers (R1 – R13, LR, and PC) and a short explanation: 2pt
   d. Copy your source code to the file after the test case screenshots and explanations.

2.
a.



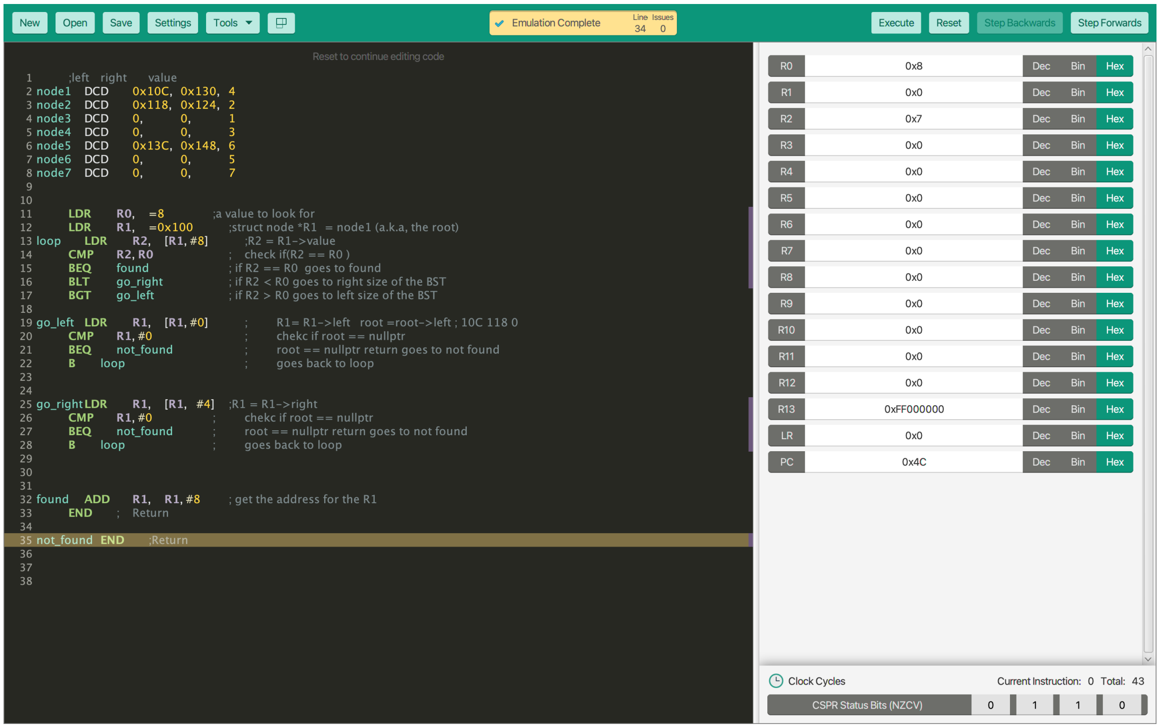
Case 1: When R0 = 0. The program will set R0 = 0, then get the value for what R1 point to, which is 0x4, and place it in R2. After that, the program will compare R2 and R0. Since 0 is less than 4, it will go to the left side of the binary search tree. Then move root point to root->left, and go back to the loop to compare R2 and R0. At this time, R2 changes to 0x2, which is still greater than R0. It will keep moving left. Then R2 will become 0x1, and the root will reach the end, which means the program does not find 0 in the BST. Out of the loop.

b.



Case 1: When R0 = 5. The program will set R0 = 0, then get the value for what R1 point to, which is 0x4, and place it in R2. After that, the program will compare R2 and R0. Since 5 is greater than 4, it will go to the right side of the binary search tree. Move the root to current = root->right, and check if it reaches the end. If not, go back to the loop, reset the value for the current node, and compare it with R0 since R6 is greater than R0. The node will go to the left side of the current node and check if it reaches the end. Then goes back to the loop to modify the current node's value. Which is 0x5, and it is equal to R0, and it will go to found and return the address of R1.

c.



Case 1: When R0 = 8. The program will set R0 = 0, then get the value for what R1 point to, which is 0x4, and place it in R2. After that, the program will compare R2 and R0. Since 8 is greater than 4, it will go to the right side of the binary search tree. Then move root point to root->right, and go back to the loop to compare R2 and R0. At this time, R2 changes to 0x6, which is still less than R0. It will keep moving right. Then R2 will become 0x7, and the root will reach the end, which means the program does not find 0 in the BST. Out of the loop.

d.

```
                ;left   right               value
node1   DCD             0x10C,  0x130,  4
node2   DCD             0x118,  0x124,  2
node3   DCD             0,                      0,                      1
node4   DCD             0,                      0,                      3
node5   DCD             0x13C,  0x148,  6
node6   DCD             0,                      0,                      5
node7   DCD             0,                      0,                      7

                LDR             R0,     =8                      ;a value to look for
                LDR             R1,     =0x100                  ;struct node *R1    = node1
(a.k.a, the root)
loop            LDR             R2,     [R1, #8]                        ;R2 = R1->value
                CMP             R2, R0                                  ;       check if(R2 ==
R0 )
                BEQ             found                                   ; if R2 == R0  goes to
found
                BLT             go_right                                        ; if R2 < R0
goes to right size of the BST
```

```
			BGT			go_left						; if R2 > R0 goes to
left size of the BST

go_left LDR		R1,		[R1, #0]				;		R1= R1->left
root =root->left ; 10C 118 0
			CMP			R1, #0							;
chekc if root == nullptr
			BEQ			not_found						;
root == nullptr return goes to not found
			B			loop
;			goes back to loop

go_right	LDR			R1,		[R1,	#4]		;R1 = R1->right
			CMP			R1, #0					;		chekc if root
== nullptr
			BEQ			not_found				;		root == nullptr
return goes to not found
			B			loop							;
goes back to loop


found	ADD			R1,		R1, #8		;getting the address for R1
			END							;		return

not_found	END				;		return
```