module pipe\_MIPS32(clk1,clk2);

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 (32X32)

reg [31:0] Mem [0:1023]; // 1024 X 32 MEMORY

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;

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

reg TAKEN\_BRANCH;

// Required to disable instructions after branch

always @(posedge clk1) // IF STAGE

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))) //check op code

begin

IF\_ID\_IR <= #2 Mem[EX\_MEM\_ALUOut]; //adrs. of branch is stored in EX\_MEM\_ALUOut,now saved in IR

TAKEN\_BRANCH <= #2 1'b1; // set to 1 as branch is taken

IF\_ID\_NPC <= #2 EX\_MEM\_ALUOut + 1;// address of next is given by +1

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

end

else // BRANCH NOT TAKEN

begin

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

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

PC <= #2 PC + 1;

end

end

always @(posedge clk2) //ID stage (instruction decode)

if (HALTED == 0) //if halt is 1 ,it is skipped

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;// latch stage shift

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

ID\_EX\_Imm <= #2 {{16{IF\_ID\_IR[15]}} , {IF\_ID\_IR[15:0]}}; //concat (sign extension),32 bit quantity(16 times replication of 16th bit concat with 16 bits)

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; // INVALID OPCODE

endcase

end

always @(posedge clk1) //EX 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: begin

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; //1 if true ,0 if false(set if less than)

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

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

endcase

end

RM\_ALU: begin

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:

begin

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

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

end

BRANCH: 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) // MEM STAGE

if (HALTED == 0)

begin

MEM\_WB\_type <= #2 EX\_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) // (DIASABLE WRITE), if 1, then dont write

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

endcase

end

always @(posedge clk1) // wb stage

begin

if (TAKEN\_BRANCH == 0) // DISABLE WRITE IF BRANCH TAKEN

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; //"rt"

HALT : HALTED <= #2 1'B1;

endcase

end

endmodule