**EXP 39 THREE STAGE PIPELINE AND OPERATION**

#include <stdio.h>

// Define the instruction structure

typedef struct {

int opcode; // Operation code

int operand1;

int operand2;

} Instruction;

// Function prototypes

void instructionFetch(Instruction\* instr);

void instructionDecode(Instruction\* instr);

void execute(Instruction\* instr);

int main() {

// Create an array of instructions for demonstration

Instruction program[] = {

{1, 10, 5}, // Example: opcode 1 represents addition, operands 10 and 5

{2, 8, 3}, // Example: opcode 2 represents multiplication, operands 8 and 3

{3, 15, 7} // Example: opcode 3 represents subtraction, operands 15 and 7

};

int numInstructions = sizeof(program) / sizeof(program[0]);

// Simulation of the three-stage pipeline for each instruction

for (int i = 0; i < numInstructions; i++) {

Instruction currentInstr = program[i];

// Stage 1: Instruction Fetch

instructionFetch(&currentInstr);

// Stage 2: Instruction Decode

instructionDecode(&currentInstr);

// Stage 3: Execute

execute(&currentInstr);

}

return 0;

}

// Simulate the Instruction Fetch stage

void instructionFetch(Instruction\* instr) {

printf("Instruction Fetch: Opcode %d\n", instr->opcode);

}

// Simulate the Instruction Decode stage

void instructionDecode(Instruction\* instr) {

printf("Instruction Decode: Operand1 %d, Operand2 %d\n", instr->operand1, instr->operand2);

}

// Simulate the Execute stage

void execute(Instruction\* instr) {

int result;

// Perform operation based on the opcode

switch (instr->opcode) {

case 1: // Addition

result = instr->operand1 + instr->operand2;

printf("Execute: Addition Result %d\n", result);

break;

case 2: // Multiplication

result = instr->operand1 \* instr->operand2;

printf("Execute: Multiplication Result %d\n", result);

break;

case 3: // Subtraction

result = instr->operand1 - instr->operand2;

printf("Execute: Subtraction Result %d\n", result);

break;

default:

printf("Invalid opcode\n");

}

}

OUTPUT:

