Autumn 2022, Homework 3 Key

Q1. (2pts) Multiplication with shift instructions

```
MOV R0, #97
MUL R2, R1, R0
```

The above code computes R2 = R1 * 97. Carry out the same computation, (i.e., a multiplication by 97) 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 opcode in VisUAL.**

```
               MOV            R1, #100       ; test setting of value 100 to R1
               LSL            R2, R1, #6     ; R2 = R1*64
               LSL            R3, R1, #5     ; R3 = R1*32
               ADD            R2, R2, R3     ; R2 = R2 + R3 = 96*R1
               ADD            R2, R2, R1     ; R2 = R2 + R1 = 97*R1
               ; the above three lines can also be changed to
               ; ADD          R2, R2, R1, LSL #5
               ; ADD          R2, R2, R1
```

Q2. (6pts) Memory map

Let's assume that you're using Keil uVision. 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    "CSS", 0
src1           DCB    "UWB", 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
```

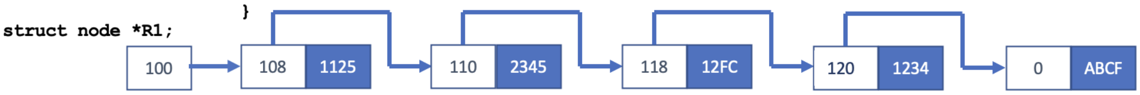
```
        B       end_prog
        END
```

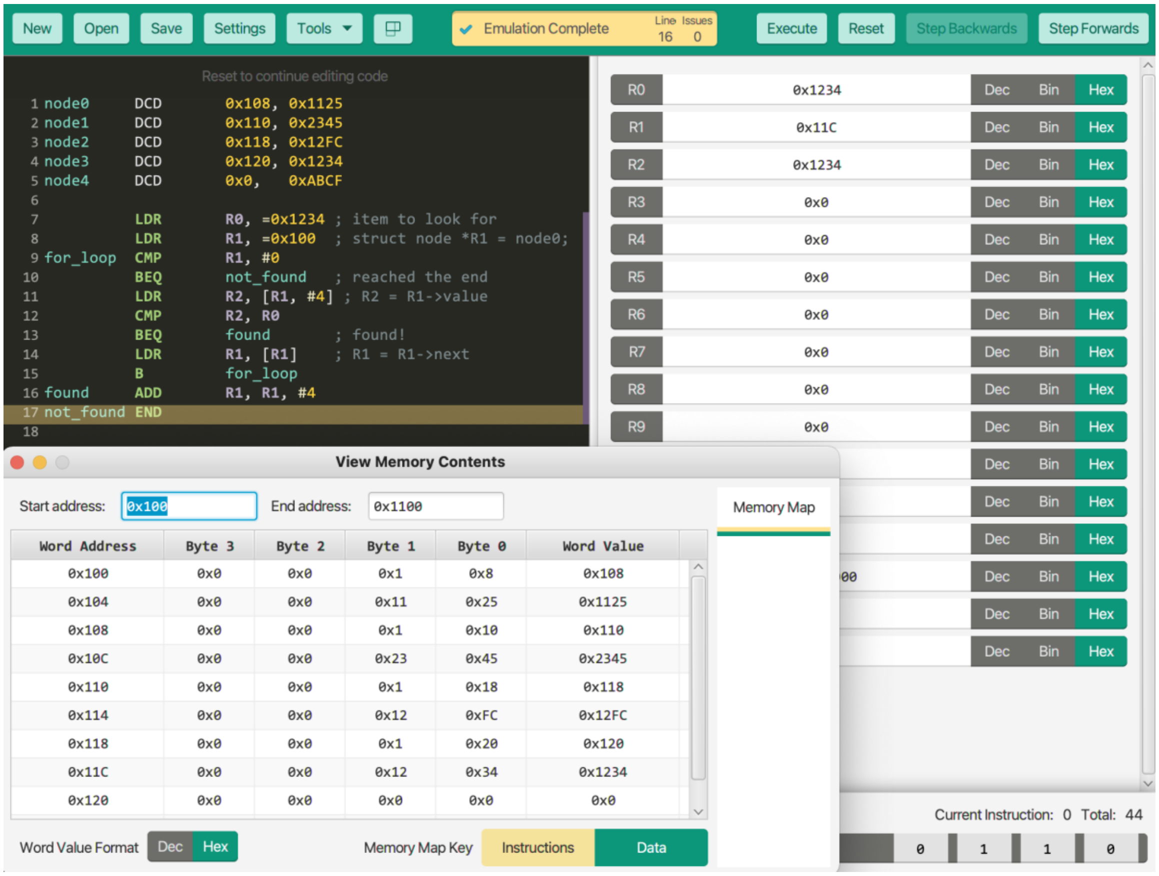
Address	Contents
0x60000000	DRAM
0x40000000	Peripherals
0x20000008	SRAM
0x20000000	
0x00000014	LDR R2, =dst or 0x4A0A
0x00000012	LDR R1, =src1 or 0x490A
0x00000010	LDR R0, =src0 or 0x4809
0x0000000C	\0 “BWU” or 0x00425755
0x00000008	\0, “SSC” or 0x00535343
0x00000004	0x00000011
0x00000000	0x20000108

Q4. (12pts) 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;
}
```





```
node0       DCD     0x108, 0x1125
node1       DCD     0x110, 0x2345
node2       DCD     0x118, 0x12FC
node3       DCD     0x120, 0x1234
node4       DCD     0x0,   0xABCF

            LDR     R0, =0x1234 ; item to look for
            LDR     R1, =0x100  ; struct node *R1 = node0;
for_loop    CMP     R1, #0
            BEQ     not_found   ; reached the end
            LDR     R2, [R1, #4] ; R2 = R1->value
            CMP     R2, R0
            BEQ     found       ; found!
            LDR     R1, [R1]    ; R1 = R1->next
            B       for_loop
found       ADD     R1, R1, #4
not_found   END
```

R0	0x1234	Dec	Bin	Hex
R1	0x11C	Dec	Bin	Hex
R2	0x1234	Dec	Bin	Hex
R3	0x0	Dec	Bin	Hex
R4	0x0	Dec	Bin	Hex
R5	0x0	Dec	Bin	Hex
R6	0x0	Dec	Bin	Hex
R7	0x0	Dec	Bin	Hex
R8	0x0	Dec	Bin	Hex
R9	0x0	Dec	Bin	Hex

View Memory Contents

Start address: 0x100 End address: 0x1100

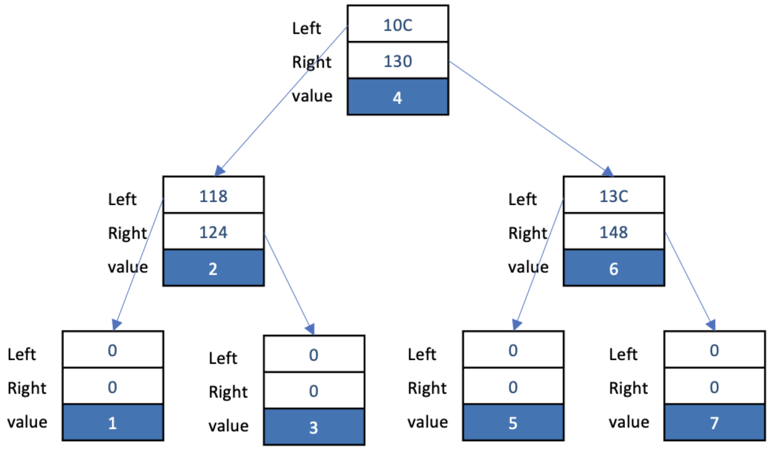
Word Address	Byte 3	Byte 2	Byte 1	Byte 0	Word Value
0x100	0x0	0x0	0x1	0x8	0x108
0x104	0x0	0x0	0x11	0x25	0x1125
0x108	0x0	0x0	0x1	0x10	0x110
0x10C	0x0	0x0	0x23	0x45	0x2345
0x110	0x0	0x0	0x1	0x18	0x118
0x114	0x0	0x0	0x12	0xFC	0x12FC
0x118	0x0	0x0	0x1	0x20	0x120
0x11C	0x0	0x0	0x12	0x34	0x1234
0x120	0x0	0x0	0x0	0x0	0x0

Word Value Format: Dec Hex

Memory Map Key: Instructions Data

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
}
```



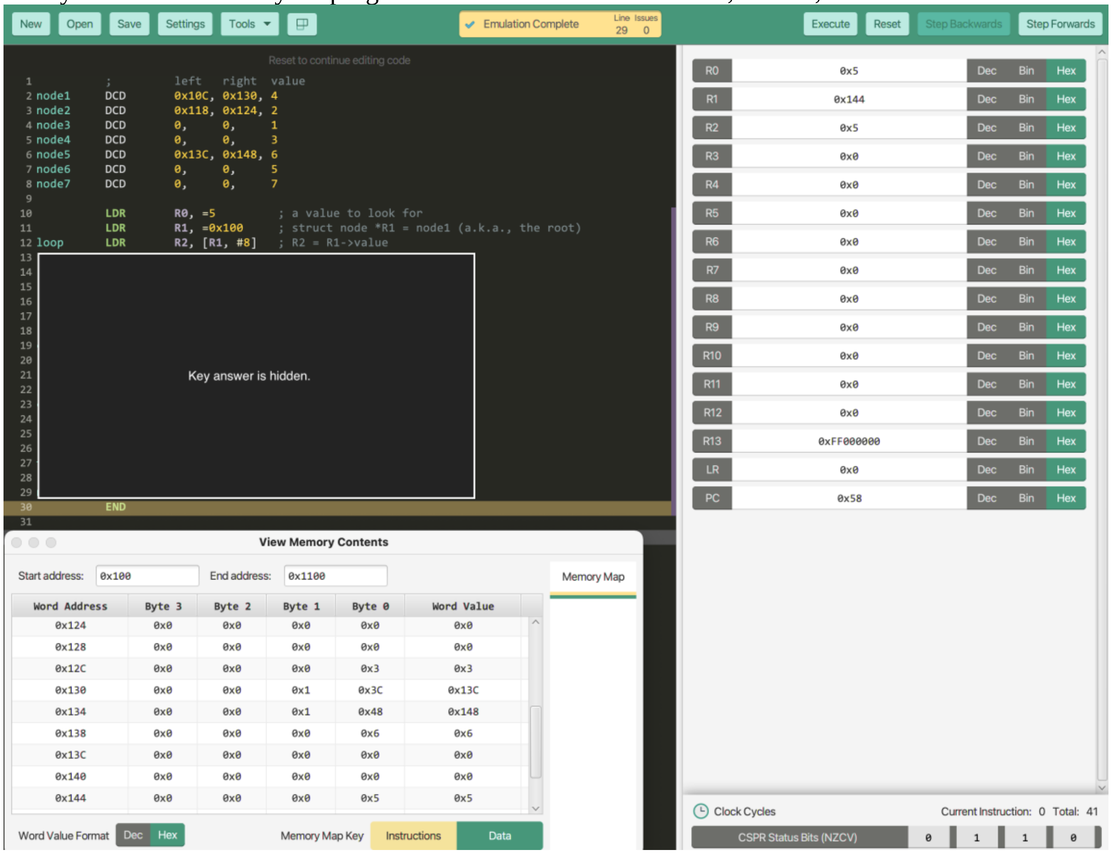
Using VisUAL, write a binary-tree search program. Your program searches for a value given in R0 and returns the address of this value into R1 (but not the address of the node). If the value was not found, it returns 0 in R1 (i.e., a null address).

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.



```
            ;          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, =5           ; 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

                 Key answer is hidden.

            END
```

View Memory Contents

Start address: 0x100 End address: 0x1100

Word Address	Byte 3	Byte 2	Byte 1	Byte 0	Word Value
0x124	0x0	0x0	0x0	0x0	0x0
0x128	0x0	0x0	0x0	0x0	0x0
0x12C	0x0	0x0	0x0	0x3	0x3
0x130	0x0	0x0	0x1	0x3C	0x13C
0x134	0x0	0x0	0x1	0x48	0x148
0x138	0x0	0x0	0x0	0x6	0x6
0x13C	0x0	0x0	0x0	0x0	0x0
0x140	0x0	0x0	0x0	0x0	0x0
0x144	0x0	0x0	0x0	0x5	0x5

R0	0x5
R1	0x144
R2	0x5
R3	0x0
R4	0x0
R5	0x0
R6	0x0
R7	0x0
R8	0x0
R9	0x0
R10	0x0
R11	0x0
R12	0x0
R13	0xFF000000
LR	0x0
PC	0x58

Current Instruction: 0 Total: 41

CSPR Status Bits (NZCV) 0 1 1 0

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 R0 = 0)'s screenshot of registers (R1 – R13, LR, and PC) and a short explanation: 2pt
   b. Test case 2 (where R0 = 5)'s screenshot of registers (R1 – R13, LR, and PC) and a short explanation: 2pt
   c. Test case 3 (where R0 = 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.

Key answer:

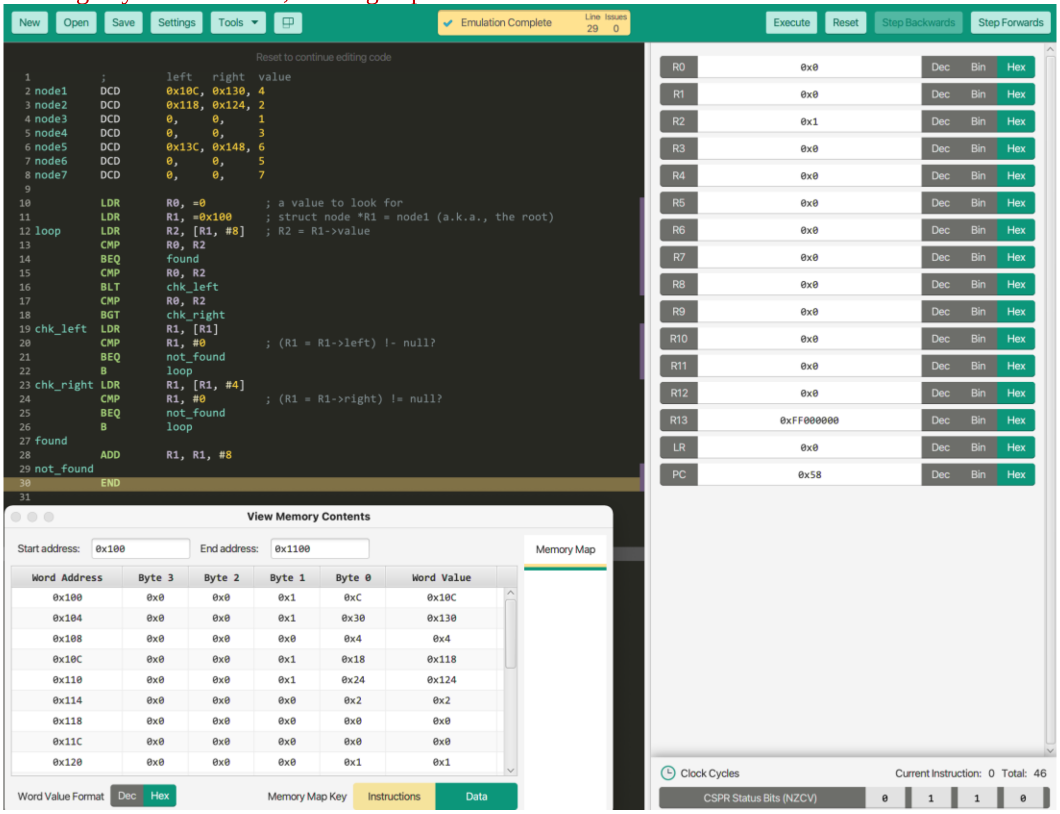
Source code (6pts):

```
                   ;                  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                R0, R2             ; compare R0 (target value) and R2 (node value)
                   BEQ                found              ; if R2 == R0, go to found
                   CMP                R0, R2             ; R0 (target value) and R2 (node value)
                   BLT                chk_left           ; if R0 < R2, go to check the left tree
                   CMP                R0, R2             ; R0 (target value) and R2 (node value)
                   BGT                chk_right          ; if R0 < R2, go to check the right tree
chk_left           LDR                R1, [R1]           ; R1 = R1->left
                   CMP                R1, #0             ; check if R1 (left node) is NULL
                   BEQ                not_found          ; if empty, the search reaches the end
                   B                  loop               ; keep searching if left node is not NULL
chk_right          LDR                R1, [R1, #4]       ; R1 = R1->right
                   CMP                R1, #0             ; check if R1 (right node) is NULL
                   BEQ                not_found          ; if empty, the search reaches the end
                   B                  loop               ; keep searching if left node is not NULL
found
                   ADD                R1, R1, #8         ; add 8 to R1 to get the value's address
not_found
                   END
```

Test case 1 (2pts)

Missing any screenshot -1, missing explanation -1.



```
          ;         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, =0          ; 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       R0, R2
          BEQ       found
          CMP       R0, R2
          BLT       chk_left
          CMP       R0, R2
          BGT       chk_right
chk_left  LDR       R1, [R1]
          CMP       R1, #0          ; (R1 = R1->left) !- null?
          BEQ       not_found
          B         loop
chk_right LDR       R1, [R1, #4]
          CMP       R1, #0          ; (R1 = R1->right) != null?
          BEQ       not_found
          B         loop
found
          ADD       R1, R1, #8
not_found
          END
```

Start address: 0x100 End address: 0x1100

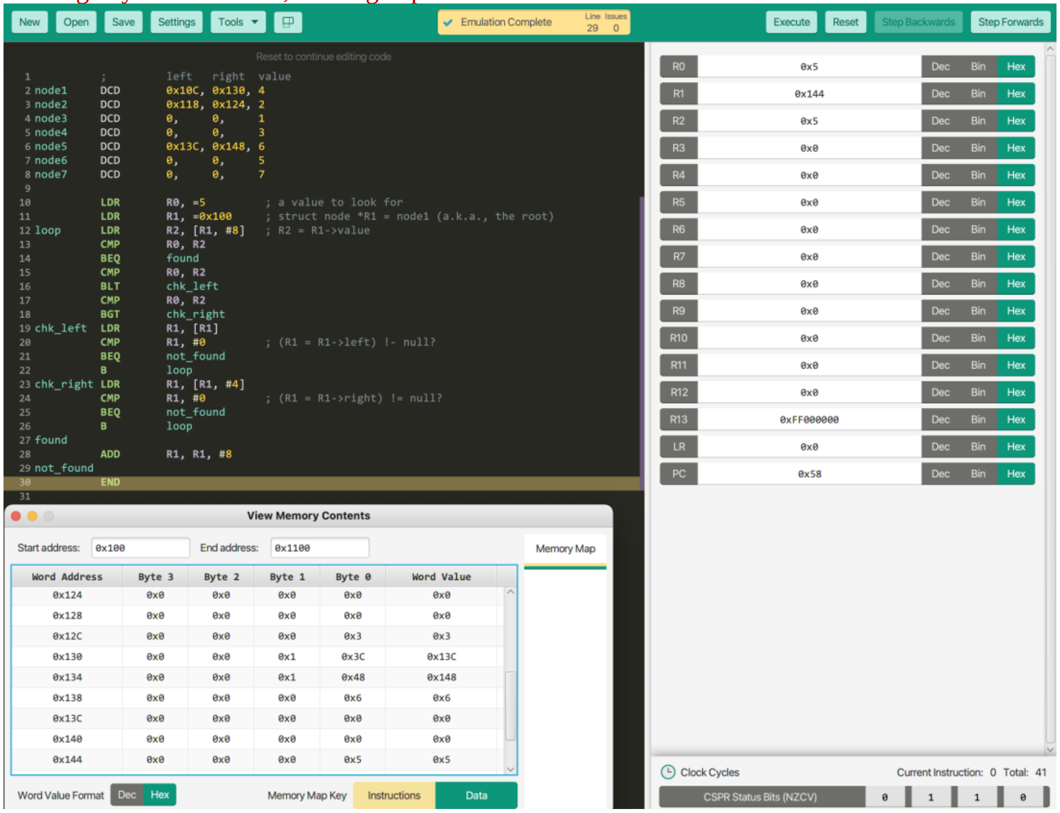
Word Address	Byte 3	Byte 2	Byte 1	Byte 0	Word Value
0x100	0x0	0x0	0x1	0xC	0x10C
0x104	0x0	0x0	0x1	0x30	0x130
0x108	0x0	0x0	0x0	0x4	0x4
0x10C	0x0	0x0	0x1	0x18	0x118
0x110	0x0	0x0	0x1	0x24	0x124
0x114	0x0	0x0	0x0	0x2	0x2
0x118	0x0	0x0	0x0	0x0	0x0
0x11C	0x0	0x0	0x0	0x0	0x0
0x120	0x0	0x0	0x0	0x1	0x1

R0	0x0	Dec	Bin	Hex
R1	0x0	Dec	Bin	Hex
R2	0x1	Dec	Bin	Hex
R3	0x0	Dec	Bin	Hex
R4	0x0	Dec	Bin	Hex
R5	0x0	Dec	Bin	Hex
R6	0x0	Dec	Bin	Hex
R7	0x0	Dec	Bin	Hex
R8	0x0	Dec	Bin	Hex
R9	0x0	Dec	Bin	Hex
R10	0x0	Dec	Bin	Hex
R11	0x0	Dec	Bin	Hex
R12	0x0	Dec	Bin	Hex
R13	0xFF000000	Dec	Bin	Hex
LR	0x0	Dec	Bin	Hex
PC	0x58	Dec	Bin	Hex

Current Instruction: 0 Total: 46

CSPR Status Bits (NZCV)	0	1	1	0

Test case 2 (2pts)
Missing any screenshot -1, missing explanation -1.



```
            ;         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, =5          ; 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       R0, R2
            BEQ       found
            CMP       R0, R2
            BLT       chk_left
            CMP       R0, R2
            BGT       chk_right
chk_left    LDR       R1, [R1]
            CMP       R1, #0          ; (R1 = R1->left) !- null?
            BEQ       not_found
            B         loop
chk_right   LDR       R1, [R1, #4]
            CMP       R1, #0          ; (R1 = R1->right) != null?
            BEQ       not_found
            B         loop
found
            ADD       R1, R1, #8
not_found
            END
```

View Memory Contents

Start address: 0x100   End address: 0x1100

Word Address	Byte 3	Byte 2	Byte 1	Byte 0	Word Value
0x124	0x0	0x0	0x0	0x0	0x0
0x128	0x0	0x0	0x0	0x0	0x0
0x12C	0x0	0x0	0x0	0x3	0x3
0x130	0x0	0x0	0x1	0x3C	0x13C
0x134	0x0	0x0	0x1	0x48	0x148
0x138	0x0	0x0	0x0	0x6	0x6
0x13C	0x0	0x0	0x0	0x0	0x0
0x140	0x0	0x0	0x0	0x0	0x0
0x144	0x0	0x0	0x0	0x5	0x5

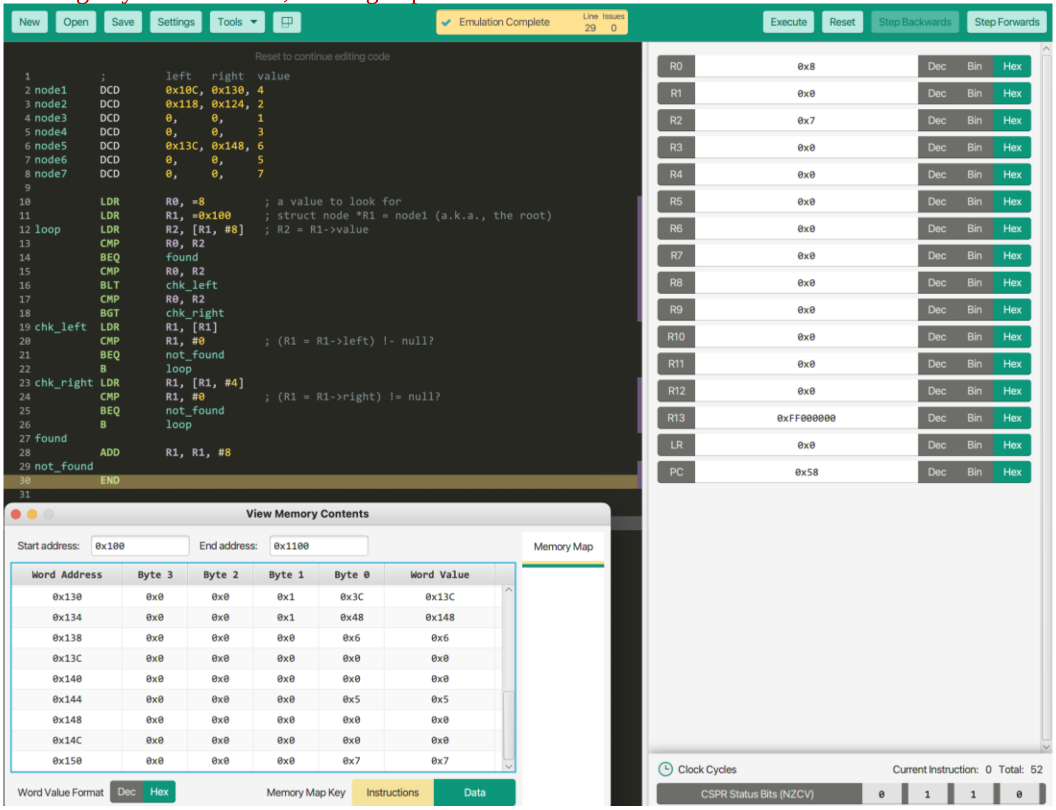
R0	0x5	Dec	Bin	Hex
R1	0x144	Dec	Bin	Hex
R2	0x5	Dec	Bin	Hex
R3	0x0	Dec	Bin	Hex
R4	0x0	Dec	Bin	Hex
R5	0x0	Dec	Bin	Hex
R6	0x0	Dec	Bin	Hex
R7	0x0	Dec	Bin	Hex
R8	0x0	Dec	Bin	Hex
R9	0x0	Dec	Bin	Hex
R10	0x0	Dec	Bin	Hex
R11	0x0	Dec	Bin	Hex
R12	0x0	Dec	Bin	Hex
R13	0xFF000000	Dec	Bin	Hex
LR	0x0	Dec	Bin	Hex
PC	0x58	Dec	Bin	Hex

Clock Cycles   Current Instruction: 0   Total: 41

CSPR Status Bits (NZCV)   0   1   1   0

Test case 3 (2pts)

Missing any screenshot -1, missing explanation -1.



Emulation Complete Line 29 Issues 0

Reset to continue editing 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

          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       R0, R2
          BEQ       found
          CMP       R0, R2
          BLT       chk_left
          CMP       R0, R2
          BGT       chk_right
chk_left  LDR       R1, [R1]
          CMP       R1, #0         ; (R1 = R1->left) !- null?
          BEQ       not_found
          B         loop
chk_right LDR       R1, [R1, #4]
          CMP       R1, #0         ; (R1 = R1->right) != null?
          BEQ       not_found
          B         loop
found
          ADD       R1, R1, #8
not_found
          END

```

Register	Value
R0	0x8
R1	0x0
R2	0x7
R3	0x0
R4	0x0
R5	0x0
R6	0x0
R7	0x0
R8	0x0
R9	0x0
R10	0x0
R11	0x0
R12	0x0
R13	0xFF000000
LR	0x0
PC	0x58

Clock Cycles Current Instruction: 0 Total: 52

CSPR Status Bits (NZCV) 0 1 1 0

View Memory Contents

Start address: 0x100 End address: 0x1100

Word Address	Byte 3	Byte 2	Byte 1	Byte 0	Word Value
0x130	0x0	0x0	0x1	0x3C	0x13C
0x134	0x0	0x0	0x1	0x48	0x148
0x138	0x0	0x0	0x0	0x6	0x6
0x13C	0x0	0x0	0x0	0x0	0x0
0x140	0x0	0x0	0x0	0x0	0x0
0x144	0x0	0x0	0x0	0x5	0x5
0x148	0x0	0x0	0x0	0x0	0x0
0x14C	0x0	0x0	0x0	0x0	0x0
0x150	0x0	0x0	0x0	0x7	0x7

Word Value Format