Indian Institute of Information Technology

Vadodara

Computer Organization and Architecture

Project

Course Instructor

Dr Kamal Kishor Jha

Title

MIPS Pipeline Simulator

Submitted By

Tushar 201951163

Table of Contents

Note – Click on the title, you will be redirected to the respective page

Abstract

In this project we have created a MIPS (Multiprocessor without Interlocked Pipeline Stages) Simulator which will help user in visualize that what happens in each stage of the pipeline and how instructions are executed in an overlapped manner. We have created a console-based application where user will input some instruction in a given format and set data of registers, the system will print what happens in each clock cycle in each stage of a pipeline.

Programming Language

C++ programming language is used.

Theory

MIPS is a computer architecture where execution of an instruction is divided into 5 pipelined stages which executes instructions in an overlapped manner. These 5 pipeline stages are

1. **Instruction Fetch**

   Instruction is fetched using address provided by Program Counter (PC) and stored in Instruction Register (IR). PC also gets incremented by 4.

2. **Instruction Decode**

   Instruction provided by IR is decoded and operands are fetched from register file.

3. **Execute**

   ALU operation are performed in this stage or executes the instruction.

4. **Memory Access**

   This stage will access memory for fetch or store operation of operands.

5. **Write Back**

   Instructions who has result or has a destination register, result gets stored in the destination register.

Instruction Format

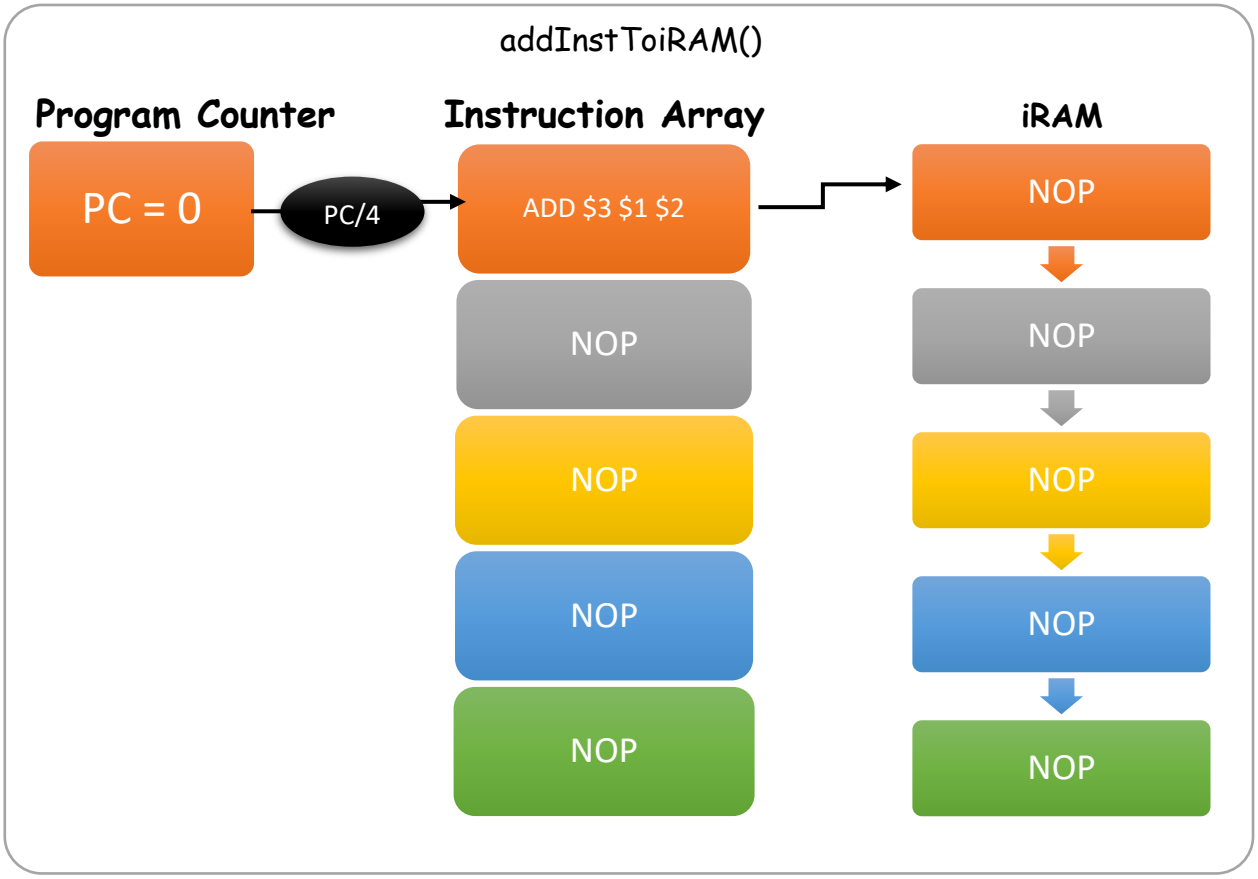
For executing Instructions follow this format table

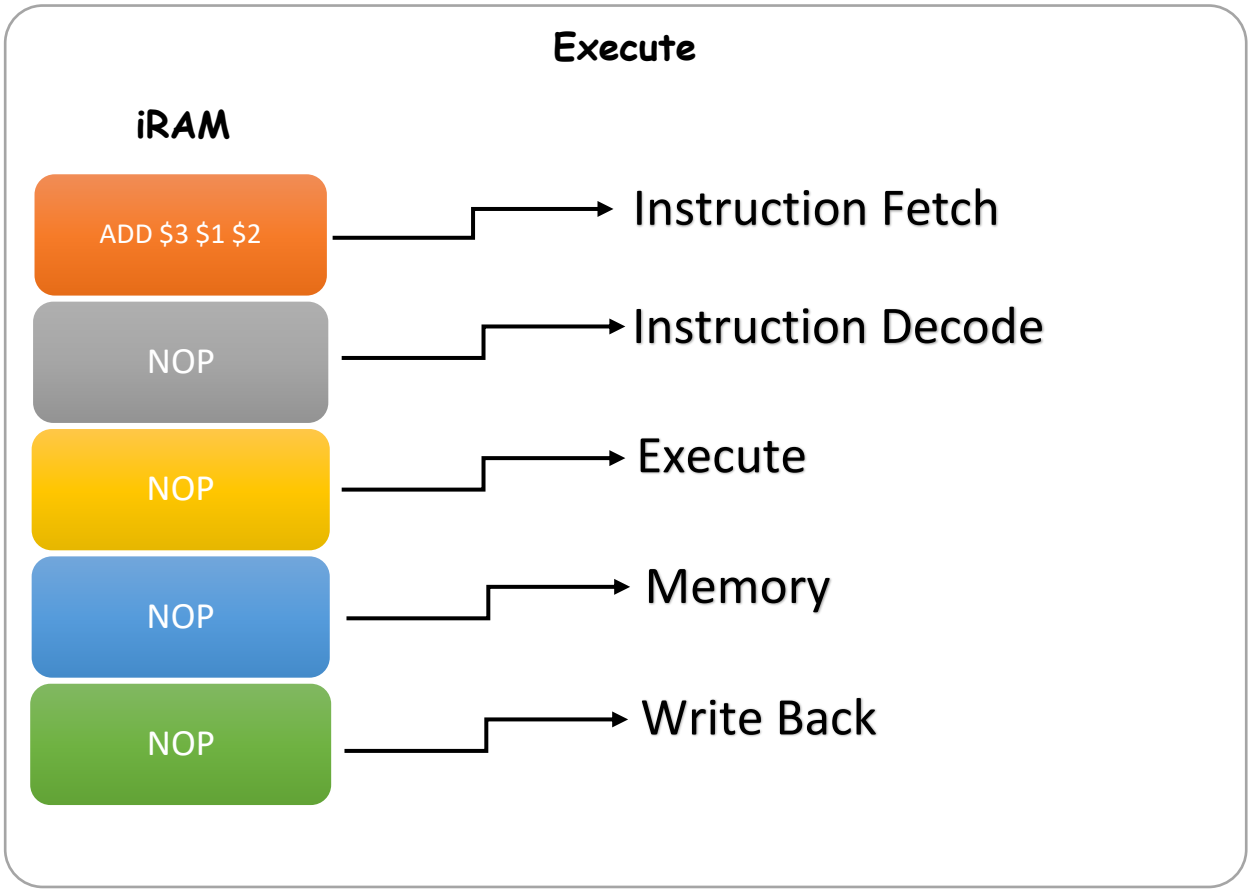
Instruction	Format	Description
ADD	ADD $4 $1 $2	$4 = $1 + $2 Terminate the program if there is data overflow
ADDU	ADDU $4 $1 $2	$4 = $1 + $2
SUB	SUB $4 $1 $2	$4 = $1 - $2 Terminate the program if there is data overflow
SUBU	SUBU $4 $1 $2	$4 = $1 - $2
MUL	MUL $4 $1 $2	$4 = $1 * $2 Terminate the program if there is data overflow
MULU	MULU $4 $1 $2	$4 = $1 * $2
DIV	DIV $4 $1 $2	$4 = $1 / $2 Terminate the program if there is data overflow
DIVU	DIVU $4 $1 $2	$4 = $1 / $2
XOR	XOR $4 $1 $2	$4 = $1 xor $2
ADDIU	ADDIU $2 $1 100	$2 = $1 + 100
LW	LW $2 100($1)	$2 = M[$1 + 100] Load data from memory using effective address and save it in
SW	SW $2 100($1)	M[$1 + 100] = $2 Save data from register 2 to memory
MOVE	MOVE $1 $2	$1 = $2 Save value of $2 to $1
BEQ	BEQ $1 $2 1	If $1 = $2 then jump to instruction no 1 (count starts from 0) For example

		I[0] = ADD $3 $1 $2 I[1] = BEQ $1 $3 0 I[1] will branch to I[0] if $1 = $3
BNE	BNE $2 $1 1	If $1 != $2 then jump to instruction no 1 (count starts from 0)
BNEZ	BNEZ $2 1	If $2 != 0 then jump to instruction no 1 (count starts from 0)
BEQZ	BEQZ $2 1	If $2 = 0 then jump to instruction no 1 (count starts from 0)

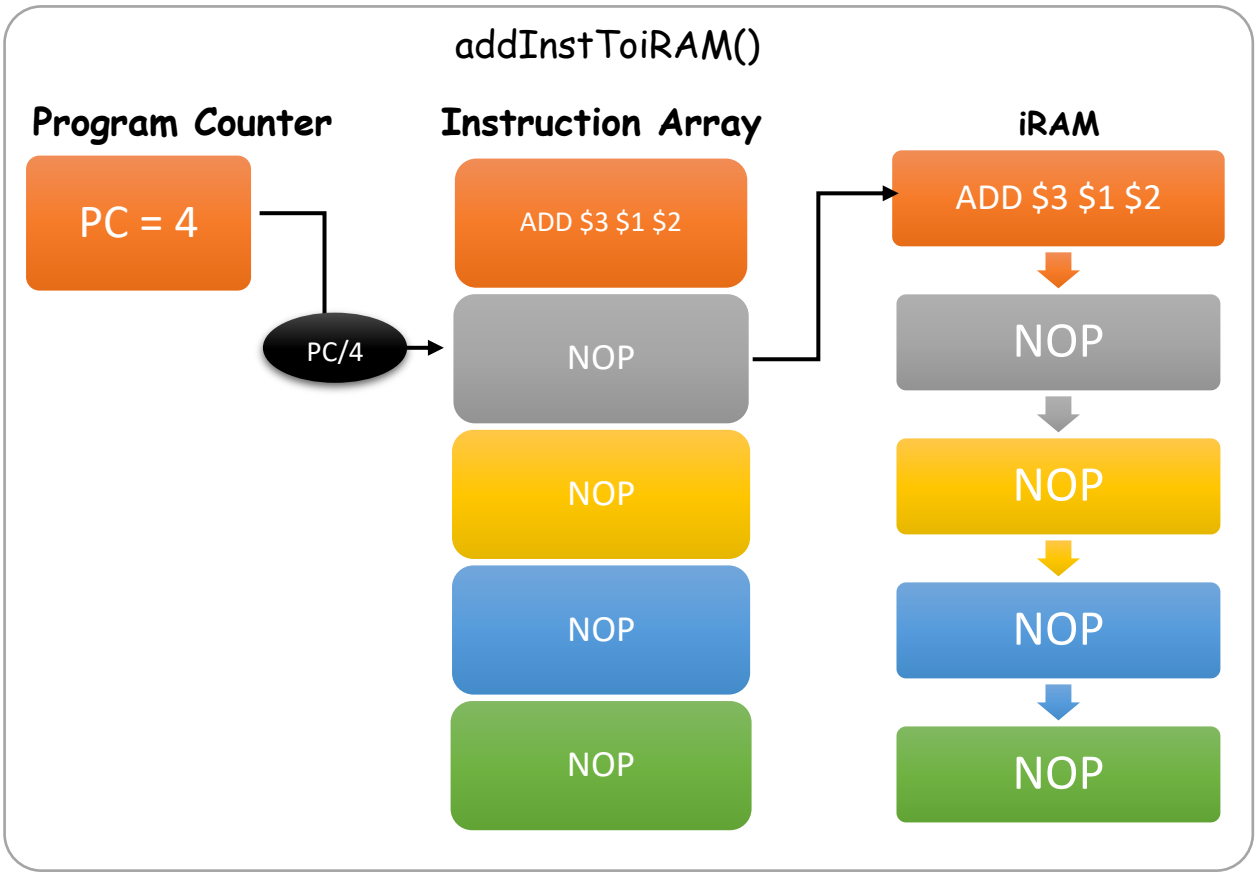
Algorithm (Graphical View)

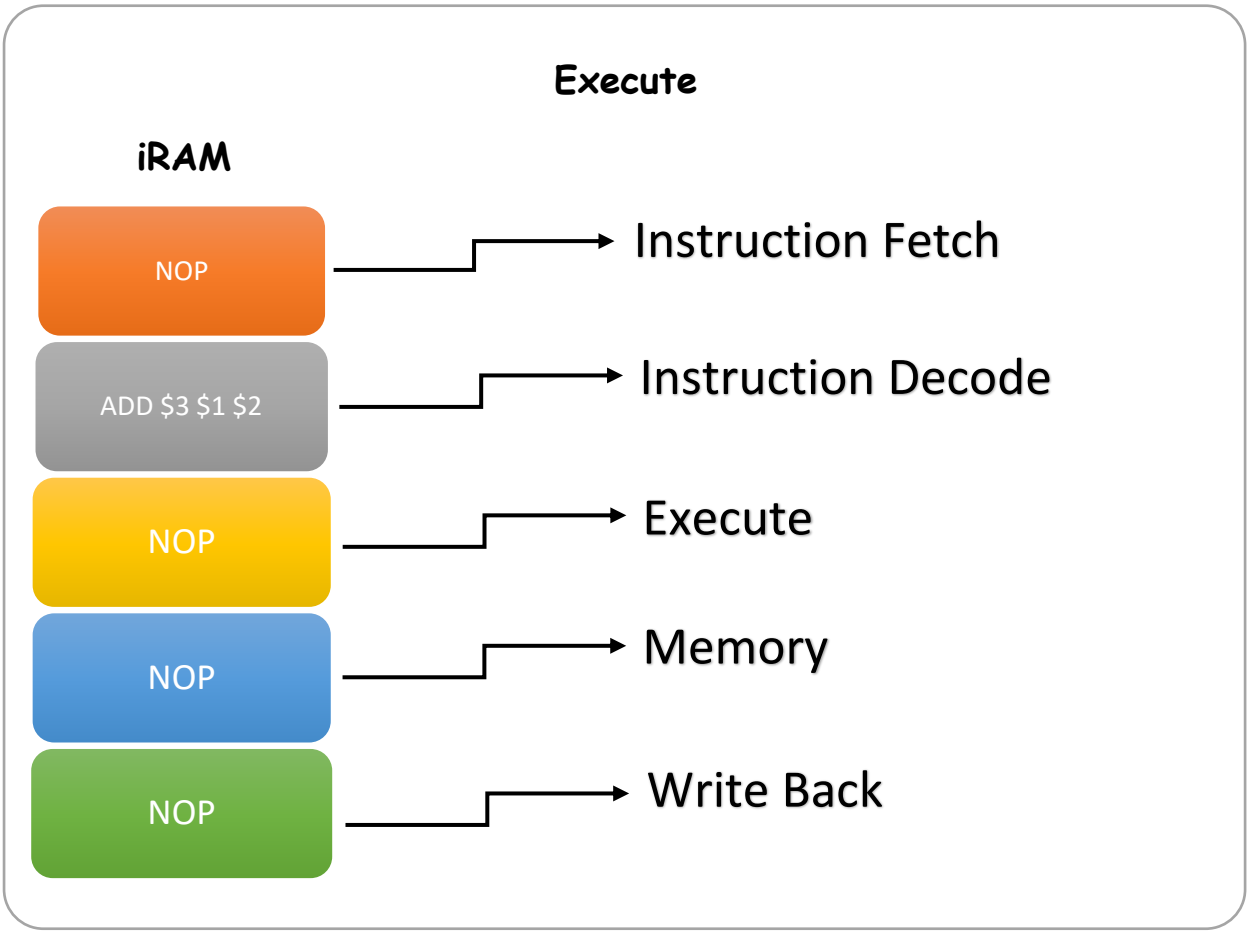
Clock 1



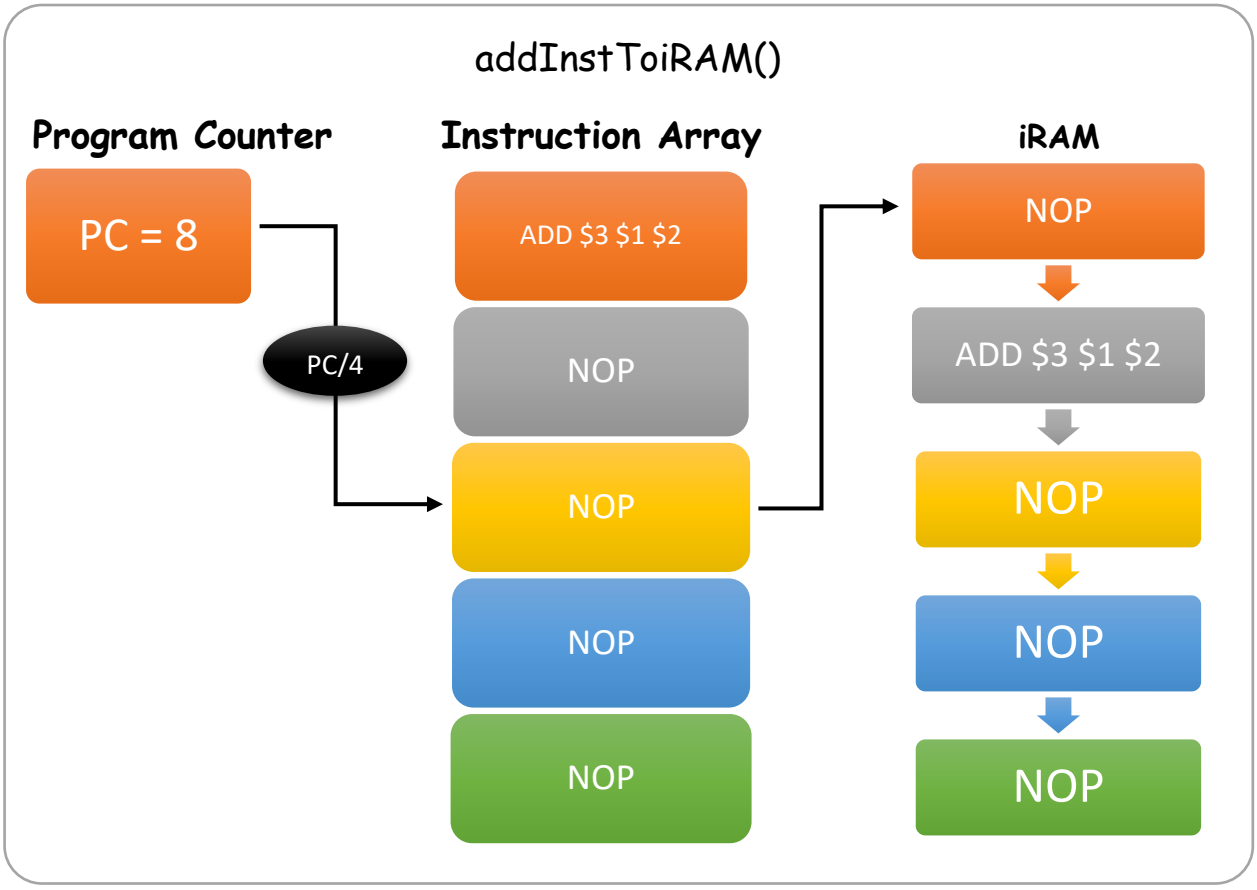


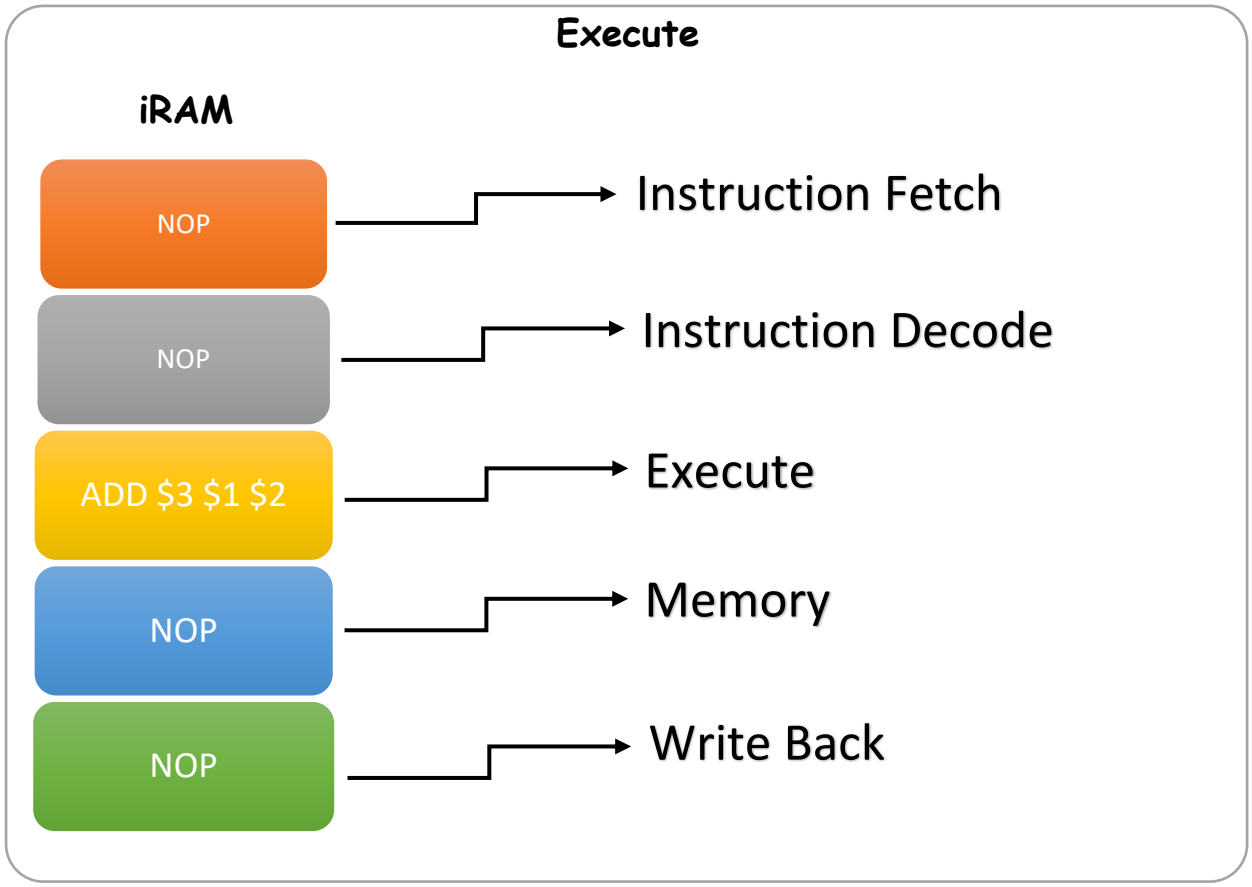
Clock 2


addInstToiRAM()
Program Counter
Instruction Array
iRAM
PC = 4
PC/4
ADD $3 $1 $2
NOP
NOP
NOP
NOP
ADD $3 $1 $2
NOP
NOP
NOP
NOP

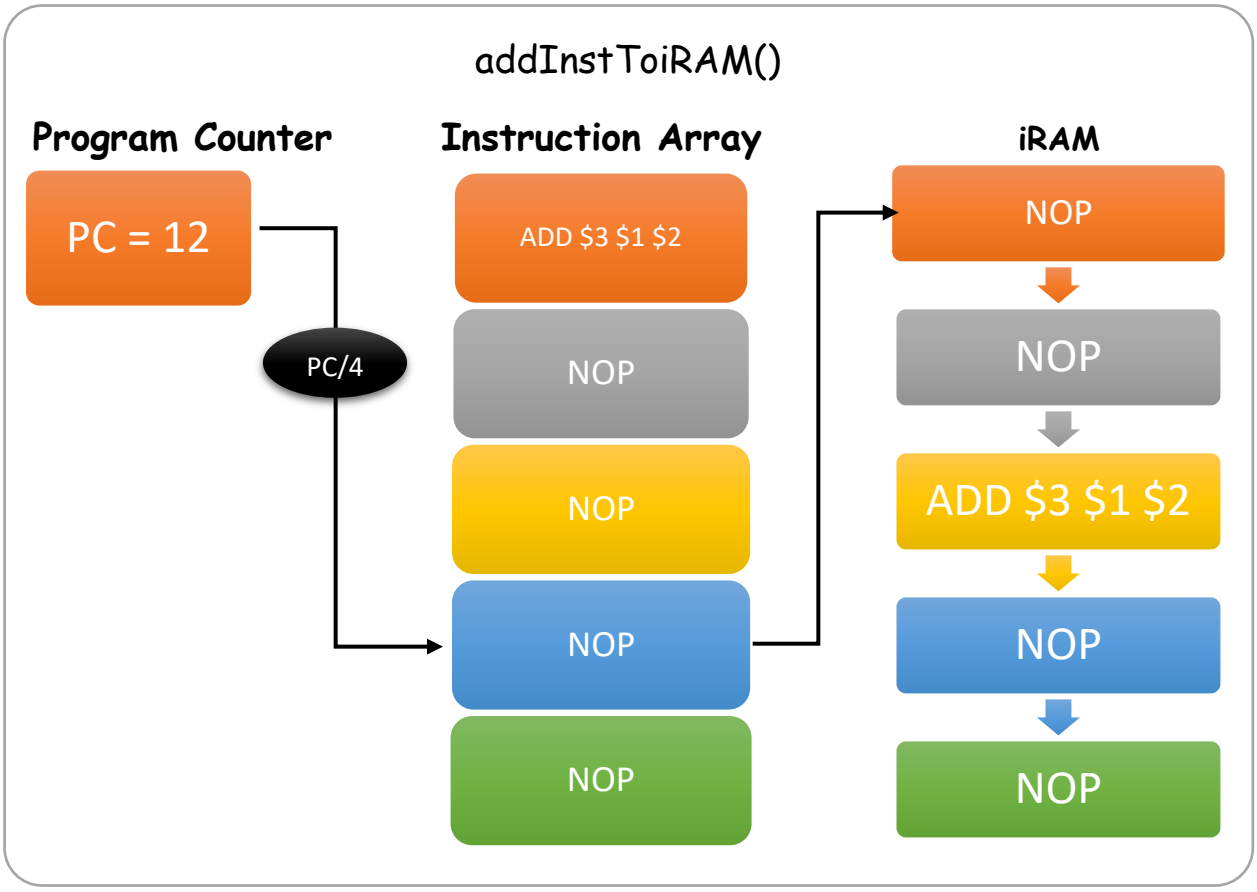

Execute
iRAM
NOP
ADD $3 $1 $2
NOP
NOP
NOP
Instruction Fetch
Instruction Decode
Execute
Memory
Write Back

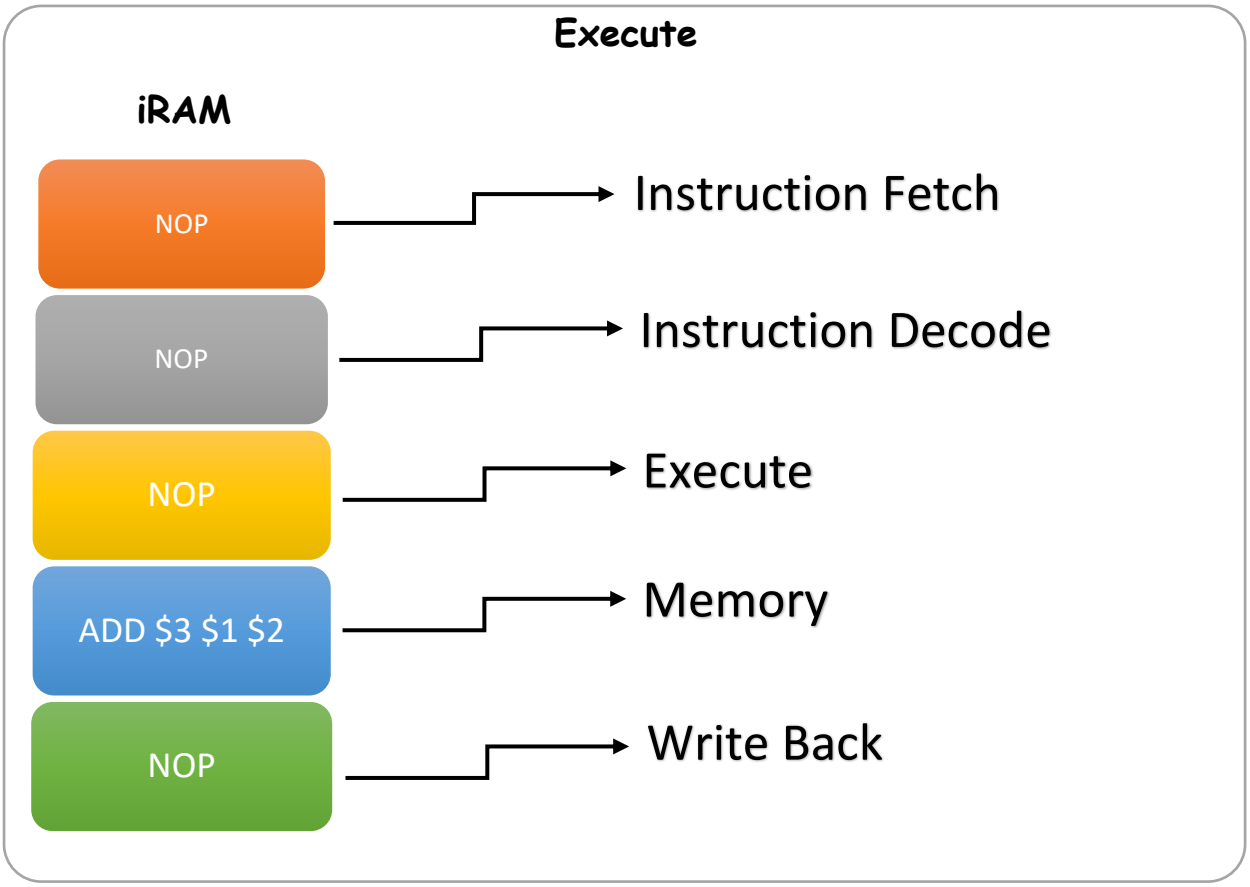
Clock 3


addInstToiRAM()
Program Counter
Instruction Array
iRAM
PC = 8
PC/4
ADD $3 $1 $2
NOP
NOP
NOP
NOP
NOP
NOP
ADD $3 $1 $2
NOP
NOP
NOP

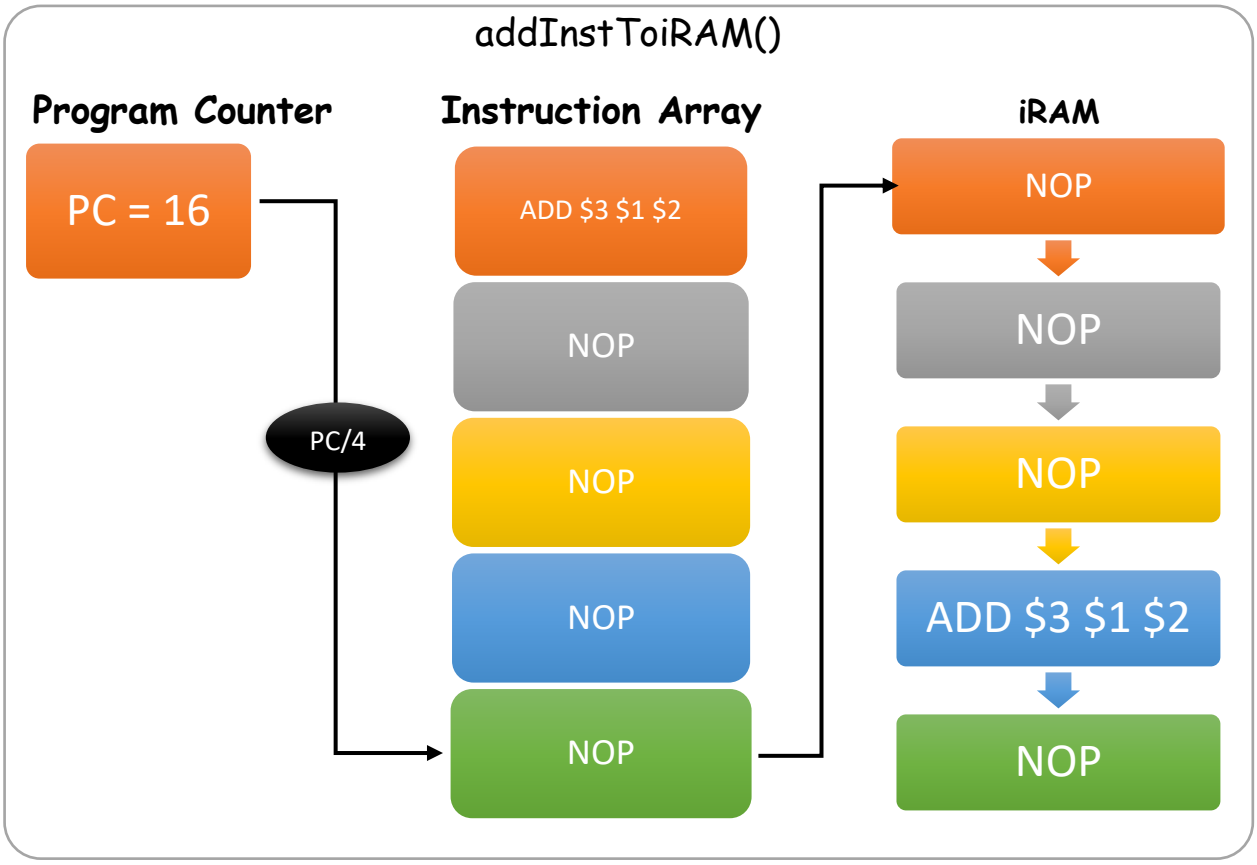

Execute
iRAM
NOP
NOP
ADD $3 $1 $2
NOP
NOP
Instruction Fetch
Instruction Decode
Execute
Memory
Write Back

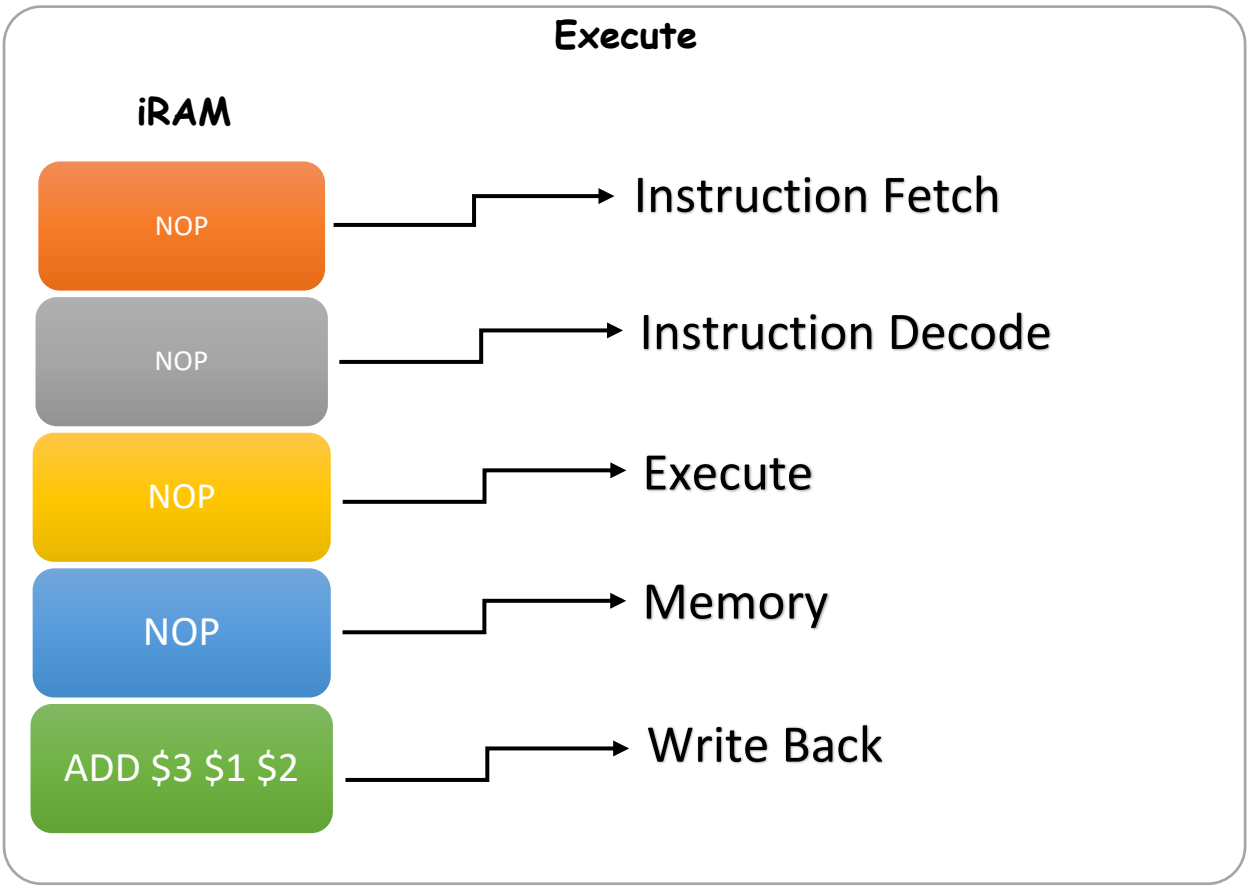
Clock 4


addInstToiRAM()
Program Counter
Instruction Array
iRAM
PC = 12
PC/4
ADD $3 $1 $2
NOP
NOP
NOP
NOP
NOP
NOP
ADD $3 $1 $2
NOP
NOP


Execute
iRAM
NOP
NOP
NOP
ADD $3 $1 $2
NOP
Instruction Fetch
Instruction Decode
Execute
Memory
Write Back

Clock 5


addInstToiRAM()
Program Counter
Instruction Array
iRAM
PC = 16
PC/4
ADD $3 $1 $2
NOP
NOP
NOP
NOP
NOP
NOP
NOP
ADD $3 $1 $2
NOP


Execute
iRAM
NOP
NOP
NOP
NOP
ADD $3 $1 $2
Instruction Fetch
Instruction Decode
Execute
Memory
Write Back

Code

```
#include <iostream>
#include <bitset>
#include <climits>
#include <map>
#include <string>

using namespace std;

#define ADD 1
#define ADDU 2
#define ADDIU 3
#define SUB 4
#define SUBU 5
#define XOR 6
#define MUL 7
#define MULU 8
#define DIV 9
#define DIVU 10
#define BEQ 11
#define BNE 12
#define BEQZ 13
#define BNEZ 14
#define LW 15
#define SW 16
#define MOVE 17
#define NOP 18
#define bin6 bitset<6>
#define bin5 bitset<5>
#define bin32 bitset<32>
#define bin16 bitset<16>
int R[32] = {0};
int PC = 0;
int M[100000] = {0};
/**
 * Each of the instruction is provided with specific number using enumeration
function
 */
enum StringValue { NotDefined,ADD_ , ADDU_, ADDIU_, SUB_,SUBU_,XOR_,MUL_,MULU_
,DIV_,DIVU_,BEQ_,BNE_,BEQZ_,BNEZ_,LW_,SW_,MOVE_,NOP_};
static std::map<std::string, StringValue> s_mapStringValues;
/**
 *  This function is used for Initialization of registers and Memory and Map.
 */
void flush();
void addInstToiRAM();
```

```
void Initialize()
{
  s_mapStringValues["ADD"] = ADD_;
  s_mapStringValues["ADDU"] = ADDU_;
  s_mapStringValues["ADDIU"] = ADDIU_;
  s_mapStringValues["SUB"] = SUB_;
  s_mapStringValues["SUBU"] = SUBU_;
  s_mapStringValues["XOR"] = XOR_;
  s_mapStringValues["MUL"] = MUL_;
  s_mapStringValues["MULU"] = MULU_;
  s_mapStringValues["DIV"] = DIV_;
  s_mapStringValues["DIVU"] = DIVU_;
  s_mapStringValues["BEQ"] = BEQ_;
  s_mapStringValues["BNE"] = BNE_;
  s_mapStringValues["BEQZ"] = BEQZ_;
  s_mapStringValues["BNEZ"] = BNEZ_;
  s_mapStringValues["LW"] = LW_;
  s_mapStringValues["SW"] = SW_;
  s_mapStringValues["MOVE"] = MOVE_;
  s_mapStringValues["NOP"] = NOP_;

  for (int i = 0; i < 100000; i++) {
    M[i] = i + 1;
  }
  for(int i=0;i<32;i++)
  {
      R[i]=0;
  }
}

class Instruction{
    public:
        int opcode = NOP;
        int rs=0;
        int rt=0;
        int rd=0;
        int IMMIDIATE = 0;

        int result=0;
        string inst = "NOP";
        bool isFlushed = false;

    public:
        void IF(){
            PC = PC + 4;
            if(opcode != 0){
                cout<<"Instruction     --> ";
```

```
            }
            switch (opcode)
            {
                case ADD:
                    cout<<"op:"<<bin6(0)<<" rs:"<<bin5(rs)<<" rt:"<<bin5(rt)<<
" rd:"<<bin5(rd)<<" shamt:"<<bin5(0)<<" funct:"<<bin6(32)<<endl;
                    break;
                case ADDU:
                    cout<<"op:"<<bin6(0)<<" rs:"<<bin5(rs)<<" rt:"<<bin5(rt)<<
" rd:"<<bin5(rd)<<" shamt:"<<bin5(0)<<" funct:"<<bin6(33)<<endl;
                    break;
                case SUB:
                    cout<<"op:"<<bin6(0)<<" rs:"<<bin5(rs)<<" rt:"<<bin5(rt)<<
" rd:"<<bin5(rd)<<" shamt:"<<bin5(0)<<" funct:"<<bin6(34)<<endl;
                    break;
                case SUBU:
                    cout<<"op:"<<bin6(0)<<" rs:"<<bin5(rs)<<" rt:"<<bin5(rt)<<
" rd:"<<bin5(rd)<<" shamt:"<<bin5(0)<<" funct:"<<bin6(35)<<endl;
                    break;
                case MUL:
                    cout<<"op:"<<bin6(0)<<" rs:"<<bin5(rs)<<" rt:"<<bin5(rt)<<
" rd:"<<bin5(rd)<<" shamt:"<<bin5(0)<<" funct:"<<bin6(36)<<endl;
                    break;
                case MULU:
                    cout<<"op:"<<bin6(0)<<" rs:"<<bin5(rs)<<" rt:"<<bin5(rt)<<
" rd:"<<bin5(rd)<<" shamt:"<<bin5(0)<<" funct:"<<bin6(37)<<endl;
                    break;
                case XOR:
                    cout <<"op:" << bin6(0)<<" rs:"<<bin5(rs)<<" rt:"<< bin5(r
t)<<" rd:"<<bin5(rd)<<" shamt:" << bin5(0) <<" funct:"<< bin6(38)<<endl;
                    break;
                case DIV:
                    cout<<"op:" << bin6(0)<<" rs:"<<bin5(rs)<<" rt:"<< bin5(rt
)<<" rd:"<<bin5(rd)<<" shamt:" << bin5(0) <<" funct:"<< bin6(39)<<endl;
                    break;
                case DIVU:
                    cout<<"op:" << bin6(0)<<" rs:"<<bin5(rs)<<" rt:"<< bin5(rt
)<<" rd:"<<bin5(rd)<<" shamt:" << bin5(0) <<" funct:"<< bin6(40)<<endl;
                    break;
                case MOVE://change
                    cout <<"op:" << bin6(0)<<" rs:"<<bin5(rs)<<" rt:"<< bin5(r
t)<<" rd:"<<bin5(rd)<<" shamt:" << bin5(0) <<" funct:"<< bin6(41)<<endl;
                    break;
                case ADDIU:
                    cout << "op:" << bin6(ADDIU) << " rs:" << bin5(rs) << " rt
:" << bin5(rt) << " IMMIDIATE/address: " << bin16(IMMIDIATE) << endl;
                    break;
                case LW:
```

```
                    cout << "op:" << bin6(LW) << " rs:" << bin5(rs) << " rt:"
<< bin5(rt) << " IMMIDIATE/address: " << bin16(IMMIDIATE) << endl;
                    break;
                case SW:
                    cout << "op:" << bin6(SW) << " rs:" << bin5(rs) << " rt:"
<< bin5(rt) << " IMMIDIATE/address: " << bin16(IMMIDIATE) << endl;
                    break;
                case NOP:
                    cout << "NOP" << endl;
                    break;
                case BEQ:
                    cout<<"op:"<<bin6(BEQ)<<" rs:"<<bin5(rs)<<" rt:"<<bin5(rt)
<<" immediate:"<<bin6(IMMIDIATE)<<endl;
                    break;
                case BNE:
                    cout<<"op:"<<bin6(BNE)<<" rs:"<<bin5(rs)<<" rt:"<<bin5(rt)
<<" immediate:"<<bin6(IMMIDIATE)<<endl;
                    break;
                case BEQZ:
                    cout<<"op:"<<bin6(BEQZ)<<" rs:"<<bin5(rs)<<" rt:"<<bin5(rt
)<<" immediate:"<<bin6(IMMIDIATE)<<endl;
                    break;
                case BNEZ:
                    cout<<"op:"<<bin6(BNEZ)<<" rs:"<<bin5(rs)<<" rt:"<<bin5(rt
)<<" immediate:"<<bin6(IMMIDIATE)<<endl;
                    break;
                default:
                    break;
            }
            cout<<"Program Counter --> "<<bin32(PC)<<" ["<<PC<<"]"<<endl;
        }

        void ID(){
            if(isFlushed){
                cout<<"Instruction Flushed"<<endl;
                return;
            }
            switch (opcode)
            {
            case ADD:
                cout<<"Register Data1: "<<bin32(R[rs])<<" ["<<R[rs]<<"]"<<"\nR
egister Data2: "<<bin32(R[rt])<<" ["<<R[rt]<<"]"<<"\nSign Extend:    "<<bin32(
(rd<<11)+32)<<" ["<<((rd<<11)+32)<<"]"<<endl;
                break;
            case ADDU:
                cout<<"Register Data1: "<<bin32(R[rs])<<" ["<<R[rs]<<"]"<<"\nR
egister Data2: "<<bin32(R[rt])<<" ["<<R[rt]<<"]"<<"\nSign Extend:    "<<bin32(
(rd<<11)+33)<<" ["<<((rd<<11)+33)<<"]"<<endl;
```

```
                break;
            case SUB:
                cout<<"Register Data1: "<<bin32(R[rs])<<" ["<<R[rs]<<"]"<<"\nR
egister Data2: "<<bin32(R[rt])<<" ["<<R[rt]<<"]"<<"\nSign Extend:    "<<bin32(
(rd<<11)+34)<<" ["<<((rd<<11)+34)<<"]"<<endl;
                break;
            case SUBU:
                cout<<"Register Data1: "<<bin32(R[rs])<<" ["<<R[rs]<<"]"<<"\nR
egister Data2: "<<bin32(R[rt])<<" ["<<R[rt]<<"]"<<"\nSign Extend:    "<<bin32(
(rd<<11)+35)<<" ["<<((rd<<11)+35)<<"]"<<endl;
                break;
            case MUL:
                cout<<"Register Data1: "<<bin32(R[rs])<<" ["<<R[rs]<<"]"<<"\nR
egister Data2: "<<bin32(R[rt])<<" ["<<R[rt]<<"]"<<"\nSign Extend:    "<<bin32(
(rd<<11)+36)<<" ["<<((rd<<11)+36)<<"]"<<endl;
                break;
            case MULU:
                cout<<"Register Data1: "<<bin32(R[rs])<<" ["<<R[rs]<<"]"<<"\nR
egister Data2: "<<bin32(R[rt])<<" ["<<R[rt]<<"]"<<"\nSign Extend:    "<<bin32(
(rd<<11)+37)<<" ["<<((rd<<11)+37)<<"]"<<endl;
                break;
            case XOR:
                cout<<"Register Data1: "<<bin32(R[rs])<<" ["<<R[rs]<<"]"<<"\nR
egister Data2: "<<bin32(R[rt])<<" ["<<R[rt]<<"]"<<"\nSign Extend:    "<<bin32(
(rd << 11)+38)<<" ["<<((rd<<11)+38)<<"]"<<endl;
                break;
            case DIV:
                cout<<"Register Data1: "<<bin32(R[rs])<<" ["<<R[rs]<<"]"<<"\nR
egister Data2: "<<bin32(R[rt])<<" ["<<R[rt]<<"]"<<"\nSign Extend:    "<<bin32(
(rd << 11)+39)<<" ["<<((rd<<11)+39)<<"]"<<endl;
                break;
            case DIVU:
                cout<<"Register Data1: "<<bin32(R[rs])<<" ["<<R[rs]<<"]"<<"\nR
egister Data2: "<<bin32(R[rt])<<" ["<<R[rt]<<"]"<<"\nSign Extend:    "<<bin32(
(rd << 11)+40)<<" ["<<((rd<<11)+40)<<"]"<<endl;
                break;
            case MOVE:
                cout<<"Register Data1: "<<bin32(R[rs])<<" ["<<R[rs]<<"]"<<"\nR
egister Data2: "<<bin32(R[rt])<<" ["<<R[rt]<<"]"<<"\nSign Extend:    "<<bin32(
(rd << 11)+41)<<" ["<<((rd<<11)+41)<<"]"<<endl;
                break;
            case ADDIU:
                cout << "Register Data1: " << bin32(R[rs])<<" ["<<R[rs]<<"]"<<
"\nRegister Data2: " << bin32(R[rt])<<" ["<<R[rt]<<"]"<<"\nSign Extend:    "<<
 bin32(IMMIDIATE)<<" ["<<(IMMIDIATE)<<"]"<<endl;
                break;
            case LW:
```

```
                cout << "Register Data1: " << bin32(R[rs])<<" ["<<R[rs]<<"]"<<
"\nRegister Data2: " << bin32(R[rt])<<" ["<<R[rt]<<"]"<<"\nSign Extend:    "<<
 bin32(IMMIDIATE) << " ["<<(IMMIDIATE)<<"]"<<endl;
                break;
            case SW:
                cout << "Register Data1: " << bin32(R[rs])<<" ["<<R[rs]<<"]"<<
"\nRegister Data2: " << bin32(R[rt])<<" ["<<R[rt]<<"]"<<"\nSign Extend:    "<<
 bin32(IMMIDIATE) << " ["<<(IMMIDIATE)<<"]"<<endl;
                break;
            case NOP:
                cout << "NOP" << endl;
                break;
            case BEQ:
                cout << "Register Data1: " << bin32(R[rs])<<" ["<<R[rs]<<"]"<<
"\nRegister Data2: " << bin32(R[rt])<<" ["<<R[rt]<<"]"<<"\nSign Extend:    "<<
 bin32(IMMIDIATE) << " ["<<(IMMIDIATE)<<"]"<<endl;
                break;
            case BNE:
                cout << "Register Data1: " << bin32(R[rs])<<" ["<<R[rs]<<"]"<<
"\nRegister Data2: " << bin32(R[rt])<<" ["<<R[rt]<<"]"<<"\nSign Extend:    "<<
 bin32(IMMIDIATE) << " ["<<(IMMIDIATE)<<"]"<<endl;
                break;
            case BNEZ:
                cout << "Register Data1: " << bin32(R[rs])<<" ["<<R[rs]<<"]"<<
"\nRegister Data2: " << bin32(R[rt])<<" ["<<R[rt]<<"]"<<"\nSign Extend:    "<<
 bin32(IMMIDIATE) << " ["<<(IMMIDIATE)<<"]"<<endl;
                break;
            case BEQZ:
                cout << "Register Data1: " << bin32(R[rs])<<" ["<<R[rs]<<"]"<<
"\nRegister Data2: " << bin32(R[rt])<<" ["<<R[rt]<<"]"<<"\nSign Extend:    "<<
 bin32(IMMIDIATE) << " ["<<(IMMIDIATE)<<"]"<<endl;
                break;
            default:
                break;
            }
        }

        void EXE(){
            if(isFlushed){
                cout<<"Instruction Flushed"<<endl;
                return;
            }
            switch (opcode)
            {
                case ADD:
                    result = R[rs] + R[rt];
                    if(R[rs] > (INT_MAX - R[rt])){
                        cout<<"Add Overflow detected"<<endl;
```

```
                throw "Overflow";
            }
            break;
        case ADDU:
            result = R[rs] + R[rt];
            break;
        case SUB:
            result = R[rs] - R[rt];
            if(((R[rs]>0) && (R[rt]<0) && (result<0)) || ((R[rs]<0) &&
 (R[rt]>0) && (result>0))){
                cout<<"Sub Overflow detected"<<endl;
                throw "Overflow";
            }
            break;
        case SUBU:
            result = R[rs] - R[rt];
            break;
        case MUL:
            result = R[rs] * R[rt];
            if(R[rs] > (INT_MAX / R[rt])){
                cout<<"Mul Overflow detected"<<endl;
                throw "Overflow";
            }
            break;
        case MULU:
            result = R[rs] * R[rt];
            break;
        case DIV:
            result=R[rt]/R[rs];
            break;
        case DIVU:
            if ( R[rt]== INT_MIN && R[rs] == -1 ){
                cout<<"possibly overflow prevented"<<endl;
            }
            else{
                result=R[rt]/R[rs];
            }
            break;
        case XOR:
            result = (R[rs]^R[rt]);
            break;
        case ADDIU:
            result = R[rs] + IMMIDIATE;
            break;
        case LW:
            result = R[rs] + IMMIDIATE;//EFFECTIVE ADDRESS IS CALCULAT
ED HERE
            break;
```

```
                case SW:
                    result = R[rs] + IMMIDIATE;//EFFECTIVE ADDRESS IS CALCULAT
ED HERE
                    break;
                case MOVE://change
                    result = rs + rt;               //EFFECTIVE ADDRESS IS CAL
CULATED HERE
                    break;
                case NOP:
                    cout << "NOP" << endl;
                    break;
                case BEQ:
                    if(R[rs] == R[rt]){
                        flush();
                        PC = IMMIDIATE*4;
                    }
                    if(R[rs]>R[rt]){
                      result = R[rs];
                    }else{
                      result = R[rt];
                    }
                    break;
                case BNE:
                    if(R[rs] != R[rt]){
                        flush();
                        PC = IMMIDIATE*4;
                    }
                    if(R[rs]>R[rt]){
                      result = R[rs];
                    }else{
                      result = R[rt];
                    }
                    break;
                case BEQZ:
                    if(R[rs] == 0){
                        flush();
                        PC = IMMIDIATE*4;
                    }
                    if(R[rs]>R[rt]){
                      result = R[rs];
                    }else{
                      result = R[rt];
                    }
                    break;
                case BNEZ:
                    if(R[rs] != 0){
                        flush();
                        PC = IMMIDIATE*4;
```

```
                    }
                    if(R[rs]>R[rt]){
                      result = R[rs];
                    }else{
                      result = R[rt];
                    }
                    break;
                default:
                    break;
            }
            cout<<"ALU Output: "<<bin32(result)<<" ["<<result<<"]"<<endl;
        }

        void MEM(){
            if(isFlushed){
                cout<<"Instruction Flushed"<<endl;
                return;
            }
            switch (opcode)
            {
                case ADD:
                case ADDU:
                case SUB:
                case SUBU:
                case MUL:
                case MULU:
                case XOR:
                case ADDIU:
                case BNE:
                case BEQ:
                case BNEZ:
                case BEQZ:
                case DIV:
                case DIVU:
                    cout<<"Data Bypassed to WB stage"<<endl;
                    break;
                case LW:
                    cout << "Accessing Effective Address: " << bin32(result) <
<" ["<<(result)<<"]"<< "\nIN THE MEMORY HAVING VALUE: " << bin32(M[result])<<"
 ["<<(M[result])<<"]"<<endl;
                    break;
                case SW://change
                    cout << "Accessing Effective Memory Address: " << bin32(re
sult) <<" ["<<(result)<<"]"<<endl;
                    break;
                case NOP:
                    cout << "NOP" << endl;
                    break;
```

```
                case MOVE://change
                    cout << "Data Bypassed to Write Back Stage "<<endl;
                    break;
                default:
                    break;
            }
        }

        void WB(){
            if(isFlushed){
                cout<<"Instruction Flushed"<<endl;
                return;
            }
            switch (opcode)
            {
                case ADD:
                case ADDU:
                case SUB:
                case SUBU:
                case MUL:
                case MULU:
                case XOR:
                case DIV:
                case DIVU:
                    R[rd] = result;
                    cout<<"Register "<<rd<<" value : "<<bin32(R[rd])<<" ["<<R[
rd]<<"]"<<endl;
                    break;
                case ADDIU:
                    R[rt]=result;
                    cout<<"Register "<<rt<<" value : "<<bin32(R[rt])<<" ["<<R[
rt]<<"]"<<endl;
                    break;
                case LW:
                    cout << "MEMORY ADDRESS " << bin32(result)<<" ["<<(result)
<<"]"<< "\nLOADED INTO REGISTER ADDRESS  " << bin32(rt) <<" ["<<(rt)<<"]"<< en
dl;
                    R[rt] = M[result];
                    break;
                case SW:
                    cout << "REGISTER ADDRESS " << bin32(rt)<<" ["<<(rt)<<"]"<
< "\nSTORED INTO MEMORY ADDRESS  " << bin32(result)<<" ["<<(result)<<"]"<<  en
dl;
                    M[result] = R[rt];
                    cout<<"THE VALUE STORED IN THE MEMORY IS "<<M[result]<<end
l;
                    break;
                case NOP:
```

```
                    cout << "NOP" << endl;
                    break;
                case MOVE://change
                    cout << "MOVING THE REGISTER FROM THE ADDRESS " << bin32(r
s)<<" ["<<(rs)<<"]"<< "\nTO THE ADDRESS "<< bin32(rt)<<" ["<<(rd)<<"]"<< endl;
                    R[rd] = R[result];
                    break;
                case BNE:
                case BEQ:
                case BNEZ:
                case BEQZ:
                    cout<<"Nothing Happen In Write Back Stage"<<endl;
                default:
                    break;
            }
        }

};

/**
 * insts is a pointer to array of instructions provided by User
 */
Instruction* insts;
/**
 * iRAM is a array of length 5 which is containing all five instructions
 * which are executing in the current clock_c
 * iRAM[0] will contain newest which should be in stage Instruction fetch
 * and
 * iRAM[4] will contain the oldest instruction which should be in Write Back s
tage
 */
Instruction iRAM[5];
/**
 * N is no of instruction provided by user
 * including nop instructions
 */
int N;
/**
 * clock_c is containg information about clock_c cycle no
 */
int clock_c = 1;
string mDivider = "=============================================================
==============================";
string lDivider = "-------------------------------------------------------------
------------------------------";

void executeInstructions(){
    string stages[5] = {"Instruction Fetch  --> ","Instruction Decode --> ",
```

```
                        "Execute            --> ","Memory             --> ",
                        "Write Back         --> "};
    while(PC < (4*N)){
        /**
         * First of all add instruction to iRAM using PC
         * so that we can execute it
         */

        addInstToiRAM();

        cout<<endl;
        cout<<mDivider<<endl;
        cout<<"Clock Cycle "<<clock_c<<endl;
        cout<<mDivider<<endl;

        for(int i=0;i<5;i++){
            cout<<stages[i]<< (iRAM[i].inst) <<endl;
            switch(i){
                case 0:
                    iRAM[i].IF();
                    break;
                case 1:
                    iRAM[i].ID();
                    break;
                case 2:
                    iRAM[i].EXE();
                    break;
                case 3:
                    iRAM[i].MEM();
                    break;
                case 4:
                    iRAM[i].WB();
                    break;
            }
            cout<<lDivider<<endl;
        }
        cout<<endl;

        clock_c++;
    }
}

void flush(){
    /**
     * instruction in iRAM[2] has called flush and we have to flush
     * iRAM[1] and iRAM[0]
     */
    iRAM[0].isFlushed = true;
```

```
    iRAM[1].isFlushed = true;
}

void addInstToiRAM(){
    /**
     * This method will insert new instruction using program counter
     * And shift all instruction backward
     */
    iRAM[4] = iRAM[3];
    iRAM[3] = iRAM[2];
    iRAM[2] = iRAM[1];
    iRAM[1] = iRAM[0];
    iRAM[0] = insts[PC/4];
}

int main(){

    Initialize();
    Instruction *input;
    cout<<"Enter the number of Register to change"<<endl;
    int rc;
    cin>>rc;
    int *rtc = new int[rc];
    cout<<"Enter the indexes of the Register to change"<<endl;
    //input format -->1 2 3 4 5
    for(int i=0;i<rc;i++)
    {
        cin>>rtc[i];
    }
    //input format -->2 4 6 8 9
    cout<<"Enter the values of the register"<<endl;
    for(int i=0;i<rc;i++)
    {
        cin>>R[rtc[i]];
    }
    cout << "Enter the Number of Instruction" << endl;
    int n;
    cin >> n;
    input = new Instruction[n + 4];
    string insist = "";
    char ch;
    int i1 =0;
    int i2 =0;
    int i3 = 0;
    cout << "Enter the Instructions" << endl;
    for (int i = 0; i < n; i++) {
        cin >> insist;
        switch (s_mapStringValues[insist]) {
```

```
            case NOP_:
                input[i].opcode = NOP;
                break;
            case ADD_:
            //input format ADD $1 $2 $3
                input[i].opcode = ADD;
                cin >> ch;
                cin >> i1;
                cin >> ch;
                cin >> i2;
                cin >> ch;
                cin >> i3;
                input[i].rs = i2;
                input[i].rt = i3;
                input[i].rd = i1;
                input[i].inst= insist +" "+"$"+to_string(input[i].rd)+" "+"$"+
to_string(input[i].rs)+" "+"$"+to_string(input[i].rt);
                // cout<<input[i].inst<<endl;
                break;
            case ADDU_:
            //input format ADDU $1 $2 $3
                input[i].opcode = ADDU;
                cin >> ch;
                cin >> i1;
                cin >> ch;
                cin >> i2;
                cin >> ch;
                cin >> i3;
                input[i].rs = i2;
                input[i].rt = i3;
                input[i].rd = i1;
                input[i].inst= insist +" "+"$"+to_string(input[i].rd)+" "+"$"+
to_string(input[i].rs)+" "+"$"+to_string(input[i].rt);
                // cout<<input[i].inst<<endl;
                break;
            case SUB_:
            //input format SUB $1 $2 $3
                input[i].opcode = SUB;
                cin >> ch;
                cin >> i1;
                cin >> ch;
                cin >> i2;
                cin >> ch;
                cin >> i3;
                input[i].rs = i2;
                input[i].rt = i3;
                input[i].rd = i1;
```

```
                input[i].inst= insist +" "+"$"+to_string(input[i].rd)+" "+"$"+
to_string(input[i].rs)+" "+"$"+to_string(input[i].rt);
                break;
            case SUBU_:
            //input format SUBU $1 $2 $3
                input[i].opcode = SUBU;
                cin >> ch;
                cin >> i1;
                cin >> ch;
                cin >> i2;
                cin >> ch;
                cin >> i3;
                input[i].rs = i2;
                input[i].rt = i3;
                input[i].rd = i1;
                input[i].inst= insist +" "+"$"+to_string(input[i].rd)+" "+"$"+
to_string(input[i].rs)+" "+"$"+to_string(input[i].rt);
                break;
            case MUL_:
            //input format MUL $1 $2 $3
                input[i].opcode = MUL;
                cin >> ch;
                cin >> i1;
                cin >> ch;
                cin >> i2;
                cin >> ch;
                cin >> i3;
                input[i].rs = i2;
                input[i].rt = i3;
                input[i].rd = i1;
                input[i].inst= insist +" "+"$"+to_string(input[i].rd)+" "+"$"+
to_string(input[i].rs)+" "+"$"+to_string(input[i].rt);
                break;
            case MULU_ :
            //input format MULU $1 $2 $3
                input[i].opcode = MULU;
                cin >> ch;
                cin >> i1;
                cin >> ch;
                cin >> i2;
                cin >> ch;
                cin >> i3;
                input[i].rs = i2;
                input[i].rt = i3;
                input[i].rd = i1;
                input[i].inst= insist +" "+"$"+to_string(input[i].rd)+" "+"$"+
to_string(input[i].rs)+" "+"$"+to_string(input[i].rt);
                break;
```

```
            case DIV_:
            //input format DIV $1 $2 $3
                input[i].opcode = DIV;
                cin >> ch;
                cin >> i1;
                cin >> ch;
                cin >> i2;
                cin >> ch;
                cin >> i3;
                input[i].rs = i2;
                input[i].rt = i3;
                input[i].rd = i1;
                input[i].inst= insist +" "+"$"+to_string(input[i].rd)+" "+"$"+
to_string(input[i].rs)+" "+"$"+to_string(input[i].rt);
                break;
            case DIVU_:
            //input format DIVU $1 $2 $3
                input[i].opcode = DIVU;
                cin >> ch;
                cin >> i1;
                cin >> ch;
                cin >> i2;
                cin >> ch;
                cin >> i3;
                input[i].rs = i2;
                input[i].rt = i3;
                input[i].rd = i1;
                input[i].inst= insist +" "+"$"+to_string(input[i].rd)+" "+"$"+
to_string(input[i].rs)+" "+"$"+to_string(input[i].rt);
                break;
            case XOR_:
            //input format XOR $1 $2 $3
                input[i].opcode = XOR;
                cin >> ch;
                cin >> i1;
                cin >> ch;
                cin >> i2;
                cin >> ch;
                cin >> i3;
                input[i].rs = i2;
                input[i].rt = i3;
                input[i].rd = i1;
                input[i].inst= insist +" "+"$"+to_string(input[i].rd)+" "+"$"+
to_string(input[i].rs)+" "+"$"+to_string(input[i].rt);
                break;
            case ADDIU_:
            //input format ADDIU $1 $2 IMMIDIATE
                input[i].opcode = ADDIU;
```

```
                cin >> ch;
                cin >> i1;
                cin >> ch;
                cin >> i2;
                cin >> i3;
                input[i].rs = i2;
                input[i].rt = i1;
                input[i].IMMIDIATE = i3;
                input[i].inst= insist+" "+"$"+to_string(input[i].rt)+" "+"$"+t
o_string(input[i].rs)+" "+to_string(input[i].IMMIDIATE);
                break;
            case LW_:
            //input format LW $1 IMMIDIATE ($2)
                input[i].opcode = LW;
                cin >> ch;
                cin >> i1;
                cin >> i2;
                cin >> ch;
                cin >> ch;
                cin >> i3;
                cin >> ch;
                input[i].rt = i1;
                input[i].rs = i3;
                input[i].IMMIDIATE = i2;
                input[i].inst= insist+" "+"$"+to_string(input[i].rt)+" "+to_st
ring(input[i].IMMIDIATE)+" ($"+to_string(input[i].rs)+") ";
                break;
            case SW_:
            //innput format SW $1 IMMIDIATE ($2)
                input[i].opcode = SW;
                cin >> ch;
                cin >> i1;
                cin >> i2;
                cin >> ch;
                cin >> ch;
                cin >> i3;
                cin >> ch;
                input[i].rt = i1;
                input[i].rs = i3;
                input[i].IMMIDIATE = i2;
                input[i].inst= insist+" "+"$"+to_string(input[i].rt)+" "+to_st
ring(input[i].IMMIDIATE)+" "+"($"+to_string(input[i].rs)+")";
                break;
            case MOVE_://change 1
            //input MOVE $1 $2
                input[i].opcode = MOVE;
                cin >> ch;
                cin >> i1;
```

```
                cin >> ch;
                cin >> i2;
                input[i].rd = i1;
                input[i].rs = i2;
                input[i].inst= insist+" "+"$"+to_string(input[i].rd)+" "+"$"+t
o_string(input[i].rs);
                break;
            case BEQ_:
            //input format BEQ $1 $2 IMMIDIATE
                input[i].opcode = BEQ;
                cin >> ch;
                cin >> i1;
                cin >> ch;
                cin >> i2;
                cin >> i3;
                input[i].rt = i1;
                input[i].rs = i2;
                input[i].IMMIDIATE = i3;
                input[i].inst= insist+" "+"$"+to_string(input[i].rt)+" "+"$"+t
o_string(input[i].rs)+" "+to_string(input[i].IMMIDIATE);
                break;
            case BEQZ_:
            //input format BEQZ $2 IMMIDIATE
                input[i].opcode = BEQZ;
                cin >> ch;
                cin >> i1;
                cin >> i2;
                input[i].rs = i1;
                input[i].IMMIDIATE = i2;
                input[i].inst= insist+" "+"$"+to_string(input[i].rs)+" "+to_st
ring(input[i].IMMIDIATE);
                break;
            case BNE_:
            //input format BNE $1 $2 IMMIDIATE
                input[i].opcode = BNE;
                cin >> ch;
                cin >> i1;
                cin >> ch;
                cin >> i2;
                cin >> i3;
                input[i].rt = i1;
                input[i].rs = i2;
                input[i].IMMIDIATE = i3;
                input[i].inst= insist+" "+"$"+to_string(input[i].rt)+" "+"$"+t
o_string(input[i].rs)+" "+to_string(input[i].IMMIDIATE);
                break;
            case BNEZ_:
            //input format BNEZ $2 IMMIDIATE
```

```
                input[i].opcode = BNEZ;
                cin >> ch;
                cin >> i2;
                cin >> i3;
                input[i].rs = i2;
                input[i].IMMIDIATE = i3;
                input[i].inst= insist+" "+"$"+to_string(input[i].rs)+" "+to_st
ring(input[i].IMMIDIATE);
                cout<<input[i].inst<<endl;
                break;
            default:
                cout << "Entered Wrong Instruction" << endl;
                break;
        }
    }
    insts = input;
    N = n+4;

    cin;
    executeInstructions();

    for(int i=0;i<32;i++){
        if(i%8 == 0){
            cout<<endl;
        }
        cout<<"R["<<i<<"] = "<<R[i]<<"\t";
    }
    return 1;

}
```

Output

```
PS C:\Users\Education\Documents\c cpp> cd "c:\Users\Education\Documents\c cpp\" ; if ($?) { g++ test.cpp -o test } ; if ($?) { .\test }
Enter the number of Register to change
2
Enter the indexes of the Register to change
1 2
Enter the values of the register
5 5
Enter the Number of Instruction
3
Enter the Instructions
BEQ $1 $2 2
ADD $5 $1 $2
MUL $5 $1 $2
```

```
================================================================================================
Clock Cycle 1
================================================================================================
Instruction Fetch  --> BEQ $1 $2 2
Instruction     --> op:001011 rs:00010 rt:00001 immediate:000010
Program Counter --> 00000000000000000000000000000100 [4]
------------------------------------------------------------------------------------------------
Instruction Decode --> NOP
NOP
------------------------------------------------------------------------------------------------
Execute            --> NOP
NOP
ALU Output: 00000000000000000000000000000000 [0]
------------------------------------------------------------------------------------------------
Memory             --> NOP
NOP
------------------------------------------------------------------------------------------------
Write Back         --> NOP
NOP
------------------------------------------------------------------------------------------------


================================================================================================
Clock Cycle 2
================================================================================================
Instruction Fetch  --> ADD $5 $1 $2
Instruction     --> op:000000 rs:00001 rt:00010 rd:00101 shamt:00000 funct:100000
Program Counter --> 00000000000000000000000000001000 [8]
------------------------------------------------------------------------------------------------
Instruction Decode --> BEQ $1 $2 2
Register Data1: 00000000000000000000000000000101 [5]
Register Data2: 00000000000000000000000000000101 [5]
Sign Extend:    00000000000000000000000000000010 [2]
------------------------------------------------------------------------------------------------
Execute            --> NOP
NOP
ALU Output: 00000000000000000000000000000000 [0]
------------------------------------------------------------------------------------------------
Memory             --> NOP
NOP
------------------------------------------------------------------------------------------------
Write Back         --> NOP
NOP
------------------------------------------------------------------------------------------------
```

```
================================================================================
Clock Cycle 3
================================================================================
Instruction Fetch  --> MUL $5 $1 $2
Instruction     --> op:000000 rs:00001 rt:00010 rd:00101 shamt:00000 funct:100100
Program Counter --> 00000000000000000000000000001100 [12]
--------------------------------------------------------------------------------
Instruction Decode --> ADD $5 $1 $2
Register Data1: 00000000000000000000000000000101 [5]
Register Data2: 00000000000000000000000000000101 [5]
Sign Extend:    00000000000000000010100000100000 [10272]
--------------------------------------------------------------------------------
Execute            --> BEQ $1 $2 2
ALU Output: 00000000000000000000000000000101 [5]
--------------------------------------------------------------------------------
Memory             --> NOP
NOP
--------------------------------------------------------------------------------
Write Back         --> NOP
NOP
--------------------------------------------------------------------------------


================================================================================
Clock Cycle 4
================================================================================
Instruction Fetch  --> MUL $5 $1 $2
Instruction     --> op:000000 rs:00001 rt:00010 rd:00101 shamt:00000 funct:100100
Program Counter --> 00000000000000000000000000001100 [12]
--------------------------------------------------------------------------------
Instruction Decode --> MUL $5 $1 $2
Instruction Flushed
--------------------------------------------------------------------------------
Execute            --> ADD $5 $1 $2
Instruction Flushed
--------------------------------------------------------------------------------
Memory             --> BEQ $1 $2 2
Data Bypassed to WB stage
--------------------------------------------------------------------------------
Write Back         --> NOP
NOP
--------------------------------------------------------------------------------
```

```
================================================================================================
Clock Cycle 5
================================================================================================
Instruction Fetch  --> NOP
Instruction     --> NOP
Program Counter --> 00000000000000000000000000010000 [16]
------------------------------------------------------------------------------------------------
Instruction Decode --> MUL $5 $1 $2
Register Data1: 00000000000000000000000000000101 [5]
Register Data2: 00000000000000000000000000000101 [5]
Sign Extend:    00000000000000000010100000100100 [10276]
------------------------------------------------------------------------------------------------
Execute            --> MUL $5 $1 $2
Instruction Flushed
------------------------------------------------------------------------------------------------
Memory             --> ADD $5 $1 $2
Instruction Flushed
------------------------------------------------------------------------------------------------
Write Back         --> BEQ $1 $2 2
Nothing Happen In Write Back Stage
------------------------------------------------------------------------------------------------


================================================================================================
Clock Cycle 6
================================================================================================
Instruction Fetch  --> NOP
Instruction     --> NOP
Program Counter --> 00000000000000000000000000010100 [20]
------------------------------------------------------------------------------------------------
Instruction Decode --> NOP
NOP
------------------------------------------------------------------------------------------------
Execute            --> MUL $5 $1 $2
ALU Output: 00000000000000000000000000011001 [25]
------------------------------------------------------------------------------------------------
Memory             --> MUL $5 $1 $2
Instruction Flushed
------------------------------------------------------------------------------------------------
Write Back         --> ADD $5 $1 $2
Instruction Flushed
------------------------------------------------------------------------------------------------
```

```
================================================================================
Clock Cycle 7
================================================================================
Instruction Fetch  --> NOP
Instruction     --> NOP
Program Counter --> 00000000000000000000000000011000 [24]
--------------------------------------------------------------------------------
Instruction Decode --> NOP
NOP
--------------------------------------------------------------------------------
Execute            --> NOP
NOP
ALU Output: 00000000000000000000000000000000 [0]
--------------------------------------------------------------------------------
Memory             --> MUL $5 $1 $2
Data Bypassed to WB stage
--------------------------------------------------------------------------------
Write Back         --> MUL $5 $1 $2
Instruction Flushed
--------------------------------------------------------------------------------


================================================================================
Clock Cycle 8
================================================================================
Instruction Fetch  --> NOP
Instruction     --> NOP
Program Counter --> 00000000000000000000000000011100 [28]
--------------------------------------------------------------------------------
Instruction Decode --> NOP
NOP
--------------------------------------------------------------------------------
Execute            --> NOP
NOP
ALU Output: 00000000000000000000000000000000 [0]
--------------------------------------------------------------------------------
Memory             --> NOP
NOP
--------------------------------------------------------------------------------
Write Back         --> MUL $5 $1 $2
Register 5 value : 00000000000000000000000000011001 [25]
--------------------------------------------------------------------------------


R[0] = 0        R[1] = 5        R[2] = 5        R[3] = 0        R[4] = 0        R[5] = 25       R[6] = 0        R[7] = 0
R[8] = 0        R[9] = 0        R[10] = 0       R[11] = 0       R[12] = 0       R[13] = 0       R[14] = 0       R[15] = 0
R[16] = 0       R[17] = 0       R[18] = 0       R[19] = 0       R[20] = 0       R[21] = 0       R[22] = 0       R[23] = 0
R[24] = 0       R[25] = 0       R[26] = 0       R[27] = 0       R[28] = 0       R[29] = 0       R[30] = 0       R[31] = 0
PS C:\Users\Education\Documents\c cpp>
```

END