Q1. Embedded systems always require the user to manipulate bits in registers or variables. Given an integer variable a, write two code fragments in C. The first should set bit 3 of a. The second should clear bit 3 of a. In both cases, the remaining bits should be unmodified.

1 a = a | 1 << 3

2 a = a &~ (1 << 3)

Q2. Develop a sequence of instructions that sets the rightmost four bits of R3, clears the leftmost three bits of R3, and inverts bit positions 7,8 and 9 of R3. Assuming that R3 is 16-bit register.

① Rightmost four bits → 0xF ⟹ ORR R3, R3, 0xF

② leftmost three bits → 0x1FFF ⟹ AND R3, R3, 0x1FFF

③ bit positions 7, 8, 9 → 0000 0011 1000 0000 ⟹ EOR R3, R3, 0x0380

#Note : leftmost bit is
bit position zero

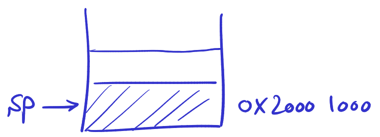
Q3. When does the LR have to be pushed on the stack?

→ Nested function

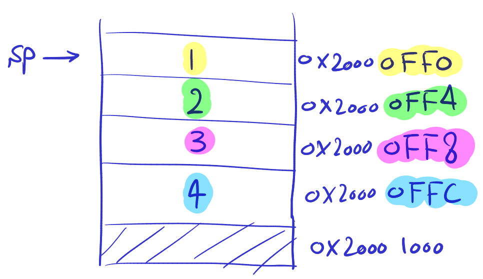
Q4. Show the SP value and the content of stack after executing this instruction PUSH {R4, R6-R8} assuming the SP initially equals 0x2000.1000 and R4=1, R6=2, R7=3, R8=4. What will be the values of the registers R0-R4 after executing this instruction POP{R0-R3}?

* R4 = 1
* R6 = 2
* R7 = 3
* R8 = 4

① initially :




② After push :




③ After pop : R0 = 1
R1 = 2
R2 = 3
R3 = 4

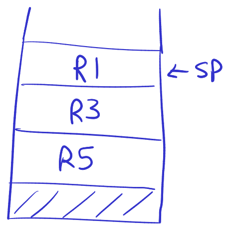
Q5. Explain how does the return from subroutine work in these two functions?

| Function PUSH {R4,LR} | Function2 |
|---|---|
| ;stuff | ;stuff |
| POP {R4,PC} | BX LR |

| Function PUSH | Function2 |
|---|---|
| push{lr} is putting the return address, in the link register, onto the stack when the subroutine is called.<br><br>pop{pc} is fetching that return address from the stack and putting it into the program counter, thus returning control back to the place the subroutine was called from. | At the end of the subroutine, the BX LR instruction will retrieve the return address from the LR register, returning the program to the place from which the subroutine was called.<br><br>More precisely, it returns to the instruction immediately after the instruction that performed the subroutine call. |

Q6. Write assembly code that pushes registers R1, R3, and R5 onto the stack.

→ PUSH {R1,R3,R5} =




Q7. What are the addressing modes used in each of the following instructions?

```
LDR R0, [R1]
LDR R2, [R1, #4]
MOV   R3, #100
BL function
MOV R0, #1
LDRB   R0, [PC, #0x30]
LDR R0, =1234567
```

① Regular Register indirect

② Regular indirect with immediate offset

③ Immediate addressing

④ PC Relative addressing

Q8. Write a complete ARM assembly program for the procedure func2. The procedure func2 calculates this C expression ((X+Y)>>3) – Z and stores its value in R0. Assume X, Y, Z are 32-bit signed numbers. X, Y, Z are defined in the memory as shown

```
        AREA   mydata, DATA, READONLY
X         DCD    -20
Y         DCD    -60
Z         DCD    -20
```

```
Func2       LDR R0, =X
            LDR R1, [R0]         ;R1=X
            LDR R0, =Y
            LDR R2, [R0]         ;R2=Y
            LDR R0, =Z
            LDR R3, [R0]         ;R3=Z

            ADD R0, R1, R2       ;R0=X+Y
            ASR R0, R0, #3       ;R0>>3   ==  (X+Y)>>3
            SUB R0, R0, R3       ;R0=R0-Z == ((X+Y)>>3)-Z

            BX LR
```

Q9. Write a complete ARM assembly program that calls the procedure func1 which in turn calls a procedure func2. The procedure func2 is defined in Q8 of Sheet 3.

```
Reset_Handler
            BL  Func1

DeadLoop    B  Deadloop
```

```
Func1       PUSH {LR}
            BL  Func2
            POP  {PC}            ; == POP {LR}
                                 ;    BX LR
```

```
Func2       LDR R0, =X
            ...
            BX LR
```

Juba