Arquivo com os Módulos das Memórias

Módulos do Arquivo `imem_rf_dmem.v`:

- Arquivo `imem_rf_dmem.v`: Módulos referentes às memórias da CPU RISC-V.
  - Módulo `reg_file`: **Conjunto de registradores** (*register File*) com **32 registradores**.
  - Módulo `instr_mem`: **Memória ROM** para armazenar os **PROGRAMAS (*program memory*)**. Utiliza os blocos de memória **M9K**.
  - Módulo `data_mem_single`: **Memória RAM** para armazenamento de **DADOS (*data memory*)**. Utiliza os blocos de memória **M9K**.

---

1 - Módulo `reg_file`

- **Conjunto de registradores** (*register File*) com **32 registradores**.
  - **Constantes**:
    - `DATA_WIDTH`: Comprimento, em bits, do *array* de dados.
    - `END_IDX`: Índice do último elemento do array com os dados.
  - **Entradas**:
    - `clk`: Entrada que recebe os pulsos de *clock*.
    - `wr_en`: Sinal indicando se o valor da entrada `w_data` deverá ser escrito no registrador `w_addr`.
    - `r_addr1`: Endereço do registrador onde está armazenado o valor `r_data1`.
    - `r_addr2`: Endereço do registrador onde está armazenado o valor `r_data2`.
    - `w_addr`: Endereço do registrador onde será escrito o valor `w_data`.
    - `w_data`: *Array* com o valor a ser escrito no registrador de endereço `w_addr`.
  - **Saídas**:
    - `r_data1`: *Array* com o valor lido no registrador de endereço `r_addr1`.
    - `r_data2`: *Array* com o valor lido no registrador de endereço `r_addr2`.

```
module reg_file #( parameter DATA_WIDTH = 32, parameter END_IDX=DATA_WIDTH-1 )
                 ( input  wire             clk,
                   input  wire             wr_en,
                   input  wire       [4:0] r_addr1,
                   input  wire       [4:0] r_addr2,
                   input  wire       [4:0] w_addr,
                   input  wire [END_IDX:0] w_data,
```

```
                    output wire [END_IDX:0] r_data1,
                    output wire [END_IDX:0] r_data2 );
//------------------------------------------------------------------------
    // Constantes:
    parameter VAL_0 = { DATA_WIDTH { 1'b0 } };

    // --> Matriz com os 32 REGISTRADORES que compõe o register file
    // Sobre a expressao '(* ramstyle = "logic" *)': Mandar criar o register file usando os elementos l
    (* ramstyle = "logic" *) reg [END_IDX:0] RF [31:0];

    // --> Register file com 3 ports
    // Os 2 ports de leitura são lidos usando lógica combinacional (r_addr1/r_data1, r_addr2/r_data2)
    // As operações de escrita somente ocorrem nas bordas de subida do clock (w_addr/w_data/wr_en)
    // Registrador RF[0] é hardwired em 0
    always @( posedge clk ) begin
        if( wr_en ) begin
            RF[w_addr] <= w_data;
        end
    end

    // --> Operações de leitura dos dados armazenados nos registradores
    // --> Caso 'r_addr1' ou 'r_addr', retornar o valor 0
    assign r_data1 = (r_addr1 != 5'd0) ? RF[r_addr1] : VAL_0;
    assign r_data2 = (r_addr2 != 5'd0) ? RF[r_addr2] : VAL_0;
endmodule
```

---

2 - Módulo `instr_mem`

- **Memória ROM** para armazenar os **PROGRAMAS (*program memory*)**. Utiliza os blocos de memória **M9K**.
  - **Constantes**:
    - `DATA_WIDTH`: Comprimento, em bits, do *array* de dados.
    - `END_IDX`: Índice do último elemento do array com os dados.
  - **Entradas**:
    - `addr`: Endereço de memória onde irá ocorrer uma operação de leitura ou de escrita.
  - **Saídas**:
    - `instr`: Instrução lida no endereço `addr` da memória de programa.

```
module instr_mem #( parameter DATA_WIDTH=32, parameter END_IDX=DATA_WIDTH-1,
                    parameter ADDR_WIDTH=10, parameter HEX_FILE="riscvtest.hex" )
                  ( input  wire [END_IDX:0] addr,
                    output wire [END_IDX:0] instr );
//------------------------------------------------------------------------------------------------------
    parameter N_WORDS = (2**ADDR_WIDTH)-1;
    parameter LAST_ADDR = ADDR_WIDTH-1;

    // --> Matriz com a 'instruction memory'
    // Sobre a expressao '(* ramstyle = "M9K" *)': Mandar implementar a ROM nos blocos de memoria M9K d
```

```
    (* ramstyle = "M9K" *) reg [END_IDX:0] ROM [N_WORDS:0];

    // --> Inserir o conteudo do arquivo .hex no array 'ROM'
      initial begin
            $readmemh(HEX_FILE, ROM);
      end

    // --> Endereco de leitura/escrita com 'ADDR_WIDTH' bits
    wire [LAST_ADDR:0] address_div4;
    assign address_div4 = { 2'b00, addr[LAST_ADDR:2] };

    // --> Atribuir a 'instr' o valor em 'address_div4'
    assign instr = ROM[ address_div4 ];
endmodule
```

---

3 - Módulo `data_mem_single`

- **Memória RAM** para armazenamento de **DADOS (*data memory*)**. Utiliza os blocos de memória **M9K**.
  - **Constantes**:
    - `DATA_WIDTH`: Comprimento, em bits, do *array* de dados.
    - `END_IDX`: Índice do último elemento do array com os dados.
  - **Entradas**:
    - `clk`: Entrada que recebe os pulsos de *clock*.
    - `w_en`: Sinal que habilita a escrita do valor `w_data` no endereço de memória `addr`.
    - `addr`: Endereço de memória onde irá ocorrer uma operação de leitura ou de escrita.
    - `w_data`: *Array* com o valor a ser escrito no endereço `addr` da memória.
  - **Saídas**:
    - `r_data`: *Array* com o valor lido no endereço de memória `addr`.

```
// synopsys translate_off
``timescale 1 ps / 1 ps

// synopsys translate_on
module data_mem_single #( parameter DATA_WIDTH=32, parameter END_IDX=DATA_WIDTH-1, parameter ADDR_WIDTH=
                      ( clk, w_en, addr, w_data, r_data );
//-------------------------------------------------------------------------------------------------
    input                       clk;
    input                       w_en;
    input  [(ADDR_WIDTH-1):0] addr;
    input    [END_IDX:0] w_data;
    output [END_IDX:0] r_data;

``ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
```

```
``endif
    tri1      clk;
    tri0      w_en;
``ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
``endif
//-------------------------------------------------------------------------------------------------------------
    // --> Constante: Numero de words:
    parameter N_WORDS = (2**ADDR_WIDTH)-1;
    parameter LAST_ADDR = ADDR_WIDTH-1;

    // --> Endereco de leitura/escrita com 'ADDR_WIDTH' bits
    wire [LAST_ADDR:0] address_div4;
    assign address_div4 = { 2'b00, addr[LAST_ADDR:2] };

    wire [END_IDX:0] sub_wire0;
    //wire [END_IDX:0] r_data = sub_wire0[(DATA_WIDTH-1):0];
    assign r_data = sub_wire0[END_IDX:0];

    // --> Instanciacao da memoria
    altsyncram  altsyncram_component (
                //.address_a( address_div4 ),
                .address_a( address_div4 ),
                .clock0( clk ),
                .data_a( w_data ),
                .wren_a( w_en ),
                .q_a( sub_wire0 ),
                .aclr0( 1'b0 ),
                .aclr1( 1'b0 ),
                .address_b( 1'b1 ),
                .addressstall_a( 1'b0 ),
                .addressstall_b( 1'b0 ),
                .byteena_a( 1'b1 ),
                .byteena_b( 1'b1 ),
                .clock1( 1'b1 ),
                .clocken0( 1'b1 ),
                .clocken1( 1'b1 ),
                .clocken2( 1'b1 ),
                .clocken3( 1'b1 ),
                .data_b( 1'b1 ),
                .eccstatus(),
                .q_b(  ),
                .rden_a( 1'b1 ),
                .rden_b( 1'b1 ),
                .wren_b( 1'b0 ) );
    // -> Definicao dos parametos
    defparam
        altsyncram_component.clock_enable_input_a = "BYPASS",
        altsyncram_component.clock_enable_output_a = "BYPASS",
        //altsyncram_component.intended_device_family = "Cyclone IV E",                 // Sera utiliz
        altsyncram_component.intended_device_family = "MAX 10",                        // Sera utilizad
        //altsyncram_component.init_file = "RAM_init.mif",                             // Arquivo .mif
        //altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
```

```
        altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=YES,INSTANCE_NAME=1234",
        altsyncram_component.lpm_type = "altsyncram",
        altsyncram_component.ram_block_type = "M9K",                                    // Usar blocos do
        altsyncram_component.numwords_a = N_WORDS,                                      // Usada aqui a co
        altsyncram_component.operation_mode = "SINGLE_PORT",
        altsyncram_component.outdata_aclr_a = "NONE",
        altsyncram_component.outdata_reg_a = "UNREGISTERED",
        //altsyncram_component.power_up_uninitialized = "TRUE",
        altsyncram_component.power_up_uninitialized = "FALSE",
        altsyncram_component.read_during_write_mode_port_a = "NEW_DATA_NO_NBE_READ", // Uma operacao de
        altsyncram_component.widthad_a = ADDR_WIDTH,                                    // Usada aqui a co
        altsyncram_component.width_a = DATA_WIDTH,                                      // Usada aqui a co
        altsyncram_component.width_byteena_a = 1;
endmodule
```