**Name- Rabindra Kharga**

**Roll: 21204407**

**Electrical Engineering (MVLSI)**

**IIT KANPUR**

**A logo with text and symbols

Description automatically generated**

**Project Title:** Pipeline implementation of MIPS32 RISC processor and performing operations like addition and factorial.

**INTRODUCTION:**

* We are implementing MIPS32 based on which is a popular 32-bit processor.
* Look at instruction types and how instructions are encoded.
* Finally applying operations like addition and factorial using Verilog code

**BASICS:**

* MIPS32 registers has 32-bit general-purpose registers R0 to R 31, wherein R0 has value 0 and cannot be written.
* A special purpose 32 bit program counter (PC) which points next instructions to fetch from memory and execute it as per the instructions in memory.
* No flag registers (zero, carry, sign).
* Memory size is 32 bits.

**ARCHITECTURE:**

**A diagram of a block diagram

Description automatically generated**

**Behavioral Verilog Code for Pipeline MIPS32 RISC processor:**

**//File Name: MIPS32 RISC Processor**

**// Type: Module**

**// Department : Electrical engineering**

**// Author: Rabindra**

**//Author's Email ID: khargarabindra@gmail.com**

**// Purpose: ALU operation**

**// Operation : Addition & Factorial**

**// Date : 1 July 2023**

module pipe\_MIPS32 (clk1, clk2); // Define two clock signals clk1 and clk2 are non-overlapping

input clk1, clk2; // Two-phase clock

reg [31:0] PC, IF\_ID\_IR, IF\_ID\_NPC;

reg [31:0] ID\_EX\_IR, ID\_EX\_NPC, ID\_EX\_A, ID\_EX\_B, ID\_EX\_Imm;

reg [2:0] ID\_EX\_type, EX\_MEM\_type, MEM\_WB\_type;

reg [31:0] EX\_MEM\_IR, EX\_MEM\_ALUOut, EX\_MEM\_B;

reg EX\_MEM\_cond;

reg [31:0] MEM\_WB\_IR, MEM\_WB\_ALUOut, MEM\_WB\_LMD;

reg [31:0] Reg [0:31]; // Register bank (32 x 32)

reg [31:0] Mem [0:1023]; // 1024 x 32 memory

parameter ADD\_op=6'b000000; // Addition opcode in machine language

parameter SUB\_op=6'b000001; // Subtraction opcode in machine language

parameter AND\_op=6'b000010; // And opcode in machine language

parameter OR\_op=6'b000011; // OR opcode in machine language

parameter SLT\_op=6'b000100; // Set less than opcode in machine language

parameter MUL\_op=6'b000101; // multiplication opcode in machine language

parameter HLT\_op=6'b111111; // Halt opcode in machine language

parameter LW\_op=6'b001000;

parameter SW\_op=6'b001001;

parameter ADDI\_op=6'b001010; // Addition instruction opcode in machine language

parameter SUBI\_op=6'b001011; // Substraction instruction opcode in machine language

parameter SLTI\_op=6'b001100; // Set less than instruction opcode in machine language

parameter BNEQZ\_op=6'b001101; // Branch not equal to zero opcode in machine language

parameter BEQZ\_op=6'b001110; // Branch equal to zero opcode in machine language

parameter RR\_ALU\_op=3'b000;

parameter RM\_ALU\_op=3'b001;

parameter LOAD\_op=3'b010; // load opcode in machine language

parameter STORE\_op=3'b011; // Store opcode in machine language

parameter BRANCH\_op=3'b100;

parameter HALT\_op=3'b101; // Halt opcode in machine language

reg HALTED;

// Set after HLT instruction is completed (in WB stage)

reg TAKEN\_BRANCH;

// Required to disable instructions after branch

always @(posedge clk1) // Instruction fetch Stage

if (HALTED == 0)

begin

if (((EX\_MEM\_IR[31:26] == BEQZ\_op) && (EX\_MEM\_cond == 1)) ||

((EX\_MEM\_IR[31:26] == BNEQZ\_op) && (EX\_MEM\_cond == 0)))

begin

IF\_ID\_IR <= #2 Mem[EX\_MEM\_ALUOut];

TAKEN\_BRANCH <= #2 1'b1;

IF\_ID\_NPC <= #2 EX\_MEM\_ALUOut + 1;

PC <= #2 EX\_MEM\_ALUOut + 1;

end

else

begin

IF\_ID\_IR <= #2 Mem[PC];

IF\_ID\_NPC <= #2 PC + 1;

PC <= #2 PC + 1;

end

end

always @(posedge clk2) // Instrcution Decode Stage

if (HALTED == 0)

begin

if (IF\_ID\_IR[25:21] == 5'b00000) ID\_EX\_A <= 0;

else ID\_EX\_A <= #2 Reg[IF\_ID\_IR[25:21]]; // "rs"

if (IF\_ID\_IR[20:16] == 5'b00000) ID\_EX\_B <= 0;

else ID\_EX\_B <= #2 Reg[IF\_ID\_IR[20:16]]; // "rt"

ID\_EX\_NPC <= #2 IF\_ID\_NPC;

ID\_EX\_IR <= #2 IF\_ID\_IR;

ID\_EX\_Imm <= #2 {{16{IF\_ID\_IR[15]}}, {IF\_ID\_IR[15:0]}};

case (IF\_ID\_IR[31:26])

ADD\_op, SUB\_op, AND\_op, OR\_op, SLT\_op, MUL\_op: ID\_EX\_type <= #2 RR\_ALU\_op;

ADDI\_op, SUBI\_op, SLTI\_op: ID\_EX\_type <= #2 RM\_ALU\_op;

LW\_op: ID\_EX\_type <= #2 LOAD\_op;

SW\_op: ID\_EX\_type <= #2 STORE\_op;

BNEQZ\_op, BEQZ\_op: ID\_EX\_type <= #2 BRANCH\_op;

HLT\_op: ID\_EX\_type <= #2 HALT\_op;

default: ID\_EX\_type <= #2 HALT\_op;

// Invalid opcode

endcase

end

always @(posedge clk1) // Execution Stage

if (HALTED == 0)

begin

EX\_MEM\_type <= #2 ID\_EX\_type;

EX\_MEM\_IR <= #2 ID\_EX\_IR;

TAKEN\_BRANCH <= #2 0;

case (ID\_EX\_type)

RR\_ALU\_op: begin

case (ID\_EX\_IR[31:26]) // "opcode"

ADD\_op: EX\_MEM\_ALUOut <= #2 ID\_EX\_A + ID\_EX\_B;

SUB\_op: EX\_MEM\_ALUOut <= #2 ID\_EX\_A - ID\_EX\_B;

AND\_op: EX\_MEM\_ALUOut <= #2 ID\_EX\_A & ID\_EX\_B;

OR\_op: EX\_MEM\_ALUOut <= #2 ID\_EX\_A | ID\_EX\_B;

SLT\_op: EX\_MEM\_ALUOut <= #2 ID\_EX\_A < ID\_EX\_B;

MUL\_op: EX\_MEM\_ALUOut <= #2 ID\_EX\_A \* ID\_EX\_B;

default: EX\_MEM\_ALUOut <= #2 32'hxxxxxxxx;

endcase

end

RM\_ALU\_op: begin

case (ID\_EX\_IR[31:26]) // "opcode"

ADDI\_op: EX\_MEM\_ALUOut <= #2 ID\_EX\_A + ID\_EX\_Imm;

SUBI\_op: EX\_MEM\_ALUOut <= #2 ID\_EX\_A - ID\_EX\_Imm;

SLTI\_op: EX\_MEM\_ALUOut <= #2 ID\_EX\_A < ID\_EX\_Imm;

default: EX\_MEM\_ALUOut <= #2 32'hxxxxxxxx;

endcase

end

LOAD\_op, STORE\_op:

begin

EX\_MEM\_ALUOut <= #2 ID\_EX\_A + ID\_EX\_Imm;

EX\_MEM\_B <= #2 ID\_EX\_B;

end

BRANCH\_op: begin

EX\_MEM\_ALUOut <= #2 ID\_EX\_NPC + ID\_EX\_Imm;

EX\_MEM\_cond <= #2 (ID\_EX\_A == 0);

end

endcase

end

always @(posedge clk2) // Memory Stage

if (HALTED == 0)

begin

MEM\_WB\_type <= EX\_MEM\_type;

MEM\_WB\_IR <= #2 EX\_MEM\_IR;

case (EX\_MEM\_type)

RR\_ALU\_op, RM\_ALU\_op:

MEM\_WB\_ALUOut <= #2 EX\_MEM\_ALUOut;

LOAD\_op: MEM\_WB\_LMD <= #2 Mem[EX\_MEM\_ALUOut];

STORE\_op: if (TAKEN\_BRANCH == 0) // Disable write

Mem[EX\_MEM\_ALUOut] <= #2 EX\_MEM\_B;

endcase

end

always @(posedge clk1) // Write back Stage

begin

if (TAKEN\_BRANCH == 0) // Disable write if branch taken

case (MEM\_WB\_type)

RR\_ALU\_op: Reg[MEM\_WB\_IR[15:11]] <= #2 MEM\_WB\_ALUOut; // "rd"

RM\_ALU\_op: Reg[MEM\_WB\_IR[20:16]] <= #2 MEM\_WB\_ALUOut; // "rt"

LOAD\_op: Reg[MEM\_WB\_IR[20:16]] <= #2 MEM\_WB\_LMD; // "rt"

HALT\_op: HALTED <= #2 1'b1;

endcase

end

endmodule

**TESTBENCH for Addition operation:**

**//File Name: MIPS32 RISC Processor\_ test bench for addition operation**

**// Type: test\_bench\_example\_1**

**// Department : Electrical engineering**

**// Author: Rabindra**

**//Author's Email ID: khargarabindra@gmail.com**

**// Purpose: ALU addition opeartion with dummy instructions for avoiding hazards**

**// Operation : Addition**

**// Date : 1 July 2023**

`include "MIPS32.v"

module test\_MIPS32;

reg clk1, clk2;

integer k;

pipe\_MIPS32 mips (clk1, clk2);

initial

begin

clk1 = 0; clk2 = 0;

repeat (20) // Generating two-phase clock

begin

#5 clk1 = 1; #5 clk1 = 0;

#5 clk2 = 1; #5 clk2 = 0;

end

end

initial

begin

for (k=0; k<31; k++)

mips.Reg[k] = k;

mips.Mem[0] = 32'h2801000a; // ADDI R1,R0,10

mips.Mem[1] = 32'h28020014; // ADDI R2,R0,20

mips.Mem[2] = 32'h28030019; // ADDI R3,R0,25

mips.Mem[3] = 32'h0ce77800; // OR R7,R7,R7 -- dummy instr.

mips.Mem[4] = 32'h0ce77800; // OR R7,R7,R7 -- dummy instr.

mips.Mem[5] = 32'h00222000; // ADD R4,R1,R2

mips.Mem[6] = 32'h0ce77800; // OR R7,R7,R7 -- dummy instr.

mips.Mem[7] = 32'h00832800; // ADD R5,R4,R3

mips.Mem[8] = 32'hfc000000; // HLT

mips.HALTED = 0;

mips.PC = 0;

mips.TAKEN\_BRANCH = 0;

#280

for (k=0; k<6; k++)

$display ("R%1d - %2d", k, mips.Reg[k]);

end

initial

begin

$dumpfile ("mips\_add.vcd");

$dumpvars (0, test\_MIPS32);

#300 $finish;

end

endmodule

**SIMULATION OUTPUT:**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**TESTBENCH for factorial program:**

**//File Name: MIPS32 RISC Processor\_ test bench for Factorial operation**

**// Type: test\_bench\_example\_2**

**// Department: Electrical engineering**

**// Author: Rabindra**

**//Author's Email ID: khargarabindra@gmail.com**

**// Purpose: ALU factorial operation with dummy instructions for avoiding hazards**

**// Operation: factorial**

**// Date : 1 July 2023**

`include "MIPS32.v"

module test1\_mips32;

reg clk1, clk2;

integer k;

pipe\_MIPS32 mips (clk1, clk2);

initial

begin

clk1 = 0; clk2 = 0;

repeat (50) // Generating two-phase clock

begin

#5 clk1 = 1; #5 clk1 = 0;

#5 clk2 = 1; #5 clk2 = 0;

end

end

initial

begin

for (k=0; k<31; k++)

mips.Reg[k] = k;

mips.Mem[0] = 32'h280a00c8; // ADDI R10,R0,200

mips.Mem[1] = 32'h28020001; // ADDI R2,R0,1

mips.Mem[2] = 32'h0e94a000; // OR R20,R20,R20 -- dummy instr.

mips.Mem[3] = 32'h21430000; // LW R3,0(R10)

mips.Mem[4] = 32'h0e94a000; // OR R20,R20,R20 -- dummy instr.

mips.Mem[5] = 32'h14431000; // Loop: MUL R2,R2,R3

mips.Mem[6] = 32'h2c630001; // SUBI R3,R3,1

mips.Mem[7] = 32'h0e94a000; // OR R20,R20,R20 -- dummy instr.

mips.Mem[8] = 32'h3460fffc; // BNEQZ R3,Loop (i.e. -4 offset)

mips.Mem[9] = 32'h2542fffe; // SW R2,-2(R10)

mips.Mem[10] = 32'hfc000000; // HLT

mips.Mem[200] = 7; // Find factorial of 7

mips.PC = 0;

mips.HALTED = 0;

mips.TAKEN\_BRANCH = 0;

#2000 $display ("Mem[200] = %2d, Mem[198] = %6d",

mips.Mem[200], mips.Mem[198]);

end

initial

begin

$dumpfile ("mips\_factorial.vcd");

$dumpvars (0, test1\_mips32);

$monitor ("R2: %4d", mips.Reg[2]);

#3000 $finish;

end

endmodule

**A screenshot of a computer

Description automatically generated**

A screen shot of a computer

Description automatically generated

**CONCLUSION:**

We have successfully Implemented a pipelined RISC based MIPS32 processor which includes component like instruction memory (I.M), register bank, Multiplexers, Arithmetic logic unit (ALU), latches and computed operations like addition and factorial written in machine language.