Time: 45 Minutes **Total Marks: 10 + 8 + 17 + 5 + 5 = 45 Marks**

5 Questions

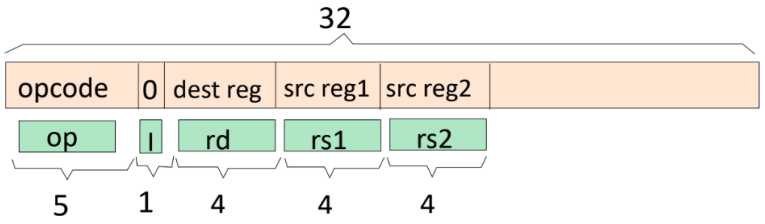
Q1. 32-bit split for the following instructions corresponding to an ISA having 16 registers and 21 instructions is below. Match the instruction with the correct encoding corresponding to that instruction:

1. mov rd, rs (move contents of rs to rd)
2. nop (no operation)
3. add rd, rs1, rs2 (add the values in registers rs1+rs2 and store in rd)
4. ld rd, #imm[rs] (load rd with contents at memory location obtained by adding #imm and contents of rs)

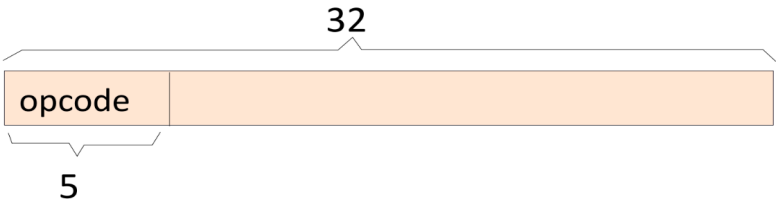
[2*4 = 8 Marks]

Note: *Wherever explicitly visible, the 6th bit equal to 1 in the instruction code means an immediate value is present as one of the operands. 0 means all operands are registers.*

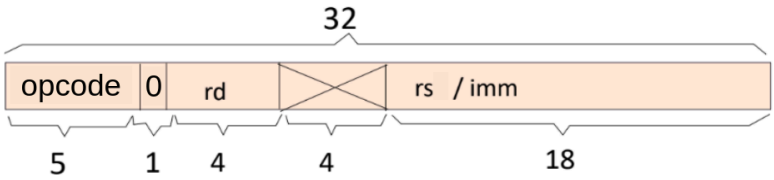
A.



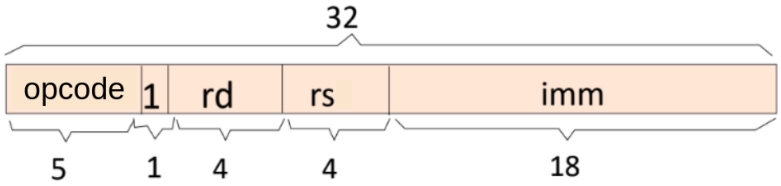
B.



C.



D.



Solution:

SETA:

1. C
2. B
3. A
4. D

SETB:

1. B
2. C
3. A
4. D

Q2. Consider a stack that starts from memory location **0x3FF (hexadecimal) [SETA] 0x2EF [SETB]**. Consider the two instructions below for stack operations.
[1*10 = 10 Marks]

Syntax	Semantics
PUSH reg1	Pushes the value of register reg1 into the stack.
POP reg1	Pops the value from the top of the stack into register reg1

Whenever a value is put into the stack via **PUSH** instruction, the value of the Stack Pointer (SP) **decrements** by one (binary). Whenever a value is moved out of the stack via **POP** instruction, the value of SP **increments** by one (binary). For the given program, determine the value of SP for each instruction.

Solution:
SETA:

Instruction	Updated value of SP (Hex)
1. POP r2	0x400
2. POP r2	0x401
3. PUSH r1	0x400
4. POP r1	0x401
5. POP r5	0x402
6. POP r4	0x403
7. PUSH r2	0x402
8. POP r3	0x403
9. PUSH r5	0x402
10. POP r7	0x403

SETB:

Instruction	Updated value of SP (Hex)
1. PUSH r12	0x2EE
2. PUSH r3	0x2ED
3. POP r15	0x2EE
4. PUSH r1	0x2ED
5. POP r4	0x2EE
6. PUSH r14	0x2ED
7. PUSH r3	0x2EC
8. PUSH r4	0x2EB
9. POP r8	0x2EC
10. POP r9	0x2ED

Q3.Study the ISA given below and answer the questions that follow. **[17 Marks]**

S. No.	Instruction (Mnemonic, Operands)	Operation Performed
1.	ADD R1, R2, R3	Add contents of registers R2 and R3 and store the result in R1.
2.	SUBI R1, #Imm	Subtract the content of the specified register(R1) by immediate value (imm).
3.	BNZ R1, Address	Branch to address if the content of specified register(R1) is not equal to zero.
4.	MUL R1, R2	Multiply the contents of registers R1, R2 and store the result in the specified register, R1.
5.	MOV R1, R2	Copy the content of the register R2 into the register R1.
6.	LD R1, Address	Load the content specified at the mentioned Address into the register (R1).
7.	ST R1, Address	Store the content of register R1 to the specified address.
8.	MVI R1, #Imm	Copy the Immediate value into the specified register (R1).
9.	BL Address	Branch to the mentioned address and store the return address (address of next instruction) in the Link register.

(a) Study the pseudo-code and corresponding assembly code given below and fill in the blanks (marked as "**BLANK**") based on the given ISA, pseudo-code, and associated assumptions. You can fill in assembly instructions or immediate value or register in the blanks, whichever is correct. **[6*2 = 12 Marks]**

Assume the following for the assembly code:

1. All registers are initialized to value 0 by default. There are 16 registers (r0 to r15) in total.
2. The final result will be a quotient retaining only the integral value. E.g. for dividend 5 and divisor 2, the assembly code should give 2 as a result (quotient) rather than 2.5.

***Note: "//"** indicates the beginning of an in-line comment.*

Pseudo-code:

```
main:
        temp_number = number_which_has_to_be_divided_by_2
        quotient_value = 0
        final_answer = sub_again (temp_number)
        store final_answer at roll_number
sub_again:
        temp_number = temp_number - 2
        quotient_value = quotient_value + 1
        if (temp_number == 0) then // for even number, repeated subtraction by 2 results in remainder = 0.
                return quotient_value and go back to the caller
        temp_number_2 = temp_number - 1 // for odd number, repeated subtraction by 2 results in remainder = 1.
        if(temp_number_2 ==0) then // checking if remainder - 1 == 0 for odd number.
                return quotient_value and go back to the caller
        call sub_again
end
```

Assembly-code: ***(line numbers do not imply addresses; they are just for visual clarity.)***

```
main:
1.    mvi r3 #1
2.    mvi BLANK #10 // Here, the dividend is taken to be 10.
3.    BLANK sub_again // calling sub_again.
4.    st r14 #roll_no // storing the quotient in #roll_no.

sub_again:
5.    subi r15 #2 // repeated subtraction of dividend with divisor.
6.    add  r14 r14 BLANK // iterating the quotient.
7.    mov  r13 r15
8.    BLANK  r15 sub_again // For even no., dividend will become 0 in last iteration and this loop won't get reiterated.
9.    subi r13 BLANK
10.   bnz  r13 BLANK // For odd no., r13 will become 0 in last iteration and this loop won't get reiterated.
11.   mov r10, r7 // Updating PC (program-counter) to return back to the caller.
```

(b) Report all the possible caller function(s) and callee function(s) in your code. **[1*2 = 2 Marks]**

(c) After studying the assembly code, identify the registers corresponding to the program counter (PC) and link register. **[1.5*2 = 3 Marks]**

Solution: ***(common for both the sets)***

(a) ***Note:*** *If any other combination of blanks fetches the same result, it should be marked as correct.*

```
main:
1.    mvi r3 #1
2.    mvi r15 #10 // Here, the dividend is taken to be 10
3.    bl sub_again // calling sub_again
4.    st r14 #roll_no // storing the quotient in #roll_no
sub_again:
5.    subi r15 #2 // repeated subtraction of dividend with divisor
6.    add  r14 r14 r3 // iterating the quotient
7.    mov  r13 r15
8.    bnz   r15 sub_again // For even no., r15 will become 0 in last iteration and this loop won't get re-iterated.
9.    subi r13 #1
10.   bnz  r13 sub_again // For odd no., r13 will become 0 in last iteration and this loop won't get reiterated.
11.   mov r10, r7 // passing the link register value to PC to jump back to the caller.
```

(b) Caller: main **[1 Mark]**
Callee: sub_again **[1 Mark]**

(c) PC: r10 **[1.5 Marks]**
Link register: r7 **[1.5 Marks]**

Since the value of r7 is being passed to r10 for the return to the caller, this means that when BL was executed, the address to be returned was stored in r7.

Q4. Using the ISA in Q3, write an assembly code to implement the given pseudo-code: **[5 Marks]**

Pseudo-code: ***(line numbers do not imply addresses; they are just for visual clarity.)***

```
1. main:
2.      number1 = last_digit_of_your_roll_number + 1
3.      number2 = last_second_digit_of_your_roll_number
4.      number3 = number1 + number2
5. label1:
6.      number1=number1 + number2
7.      number3=number3 - 1
8.      if(number3 !=0) then
9.              go_to label1
10. exit
```

Solution: ***(common for both the sets)***

Note: *There can be multiple solutions; any other code fetching the same functionality should be marked as correct.*

Assembly-code:

```
main:
      mvi r1 #7
      mvi r4 #1
      add r1 r1 r4
      mvi r2 #0
      add r3, r2, r1
label1:
      add r1 r2 r1
      subi r3 #1
      bnz r3 label1
      bl exit
exit:
```

Marking Scheme:

1. **1 Mark**: Correctly implementing pseudo-code lines 2, 3.
2. **0.5 marks**: Correctly implementing pseudo-codes line 4.
3. **1 mark**: Correctly implementing pseudo-codes lines 6, 7.
4. **2 Marks**: Correct condition checking and branching to label1 in pseudo-code lines 8 and 9, respectively.
5. **0.5 marks**: Correctly exiting the pseudocode line 10. It is not compulsory to use the “bl” instruction to exit.

Q5. A 32-bit ISA has 16 registers (R0, R1..) and 21 instructions with encoding same as in Q1. Of these 21 instructions, the first 9 are provided in a table in Q3. Decode the binary instruction codes provided below and identify the operation being performed (opcode), the source registers, the destination registers, and the immediate value. **[5 Marks]**

Note: *The 6th bit equal to 1 in the instruction code means an immediate value is present as one of the operands. 0 means all operands are registers.*

SETA:

1. 0010_0000_0110_0000_0000_0000_0000_0000
2. 0000_0010_0110_0000_0000_0000_0000_0000

SETB:

1. 0011_0000_1110_0011_1111_1111_1111_1111
2. 0001_1010_0110_0100_0100_0000_0000_0000

Hint*: For e.g., for the 3rd instruction in the table in Q3, the 5-bit opcode will be = 00010 (BNZ).*

Solution:

SETA:

1. 00100: MOV **[1 Mark]**
   0001: R1 (Destination) **[0.5 Marks]**
   0000: R0 (Source) **[1 Mark]**

2. 00000: ADD **[1 Mark]**
   1001: R9 (Destination) **[0.5 Marks]**
   1000: R8 (Source1) **[0.5 Marks]**
   0000: R0 (Source2) **[0.5 Marks]**

SETB:

1. 00110: MOV **[1 Mark]**
   0011: R3 (Destination) **[0.5 Marks]**
   1111: R15 (Source) **[1 Mark]**

2. 00011: ADD **[1 Mark]**
   1001: R9 (Destination) **[0.5 Marks]**
   1001: R9 (Source1) **[0.5 Marks]**
   0001: R1 (Source2) **[0.5 Marks]**