**Q2 . Functionalities ( code Flow) :-**

• In compiler construction, control flow refers to the order in which the program's statements are executed. It involves managing the flow of control through the generated code, ensuring that the program executes in the correct sequence

• CODE :- class Compiler: def \_\_init\_\_(self): self.label\_count = 0 def compile\_if\_else(self, condition, true\_branch, false\_branch): # Generate unique labels for branches true\_label = f"LABEL\_TRUE\_{self.label\_count}" false\_label = f"LABEL\_FALSE\_{self.label\_count}" end\_label = f"LABEL\_END\_{self.label\_count}" # Increment label count for uniqueness self.label\_count += 1 # Compile the condition self.compile\_condition(condition, true\_label, false\_label) # Emit code for the true branch self.emit\_label(true\_label) self.compile(true\_branch) self.emit\_jump(end\_label) # Emit code for the false branch self.emit\_label(false\_label) self.compile(false\_branch) # Emit the end label self.emit\_label(end\_label) def compile\_condition(self, condition, true\_label, false\_label): # Simplified: Assume condition is a variable or constant self.emit("LOAD", condition) self.emit\_jump\_if\_zero(false\_label) def compile(self, code): # Simplified compilation of code print("Compiling:", code) def emit(self, operation, operand=None): print(f"{operation} {operand}") def emit\_jump(self, label): print(f"JUMP {label}") def emit\_jump\_if\_zero(self, label): print(f"JUMP\_IF\_ZERO {label}") def emit\_label(self, label): print(f"{label}:") # Example usage compiler = Compiler() if\_else\_code = "x 0 JUMP\_IF\_ZERO LABEL\_TRUE\_0 LABEL\_FALSE\_0" compiler.compile\_if\_else("x", "true\_branch\_code", "false\_branch\_code")

