**FOUR STAGE PIPE LINE**

**EXP NO: 36**

**AIM:** To write a C program to implement four stage pipeline.

**ALGORITHM:**

**1. Instruction Fetch (IF):**

- Fetch the instruction from memory based on the program counter (PC).

- Increment the PC for the next instruction.

- Place the fetched instruction in the Instruction Register (IR).

**2. Instruction Decode (ID):**

- Decode the opcode of the instruction in the IR to determine the operation to be performed.

- If the instruction involves data, read the operands from the register file or memory.

- Prepare the operands for the next stage.

**3. Execute (EX):**

- Execute the operation specified by the opcode using the provided operands.

- Perform arithmetic or logical operations, or calculate memory addresses.

- Generate the result of the operation.

**4. Write Back (WB):**

- Write the result of the operation back to the register file or memory.

- Update the register file with the new values if necessary.

- Prepare for the next instruction**.**

**Pipeline Operation:**

- While one instruction is in the Write Back (WB) stage, the next instruction can be in the Execute (EX) stage, the one after in the Decode (ID) stage, and the one after that in the Fetch (IF) stage.

- Each clock cycle, instructions move one stage down the pipeline.

- New instructions are fetched in every clock cycle, creating a continuous flow of instructions through the pipeline**.**

**Handling Hazards:**

- Detect and handle data hazards, such as read-after-write (RAW) hazards, through techniques like forwarding and stalling.

- Detect and handle control hazards, such as branch instructions, through techniques like branch prediction.

**Overview:**

1. Initialize the pipeline stages, registers, and memory.

2. Begin the clock cycle loop.

- Execute the operations of each stage in parallel for the current set of instructions.

- Handle data hazards and control hazards as needed.

- Move instructions to the next stage.

- Fetch new instructions into the Instruction Fetch (IF) stage.

3. Repeat the clock cycle loop until the program completes.

**PROGRAM:**

#include<stdio.h>

int main(){

int counter=0;

int input;

int num1,num2;

int op;

int res;

int ins;

int performance\_measure=0;

printf("\n Enter 1st value:");

scanf("%d",&num1);

counter+=1;

printf("\n Enter the 2ndvalue: ");

scanf("%d",&num2);

counter+=1;

printf("\n Enter the option:\n1)Addition\n2)Subraction\n3)Multiplication\n4)Division");

scanf("%d",&op);

switch(op){

case 1:

printf("Performing addition operation");

res=num1+num2;

counter+=1;

break;

case 2:

printf("Performing subraction operation");

res=num1-num2;

counter+=1;

break;

case 3:

printf("Performing multiplication operation");

res=num1\*num2;

counter+=1;

break;

case 4:

if(num2==0){

printf("\nDenominator can't be zero");

}

else{

printf("Performing division operation");

res=num1/num2;

counter+=1;

break;

}

default:

printf("Invalidcase...");

counter+=3;

break;

}

printf("\n CYCLE VALUE IS :%d",counter);

printf("Enter the no.instruction");

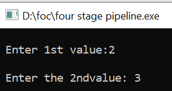
scanf("%d",&ins);

performance\_measure=ins/counter;

printf("\n Performance Measure is: %d",performance\_measure);

}

**INPUT:**



**OUTPUT:**

**A screenshot of a computer

Description automatically generated**

**RESULT:** Thus the program was executed successfully using DevC++.