class AssemblerPassOne:

    def \_\_init\_\_(self):

        self.symbol\_table = {}       # For storing labels and variables (symbols)

        self.intermediate\_code = []  # For storing intermediate code

        self.opcode\_table = {

            "ADD": "01",

            "SUB": "02",

            "MUL": "03",

            "DIV": "04",

            "LOAD": "05",

            "STORE": "06",

            "START": "00",

            "END": "FF",

            "ORIGIN": "A0",  # Just a placeholder for ORIGIN directive opcode

            "LTORG": "B0",   # Placeholder for LTORG directive opcode

            "DECLARE": "C0", # Placeholder for DECLARE directive

        }

        self.literal\_table = []      # For storing literals

        self.pool\_table = []         # For storing literal pool indexes

        self.current\_address = 0     # Tracks the current memory address

        self.literal\_index = 0       # Index for literals

    def pass\_one(self, source\_code):

        for line in source\_code:

            parts = line.strip().split()

            if not parts:

                continue

            # Handle ORIGIN directive (sets starting address)

            if parts[0] == "ORIGIN":

                self.current\_address = int(parts[1]) if len(parts) > 1 else self.current\_address

                continue

            # Handle LTORG directive (process literals)

            if parts[0] == "LTORG":

                self.process\_literals()

                continue

            # Handle END directive

            if parts[0] == "END":

                self.intermediate\_code.append((self.current\_address, "FF"))  # Using FF for END opcode

                break

            # Handle labels (symbol table)

            if parts[0].endswith(":"):

                label = parts[0][:-1]  # Remove the colon from the label

                self.symbol\_table[label] = self.current\_address

                parts = parts[1:]  # Remove the label part from the instruction

            # Handle DECLARE statements (constant and storage declarations)

            if parts[0] == "DECLARE":

                if len(parts) >= 2:

                    declaration\_type = parts[1]

                    if declaration\_type == "CONSTANT":

                        value = parts[2] if len(parts) > 2 else "0"

                        self.symbol\_table[parts[2]] = self.current\_address

                    elif declaration\_type == "STORAGE":

                        variable = parts[2] if len(parts) > 2 else "UNKNOWN"

                        self.symbol\_table[variable] = self.current\_address

                    self.current\_address += 1

                continue

            # Handle opcodes and add to intermediate code

            if parts[0] in self.opcode\_table:

                opcode = self.opcode\_table[parts[0]]

                self.intermediate\_code.append((self.current\_address, opcode))

                self.current\_address += 1

            # Handle literals (e.g., =5, =A)

            for part in parts:

                if part.startswith('='):

                    literal = part[1:]

                    if literal not in self.literal\_table:

                        self.literal\_table.append(literal)

                        if not self.pool\_table or self.pool\_table[-1] != self.literal\_index:

                            self.pool\_table.append(self.literal\_index)

                        self.literal\_index += 1

    def process\_literals(self):

        # This method processes literals when LTORG is encountered

        for literal in self.literal\_table:

            self.intermediate\_code.append((self.current\_address, f"= {literal}"))

            self.current\_address += 1

        self.literal\_table.clear()  # Clear literal table after processing

    def display\_tables(self):

        print("Symbol Table:")

        for label, address in self.symbol\_table.items():

            print(f"{label}: {address}")

        print("\nLiteral Table:")

        for i, literal in enumerate(self.literal\_table):

            print(f"Literal {i + 1}: {literal}")

        print("\nPool Table:")

        for i, index in enumerate(self.pool\_table):

            print(f"Pool {i + 1}: Index {index}")

        print("\nIntermediate Code:")

        for code in self.intermediate\_code:

            print(f"Address: {code[0]}, Opcode: {code[1]}")

        # Constants and Storage Declarations are omitted from final output

# To run pass one

if \_\_name\_\_ == "\_\_main\_\_":

    source\_code = [

        "START 100",

        "LOOP:  ADD A, B",

        "DECLARE CONSTANT 1000",

        "DECLARE STORAGE a",

        "DECLARE STORAGE b",

        "ORIGIN 200",

        "LTORG",

        "END"

    ]

    assembler = AssemblerPassOne()

    assembler.pass\_one(source\_code)

    assembler.display\_tables()

    print("\n=== Code Execution Successful ===")