ES1 EE3400.

1. For the counter machine with instruction set:
CLR (r), INC (r), JE (r_j, r_k, z)

- INC (r): INCrement the contents of register *r*.
- CLR (r): CLeaR register *r*. (Set *r* to zero.)
- JE (r_j, r_k, z): IF the contents of register r_j Equals the contents of register r_k THEN Jump to instruction *z* ELSE continue in sequence.

Write a program to
A. Add two numbers.
B. Multiply two numbers.
Assume unsigned binary representation and two numbers given in registers r2 and r3, the result should be stored in r1. Register r0 is hardwired to 0.

2. Write the following expression in reverse polish notation and verify the correctness of results when fed to a stack machine.
((10 * (6 / ((9 + 3) * -11))) + 17) + 5

3. For the stack machine having the following instructions (apart from the operators +, -, *),

- Dump shows the stack at any point of time
- DUP duplicates the top of stack (TOS)
- SWAP interchanges the top two elements
- DROP pops the TOS discarding it
- "." pops and prints the TOS
- "=" pops the top two elements, compares them, and push 1 onto the stack if there are equal, and 0 if no
- IF [instructions] THEN, check the TOS, if it is 0, then execute instructions that are listed till THEN.

Construct a program to
A. Calculate the sum of the cube of first n numbers.
B. Calculate the factorial of a number by recursion.

4. Run the following code in python interpreter to look at bytecode generated

```python
def add(a, b):
    return a + b
import dis
dis.dis(add)
```

It should give you:

```
2        0 LOAD_FAST            0 (a)
         2 LOAD_FAST            1 (b)
         4 BINARY_ADD
         6 RETURN_VALUE
```

You can also get more information using:

```python
import opcode
code = add.__code__.co_code
for i in range(0, len(code)-1, 2):
    opcode_value, oparg = code[i], code[i+1]
    print(f"opcode_value: {opcode_value}, opcode_name:
{opcode.opname[opcode_value]}, oparg: {oparg}")
```

5. For the turing machine discussed in class draw the diagram of FSMs to achieve the following:
A. Starting from the initial tape location of a blank, moving toward the right hand side you are given two binary numbers separated by a blank. Add them and store the result in place of first number.
For example: - - - - 1 0 1 - 1 1 1 - - -
Should yield: - - - 1 1 0 0 - 1 1 1 - - -

B. For detection of the string {0^n1^n} where n is not given to you. The binary string lies to the right of the initial position and followed by blank symbols in the end. If there is a match, place 1 just right to the initial position, else place 0.
Ex: - - - - 0 0 0 1 1 1 - - -
Gives - - - - 1 0 0 1 1 1 - - -