

## Question 2.1.1

PRITISH WADHWA

1)

$$(1010\ 1011\ 1100\ 1101)_2 = (1 \times 2^{15}) + (0 \times 2^{14}) + (1 \times 2^{13}) + (0 \times 2^{12}) + (1 \times 2^{11}) + (0 \times 2^{10}) + (1 \times 2^9) + (1 \times 2^8) + (1 \times 2^7) + (1 \times 2^6) + (0 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = (43981)_{10}$$

2)

$$(1111\ 1110\ 1101\ 1100)_2 = (1 \times 2^{15}) + (1 \times 2^{14}) + (1 \times 2^{13}) + (1 \times 2^{12}) + (1 \times 2^{11}) + (1 \times 2^{10}) + (1 \times 2^9) + (0 \times 2^8) + (1 \times 2^7) + (1 \times 2^6) + (0 \times 2^5) + (1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0) = (65244)_{10}$$

3)

$$(0111\ 1101\ 1111\ 1000)_2 = (0 \times 2^{15}) + (1 \times 2^{14}) + (1 \times 2^{13}) + (1 \times 2^{12}) + (1 \times 2^{11}) + (1 \times 2^{10}) + (0 \times 2^9) + (1 \times 2^8) + (1 \times 2^7) + (1 \times 2^6) + (1 \times 2^5) + (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (0 \times 2^0) = (32248)_{10}$$

4)

$$(0011\ 0000\ 0011\ 1001)_2 = (0 \times 2^{15}) + (0 \times 2^{14}) + (1 \times 2^{13}) + (1 \times 2^{12}) + (0 \times 2^{11}) + (0 \times 2^{10}) + (0 \times 2^9) + (0 \times 2^8) + (0 \times 2^7) + (0 \times 2^6) + (1 \times 2^5) + (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = (12345)_{10}$$

## Question 2.1.2

PRITISH WADHWA

1)  
 $(A000)_{16} = (10 \times 16^3) + (0 \times 16^2) + (0 \times 16^1) + (0 \times 16^0) = (40960)_{10}$


2)  
 $(8A89)_{16} = (8 \times 16^3) + (10 \times 16^2) + (8 \times 16^1) + (9 \times 16^0) = (35465)_{10}$


3)  
 $(0190)_{16} = (0 \times 16^3) + (1 \times 16^2) + (9 \times 16^1) + (0 \times 16^0) = (400)_{10}$


4)  
 $(AFCD)_{16} = (10 \times 16^3) + (15 \times 16^2) + (12 \times 16^1) + (13 \times 16^0)$   
 $= (45005)_{10}$


## Question 2.1.3


PRITISH WADHWA

 To convert Hex to Octal, I will first convert hex to binary and then I will convert the corresponding binary sequence to octal.

 1)  
A      0      0      0  
1010 0000 0000 0000  
  
001 010 000 000 000 000  
1    2    0    0    0    0  
  
 $(A000)_{16} = (120000)_8$

 2)  
8      A      8      9  
1000 1010 1000 1001  
  
001 000 101 010 001 001  
1    0    5    2    1    1  
  
 $(8A89)_{16} = (105211)_8$

 3)  
0      1      9      0  
0000 0001 1001 0000  
  
000 000 000 110 010 000  
0    0    0    6    2    0  
  
 $(0190)_{16} = (000620)_8$

 4)  
A      F      C      D  
1010 1111 1100 1101  
  
001 010 111 111 001 101  
1    2    7    7    1    5  
  
 $(AFCD)_{16} = (127715)_8$

## Question 2.1.4

PRITISH WADHWA

1)

001	010	101	111	001	101
1	2	5	7	1	5

$$(1010\ 1011\ 1100\ 1101)_2 = (125715)_8$$

2)

001	111	111	011	011	100
1	7	7	3	3	4

$$(1111\ 1110\ 1101\ 1100)_2 = (177334)_8$$

3)

000	111	110	111	111	000
0	7	6	7	7	0

$$(0111\ 1101\ 1111\ 1000)_2 = (076770)_8$$

4)

000	011	000	000	111	001
0	3	0	0	7	1

$$(0011\ 0000\ 0011\ 1001)_2 = (030071)_8$$

## Question 2.2.2 - ARMSim

PRITISH WADHWA

ARMSim code :

```
MOV R1, #100          @ R1 = 100
MOV R2, #1            @ R2 = 1
MOV R4, #0            @ R4 = 0
@ start the iterations
.LOOP:
    @ extract the LSB and compare
    AND R3, R1, #1     @ R3 = R1 && 1
    CMP R3, #1         @ COMPARE R3, 1

    @ increment the counter
    ADDEQ R4, R4, #1    @ ADD(R4 = R4 + 1) IF Z = 1

    @ prepare for the next iteration
    MOV R1, R1, LSR #1 @ R1 = R1 WITH LSR 1
    ADD R2, R2, #1      @ R2 = R2 + 1

    @ loop condition
    CMP R2, #32         @ R2, 32
    BLE .LOOP           @ BRANCH TO LOOP IF R2<32

MOV R1, R4             @ R1 = R4
SWI 0x6b               @ PRINT
```

```
1  MOV    R1, #100          @ R1 = 100
2  MOV    R2, #1            @ R2 = 1
3  MOV    R4, #0            @ R4 = 0
4  @ start the iterations
5  .LOOP:
6      @ extract the LSB and compare
7      AND    R3, R1, #1     @ R3 = R1 && 1
8      CMP    R3, #1         @ COMPARE R3, 1
9
10     @ increment the counter
11     ADDEQ   R4, R4, #1    @ ADD(R4 = R4 + 1) IF Z = 1
12
13     @ prepare for the next iteration
14     MOV     R1, R1, LSR #1 @ R1 = R1 WITH LSR 1
15     ADD     R2, R2, #1    @ R2 = R2 + 1
16
17     @ loop condition
18     CMP     R2, #32       @ R2, 32
19     BLE     .LOOP        @ BRANCH TO LOOP IF R2<32
20
21     MOV     R1, R4        @ R1 = R4
22     SWI     0x6b         @ PRINT
```



## Question 2.2.2 - ARMsims

PRITISH WADHWA

The screenshot displays the ARMsim interface with three main panels:

- RegistersView:** Shows the state of 16 registers (R0-R15) and the CPSR register. R0-R14 are in hexadecimal, while R15 (PC) is in decimal. The CPSR register shows various flags like Negative(N), Zero(Z), Carry(C), Overflow(V), IRQ Disable, FIQ Disable, Thumb(T), and CPU Mode (System).
- CodeView:** Displays the assembly code for the program. The code is in ARMv7 assembly syntax and includes comments explaining the operations. The code starts with setting R1 to 100, R2 to 1, and R4 to 0. It then enters a loop that extracts the LSB of R1, compares it with 1, increments R4, and shifts R1 right by 1 bit. The loop continues until R2 reaches 32. Finally, it prints the value of R4.
- OutputView:** Shows the output of the program, which is the value 3.

**RegistersView Details:**

Register	Value
R0	:00000000
R1	:00000003
R2	:00000021
R3	:00000000
R4	:00000003
R5	:00000000
R6	:00000000
R7	:00000000
R8	:00000000
R9	:00000000
R10(s1)	:00000000
R11(fp)	:00000000
R12(ip)	:00000000
R13(sp)	:00001400
R14(lr)	:00000000
R15(pc)	:00011400

**CPSR Register:**

Flag	Value
Negative(N)	:0
Zero(Z)	:0
Carry(C)	:0
Overflow(V)	:0
IRQ Disable	:1
FIQ Disable	:1
Thumb(T)	:0
CPU Mode	:System

**CodeView Details:**

```
lab1_1_a.  
00001000:E3A01064MOV R1, #100 @ R1 = 100  
00001004:E3A02001MOV R2, #1 @ R2 = 1  
00001008:E3A04000MOV R4, #0 @ R4 = 0  
    @ start the iterations  
    .LOOP:  
    @ extract the LSB and compare  
0000100C:E2013001AND R3, R1, #1 @ R3 = R1 && 1  
00001010:E3530001CMP R3, #1 @ COMPARE R3, 1  
  
    @ increment the counter  
00001014:02844001ADDEQ R4, R4, #1 @ ADD(R4 = R4 + 1) IF Z = 1  
  
    @ prepare for the next iteration  
00001018:E1A010A1MOV R1, R1, LSR #1 @ R1 = R1 WITH LSR 1  
0000101C:E2822001ADD R2, R2, #1 @ R2 = R2 + 1  
  
    @ loop condition  
00001020:E3520020CMP R2, #32 @ R2, 32  
00001024:DAFFFFF8BLE .LOOP @ BRANCH TO LOOP IF R2<32  
  
00001028:E1A01004MOV R1, R4 @ R1 = R4  
0000102C:EF000006BSWI 0x6b @ PRINT
```

**OutputView:**

```
3
```

## Question 2.2.2 - visUAL

PRITISH WADHWA

visUAL code :

```
MOV      R1, #100      ; R1 = 100
MOV      R2, #1        ; R2 = 1
MOV      R4, #0        ; R4 = 0
;        start the iterations

LOOP

;        extract the LSB and compare
AND      R3, R1, #1     ; R3 = R1 && 1
CMP      R3, #1         ; COMPARE R3, 1

;        increment the counter
ADDEQ    R4, R4, #1     ; ADD(R4 = R4 + 1) IF Z = 1

;        prepare for the next iteration
MOV      R1, R1, LSR #1 ; R1 = R1 WITH LSR 1
ADD      R2, R2, #1     ; R2 = R2 + 1

;        loop condition
CMP      R2, #32        ; R2, 32
BLE      LOOP           ; BRANCH TO LOOP IF R2<32

MOV      R1, R4         ; R1 = R4
```

The screenshot displays the visUAL emulator interface. The top bar includes buttons for 'New', 'Open', 'Save', 'Settings', 'Tools', and a status bar showing 'Emulation Complete' with 'Line Issues: 21 0'. On the right, there are buttons for 'Execute', 'Reset', 'Step Backwards', and 'Step Forwards'.

The main window is divided into two panels. The left panel shows the assembly code being executed, with line numbers 1 through 22. The code is as follows:

```
1 MOV      R1, #100      ; R1 = 100
2 MOV      R2, #1        ; R2 = 1
3 MOV      R4, #0        ; R4 = 0
4 ;        start the iterations
5 LOOP
6 ;        extract the LSB and compare
7 AND      R3, R1, #1     ; R3 = R1 && 1
8 CMP      R3, #1         ; COMPARE R3, 1
9
10 ;        increment the counter
11 ADDEQ    R4, R4, #1     ; ADD(R4 = R4 + 1) IF Z = 1
12
13 ;        prepare for the next iteration
14 MOV      R1, R1, LSR #1 ; R1 = R1 WITH LSR 1
15 ADD      R2, R2, #1     ; R2 = R2 + 1
16
17 ;        loop condition
18 CMP      R2, #32        ; R2, 32
19 BLE      LOOP           ; BRANCH TO LOOP IF R2<32
20
21 MOV      R1, R4         ; R1 = R4
22
```


















The right panel shows the state of the registers (R0 through R13) and the Program Counter (PC). The values are as follows:

Register	Value	Dec	Bin	Hex
R0	0	0	00000000	00000000
R1	3	3	00000011	00000003
R2	33	33	00010001	00000021
R3	0	0	00000000	00000000
R4	3	3	00000011	00000003
R5	0	0	00000000	00000000
R6	0	0	00000000	00000000
R7	0	0	00000000	00000000
R8	0	0	00000000	00000000
R9	0	0	00000000	00000000
R10	0	0	00000000	00000000
R11	0	0	00000000	00000000
R12	0	0	00000000	00000000
R13	-16777216	-16777216	FF000000	FFFFFFFF
LR	0	0	00000000	00000000
PC	52	52	00000100	00000034



## Question 2.2.3 – Code Explanation

PRITISH WADHWA

-  The code is well explained using comments, still I will give a brief description about the same.
-  I would be using the code written in Sublime Text(Image attached previously) to explain my code.
-  The code starts at line 1 when '100' is transferred to R1.
-  In line 2, I am transferring '1' to R2. R2 would behave as the index for my program.
-  In line 3, I am transferring '0' to R4. R4 would function as my counter.
-  The main loop starts at line 5.
-  In line 7, I Bitwise AND my number(100) stored in R1 with '1'. This generates the LSB of my number(100) and stores it in R3.
-  Line 8 compares the value stored in R3 with '1' and updates the flag 'Z' accordingly.
-  The value of the flag 'Z' is '1', that is the value in R3 is equal to '1' else the value of the flag 'Z' is '0'(When the value in R3 is not equal to '1').
-  Line 11 checks the value of flag 'Z', if it is equal, '1' is added to R4(counter) and the value is stored back in R4.
-  In line 14, using the LSR(Logical Shift Right) command, the LSB of R1 is removed and the updated R1 is stored in R1 itself.
-  In line 15, the value of R2(index of the loop) is incremented by '1'.
-  Line 18 compares the value stored in R2 with the number '32'.
-  If the value in R2 is less than '32', the BLE command in the line 19 transfers back the control to line 5.
-  The above series of steps is repeated until R2 becomes equal to '32'.
-  Following the above statement, the control of the program moves to line 21 where the value stored in R4(the final answer) is finally transferred to R1.
-  Line 22 completes the program by printing the value stored in R1

## Question 2.3.1 - ARMsims

PRITISH WADHWA

ARMsims :

```
ARR :      .WORD 10, 24, 3, 14
LDR        R10, =ARR
MOV        R4, #0
MOV        R2, #0
```

```
.LOOP:
    LDR        R3, [R10, R2, LSL #2]
    ADD        R2, R2, #1
    ADD        R4, R4, R3
    CMP        R2, #4
    BNE        .LOOP
    MOV        R1, R4
    SWI        0x6b
```

```
1  ARR :      .WORD 10, 24, 3, 14
2  LDR        R10, =ARR
3  MOV        R4, #0
4  MOV        R2, #0
5
6  .LOOP:
7      LDR        R3, [R10, R2, LSL #2]
8      ADD        R2, R2, #1
9      ADD        R4, R4, R3
10     CMP        R2, #4
11     BNE        .LOOP
12     MOV        R1, R4
13     SWI        0x6b
```

The screenshot shows the Keil uVision IDE interface. The top toolbar contains standard development icons. The main workspace is divided into three panes:

- RegistersView (Left):** Displays the state of ARM registers. The 'General' tab is selected. Registers R0 through R15 are listed with their current values in hexadecimal. R10 is highlighted in red. The CPSR register is also shown with its flags (Negative, Zero, Carry, Overflow) and the Thumb bit. The CPU Mode is set to 'System'.
- CodeView (Middle):** Displays the assembly code for the 'labco.' section. The code includes instructions for loading, moving, adding, and comparing values, along with a loop structure. The address 00001000 is highlighted in red.
- OutputView (Bottom):** Currently empty, showing the console output area.

## Question 2.3.1 - visUAL

PRITISH WADHWA

visUAL code :

```
ARR      DCD      10, 24, 3, 14
          LDR      R10, =ARR
          MOV      R4, #0
          MOV      R2, #0
```

LOOP

```
          LDR      R3, [R10, R2, LSL #2]
          ADD      R2, R2, #1
          ADD      R4, R4, R3
          CMP      R2, #4
          BNE      LOOP
          MOV      R1, R4
```

The screenshot displays the visUAL emulator interface. The top bar includes menu items (New, Open, Save, Settings, Tools) and a status bar indicating 'Emulation Complete' with 13 lines and 0 issues. The main window is divided into two panes. The left pane shows the assembly code with line numbers 1 through 14. The right pane displays a register table with 14 registers (R0-R13), LR, and PC, each with a value and format (Dec, Bin, Hex).

Assembly Code:

```
1 ARR      DCD      10, 24, 3, 14
2          LDR      R10, =ARR
3          ;MOV     R0, =ARR/
4          MOV      R4, #0
5          MOV      R2, #0
6
7 LOOP
8          LDR      R3, [R10, R2, LSL #2]
9          ADD      R2, R2, #1
10         ADD      R4, R4, R3
11         CMP      R2, #4
12         BNE      LOOP
13         MOV      R1, R4
14
```

Register Table:

Register	Value	Dec	Bin	Hex
R0	0	Dec	Bin	Hex
R1	51	Dec	Bin	Hex
R2	4	Dec	Bin	Hex
R3	14	Dec	Bin	Hex
R4	51	Dec	Bin	Hex
R5	0	Dec	Bin	Hex
R6	0	Dec	Bin	Hex
R7	0	Dec	Bin	Hex
R8	0	Dec	Bin	Hex
R9	0	Dec	Bin	Hex
R10	256	Dec	Bin	Hex
R11	0	Dec	Bin	Hex
R12	0	Dec	Bin	Hex
R13	-16777216	Dec	Bin	Hex
LR	0	Dec	Bin	Hex
PC	44	Dec	Bin	Hex

## Question 2.3.2 - ARMsims

PRITISH WADHWA

ARMsim code:

ADDMUL :

ADD R5, R2, R3

MUL R5, R5, R4

BX LR

MAIN :

MOV R2, #11

MOV R3, #22

MOV R4, #33

BL ADDMUL

MOV R1,R5

SWI 0x6b

```
ADDMUL :  
    ADD R5, R2, R3  
    MUL R5, R5, R4  
    BX LR
```

```
MAIN :  
    MOV R2, #11  
    MOV R3, #22  
    MOV R4, #33  
    BL ADDMUL  
    MOV R1,R5  
    SWI 0x6b
```

The screenshot shows the Keil uVision IDE interface. The top toolbar contains icons for file operations, execution, and debugging. The main workspace is divided into three panes:

- RegistersView (Left):** Displays the state of 16 registers (R0-R15) and the CPSR register. The registers are shown in hexadecimal. R0-R15 values are: R0: 00000000, R1: 00000441, R2: 0000000b, R3: 00000016, R4: 00000021, R5: 00000441, R6: 00000000, R7: 00000000, R8: 00000000, R9: 00000000, R10(s1): 00000000, R11(fp): 00000000, R12(ip): 00000000, R13(sp): 00001400, R14(lr): 0000101c, R15(pc): 00011400. The CPSR register shows: Negative(N): 0, Zero(Z): 0, Carry(C): 0, Overflow(V): 0, IRQ Disable: 1, FIQ Disable: 1, Thumb(T): 0, CPU Mode: System. The current PC value is 0x000000df.
- CodeView (Top Right):** Displays the assembly code for the 'Lab1\_2\_2' target. It shows two sections: 'ADDMUL' and 'MAIN'. The 'ADDMUL' section contains three instructions: '00001000:E0825003ADD R5, R2, R3', '00001004:E0050495MUL R5, R5, R4', and '00001008:E12FFF1EBX LR'. The 'MAIN' section contains seven instructions: '0000100C:E3A0200BM0V R2, #11', '00001010:E3A03016M0V R3, #22', '00001014:E3A04021M0V R4, #33', '00001018:EBFFFFFF8BL ADDMUL', '0000101C:E1A01005M0V R1,R5', and '00001020:EF00006BSWI 0x6b'.
- OutputView (Bottom Right):** Currently empty, showing the 'Console' tab.

## Question 2.3.2 - visUAL

PRITISH WADHWA

visUAL code:

```
MOV    R1, #11
MOV    R2, #22
MOV    R3, #33
BL     ADDMUL
END
```

ADDMUL

```
ADD    R4, R1, R2
MOV    R5, #0
```

LOOP

```
ADD    R5, R5, R4
SUB    R3, R3, #1
CMP    R3, #0
BNE    LOOP
MOV    R10, R5
END
```

The screenshot displays the visUAL emulator interface. The top toolbar includes buttons for 'New', 'Open', 'Save', 'Settings', 'Tools', 'Emulation Complete' (with a checkmark), 'Line Issues' (15, 0), 'Execute', 'Reset', 'Step Backwards', and 'Step Forwards'. The main window is split into two panes. The left pane shows assembly code with line numbers 1 through 18. The right pane shows a register table with 14 registers (R0-R13), LR, and PC, each with a value and three format buttons (Dec, Bin, Hex).

Assembly Code (Left Pane):

```
1  MOV    R1, #11
2  MOV    R2, #22
3  MOV    R3, #33
4  BL     ADDMUL
5  END
6
7  ADDMUL
8      ADD    R4, R1, R2
9      MOV    R5, #0
10 LOOP
11      ADD    R5, R5, R4
12      SUB    R3, R3, #1
13      CMP    R3, #0
14      BNE    LOOP
15      MOV    R10, R5
16  END
17
18
```

Register Table (Right Pane):

Register	Value	Dec	Bin	Hex
R0	0	Dec	Bin	Hex
R1	11	Dec	Bin	Hex
R2	22	Dec	Bin	Hex
R3	0	Dec	Bin	Hex
R4	33	Dec	Bin	Hex
R5	1089	Dec	Bin	Hex
R6	0	Dec	Bin	Hex
R7	0	Dec	Bin	Hex
R8	0	Dec	Bin	Hex
R9	0	Dec	Bin	Hex
R10	1089	Dec	Bin	Hex
R11	0	Dec	Bin	Hex
R12	0	Dec	Bin	Hex
R13	-16777216	Dec	Bin	Hex
LR	16	Dec	Bin	Hex
PC	64	Dec	Bin	Hex