

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.

```
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
src1           DCB       "CSS", 0
src1           DCB       "UWB", 0

                ALIGN
                ENTRY
Reset_Handler  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
```

```

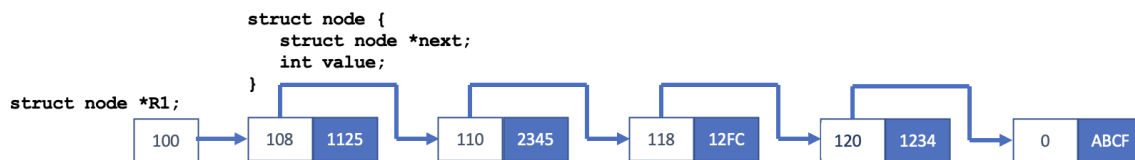
B      end_prog
END

```

Address	Contents
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).



Emulation Complete Line Issues 16 0

Execute Reset Step Backwards Step Forwards

Reset to continue editing code

```

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

```

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

Current Instruction: 0 Total: 44

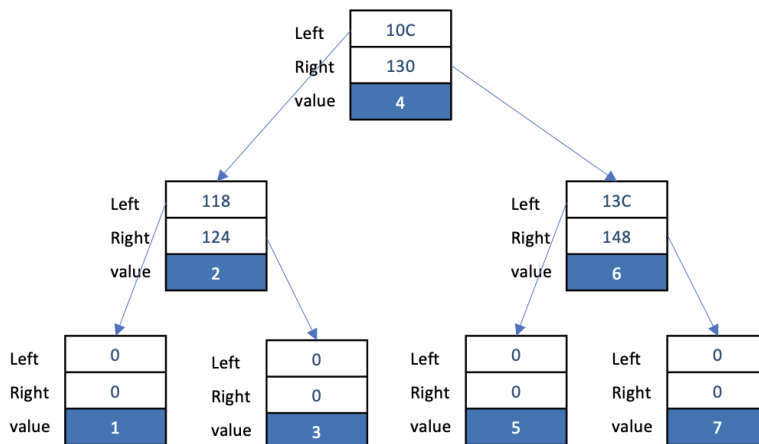
How about traversing a binary search tree?

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.

The screenshot shows an ARM assembly emulator interface. The top bar includes buttons for 'New', 'Open', 'Save', 'Settings', 'Tools', and a status bar indicating 'Emulation Complete' with 29 lines and 0 issues. Below the bar, the main window is divided into three sections:

- Source Code:** Displays assembly code for initializing a binary tree. It defines nodes 1 through 7 with their left, right, and value pointers. The code includes comments and a loop structure. A message 'Key answer is hidden.' is visible in a dark box.
- Registers:** A table showing the state of registers R0 through R13, LR, and PC. R0 contains 0x5, R1 contains 0x144, and R2 contains 0x5. Other registers are empty (0x0).
- View Memory Contents:** A window showing memory data from address 0x100 to 0x110. It displays a table with columns for Word Address, Byte 3, Byte 2, Byte 1, Byte 0, and Word Value. The data shows the tree structure being built in memory.

At the bottom, there are tabs for 'Word Value Format' (Dec, Hex), 'Memory Map Key', 'Instructions', and 'Data'. The status bar at the very bottom shows 'Clock Cycles' and 'Current Instruction: 0 Total: 41'.

What to submit: source code, screen shots, 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.

The screenshot displays an ARM emulator interface with the following components:

- Assembly Code:** Lines 1-31 of assembly code are visible, including node definitions, a search loop, and conditional branches for left and right tree traversal.
- Registers:** A table on the right shows registers R0 through R13, LR, and PC. R0-R13 contain 0x0, LR contains 0x0, and PC contains 0x58.
- Memory Map:** A table at the bottom shows memory contents from address 0x100 to 0x120. It lists word addresses, byte values, and word values in hexadecimal.
- Emulation Status:** The top bar indicates "Emulation Complete" with 29 lines and 0 instructions.
- Execution Controls:** Buttons for "Execute", "Reset", "Step Backwards", and "Step Forwards" are present.

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

NewOpenSaveSettingsTools

Emulation CompleteLine Issues290

ExecuteResetStep BackwardsStep Forwards

Reset to continue editing code

```
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 CMP R0, R2
14 BEQ found
15 CMP R0, R2
16 BLT chk_left
17 CMP R0, R2
18 BGT chk_right
19 chk_left LDR R1, [R1]
20 CMP R1, #0 ; (R1 = R1->left) != null?
21 BEQ not_found
22 B loop
23 chk_right LDR R1, [R1, #4]
24 CMP R1, #0 ; (R1 = R1->right) != null?
25 BEQ not_found
26 B loop
27 found ADD R1, R1, #8
28 not_found
29
30 END
31
```

View Memory Contents

Start address: 0x100End 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: DecHexMemory Map KeyInstructionsData

R00x5DecBinHex

R10x144DecBinHex

R20x5DecBinHex

R30x0DecBinHex

R40x0DecBinHex

R50x0DecBinHex

R60x0DecBinHex

R70x0DecBinHex

R80x0DecBinHex

R90x0DecBinHex

R100x0DecBinHex

R110x0DecBinHex

R120x0DecBinHex

R130xFF000000DecBinHex

LR0x0DecBinHex

PC0x58DecBinHex

Clock Cycles

Current Instruction: 0Total: 41

CSPR Status Bits (NZCV)0110

Test case 3 (2pts)

Missing any screenshot -1, missing explanation -1.

The screenshot displays an ARM emulator interface. The main window shows assembly code for a binary search algorithm. The code defines a linked list structure and implements a search function. The registers R0, R1, and R2 are used to track the search progress. The memory map shows the current state of memory, with the search value 0x100 stored at address 0x130.

Assembly Code:

```
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, =8 ; 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 CMP R0, R2
14 BEQ found
15 CMP R0, R2
16 BLT chk_left
17 CMP R0, R2
18 BGT chk_right
19 chk_left LDR R1, [R1]
20 CMP R1, #0 ; (R1 = R1->left) != null?
21 BEQ not_found
22 B loop
23 chk_right LDR R1, [R1, #4]
24 CMP R1, #0 ; (R1 = R1->right) != null?
25 BEQ not_found
26 B loop
27 found ADD R1, R1, #8
28 not_found
29 not_found
30 END
```

Registers:

Register	Value	Dec	Bin	Hex
R0	8	8	00000000	0x8
R1	0x100	256	00000100	0x100
R2	0x7	7	00000007	0x7
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	0xFF000000
LR	0x0	0	00000000	0x0
PC	0x58	88	00000000 00000100	0x58

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: Dec Hex

Memory Map Key: Instructions Data

Clock Cycles: Current Instruction: 0 Total: 52

CSPR Status Bits (NZCV): 0 1 1 0