

## 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.

```

StackSize      THUMB EQU      0x00000100

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

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

dst            AREA          MYDATA, DATA, READWRITE
SPACE          8

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

Reset_Handler  ALIGN
ENTRY
EXPORT         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

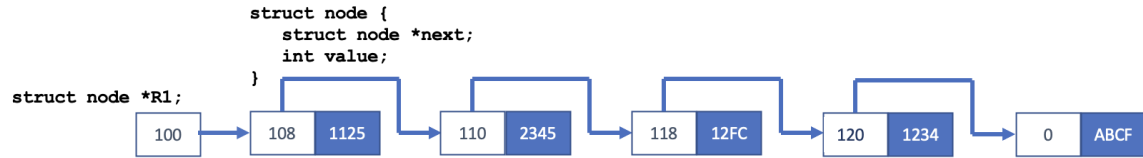
end_prog       B              end_prog
               END

```

Address	Contents
	DRAM
0x60000000	
	Peripherals
0x40000000	
	SRAM
0x20000008	
0x20000000	...
	...
	...
0x00000014	~
0x00000012	~
0x00000010	~
0x0000000C	~
0x00000008	~
0x00000004	~
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 traverse 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).



The screenshot shows an ARM emulator with the following components:

- Assembly Code (Left):**

```

1 node0 DCD 0x108, 0x1125
2 node1 DCD 0x110, 0x2345
3 node2 DCD 0x118, 0x12FC
4 node3 DCD 0x120, 0x1234
5 node4 DCD 0x0, 0xABCF
6
7 LDR R0, =0x1234 ; item to look for
8 LDR R1, =0x100 ; struct node *R1 = node0;
9 for_loop CMP R1, #0
10 BEQ not_found ; reached the end
11 LDR R2, [R1, #4] ; R2 = R1->value
12 CMP R2, R0
13 BEQ found ; found!
14 LDR R1, [R1] ; R1 = R1->next
15 B for_loop
16 found ADD R1, R1, #4
17 not_found END
18

```
- Registers (Right):**

Register	Value	Dec	Bin	Hex
R0	0x1234			
R1	0x11C			
R2	0x1234			
R3	0x0			
R4	0x0			
R5	0x0			
R6	0x0			
R7	0x0			
R8	0x0			
R9	0x0			
- View Memory Contents (Bottom):**

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

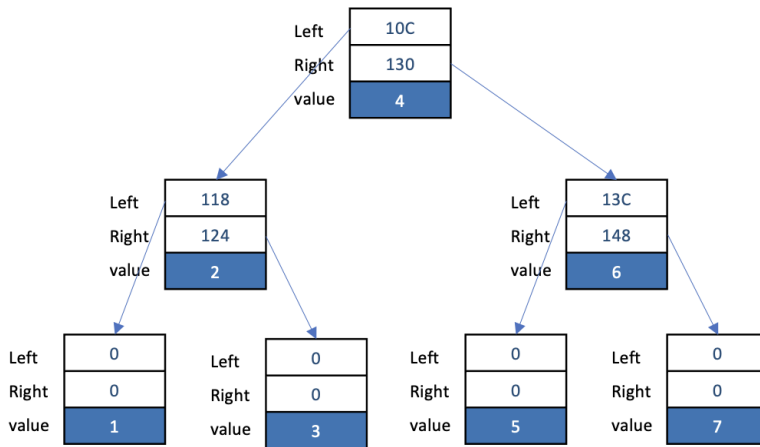
How about traversing a binary search tree?

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.

Emulation Complete 29 0

Execute Reset Step Backwards Step Forwards

```

1 ;          left right value
2 node1 DCD 0x10C, 0x130, 4
3 node2 DCD 0x118, 0x124, 2
4 node3 DCD 0, 0, 1
5 node4 DCD 0, 0, 3
6 node5 DCD 0x13C, 0x148, 6
7 node6 DCD 0, 0, 5
8 node7 DCD 0, 0, 7
9
10 LDR R0, #5 ; a value to look for
11 LDR R1, #0x100 ; struct node *R1 = node1 (a.k.a., the root)
12 loop LDR R2, [R1, #8] ; R2 = R1->value
13
14
15
16
17
18
19
20
21 Key answer is hidden.
22
23
24
25
26
27
28 END
29
30
31
  
```

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

Word Value Format: Dec Hex Memory Map Key Instructions Data

Register	Value	Dec	Bin	Hex
R0	0x5	5	101	0x5
R1	0x144	324	10101000	0x144
R2	0x5	5	101	0x5
R3	0x0	0	0	0x0
R4	0x0	0	0	0x0
R5	0x0	0	0	0x0
R6	0x0	0	0	0x0
R7	0x0	0	0	0x0
R8	0x0	0	0	0x0
R9	0x0	0	0	0x0
R10	0x0	0	0	0x0
R11	0x0	0	0	0x0
R12	0x0	0	0	0x0
R13	0xFF000000	4294967296	1111111100000000	0xFF000000
LR	0x0	0	0	0x0
PC	0x58	88	1001100	0x58

Clock Cycles: 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 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.

The screenshot shows an ARM assembler emulator interface. On the left, the assembly code is displayed with line numbers 1 through 38. The code defines a binary search tree (BST) with 8 nodes and implements a search function. On the right, a register window shows the values of R0 through R13, LR, and PC. R0 is 0x0, R1 is 0x0, R2 is 0x1, R3 is 0x0, R4 is 0x0, R5 is 0x0, R6 is 0x0, R7 is 0x0, R8 is 0x0, R9 is 0x0, R10 is 0x0, R11 is 0x0, R12 is 0x0, R13 is 0xFF000000, LR is 0x0, and PC is 0x4C. The status bar at the bottom indicates 46 clock cycles and the current instruction is 0.

```

1  left_right  value
2 node1 DCD 0x10C, 0x130, 4
3 node2 DCD 0x118, 0x124, 2
4 node3 DCD 0, 0, 1
5 node4 DCD 0, 0, 3
6 node5 DCD 0x13C, 0x148, 6
7 node6 DCD 0, 0, 5
8 node7 DCD 0, 0, 7
9
10
11 LDR R0, =0 ; a value to look for
12 LDR R1, =0x100 ; struct node *R1 = node1 (a.k.a, the root)
13 loop LDR R2, [R1, #8] ; R2 = R1->value
14 CMP R2, R0 ; check if (R2 == R0)
15 BEQ found ; if R2 == R0 goes to found
16 BLT go_right ; if R2 < R0 goes to right size of the BST
17 BGT go_left ; if R2 > R0 goes to left size of the BST
18
19 go_left LDR R1, [R1, #0] ; R1 = R1->left root=root->left ; 10C 118 0
20 CMP R1, #0 ; chekc if root == nullptr
21 BEQ not_found ; root == nullptr return goes to not found
22 B loop ; goes back to loop
23
24
25 go_right LDR R1, [R1, #4] ; R1 = R1->right
26 CMP R1, #0 ; chekc if root == nullptr
27 BEQ not_found ; root == nullptr return goes to not found
28 B loop ; goes back to loop
29
30
31
32 found ADD R1, R1, #8 ; get the address for the R1
33 END ; Return
34
35 not_found END ;Return
36
37
38

```

Register	Value	Dec	Bin	Hex
R0	0x0	0	00000000	0x0
R1	0x0	0	00000000	0x0
R2	0x1	1	00000001	0x1
R3	0x0	0	00000000	0x0
R4	0x0	0	00000000	0x0
R5	0x0	0	00000000	0x0
R6	0x0	0	00000000	0x0
R7	0x0	0	00000000	0x0
R8	0x0	0	00000000	0x0
R9	0x0	0	00000000	0x0
R10	0x0	0	00000000	0x0
R11	0x0	0	00000000	0x0
R12	0x0	0	00000000	0x0
R13	0xFF000000	4294967296	11111111 00000000 00000000 00000000	0xFF000000
LR	0x0	0	00000000	0x0
PC	0x4C	76	00000100 1100	0x4C

Clock Cycles: 46  
Current Instruction: 0 Total: 46  
CSPR Status Bits (NZCV): 0 1 1 0

**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.

The screenshot shows an ARM assembly emulator with the following components:

- Assembly Code (Left Panel):**

```

1  ;left right value
2 node1 DCD 0x10C, 0x130, 4
3 node2 DCD 0x118, 0x124, 2
4 node3 DCD 0, 0, 1
5 node4 DCD 0, 0, 3
6 node5 DCD 0x13C, 0x148, 6
7 node6 DCD 0, 0, 5
8 node7 DCD 0, 0, 7
9
10
11 LDR R0, =5 ;a value to look for
12 LDR R1, =0x100 ;struct node *R1 = node1 (a.k.a, the root)
13 loop LDR R2, [R1, #8] ;R2 = R1->value
14 CMP R2, R0 ; check if(R2 == R0)
15 BEQ found ; if R2 == R0 goes to found
16 BLT go_right ; if R2 < R0 goes to right size of the BST
17 BGT go_left ; if R2 > R0 goes to left size of the BST
18
19 go_left LDR R1, [R1, #0] ; R1 = R1->left root=root->left ; 10C 118 0
20 CMP R1, #0 ; check if root == nullptr
21 BEQ not_found ; root == nullptr return goes to not found
22 B loop ; goes back to loop
23
24
25 go_right LDR R1, [R1, #4] ; R1 = R1->right
26 CMP R1, #0 ; check if root == nullptr
27 BEQ not_found ; root == nullptr return goes to not found
28 B loop ; goes back to loop
29
30
31 found ADD R1, R1, #8 ; get the address for the R1
32 END ; Return
33
34
35 not_found END ;Return
36
37
38

```
- Register Window (Right Panel):**

Register	Value	Dec	Bin	Hex
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	0x48			
- Execution Controls (Top Right):** Execute, Reset, Step Backwards, Step Forwards.
- Status Bar (Bottom):** Clock Cycles: 0 / Total: 38. CSRR Status Bits (NZCV): 0 | 1 | 1 | 0.

**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.

```

1 ;left right value
2 node1 DCD 0x10C, 0x130, 4
3 node2 DCD 0x118, 0x124, 2
4 node3 DCD 0, 0, 1
5 node4 DCD 0, 0, 3
6 node5 DCD 0x13C, 0x148, 6
7 node6 DCD 0, 0, 5
8 node7 DCD 0, 0, 7
9
10
11 LDR R0, #8 ;a value to look for
12 LDR R1, #0x100 ;struct node *R1 = node1 (a.k.a, the root)
13 loop LDR R2, [R1, #8] ;R2 = R1->value
14 CMP R2, R0 ; check if(R2 == R0)
15 BEQ found ; if R2 == R0 goes to found
16 BLT go_right ; if R2 < R0 goes to right size of the BST
17 BGT go_left ; if R2 > R0 goes to left size of the BST
18
19 go_left LDR R1, [R1, #0] ; R1 = R1->left root=root->left ; 10C 118 0
20 CMP R1, #0 ; chekc if root == nullptr
21 BEQ not_found ; root == nullptr return goes to not found
22 B loop ; goes back to loop
23
24
25 go_right LDR R1, [R1, #4] ; R1 = R1->right
26 CMP R1, #0 ; chekc if root == nullptr
27 BEQ not_found ; root == nullptr return goes to not found
28 B loop ; goes back to loop
29
30
31 found ADD R1, R1, #8 ; get the address for the R1
32 END ; Return
33
34 not_found END ;Return
35
36
37
38

```

R0	0x8	Dec	Bin	Hex
R1	0x0	Dec	Bin	Hex
R2	0x7	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	0x4C	Dec	Bin	Hex

Clock Cycles: 0 Current Instruction: 0 Total: 43  
CSPR Status Bits (NZCV): 0 1 1 0

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.

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

```

left size of the BST      BGT      go_left      ; if R2 > R0 goes to
                           ;
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
== nullptr    CMP      R1, #0      ;      ;
                           BEQ      not_found      ;      ;
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

```