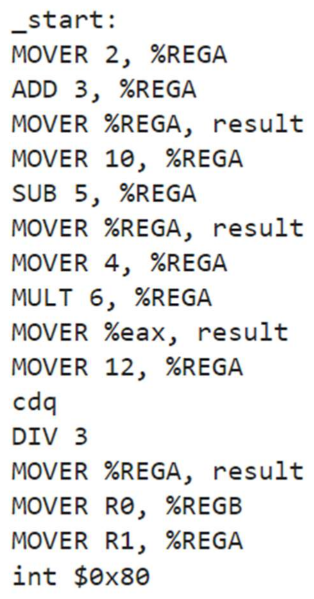
Experiment no 7

```
class CodeGenerator:
    def __init__(self, intermediate_code):
        self.intermediate_code = intermediate_code
        self.generated_code = []

    def generate_code(self):
        self.generated_code.append('_start:')
        for instruction in self.intermediate_code:
            opcode = instruction['opcode']
            operands = instruction['operands']
            if opcode == 'ADD':
                self.add(operands)
            elif opcode == 'SUB':
                self.sub(operands)
            elif opcode == 'MULT':
                self.mult(operands)
            elif opcode == 'DIV':
                self.div(operands)
            else:
                raise ValueError(f"Invalid opcode '{opcode}'")
        self.generated_code.append('MOVER R0, %REGB')
        self.generated_code.append('MOVER R1, %REGA')
        self.generated_code.append('int $0x80')

    def add(self, operands):
        op1, op2, result = operands
        self.generated_code.append(f"MOVER {op1}, %REGA")
        self.generated_code.append(f"ADD {op2}, %REGA")
        self.generated_code.append(f"MOVER %REGA, {result}")

    def sub(self, operands):
        op1, op2, result = operands
        self.generated_code.append(f"MOVER {op1}, %REGA")
        self.generated_code.append(f"SUB {op2}, %REGA")
        self.generated_code.append(f"MOVER %REGA, {result}")

    def mult(self, operands):
        op1, op2, result = operands
        self.generated_code.append(f"MOVER {op1}, %REGA")
        self.generated_code.append(f"MULT {op2}, %REGA")
        self.generated_code.append(f"MOVER %eax, {result}")

    def div(self, operands):
        op1, op2, result = operands
        self.generated_code.append(f"MOVER {op1}, %REGA")
        self.generated_code.append(f"cdq")
        self.generated_code.append(f"DIV {op2}")
        self.generated_code.append(f"MOVER %REGA, {result}")

# Example intermediate code
intermediate_code = [
    {'opcode': 'ADD', 'operands': ['2', '3', 'result']},
    {'opcode': 'SUB', 'operands': ['10', '5', 'result']},
    {'opcode': 'MULT', 'operands': ['4', '6', 'result']},
    {'opcode': 'DIV', 'operands': ['12', '3', 'result']}
]

# Generate x86 assembly code
code_generator = CodeGenerator(intermediate_code)
code_generator.generate_code()
assembly_code = '\n'.join(code_generator.generated_code)

# Print generated x86 assembly code
print(assembly_code)
```

Output:



```
_start:
MOVER 2, %REGA
ADD 3, %REGA
MOVER %REGA, result
MOVER 10, %REGA
SUB 5, %REGA
MOVER %REGA, result
MOVER 4, %REGA
MULT 6, %REGA
MOVER %eax, result
MOVER 12, %REGA
cdq
DIV 3
MOVER %REGA, result
MOVER R0, %REGB
MOVER R1, %REGA
int $0x80
```