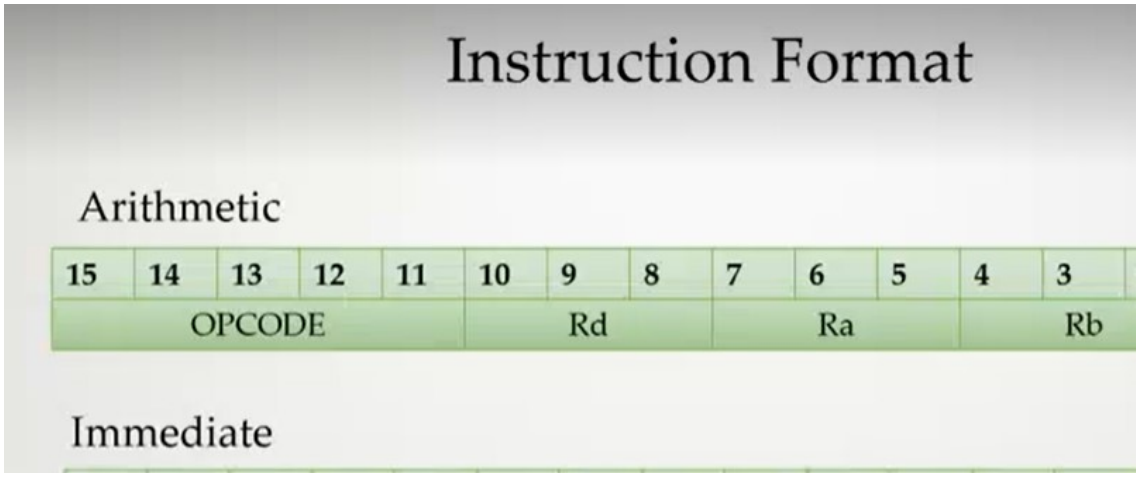
RISC Processor

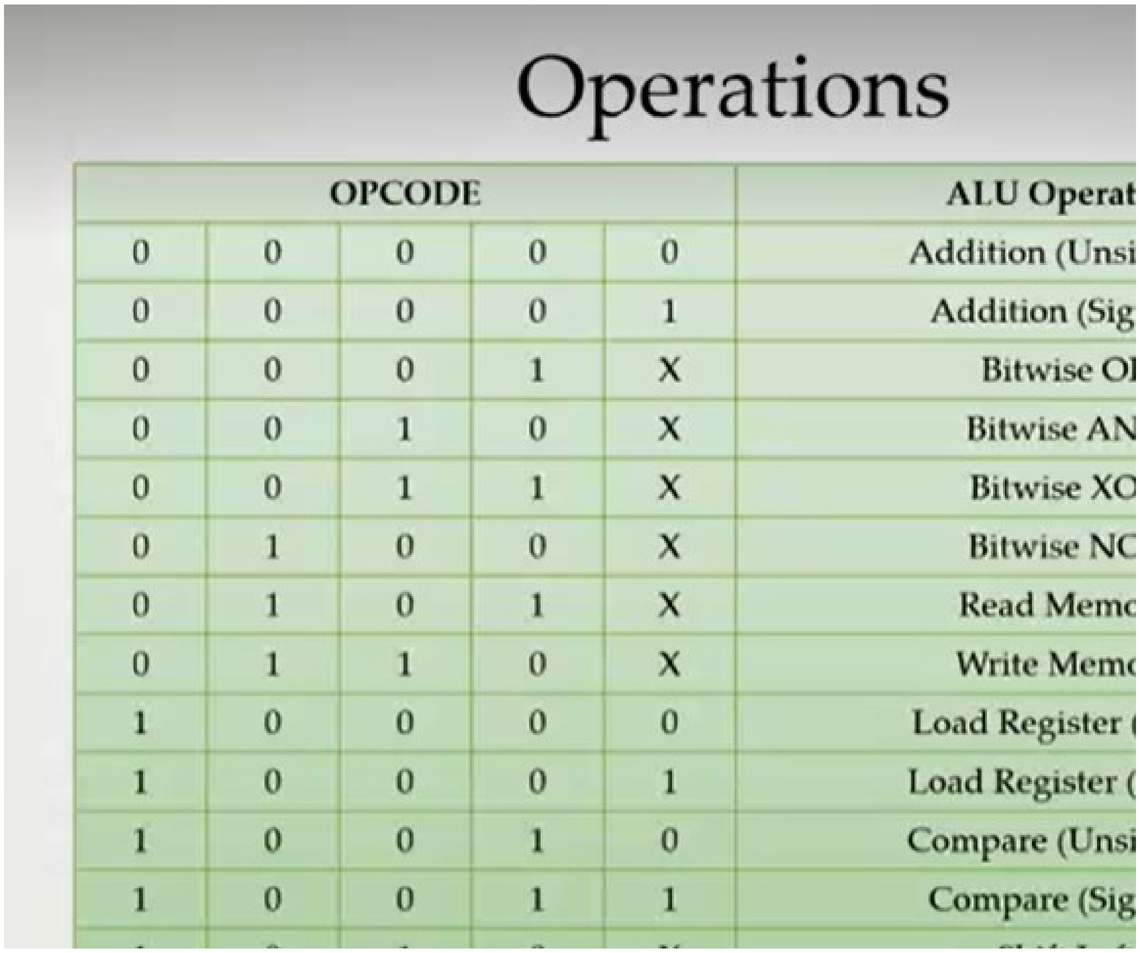


Instruction Format

Arithmetic

15	14	13	12	11	10	9	8	7	6	5	4	3
OPCODE					Rd			Ra			Rb	

Immediate



Operations

OPCODE					ALU Operat
0	0	0	0	0	Addition (Unsi
0	0	0	0	1	Addition (Sig
0	0	0	1	X	Bitwise OI
0	0	1	0	X	Bitwise AN
0	0	1	1	X	Bitwise XC
0	1	0	0	X	Bitwise NC
0	1	0	1	X	Read Memc
0	1	1	0	X	Write Memc
1	0	0	0	0	Load Register (
1	0	0	0	1	Load Register (
1	0	0	1	0	Compare (Unsi
1	0	0	1	1	Compare (Sig

Design:

Inst_dec

```
//timescale
`timescale 1ns/1ps
//module definition
module inst_dec(//inputs
            input [15:0]I_inst,
            input I_clk,
            input I_en,
//outputs
            output reg [4:0] o_aluop,
            output reg [2:0]o_selA,
            output reg [2:0]o_selB,
            output reg [2:0]o_selD,
            output reg [15:0]o_imm,
            output reg o_regwe
);
//initial block
initial begin
```

```
            o_aluop<=0;
            o_selA<=0;
            o_selB<=0;
            o_selD<=0;
            o_imm<=0;
            o_regwe<=0;
end
//instruction decoder block
always@(negedge I_clk)begin
      if(I_en)begin
                  o_aluop<=I_inst[15:11];//opcode
                  o_selA<=I_inst[10:8];//regA
                  o_selB<=I_inst[7:5];//regB
                  o_selD<=I_inst[4:2];//regD
                  o_imm<=I_inst[7:0];//Imm Data
//reg write enable
case(I_inst[15:12])
                        4'b0111:o_regwe<=0;
                        4'b1100:o_regwe<=0;
                        4'b1101:o_regwe<=0;
                        default: o_regwe<=1;
```

```
endcase
end
end
endmodule
```

ctrl_unit

```
//timescale
`timescale 1ns/1ps
//module definition
module ctrl_unit(
//inputs
            input I_clk,
            input I_reset,
//outputs
            output o_enfetch,
            output o_endec,
            output o_enrgrd,
            output o_enalu,
            output o_enrgwr,
            output o_enmem
);
```

```
//reg  decleration
              reg [5:0] state;
//initial block
initial begin
       state  <=  6'b000001;
end
//state select block
always@(posedge I_clk)begin
if(I_reset)
        state <= 6'b000001;
else begin
       case(state)
              6'b000001:  state<=    6'b000010;
               6'b000010:  state<=    6'b000100;
               6'b000100:  state<=    6'b001000;
               6'b001000:  state<=    6'b010000;
               6'b010000:  state<=    6'b100000;
               default:    state<=    6'b000001;

     endcase
     end
```

```
end
//assignment enable signals
                    assign o_enfetch=state[0];
                    assign o_endec=state[1];
                    assign o_enrgrd=state[2]|state[4];
                    assign o_enalu=state[3];
                    assign o_enrgwr=state[4];
                    assign o_enmem=state[5];

endmodule
```

alu

```
//timescale
`timescale 1ns/1ps
//module definition
module alu(
           //inputs
           input I_clk,
           input I_en,
           input[4:0]I_aluop,
           input[15:0]I_dataA,
           input [15:0]I_dataB,
           input[7:0]I_imm,
           //outputs
           output[15:0]o_dataResult,
           output reg o_shldBranch
);

//reg decleration
          reg [17:0] int_result;
          wire op_lsb;
          wire [3:0]opcode;

//parameter decleration
    localparam  Add=0,
           Sub=1,
           OR=2,
           AND=3,
           XOR=4,
           NOT=5,
           Load=8,
           Cmp=9,
                SHL=10,
                SHR=11,
                JMPA=12,
                JMPR=13;
//initial block
initial begin
    int_result <=0 ;
end
//assign values
                      assign op_lsb =I_aluop[0];
                      assign opcode = I_aluop[4:1];
                      assign o_dataResult=int_result[15:0];
//alu operations
always@(negedge I_clk)begin
   if(I_en)begin
case(opcode)
   Add:begin

int_result<=(op_lsb?($signed(I_dataA)+$signed(I_d
ataB)):(I_dataA+I_dataB));
        o_shldBranch<=0;
    end
    Sub:begin
        int_result<=(op_lsb?($signed(I_dataA)-
$signed(I_dataB)):(I_dataA-I_dataB));
        o_shldBranch<=0;
    end
    OR:begin
        int_result<=I_dataA|I_dataB;
        o_shldBranch<=0;
    end
    AND:begin
        int_result<= I_dataA & I_dataB;
        o_shldBranch<=0;
    end
    XOR:begin
        int_result<=I_dataA^I_dataB;
        o_shldBranch<=0;
    end
    NOT:begin
        int_result<=~I_dataA;
        o_shldBranch<=0;
    end
```

```
   Load:begin
      int_result<= (op_lsb
?({I_imm,8'h00}):({8'h00,I_imm}));
     // int_result<=(op_lsb ?({I_imm,8'h00}):
({8'h00,I_imm}));
      o_shldBranch <=0;
   end
   Cmp:begin
      if(op_lsb)begin
int_result[0]<=($signed(I_dataA)==$signed(I_dataB
))?1:0;
         int_result[1]<=($signed(I_dataA)==0)?1:0;
         int_result[2]<=($signed(I_dataB)==0)?1:0;

int_result[3]<=($signed(I_dataA)>$signed(I_dataB))
?1:0;
int_result[4]<=($signed(I_dataA)<$signed(I_dataB))
?1:0;
   end else begin
         int_result[0]<= (I_dataA== I_dataB)?1:0;
         int_result[1]<= (I_dataA==0)?1:0;
         int_result[2]<= (I_dataB==0)?1:0;
         int_result[3]<= (I_dataA>I_dataB)?1:0;
```

```
        int_result[4]<= (I_dataA<I_dataB)?1:0;
    end
        o_shldBranch<=0;
    end
    SHL:begin
      int_result<=I_dataA<<(I_dataB[3:0]);
      o_shldBranch<=0;
    end
    SHR:begin
      int_result<=I_dataA>>(I_dataB[3:0]);
      o_shldBranch<=0;
    end
    JMPA:begin
      int_result<=(op_lsb?I_dataA:I_imm);
      o_shldBranch<=1;
    end
    JMPR:begin
      int_result<= I_dataA;
      o_shldBranch<= I_dataB [{op_lsb ,
I_imm[1:0]}];
    end
```

```
endcase
end
end
endmodule
```

PC_UNIT

```
`timescale 1ns/1ps
//module definition
module pc_unit(
//inputs
          input I_clk,
          input [1:0]I_opcode,
          input[15:0]I_pc,
//outputs
         output reg[15:0]o_pc
);
//initial block
initial begin
o_pc<=0;
end
//program counter state
```

```
always@(negedge I_clk)begin
case(I_opcode)
            2'b00:o_pc<=o_pc;
            2'b01:o_pc<=o_pc+1;
            2'b10:o_pc<=I_pc;
            2'b11:o_pc<=0;
endcase
end
endmodule
```

REG_file

```
//timescale
`timescale 1ns/1ps
//module definition
module reg_file(
//inputs
            input I_clk,
            input I_en,
            input I_we,
            input[2:0] I_selA,
            input[2:0]I_selB,
             input[2:0]I_selD,
             input[15:0]I_dataD,
//output
             output reg[15:0]o_dataA,
             output reg[15:0]o_dataB
);
//internal reg decleration
reg [15:0]regs[7:0];
//loop variables
integer count;
//initial output
initial begin
                    o_dataA=0;
                   o_dataB=0;
for(count=0;count<8;count=count+1)begin
 regs[count]=0;
end
end
//assigning correct values to op regs
always@(negedge I_clk)begin
 if(I_en)begin
if(I_we)
            regs[I_selD]<=I_dataD;
            o_dataA<=regs[I_selA];
            o_dataB<=regs[I_selB];
end
end
endmodule
```

fake_ram

```
//timescale
`timescale 1ns/1ps
//module definition
module fake_ram(
//inputs
            input I_clk,
            input I_we,
            input [15:0]I_addr,
            input [15:0]I_data,
//outputs
            output reg [15:0]o_data
);
```

```
//memory decleration
reg [15:0] mem[8:0];
//initialize registers
initial begin
            mem[0]=16'b1000000011111110;
            mem[1]=16'b1000100111101101;
            mem[2]=16'b0010001000100000;
            mem[3]=16'b1000001100000001;
            mem[4]=16'b1000010000000001;
            mem[5]=16'b0000001101110000;
            mem[6]=16'b1100000000000100;
            mem[7]=0;
            mem[8]=0;
            o_data=16'b0000000000000000;

end
//ram opearation
always@(negedge I_clk)begin
if(I_we)begin
mem[I_addr[15:0]]<=I_data;
end
```

```
o_data<=mem[I_addr[15:0]];
end
endmodule
```

Test bench

Decode_unitest

```
//timescale
`timescale 1ns/1ps
//module definition
module decoder_unittests ();
//variable declerations
            reg  I_clk;
            reg I_en;
            reg [15:0]I_inst;
//wires
            wire[4:0]o_aluop;
            wire[2:0]o_selA;
            wire[2:0]o_selB;
            wire[2:0]o_selD;
            wire[15:0]o_imm;
            wire o_regwe;
inst_dec inst_unit(
//inputs
            I_inst,
            I_clk,
            I_en,
//outputs
            o_aluop,
            o_selA,
            o_selB,
            o_selD,
            o_imm,
            o_regwe
);

initial begin
//time=0
I_clk=0;I_en=0;
I_inst=0;
 //time=10
#10;
```

```
I_inst=16'b0001011100000100;
//time=20
#10;
I_en=1;
end

always begin
#5;
I_clk=~I_clk;

end

endmodule
```

regfile_unitest

```
//timescale
`timescale 1ns/1ps
//module definition
module regfile_unittest();
//variables decleration
```

```
//regs
                reg  I_clk;
                reg  [15:0]I_dataD;
                reg  I_en;
                reg  [2:0]I_selA;
                reg [2:0]I_selB;
                reg  [2:0]I_selD;
reg  I_we;
//wires
                wire [15:0]o_dataA;
                wire [15:0]o_dataB;



reg_file reg_test(
//inputs
          I_clk,
          I_en,
          I_we,
          I_selA,
          I_selB,
          I_selD,
          I_dataD,
//output
          o_dataA,
          o_dataB
);
initial begin
          I_clk=1'b0;
          I_dataD=0;
          I_en=0;
          I_selA=0;
          I_selD=0;
          I_selB=0;
          I_we=0;
//start test
//time=7
#7
          I_en=1'b1;
          I_selA=3'b000;
          I_selB=3'b001;
          I_selD=3'b000;
          I_dataD=16'hFFFF;
```

```
        I_we=1'b1;
//time=17
#10;
        I_we=1'b0;
        I_selD=3'b010;
        I_dataD=16'h2222;
//time =27
#10;
        I_we=1;
//time=37
#10;
        I_dataD=16'h3333;
//time=47
#10;
        I_we=0;
        I_selD=3'b000;
        I_dataD=16'hFEED;
//time=57
#10;
        I_selD=3'b100;
        I_dataD=16'h4444;
```

```
//time=67
#10;
            I_we=1;
//time=117;
#50;
            I_selA=3'b100;
            I_selB=3'b100;

end
always begin
#5
I_clk=~I_clk;
end

endmodule
```

main_test

```
//timescale
`timescale 1ns/1ps
```

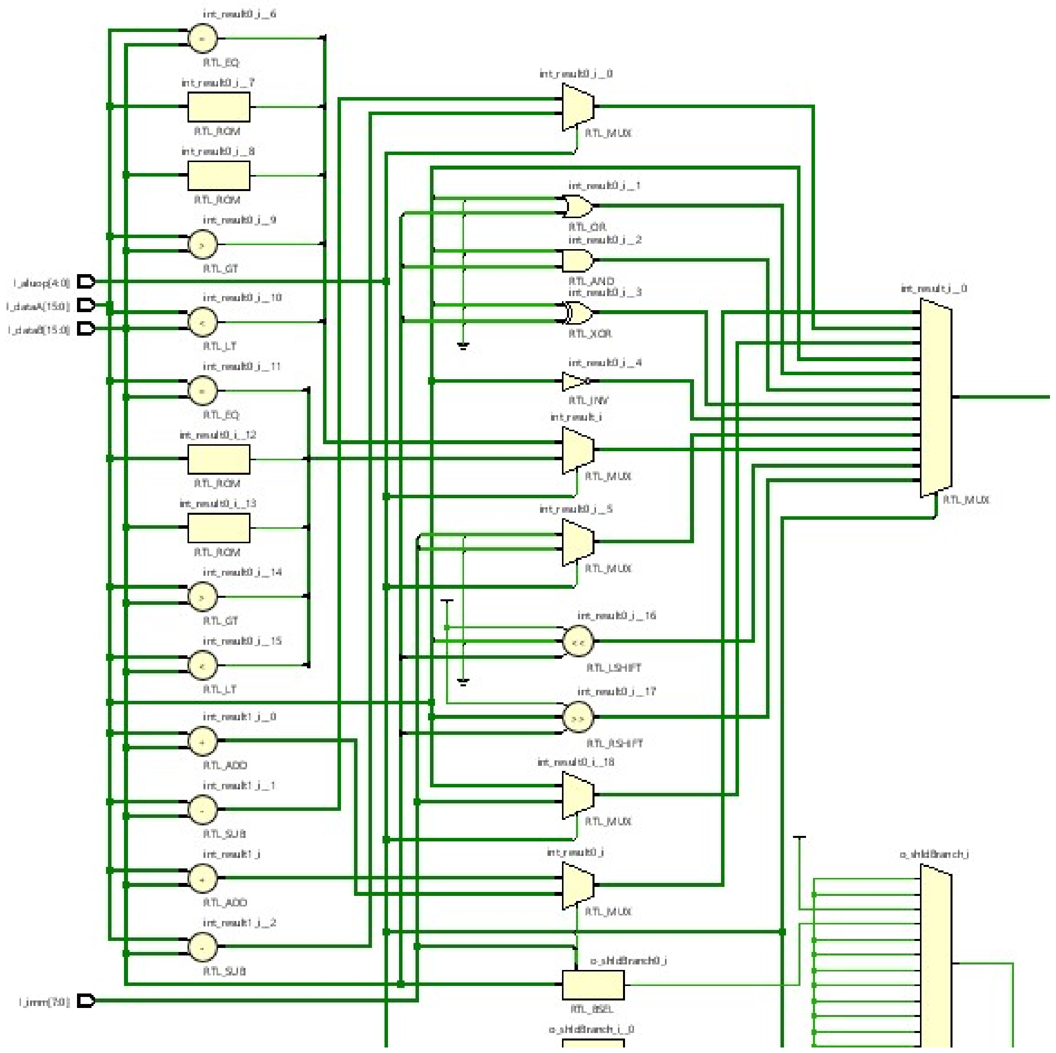
```
//module definition
module main_test();
//variable decleration
//regs

            reg clk;
            reg reset;
            reg ram_we=0;
            reg [15:0]dataI=0;

//wires
                  wire [2:0] selA;
                  wire  [2:0]selB;
                  wire [2:0] selD;
                  wire [15:0]dataA;
                  wire [15:0]dataB;
                  wire [15:0]dataD;
                  wire [1:0]opcode;
                  wire[4:0]aluop;
                  wire[7:0]imm;
                  wire [15:0]dataO;
                  wire [15:0]pcO;

                  wire shldBranch;
                  wire enfetch;
```

```
                wire enalu;
                wire endec;
                wire enmem;
                wire enrgrd;
                wire enrgwr;
                wire regwe;
                wire update;

//assigning
            assign enrgwr=regwe & update;
            assign opcode=(reset)?2'b11: ((shldBranch ?2'b10:
            ((enmem)?2'b01: 2'b00)));
//instantiations
 reg_file  main_reg(
//inputs
            clk,
            enrgrd,
            enrgwr,
            selA,
            selB,
            selD,
            dataD,
//outputs
            dataA,
            dataB
```

```
);
inst_dec main_inst(
//inputs

		clk,
		endec,
		dataO,
		//outputs
		aluop,
		selA,
		selB,
		selD,
		imm,
		regwe
);

alu main_alu(
//inputs
		clk,
		enalu,
		aluop,
		dataA,
		dataB,
		imm,
//outputs
```

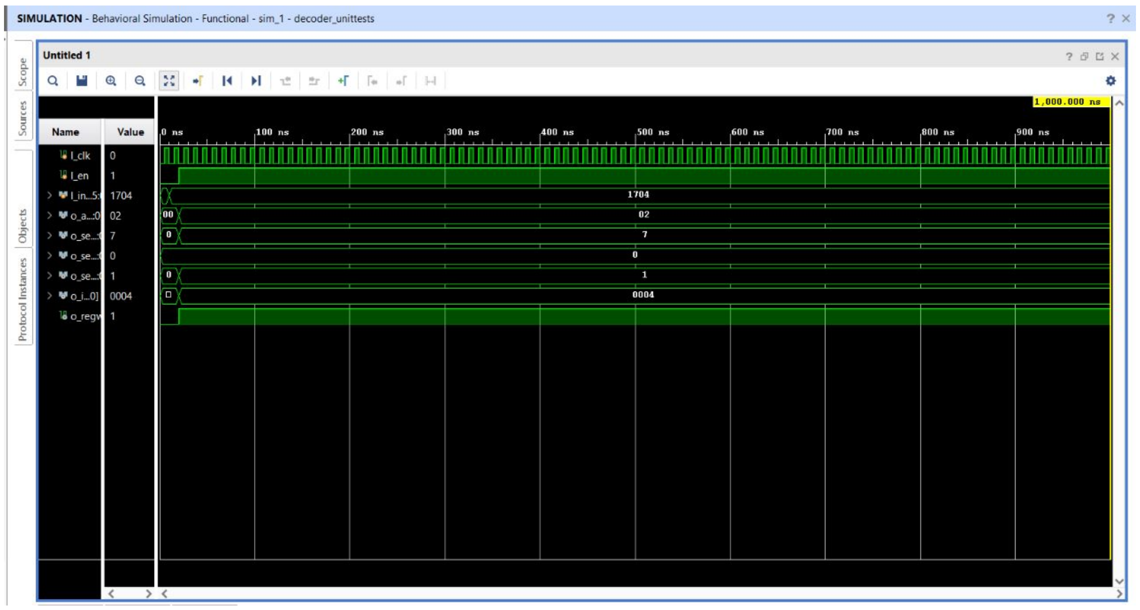
```
            dataD,
            shldBranch
);

ctrl_unit main_ctrl(
//inputs
            clk,
            reset,
            //outputs
            enfetch,
            endec,
            enrgrd,
            enalu,
            update,
            enmem
);
pc_unit main_pc(
//inputs
            clk,
            opcode,
            dataD,
//outputs
pcO
);
fake_ram main_ram(
//inputs
		clk,
		ram_we,
		pcO,
		dataI,
//outputs
		dataO
);

initial begin
		clk=0;
		reset=1;
#20
		reset=0;

end
//clock generation
always begin
	#5;
	clk=~clk;
end
endmodule
```

Simulations



int_result0_i__6
RTL_EQ
int_result0_i__7
RTL_ROM
int_result0_i__8
RTL_ROM
int_result0_i__9
RTL_GT
i_aluop[4:0]
i_dataA[15:0]
i_dataB[15:0]
int_result0_i__10
RTL_LT
int_result0_i__11
RTL_EQ
int_result0_i__12
RTL_ROM
int_result0_i__13
RTL_ROM
int_result0_i__14
RTL_GT
int_result0_i__15
RTL_LT
int_result1_i__0
RTL_ADD
int_result1_i__1
RTL_SUB
int_result1_i
RTL_ADD
int_result1_i__2
RTL_SUB
i_imm[7:0]
int_result0_i__0
RTL_MUX
int_result0_i__1
RTL_OR
int_result0_i__2
RTL_AND
int_result0_i__3
RTL_XOR
int_result0_i__4
RTL_INV
int_result_i
RTL_MUX
int_result0_i__5
RTL_MUX
int_result0_i__16
RTL_LSHIFT
int_result0_i__17
RTL_RSHIFT
int_result0_i__18
RTL_MUX
int_result0_i
RTL_MUX
o_shldBranch0_i
RTL_BSEL
o_shldBranch_i__0
int_result_i__0
RTL_MUX
o_shldBranch_i

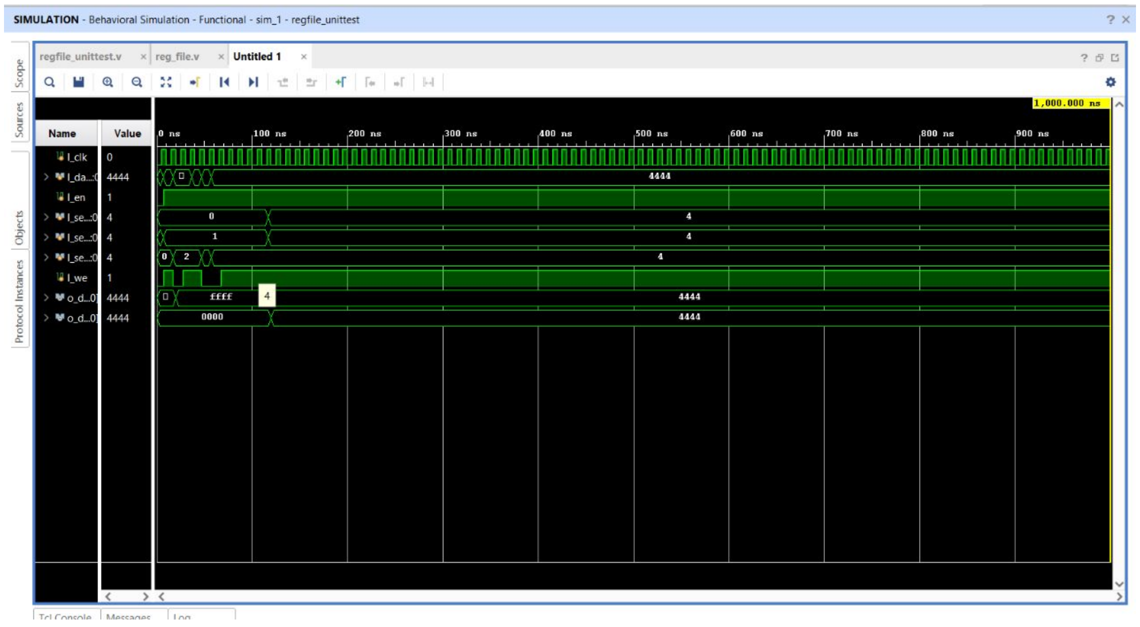
Waveforms:



SIMULATION - Behavioral Simulation - Functional - sim_1 - decoder_unittests

Untitled 1

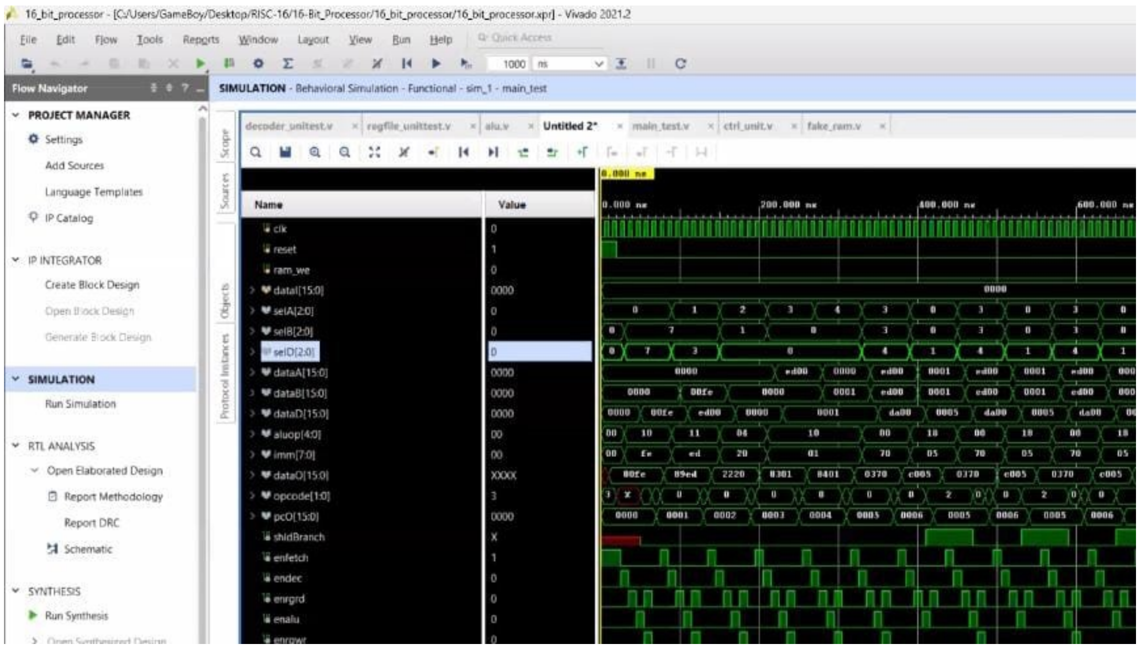
```
Name        Value
i_clk       0
i_en        1
i_in...5:   1704
o_a...:0    02
o_se...:    7
o_se...:    0
o_se...:    1
o_i...0]    0004
o_regw      1
```



SIMULATION - Behavioral Simulation - Functional - sim_1 - regfile_unittest

regfile_unittest.v × reg_file.v × Untitled 1 ×

```
Name        Value
i_clk       0
i_da...:0   4444
i_en        1
i_se...:0   4
i_se...:0   4
i_se...:0   4
i_we        1
o_d...0]    4444
o_d...0]    4444
```



16_bit_processor - [C:/Users/GameBoy/Desktop/RISC-16/16-Bit_Processor/16_bit_processor/16_bit_processor.xpr] - Vivado 2021.2

SIMULATION - Behavioral Simulation - Functional - sim_1 - main_test

Name	Value
clk	0
reset	1
ram_we	0
dataI[15:0]	0000
selA[2:0]	0
selB[2:0]	0
selD[2:0]	0
dataA[15:0]	0000
dataB[15:0]	0000
dataD[15:0]	0000
aluop[4:0]	00
imm[7:0]	00
dataO[15:0]	XXXX
opcode[1:0]	3
pcO[15:0]	0000
shldBranch	X
enfetch	1
endec	0
enrgrd	0
enalu	0
enrgwr	0