

// blue box contains Instruction Fetch & program counter //// Data Memory // need to add the **program.hex**  **data\_file.txt I used block ram**

module bram

   #(

      parameter RAM\_WIDTH     = 32,

      parameter RAM\_ADDR\_BITS    = 9,

      parameter DATA\_FILE     = "data\_file.txt",

      parameter INIT\_START\_ADDR  = 0,

      parameter INIT\_END\_ADDR    = 10

   )

   (

   input                   clock,

   input                   ram\_enable,

   input                   write\_enable,

    input      [RAM\_ADDR\_BITS-1:0]  address,

    input      [RAM\_WIDTH-1:0]   input\_data,

   output reg  [RAM\_WIDTH-1:0]   output\_data

   );

   (\* RAM\_STYLE="BLOCK" \*)

   reg [RAM\_WIDTH-1:0] ram\_name [(2\*\*RAM\_ADDR\_BITS)-1:0];

   //  The forllowing code is only necessary if you wish to initialize the RAM

   //  contents via an external file (use $readmemb for binary data)

   initial

      $readmemh(DATA\_FILE, ram\_name, INIT\_START\_ADDR, INIT\_END\_ADDR);

   always @(posedge clock)

      if (ram\_enable) begin

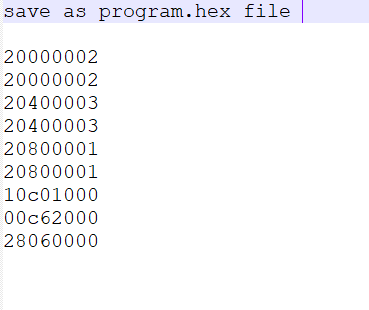
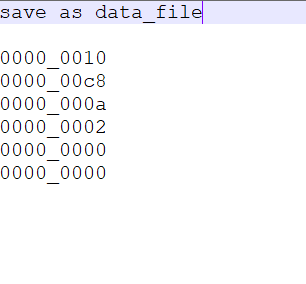
         if (write\_enable)

            ram\_name[address] <= input\_data;

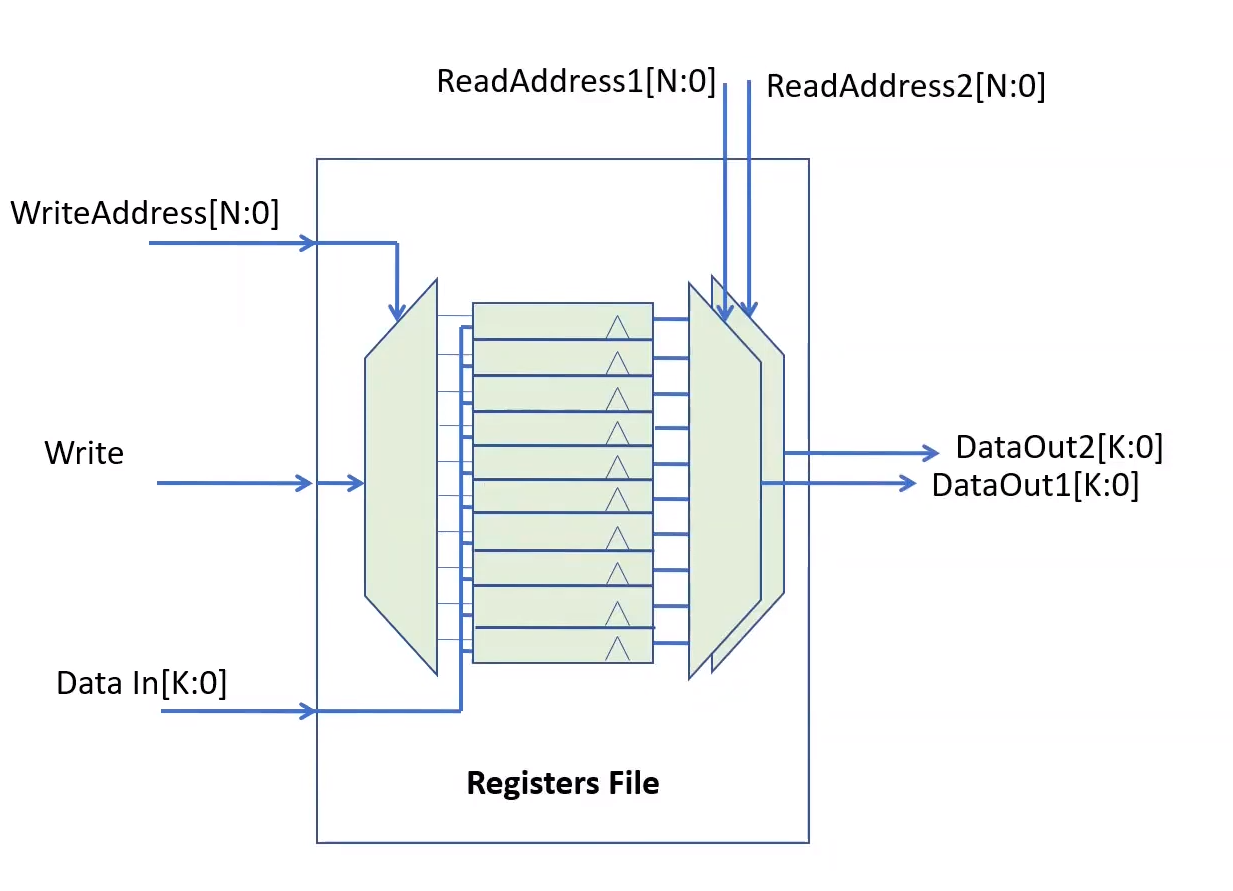
         output\_data <= ram\_name[address];

      end

endmodule

**//green box is register file**

****

**//THIS IS DATA STORAGE FILE // Operand Fetch & Write back**

//THIS IS DATA STORAGE FILE

module reg\_file(

        input           clk,

        input   [4:0]   rd\_addr1,

        input   [4:0]   rd\_addr2,

        output  [31:0]  rd\_data1,

        output  [31:0]  rd\_data2,

        input   [4:0]   wr\_addr,

        input           wr\_en,

        input   [31:0]  wr\_data

);

reg [31:0] regs[0:31];

always@(posedge clk)

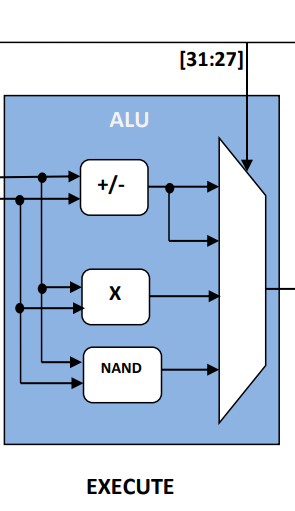
    if(wr\_en)   regs[wr\_addr] <= wr\_data;

assign  rd\_data1 = regs[rd\_addr1];

assign  rd\_data2 = regs[rd\_addr2];

endmodule

**// ALU and execute**



//THIS IS COMPELETE processor FOR NEWBIES

module processor

    #(

        parameter RAM\_WIDTH         = 32,

        parameter RAM\_ADDR\_BITS     = 9,

        parameter DATA\_FILE         = "data\_file.txt",

        parameter DATA\_START\_ADDR   = 0,

        parameter DATA\_END\_ADDR     = 5,

        parameter PROG\_FILE         = "program.hex",  // A= B + C\*D -- B=200 C=10 D=2

        parameter PROG\_START\_ADDR   = 0,

        parameter PROG\_END\_ADDR     = 8

    )

(

    input clk,

    input reset

);

parameter ADD  = 'd0;

parameter SUB  = 'd1;

parameter MUL  = 'd2;

parameter NAND = 'd3;

parameter LW   = 'd4;

parameter SW   = 'd5;

reg     [RAM\_ADDR\_BITS-1:0] prog\_cnt;

wire    [RAM\_WIDTH-1:0]     instr;

wire    [RAM\_WIDTH-1:0]     op1;

wire    [RAM\_WIDTH-1:0]     op2;

reg     [RAM\_WIDTH-1:0]     alu\_out;

reg     [RAM\_WIDTH-1:0]     reg\_data\_in;

wire    [RAM\_WIDTH-1:0]     dram\_data\_out;

reg     [4:0]               opcode;

reg                     ctl1,ctl2,ctl3;

// Instruction Fetch

always@(posedge clk)

    if(reset)   prog\_cnt <= 0;

    else        prog\_cnt <= prog\_cnt + 1;

bram

#(

    .RAM\_WIDTH      (RAM\_WIDTH          ),

    .RAM\_ADDR\_BITS  (RAM\_ADDR\_BITS      ),

    .DATA\_FILE      (PROG\_FILE          ),

    .INIT\_START\_ADDR(PROG\_START\_ADDR    ),

    .INIT\_END\_ADDR  (PROG\_END\_ADDR      )

)

prog\_mem

(

    .clock          (clk            ),

    .ram\_enable     (1'b1           ),

    .write\_enable   (1'b0           ),

    .address        (prog\_cnt       ),

    .input\_data     (0              ),

    .output\_data    (instr          )

);

// Operand Fetch & Write back

reg\_file regfile\_inst

(

        .clk            (clk),

        .rd\_addr1       (instr[21:17]),

        .rd\_addr2       (instr[16:12]),

        .rd\_data1       (op1),

        .rd\_data2       (op2),

        .wr\_addr        (instr[26:22]),

        .wr\_en          (ctl1),

        .wr\_data        (reg\_data\_in)

);

// ALU

always@\* opcode = instr[31:27];

always@\*

    case(opcode[1:0])

    ADD : alu\_out =   op1 + op2;

    SUB : alu\_out =   op1 - op2;

    MUL : alu\_out =   op1 \* op2;

    NAND: alu\_out = ~(op1 & op2);

    endcase

always@\* reg\_data\_in = ctl2 ? dram\_data\_out : alu\_out;

// Data Memory

bram

#(

    .RAM\_WIDTH      (RAM\_WIDTH      ),

    .RAM\_ADDR\_BITS  (RAM\_ADDR\_BITS  ),

    .DATA\_FILE      (DATA\_FILE      ),

    .INIT\_START\_ADDR(DATA\_START\_ADDR),

    .INIT\_END\_ADDR  (DATA\_END\_ADDR  )

)

data\_mem

(

    .clock          (clk            ),

    .ram\_enable     (1'b1           ),

    .write\_enable   (ctl3           ),

    .address        (instr[8:0]     ),

    .input\_data     (op1            ),

    .output\_data    (dram\_data\_out  )

);

// Control Signals

always@\* ctl1 = ~ctl3;

always@\* ctl2 = opcode == LW;

always@\* ctl3 = opcode == SW;

endmodule