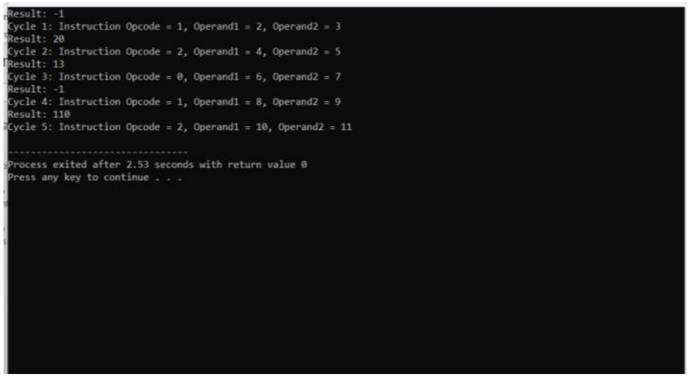
4STAGE PIPELINING:

```
#include <stdio.h>
// Structure representing an instruction
typedef struct {
    int opcode;
    int operand1;
    int operand2;
} Instruction;

// Structure representing the pipeline registers
typedef struct {
    Instruction instruction;
    int result;
} PipelineRegister;

// Function to simulate instruction fetch stage
void fetch_stage(int *instruction_count, Instruction *current_instruction) {
    // Simulating fetching instructions from memory
    // Increment instruction count
    (*instruction_count)++;
    // Simulating instruction decoding
    current_instruction->opcode = (*instruction_count) % 3;  // Example: alternating opcodes
    current_instruction->operand1 = (*instruction_count) * 2;
    current_instruction->operand2 = (*instruction_count) * 2 + 1;
}

// Function to simulate instruction decode stage
void decode_stage(Instruction *current_instruction, PipelineRegister *decode_reg) {
    // Transfer the instruction to the decode register
    decode_reg->instruction = *current_instruction;
}
```

```
// Function to simulate execute stage
void execute_stage(PipelineRegister *decode_reg, PipelineRegister *execute_reg) {
    // Simulating instruction execution
    switch (decode_reg->instruction.opcode) {
        case 0:
            execute_reg->result = decode_reg->instruction.operand1 + decode_reg-
>instruction.operand2;
            break;
        case 1:
            execute_reg->result = decode_reg->instruction.operand1 - decode_reg-
>instruction.operand2;
            break;
        case 2:
            execute_reg->result = decode_reg->instruction.operand1 * decode_reg-
>instruction.operand2;
            break;
        default:
            printf("Invalid opcode\n");
            break;
    }
}

// Function to simulate writeback stage
void writeback_stage(PipelineRegister *execute_reg) {
    // Printing the result obtained from the execution stage
    printf("Result: %d\n", execute_reg->result);
}

int main() {
    int instruction_count = 0;
    Instruction current_instruction;
    PipelineRegister decode_reg, execute_reg;

    // Perform multiple cycles of the pipeline stages
    for (int i = 0; i < 5; i++) { // Example: 5 cycles
        // Instruction fetch stage
        fetch_stage(&instruction_count, &current_instruction);

        // Instruction decode stage
        decode_stage(&current_instruction, &decode_reg);

        // Instruction execute stage
        execute_stage(&decode_reg, &execute_reg);

        // Instruction writeback stage
        writeback_stage(&execute_reg);

        // Output the current instruction being processed
        printf("Cycle %d: Instruction Opcode = %d, Operand1 = %d, Operand2 = %d\n",
               i + 1, current_instruction.opcode, current_instruction.operand1,
current_instruction.operand2);
    }

    return 0;
}
```

Input &output:



```
Result: -1
Cycle 1: Instruction Opcode = 1, Operand1 = 2, Operand2 = 3
Result: 20
Cycle 2: Instruction Opcode = 2, Operand1 = 4, Operand2 = 5
Result: 13
Cycle 3: Instruction Opcode = 0, Operand1 = 6, Operand2 = 7
Result: -1
Cycle 4: Instruction Opcode = 1, Operand1 = 8, Operand2 = 9
Result: 110
Cycle 5: Instruction Opcode = 2, Operand1 = 10, Operand2 = 11

--------------------------------
Process exited after 2.53 seconds with return value 0
Press any key to continue . . .
```