***PROJECT ON VERILOG***

***NAME:ABHAY TYAGI***

***COURSE:BTECH***

***BRANCH:ELECTRONICS AND COMMUNICATION***

***UNIVERSITY:GRAPHIC ERA DEEMED TO BE UNIVERSITY***

***PROJECT TITLE: DESIGN & SIMULATION OF RISC-BASED MIPS-32 PROCESSOR***

***ACKNOWLEDGEMENT***

***I would like to convey my heartfelt gratitude to Mr Abhay Sharma for her tremendous support and assistance in the completion of my project. I would also like to thank my senior Mr Pranav aggarwal for for providing me with this wonderful opportunity to work on a project with the topic DESIGN & SIMULATION OF RISC-BASED MIPS-32 PROCESSOR. The completion of the project would not have been possible without their help and insights.***

***Abhay Tyagi***

***CONTENTS***

***1)INTRODUCTION***

***2)STAGES OF PROCESSOR***

***3)TYPES OF INSTRUCTION***

***4)PIPELINING***

***5)HAZARDS OF PIPELINING***

***6)CODE FOR DESIGNING PROCESSOR***

***7)ELABORATED SCHEMATIC OF PROCESSOR***

***8)SYNTHESIS DESIGN OF PROCESSOR***

***9)TEST BENCH FOR CODE***

***DESIGN & SIMULATION OF RISC-BASED MIPS-32 PROCESSOR***

**ABSTRACT:**

***This project presents the design & simulation of a high-performance five-stage pipelined 32-bit Microprocessor Interlocked Pipeline Stages MIPS, which is a Reduced Instruction Set Computing (RISC) architecture-based processor. The purpose of the RISC microprocessor is to execute a diminutive batch of instructions, with the aim of procreation the celerity of the processor. This processor was designed with 5 phases of pipeline in particular Instruction Fetch (IF), Instruction Decode& Register Fetch (ID), Execution & Address Calculation (EX), Memory Access (MEM), and Write Back (WB) modules. The designing process was done using a module mips32 along with different register variables like PC, NPC, and many more, register bank, memory bank, etc. The Proposed design is developed by Verilog HDL using XILINX software 2017.4(VIVADO) and also simulated on VIVADO using different test benches.***

**OBJECTIVE:**

* ***The main objective of this project is to create the design of a RISC(REDUCED INSTRUCTION SET COMPUTER) CPU architecture based on MIPS(Microprocessor Interlock Pipeline Stages) using Verilog HDL.***
* ***It also describes the instruction set and the architecture of the processor.***
* ***To run the different test benches and verify the output.***

**MOTIVE:**

* ***RISC systems use hard-wired code with a simple instruction set that needs a less costly CPU than a CISC device.***
* ***RISC processors are used in smartphones, printers, tablets, and devices that do a specific set of repeatable activities.***
* ***It also helps in exploring the depth of knowledge of microprocessor working.***

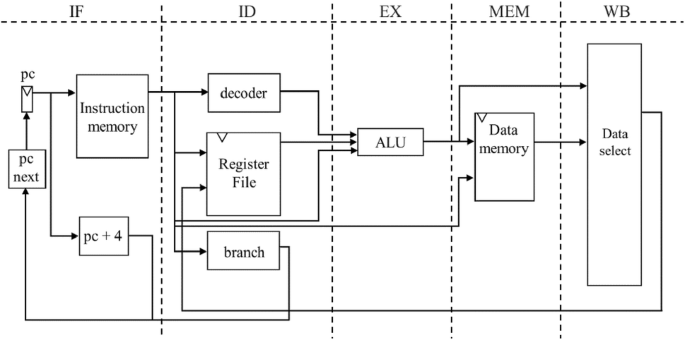
**INTRODUCTION:**

***A microprocessor is a computer processor where the data processing logic and control are included on a single integrated circuit or a small number of ICs. The microprocessor contains the arithmetic, logic, and control circuitry required to perform the functions of a computer's central processing unit.***

***RISC BASED MIPS-32 PROCESSOR::***

* ***It has a 32,32-bit general purpose register(R0 to R31).***
* ***Register R0 contains a constant value of 0.***
* ***There is no flag register present in the processor.***
* ***It has fewer addressing modes.***
* ***Only load and store instruction can access memory.***
* ***Memory word size if of 32 bit***
* ***It has a program counter to point instruction in memory.***

***BLOCK DIAGRAM:***



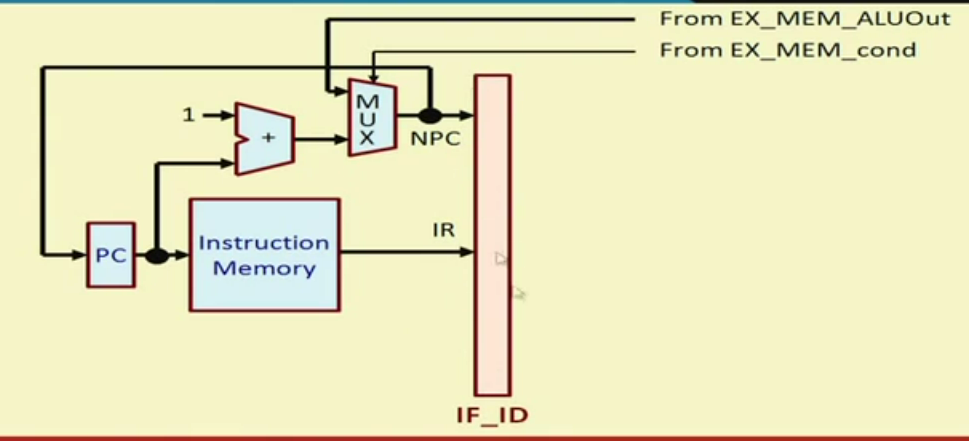
***STAGES OF PROCESSOR:***

***1)INSTRUCTION FETCH(IF)***

* ***The instruction pointed by the PC (program counter) is fetched from memory and also the next value is evaluated.***
* ***Every instruction of mips32 is 32-bit.***
* ***Every memory word is 32 bits and has a unique address.***
* ***For branch instruction, the new value of PC may be the target address so the PC is not updated in this stage it is stored in NPC.***
* ***IF: IR🡨memory[PC]***

***NPC🡨 PC+1***

* ***If byte addressable then PC is increased by 4.***

******

*HOW INSTRUCTION IS FETCHED:*

*1)REGISTER TYPE:*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *OPCODE* | *RS* | *RT* | *RD* | *SHAMT* | *FUNC* |

***RS-Source register***

***RD-Destination register***

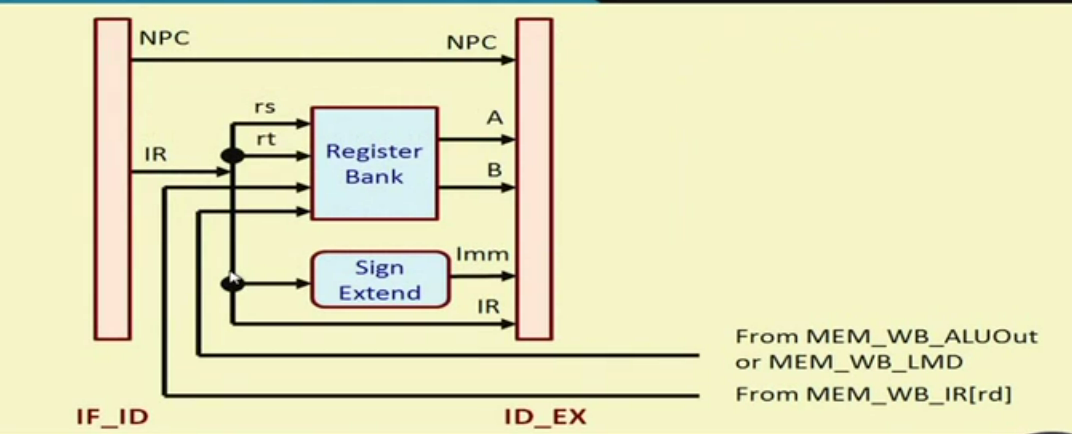
***RT-Second source register***

***Shamt-Shift amount***

***FUNC-Function to be executed***

***2)INSTRUCTION DECODE(ID)***

* ***Instruction fetched in IR is decoded.***
* ***Opcode is of 6 bits[31:26]***
* ***RS (source register) [25:21] and RT (destination register)[20:16] are of 5bits each.***
* ***16-bit immediate data from [15:0] register type operation.***
* ***26 bit immidiate data from[25:0] if immidiate type instruction.***

**

***3)EXECUTION STAGE(EX):***

* ***ALU(ARITHMETIC AND LOGICAL UNIT) is used to perform the calculations.***
* ***The ALU operates on the operands that have been already made ready in the previous cycle. [A, B, Imm].***
* ***Memory reference:***

***ALUOUT***🡨***A+Imm***

***The effective address is calculated***

* ***Register –Register ALU instruction***

***ALUOUT🡨A FUNC B***

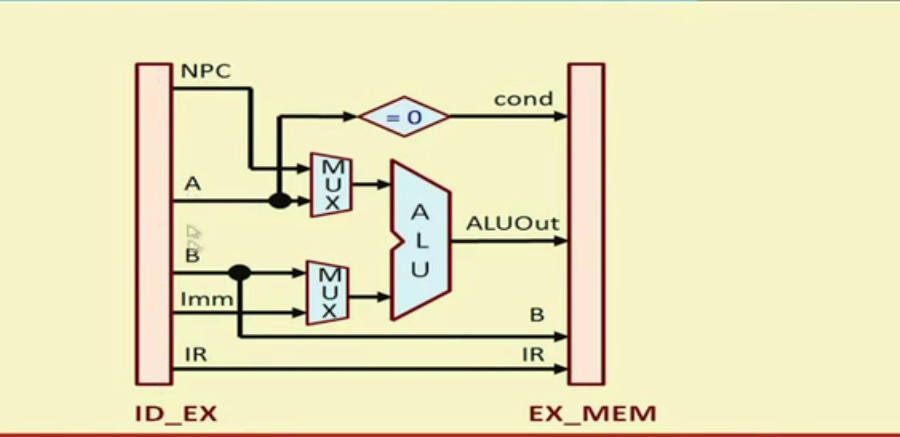
* ***Register –Immediate ALU instruction***

***ALUOUT🡨A FUNC Imm***

* ***Branch instruction***

***ALUOUT🡨NPC+Imm***

***COND🡨(A==O) O is the value used while comparing values.***



***4)MEMORY ACCESS(MEM):***

* ***Only the instruction LOAD and STORE make use of this stage***
* ***LOAD and STORE access the memory***
* ***The branch instruction updates the PC depending on the outcome.***

***PC🡨NPC***

***LMD🡨MEM[ALUOUT] (FOR LOADING)***

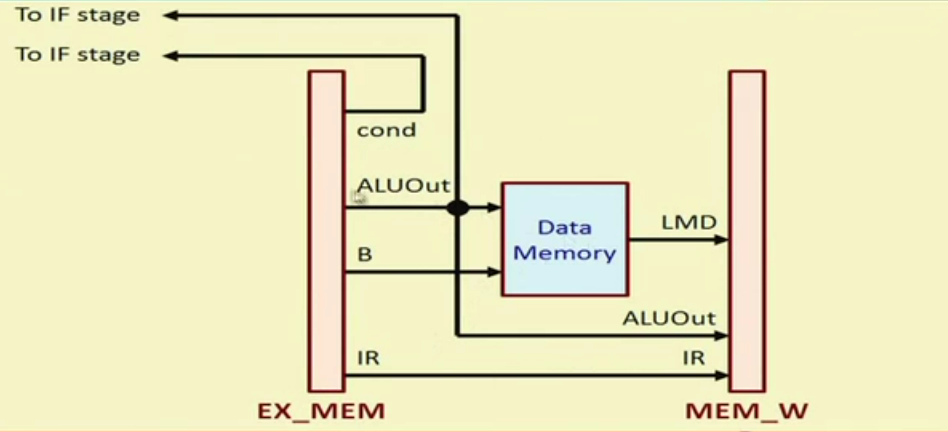
***MEM[ALUOUT]🡨B (FOR STORING)***

***BRANCH::***

***If(condition) PC🡨ALUOUT***

***Else***

***PC🡨NPC***

**

***5)WRITE BACK (WB):***

* ***The result is written back to the REGISTER file:***
* ***The result may come from ALU or the memory system.***
* ***The position of the destination register in instruction depends on the execution***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***OPCODE*** | ***RS*** | ***RT*** | ***RD*** | ***SHAMT*** | ***FUNCTION*** |

***R--TYPE***

|  |  |  |  |
| --- | --- | --- | --- |
| ***OPCODE*** | ***RS*** | ***RT*** | ***IMMEDIATE DATA*** |

***I--TYPE***

*INSTRUCTION TYPE:::*

* *REG-REG ALU*

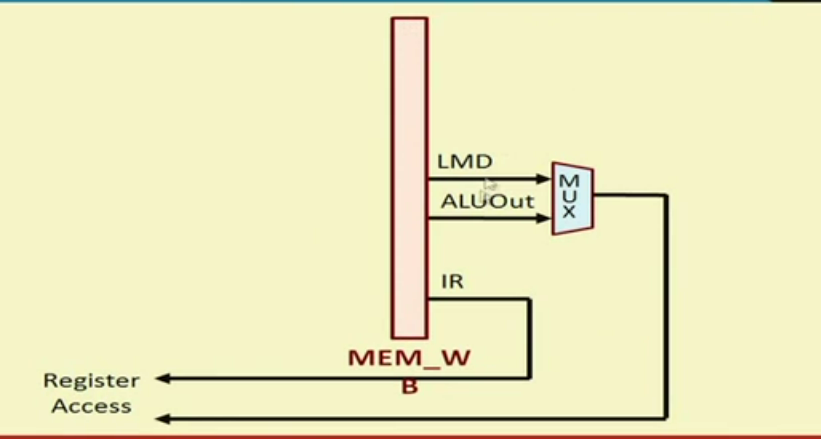
*Reg[rd]🡨ALUOUT*

* *REG-IMM ALU*

*Reg[rt]🡨ALUOUT*

* *LOAD INSTRUCTION*

*Reg[rt]🡨LMD*



***TYPE OF INSTRUCTIONS:***

***1)LOAD AND STORE INSTRUCTION:***

* *Load instruction is used to store the value in the register from a memory location*

*Syntax*

*LW Ri ,N(R)*

*So Ri register will get the value at register R+N.*

*Example:*

*LW R2,124(R8) 🡪R2=MEM[R8+128]*

* *Store instruction is used to store the value in a memory location from a register.*

*Syntax*

*SW Ri ,N(R)*

*So MEM address will get the value at register R.*

*Example:*

*SW R5,-10(R25) 🡪MEM[R25-10]=R5*

***2)ARITHMETIC AND LOGICAL OPERATION:***

***ADD***

* *Add instruction will add the content of 2 registers.*
* *Syntax :*
* *ADD RI, R2, R3*

*Here R1 will get the value R2+R3*

* *If R3 is replaced by R0 then R1=R2*

***SUB***

* *Sub-instruction will subtract the content of 2 registers.*
* *Syntax :*
* *SUB RI, R2, R3*
* *Here R1 will get the value R2-R3*

***AND***

* *This instruction will perform the logical (& and ) operation between the content of two registers.*
* *Syntax :*
* *AND R1,R2,R3*
* *Here R1 will get the values R1&R2*

***OR***

* *This instruction will perform the logical (|| or) operation between the content of two registers.*
* *Syntax :*
* *OR R1,R2,R3*
* *Here R1 will get the value R1 OR R2*

***MULTIPLY***

* *This instruction will perform the multiply operation between the content of two registers.*
* *Syntax :*
* *AND R1,R2,R3*
* *Here R1 will get the value R1 \* R2*

***SAT***

* *This instruction will compare the content of two register*
* *If the first register is less than the second register then the target register will get the value1 otherwise 0.*
* *Syntax*
* *SLT R1,R2,R3*

*Here if R2<R3 then R1=1*

*Otherwise R1=0.*

***3)ARITHMETIC AND LOGICAL OPERATION(Immidiate operation):***

***ADDI:***

* *Addi instruction will add the content of the register with some value.*
* *Syntax :*
* *ADDI RI, R2,25*

*Here R1 will get the value R2+25*

***SUBI:***

* *Subi instruction will subtract the content of the register with some value.*
* *Syntax :*
* *SUBI RI, R2,25*

*Here R1 will get the value R2-25*

***SLTI:***

* *Slti instruction will compare the register value with some value and the target register will get 1 or 0 according to output.*
* *Syntax :*
* *SLTI RI, R2<25 otherwise R1 will get zero(o).*

***4)BRANCH INSTRUCTION:***

***BEQZ:***

* *This instruction will compare the register with zero value and if true it will branch to the target instruction address otherwise PC normally gets updated.*
* *Syntax*

*BEQZ R1,LOOP*

*Here if R1 is 0 then PC will jump to the target address.*

***BNEQZ:***

* *This instruction will compare the register with zero value and if not true it will branch to the target instruction address otherwise PC normally gets updated.*
* *Syntax*

*BNEQZ R1,LOOP*

*Here if R1 is not 0 then PC will jump to the target address.*

***: 5)JUMP INSTRUCTION:***

***J :***

* *This instruction is used to jump to a target instruction without any condition output.*
* *Syntax*

*J LOOP*

*Here the PC will get updated to a targeted address.*

***: 5)MISCELLANEOUS INSTRUCTION:***

* ***HLT***

*Hlt is an instruction which is used to stop the process.*

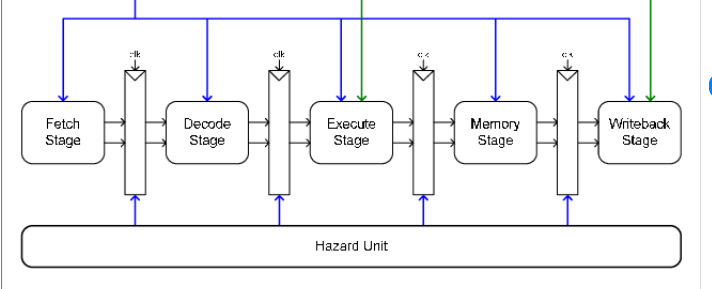
* *It is the end of the program.*

***PIPELINING OF MIPS-32 PROCESSOR:***

* *Every clock cycle a new instruction is fetched.*
* *Each of 5 steps mentioned before is executed*
* *Latches are used for pipelining different stage cells are part of latches.*
* *4 latches are used between 2 different stages.*

*.*

* *Each stage must be finished in one cycle to avoid execution errors.*



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Clock🡪*** | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** | ***6*** | ***7*** | ***8*** |
| i | ***IF*** | ***ID*** | ***EX*** | ***MEM*** | ***WB*** |  |  |  |
| i+1 |  | ***IF*** | ***ID*** | ***EX*** | ***MEM*** | ***WB*** |  |  |
| i+2 |  |  | ***IF*** | ***ID*** | ***EX*** | ***MEM*** | ***WB*** |  |
| i+3 |  |  |  | ***IF*** | ***ID*** | ***EX*** | ***MEM*** | ***WB*** |

***HAZARDS OF PIPELINING:***

* **When two (or more) instructions in the pipeline require the same resource, a structural hazard occurs. As a result, for a portion of the pipeline, instructions must be performed in series rather than parallel.**

**ADD R1,R2,R3**

**SUB R5, R1, R3 ##here R1 is required in both instructions.**

* ***In branch conditions where comparison is to be done and then the target is calculated there is an execution delay.***

***Eg: BEQZ, BNEQZ***

* ***During HLT instruction, after that, no instruction is to be executed.***

***We check for the HLT flag if high upcoming instructions do not get executed.***

* **Data hazard in pipelining arises when one instruction is dependent on the results of a preceding instruction and that result has not yet been calculated.**

***CODE FOR MIPS—32 PROCESSOR:::***

***timescale 1ns / 1ps***

***module misp32f(clk1,clk2,MEM\_WB\_ALUOut);***

***input clk1,clk2;***

***output reg[31:0]MEM\_WB\_ALUOut;***

***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\_LMD;***

***reg [31:0] Reg[0:31];***

***reg [31:0] mem[0:1023];***

***parameter***

***ADD=6'b000000,***

***SUB=6'b000001,***

***AND=6'b000010,***

***OR=6'b000011,***

***SLT=6'b000100,***

***MUL=6'b000101,***

***HLT=6'b111111,***

***LW=6'b001000,***

***SW=6'b001001,***

***ADDI=6'b001010,***

***SUBI=6'b001011,***

***SLTI=6'b001100,***

***BNEQZ=6'b001101,***

***BEQZ=6'b001110;***

***parameter RR\_ALU=3'b000,***

***RM\_ALU=3'b001,***

***LOAD=3'b010,***

***STORE=3'b011,***

***BRANCH=3'b100,***

***HALT=3'b101;***

***reg HALTED;***

***reg TAKEN\_BRANCH;***

***always @(posedge clk1)///stage1:::::***

***if(HALTED==0)***

***begin***

***if(((EX\_MEM\_IR[31:26]==BEQZ) && (EX\_MEM\_cond==1)) || ((EX\_MEM\_IR[31:26]==BNEQZ) && (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) //stage2:::***

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

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

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

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

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

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

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

***endcase***

***end***

***always @(posedge clk1)//stage3:::::***

***if(HALTED==0)***

***begin***

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

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

***TAKEN\_BRANCH<= #2 1'b0 ;***

***case(ID\_EX\_type)***

***RR\_ALU: begin//type1***

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

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

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

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

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

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

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

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

***endcase***

***end***

***RM\_ALU:begin//type2***

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

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

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

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

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

***endcase***

***end***

***LOAD, STORE: //type3***

***begin***

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

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

***end***

***BRANCH: //type4***

***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)// stage4:::::***

***if(HALTED==0)***

***begin***

***MEM\_WB\_type<=#2EX\_MEM\_type;***

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

***case(EX\_MEM\_type)***

***RR\_ALU,RM\_ALU:***

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

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

***STORE: if(TAKEN\_BRANCH==0)***

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

***endcase***

***end***

***always @(posedge clk1)//STAGE5:::***

***begin***

***if(TAKEN\_BRANCH==0) //DISABLE WRITE***

***case(MEM\_WB\_type)***

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

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

***LOAD:Reg[MEM\_WB\_IR[20:16]]<= #2 MEM\_WB\_LMD;***

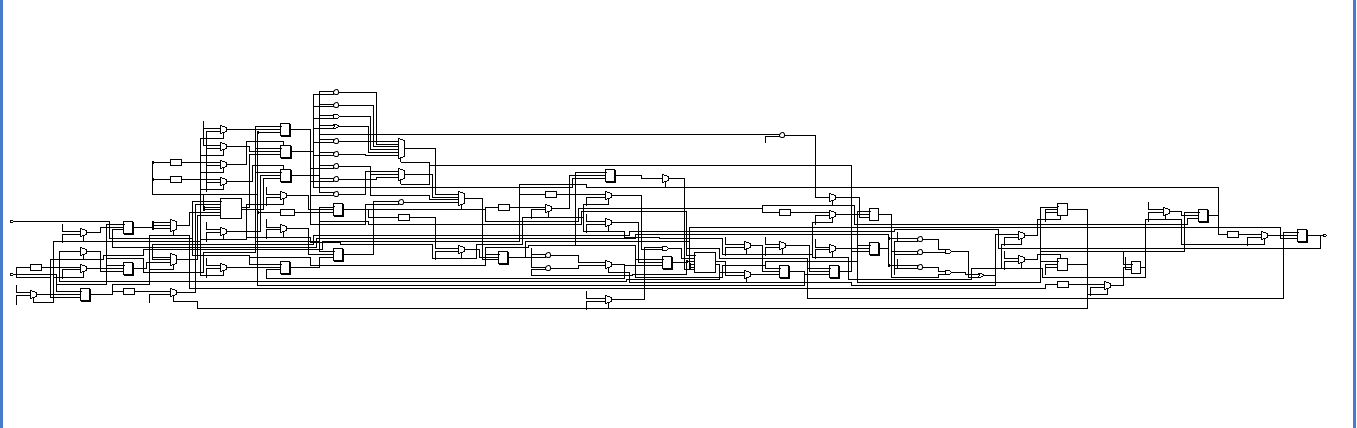
***HALT: HALTED <=#2 1'b1;***

***endcase***

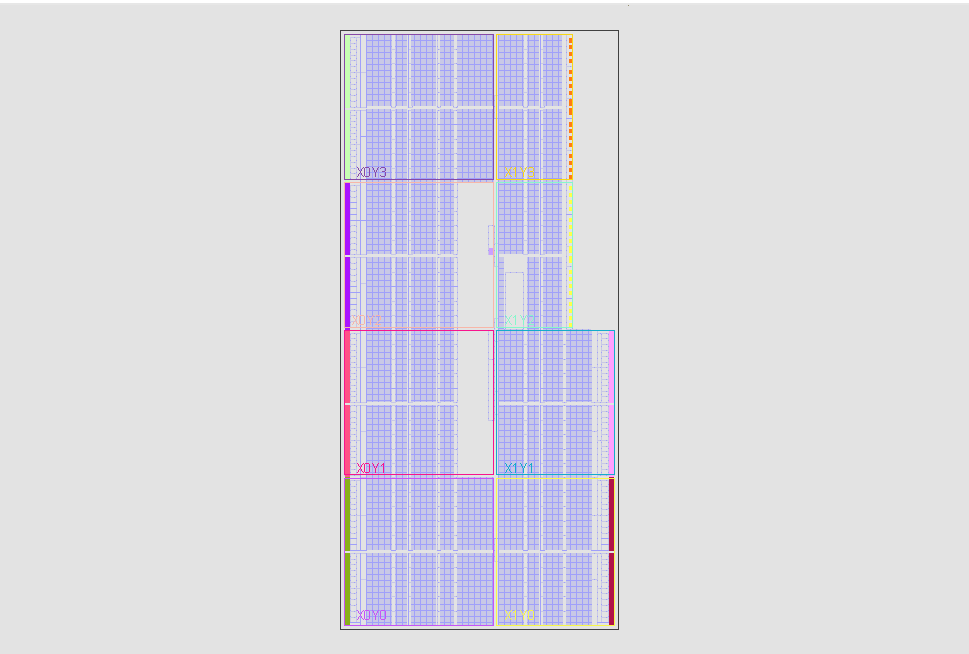
***end***

***endmodule***

***ELABORATED SCHEMATIC DESIGN OF MIPS 32 PROCESSOR USING VERILOG:***

******

***SYNTHESIS DESIGN OF MIPS-32 PROCESSOR USING VERILOG:***

******

***TEST BENCH FOR VERILOG CODE:***

***1)TEST BENCH 1:***

**Add three numbers 10, 20, and 30 stored in processor registers.**

**• The steps: -**

* **Initialize register R1 with 10. –**
* **Initialize register R2 with 20. –**
* **Initialize register R3 with 30. –**
* **Add the three numbers and store the sum in R4.**

***timescale 1ns / 1ps***

**module mipstest;**

**reg clk1,clk2;**

**integer k;**

**misp32f mips(clk1,clk2,MEM\_WB\_ALUOut);**

**initial**

**begin**

**clk1=0;clk2=0;**

**repeat(20)**

**begin //2phase clock**

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

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

**end**

**end**

**initial**

**begin**

**for (k=0; k<31; k=k+1)**

**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**

**mips.mem[4]=32'h0ce77800;// OR R7,R7,R7--DUMMY**

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

**mips.mem[6]=32'h0ce77800;// OR R7,R7,R7--DUMMY**

**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=k+1)**

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

**end**

**initial**

**begin**

**//$dumpfile("mips.vcd");**

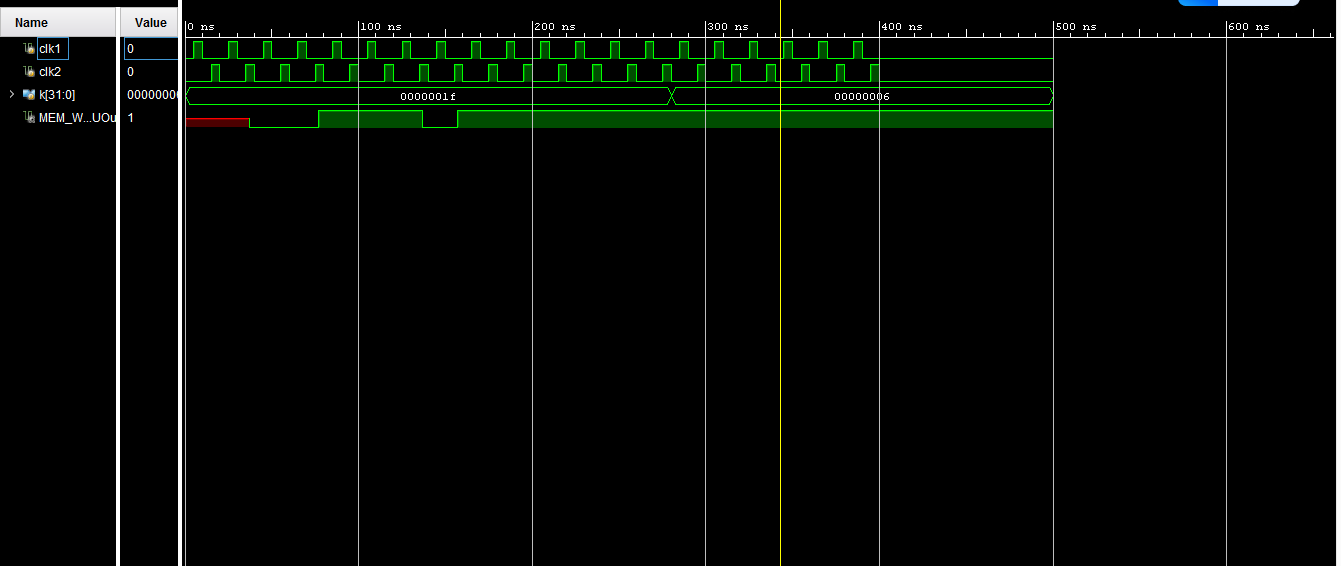
**//$dumpvars(0,mipstest);**

**#500 $finish;**

**end**

**endmodule**

***TEST BENCH 1:SIMULATION OUTPUT GRAPH:***

******

*PROCESSOR OUTPUT: TCS CONSOLE🡪*

**# run 3000ns**

**R0- 0**

**R1-10**

**R2-20**

**R3-25**

**R4-30**

**R5-55**

***2)TEST BENCH 2:***

* **Load a word stored in memory location 120, add 45 to it, and store the result in memory location 121.**
* **The steps: - Initialize register R1 with the memory address 120.**
* **Load the contents of memory location 120 into register R2. –**
* **Add 45 to register R2. –**
* **Store the result in memory location 121.**

***timescale 1ns / 1ps***

**module mipst;**

**reg clk1,clk2;**

**wire MEM\_WB\_ALUOut;**

**integer k;**

**misp32f mips(clk1,clk2,MEM\_WB\_ALUOut);**

**initial**

**begin**

**clk1=0;clk2=0;**

**repeat(20)**

**begin //2phase clock**

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

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

**end**

**end**

**initial**

**begin**

**for (k=0; k<31; k=k+1)**

**mips.Reg[k]=k;**

**//MACHINE CODE FOR INSTRUCTION: CODE**

**mips.mem[0]=32'h28010078;// ADDI R1,R0,120**

**mips.mem[1]=32'h0c631800;// OR R3,R3,R3---DUMMY**

**mips.mem[2]=32'h20220000;// LW R2,0(R1)**

**mips.mem[3]=32'h0c631800;// OR R3,R3,R3---DUMMY**

**mips.mem[4]=32'h2842002d;// ADDI R2,R2,45**

**mips.mem[5]=32'h0c631800;// OR R3,R3,R3---DUMMY**

**mips.mem[6]=32'h24220001;// SW R2,1(R1)**

**mips.mem[7]=32'hfc000000;// HLT**

**mips.mem[120]=85;**

**mips.HALTED=0;**

**mips.PC=0;**

**mips.TAKEN\_BRANCH=0;**

**#500**

**for(k=0;k<6;k=k+1)**

**$display("mem[120]: %4d \nmem[121]: %4d",mips.mem[120],mips.mem[121]);**

**end**

**initial**

**begin**

**//$dumpfile("mips.vcd");**

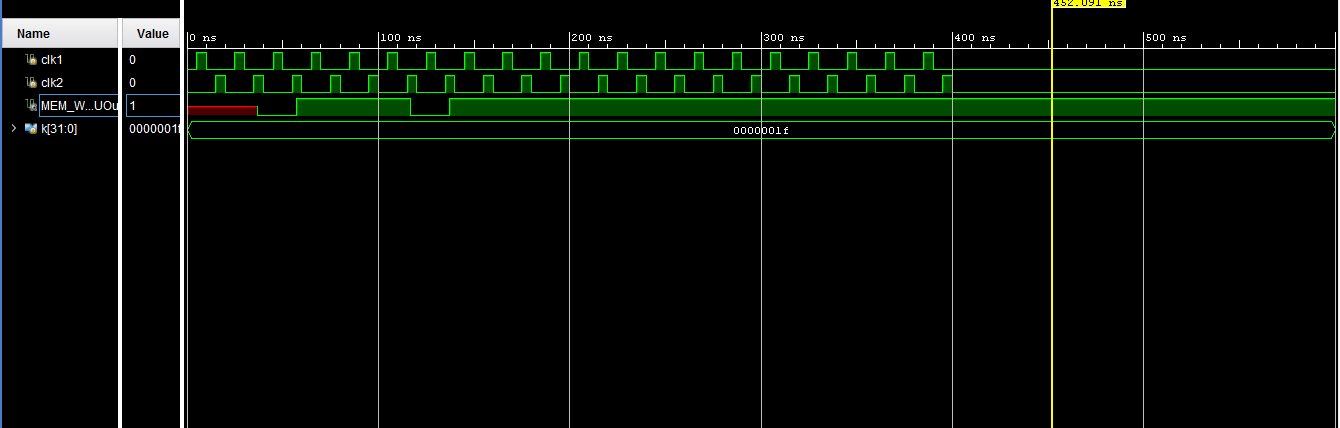
**//$dumpvars(0,mipstest);**

**#600 $finish;**

**end**

**endmodule**

***TEST BENCH 2:SIMULATION OUTPUT GRAPH:***

******

*PROCESSOR OUTPUT :TCS CONSOLE🡪*

**# run 3000ns**

**mem[120]: 85**

**mem[121]: 130**

***3)TEST BENCH 3:***

**Compute the factorial of a number N stored in memory location 200. The result will be stored in memory location 198.**

**The steps: -**

* **Initialize register R10 with the memory address 200.**
* **Load the contents of memory location 200 into register R3. –**
* **Initialize register R2 with the value 1.**
* **In a loop, multiply R2 and R3, and store the product in R2. –**
* **Decrement R3 by 1; if not zero repeat the loop. –**
* **Store the result (from R3) in memory location 198.**

***timescale 1ns / 1ps***

**module misptest3;**

**reg clk1,clk2;**

**wire MEM\_WB\_ALUOut;**

**integer k;**

**misp32f mips(clk1,clk2,MEM\_WB\_ALUOut);**

**initial**

**begin**

**clk1=0;clk2=0;**

**repeat(50)**

**begin //2phase clock**

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

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

**end**

**end**

**initial**

**begin**

**for (k=0; k<31; k=k+1)**

**mips.Reg[k]=k;**

**//MACHINE CODE FOR INSTRUCTION: CODE**

**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**

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

**mips.mem[4]=32'h0e94a000;// OR R20,R20,R20--DUMMY**

**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**

**mips.mem[8]=32'h3460fffc;// BNEQZ R3,LOOP**

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

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

**mips.mem[200]=7; // FACTORIAL OF 7**

**mips.HALTED=0;**

**mips.PC=0;**

**mips.TAKEN\_BRANCH=0;**

**#2000 $display("mem[200]= %2d, mem[198]= %6d",mips.mem[200],mips.mem[198]);**

**end**

**initial**

**begin**

**//$dumpfile("mips.vcd");**

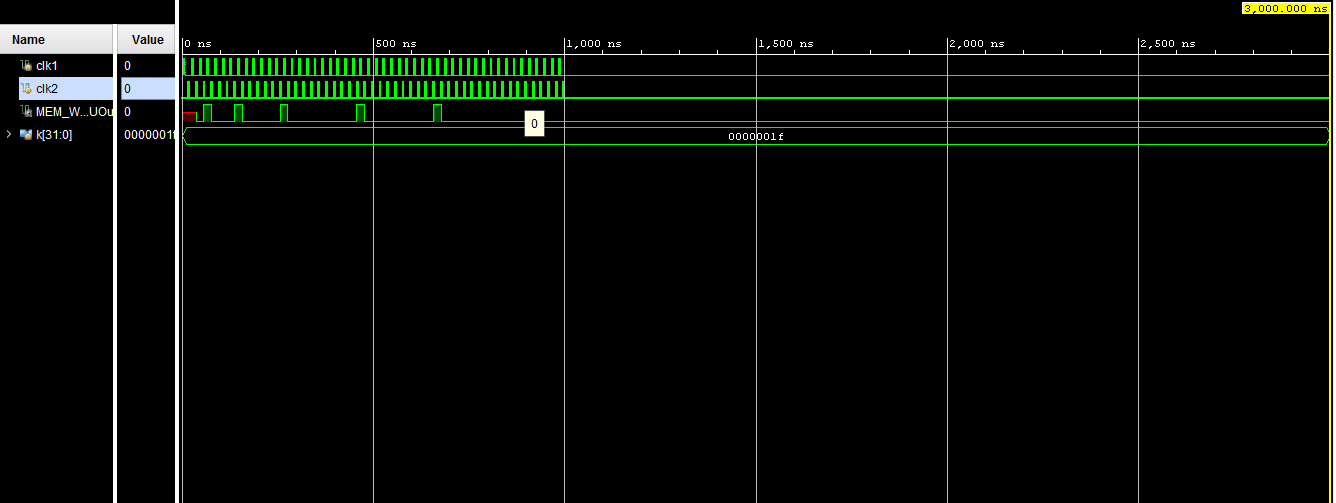
**//$dumpvars(0,mipstest);**

**#3000 $finish;**

**end**

**endmodule**

***TEST BENCH 3:SIMULATION OUTPUT GRAPH:***



*PROCESSOR OUTPUT: TCS CONSOLE🡪*

**# run 3000ns**

**R2: d 2**

**R2: d 1**

**R2: d 7**

**R2: d 42**

**R2: d 210**

**R2: d 840**

**R2: d 2520**

**R2: d 5040**

**mem[200]= 7, mem[198]= 5040**