#Verilog code for implementing 32-bit Processor Module

```
module processor(clk1,clk2);

// FOR TWO PHASE CLOCK
input clk1,clk2;

// INTERSTAGE LATCHES AT EACH PIPELINE STAGE

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 [31:0] EX_MEM_IR, EX_MEM_ALU_OUT, EX_MEM_B;
reg EX_MEM_COND;                          // FOR BRANCH CONDITION CHECK
reg [31:0] MEM_WB_IR, MEM_WB_ALU_OUT, MEM_WB_LMD;

// TYPE OF INSTRUCTION TO MAKE SOME DECISIONS AT DIFFERENT STAGES
reg [2:0] ID_EX_TYPE, EX_MEM_TYPE, MEM_WB_TYPE;

reg [31:0] Reg [0:31]; // REGISTER BANK COMPRISING OF 32 REGISTERS EACH OF SIZE 32.
reg [31:0] Mem [0:1023]; // MEMORY 1024 X 32.
```

```
// DEFINING PARAMETER VALUES FOR ALU OPERATIONS
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;

// DEFINING PARAMETER VALUES FOR TYPES OF INSTRUCTION
parameter RR_ALU=3'b000, RM_ALU=3'b001, LOAD=3'b010, STORE=3'b011,
BRANCH=3'b100,HALT=3'b101;

// SET AFTER HLT INSTRUCTION IS COMPLETED
reg HALTED;

// REQUIRED TO DISABLE INSTRUCTIONS AFTER BRANCH
reg TAKEN_BRANCH;

// INSTRUCTION FETCH (IF) STAGE
always @(posedge clk1)

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_ALU_OUT];
TAKEN_BRANCH <= #2 1'b1;
IF_ID_NPC <= #2 EX_MEM_ALU_OUT+1;
PC <= #2 EX_MEM_ALU_OUT+1;
end

else
begin
 IF_ID_IR  <= #2 Mem[PC];
 IF_ID_NPC <= #2 PC+1;
 PC        <= #2 PC+1;
end

 end

// Introduction Decode (ID) Stage

always @ (posedge clk2)
if (HALTED==0)
begin
if (IF_ID_IR[25:21]==5'b00000) ID_EX_A <= 0; // assign 0 (content of R0)
else ID_EX_A <= #2 Reg[IF_ID_IR[25:21]];

if (IF_ID_IR[20:16]==5'b00000) ID_EX_B<=0; // assign 0 (content of R0)
else ID_EX_B <= #2 Reg[IF_ID_IR[20:16]];
```

```
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]}};// Sign extension of 16 bit offset.

// Decoding the type of operation to be performed.

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
// Execution stage

always @(posedge clk1)
if (HALTED==0)
begin
EX_MEM_TYPE <= ID_EX_TYPE;
EX_MEM_IR   <= ID_EX_IR;
```

```
TAKEN_BRANCH <= #2 0;

case (ID_EX_TYPE)
RR_ALU: begin
case (ID_EX_IR[31:26]) // "opcode"
ADD: EX_MEM_ALU_OUT <= #2 ID_EX_A + ID_EX_B;
SUB: EX_MEM_ALU_OUT <= #2 ID_EX_A - ID_EX_B;
AND: EX_MEM_ALU_OUT <= #2 ID_EX_A & ID_EX_B;
OR:  EX_MEM_ALU_OUT <= #2 ID_EX_A | ID_EX_B;
SLT: EX_MEM_ALU_OUT <= #2 ID_EX_A < ID_EX_B;
MUL: EX_MEM_ALU_OUT <= #2 ID_EX_A * ID_EX_B;
default: EX_MEM_ALU_OUT <= #2 32'hxxxxxxxx;
endcase
end

RM_ALU: begin
case (ID_EX_IR[31:26]) // "opcode"
ADDI: EX_MEM_ALU_OUT <= #2 ID_EX_A + ID_EX_IMM;
SUBI: EX_MEM_ALU_OUT <= #2 ID_EX_A + ID_EX_IMM;
SLTI: EX_MEM_ALU_OUT <= #2 ID_EX_A + ID_EX_IMM;
default: EX_MEM_ALU_OUT <= #2 32'hxxxxxxxx;
endcase
end

LOAD, STORE:
```

```
begin
EX_MEM_ALU_OUT <= #2 ID_EX_A + ID_EX_IMM;
EX_MEM_B <= #2 ID_EX_B;
end

BRANCH: begin

EX_MEM_ALU_OUT <= #2 ID_EX_NPC + ID_EX_IMM;
EX_MEM_COND <= #2 (ID_EX_A==0);
end

endcase
end

// Memory Excess/Branch Completion Stage
always@(posedge clk2)

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_ALU_OUT <= #2 EX_MEM_ALU_OUT;
```

```
LOAD:
       MEM_WB_LMD <= #2 Mem[EX_MEM_ALU_OUT];
STORE:
     if(TAKEN_BRANCH==0) // Disable wite
         Mem[EX_MEM_ALU_OUT] <= #2 EX_MEM_B;
endcase
end

// Write back (WB) stage

always @(posedge clk1)

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_ALU_OUT; // destination register

RM_ALU: Reg[MEM_WB_IR[20:16]]<= #2 MEM_WB_ALU_OUT; // target register

LOAD: Reg[MEM_WB_IR[20:16]]<= #2 MEM_WB_LMD; // target register

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

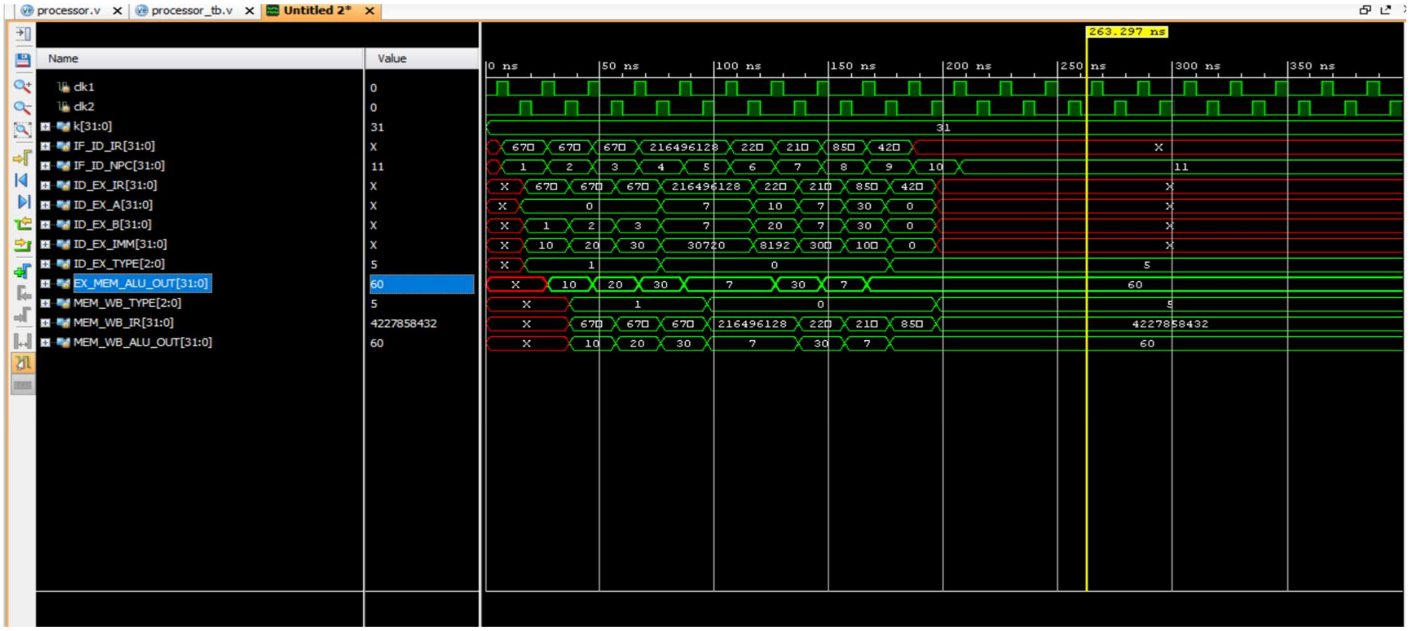
```
endcase
end

endmodule
```

# Test Bench code for performing three different operations

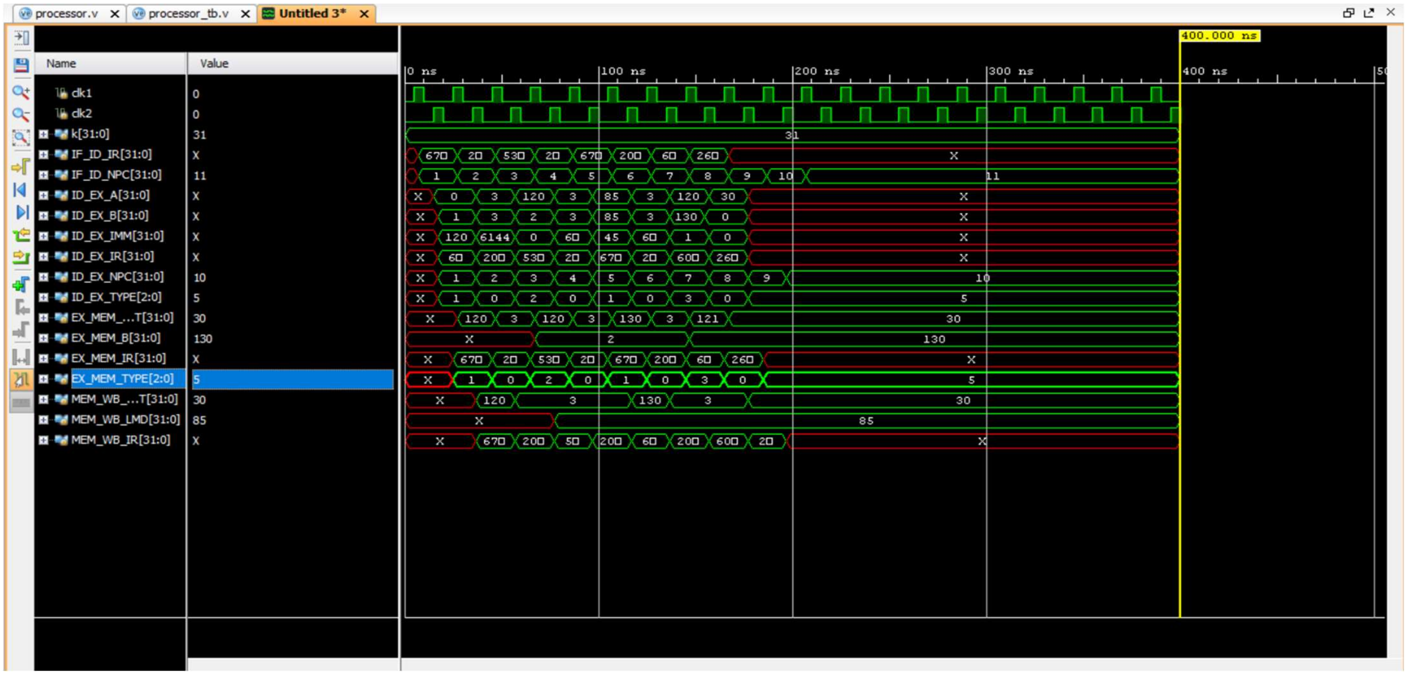
```
module processor_tb;

 reg clk1,clk2;
 integer k;

 processor DUT (clk1,clk2);

 initial
 begin
 clk1=0; clk2=0;
 repeat(20)
 begin
 #5 clk1=1; #5 clk1=0;
 #5 clk2=1; #5 clk2=0;
 end
 end

 initial
 begin
 for (k=0;k<31;k=k+1)
 processor.Reg[k]=k;
 // OPERATION ADD THREE NUMBERS 10,20 AND 30 STORED IN
PROCESSOR REGISTERS

 processor.Mem[0]=32'h2801000a; // ADDI R1,R0,10
 processor.Mem[1]=32'h28020014; // ADDI R2,R0,20
 processor.Mem[2]=32'h2803001e; // ADDI R3,R0,25
 processor.Mem[3]=32'h0ce77800; // OR  R7,R7,R7
 processor.Mem[4]=32'h0ce77800; // OR R7,R7,R7
 processor.Mem[5]=32'h00222000; // ADD R4,R1,R2
 processor.Mem[6]=32'h0ce77800; // OR R7,R7,R7
 processor.Mem[7]=32'h00832800; // AND R5,R4,R3
 processor.Mem[8]=32'hfc000000; // HLT

processor.HALTED=0;
processor.PC=0;
processor.TAKEN_BRANCH=0;

#280;
end

 /*
 // OPERATION: LOADING A WORD STORED IN MEMORY LOCATION 12,
AND ADDING 45 TO IT,
 // AND STORE THE RESULT IN MEMORY LOCATION 121

 processor.Mem[0]=32'h28010078; // ADDI R1,R0,120
 processor.Mem[1]=32'hoc631800; // OR R3,R3,R3 // DUMMY INSTRUCTION TO AVOID HAZARDS

 processor.Mem[2]=32'h20220000; // LW R2,0(R1)

 processor.Mem[3]=32'h0c631800; // OR R3,R3,R3 // DUMMY INSTRUCTION TO AVOID HAZARDS

 processor.Mem[4]=32'h2842002d; // ADDI R2,R2,45

 processor.Mem[5]=32'h0c631800; // OR R3,R3,R3 // DUMMY INSTRUCTION TO AVOID HAZARDS

 processor.Mem[6]=32'h24220001; // SW R2,1(R1)

 processor.Mem[7]=32'hfc00000; // HLT


processor.Mem[120]=85;

processor.HALTED=0;

processor.PC=0;

processor.TAKEN_BRANCH=0;


#500

end

*/




 /*

 // OPERATION : COMPUTE THE FACTORIAL OF A NUMBER N STORED IN MEMORY LOCATION 200.

 // THE RESULT WILL BE STORED IN MEMORY LOCATION 198
 processor.Mem[0]=32'h280a00c8; // ADDI R1,R0,200
 processor.Mem[1]=32'h28020001; // ADDI R2,R0,1
 processor.Mem[2]=32'h0e94a000; // OR R20,R20,R20 // DUMMY
INSTRUCTION TO AVOID HAZARDS
 processor.Mem[3]=32'h21430000; // LW  R3,0(R10)
 processor.Mem[4]=32'h0e94a000; // OR R20,R20,R20 // DUMMY
INSTRUCTION TO AVOID HAZARDS
 processor.Mem[5]=32'h14431000; // LOOP: MUL R2,R2,R3
 processor.Mem[6]=32'h2c630001; // SUBI R3,R3,1
 processor.Mem[7]=32'h0e94a000; // OR R20,R20,R20 // DUMMY
INSTRUCTION TO AVOID HAZARDS
 processor.Mem[8]=32'h3460fffc; // BNEQZ R3,LOOP (OFFSET -3)
 processor.Mem[9]=32'h2542fffe; // SW R2,-2(R10)
 processor.Mem[10]=32'hfc000000; // HLT


processor.Mem[200]=7;   // finding the factorial of data stored at memory location
200


processor.HALTED=0;
processor.PC=0;
processor.TAKEN_BRANCH=0;
#2000
end
*/


endmodule
```

# Results obtained for three operations performed by the processor.

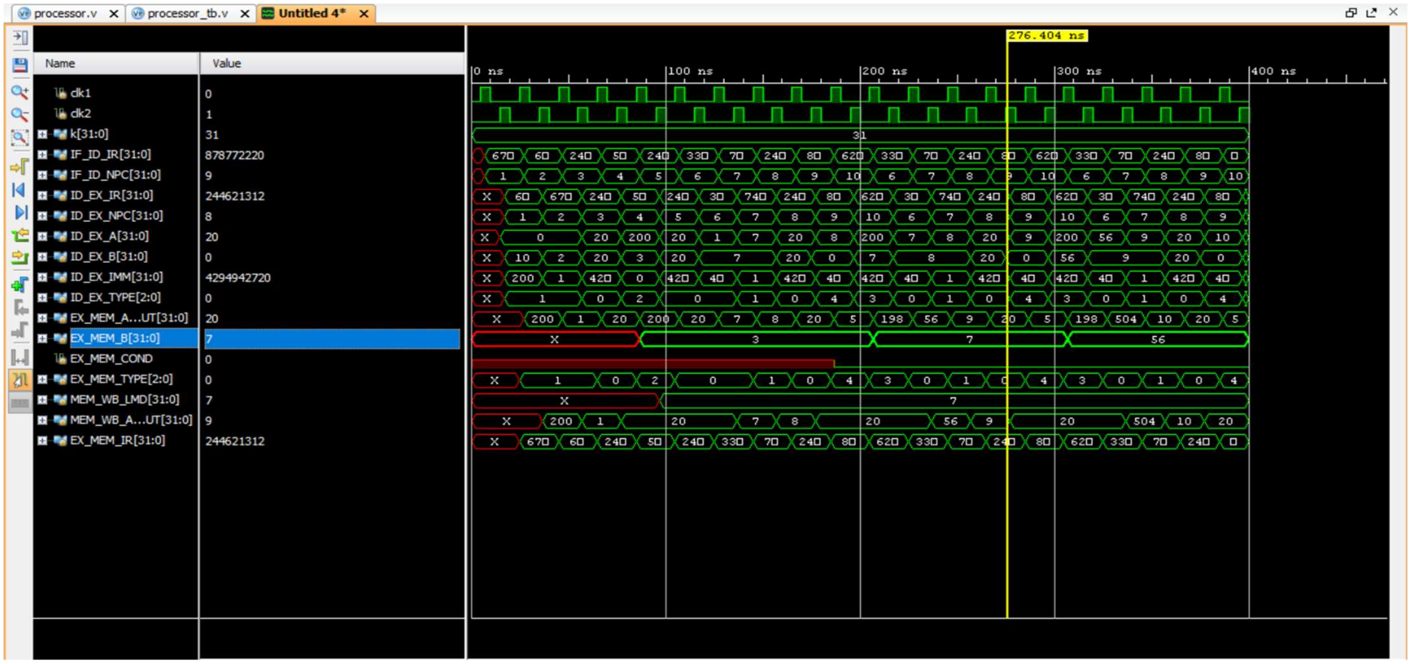
#Operation result:



# Operation 2 result



Operation 3



Name	Value
clk1	0
clk2	1
k[31:0]	31
IF_ID_IR[31:0]	878772220
IF_ID_NPC[31:0]	9
ID_EX_IR[31:0]	244621312
ID_EX_NPC[31:0]	8
ID_EX_A[31:0]	20
ID_EX_B[31:0]	0
ID_EX_IMM[31:0]	4294942720
ID_EX_TYPE[2:0]	0
EX_MEM_A...UT[31:0]	20
EX_MEM_B[31:0]	7
EX_MEM_COND	0
EX_MEM_TYPE[2:0]	0
MEM_WB_LMD[31:0]	7
MEM_WB_A...UT[31:0]	9
EX_MEM_IR[31:0]	244621312