LABORATÓRIO DE ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES I

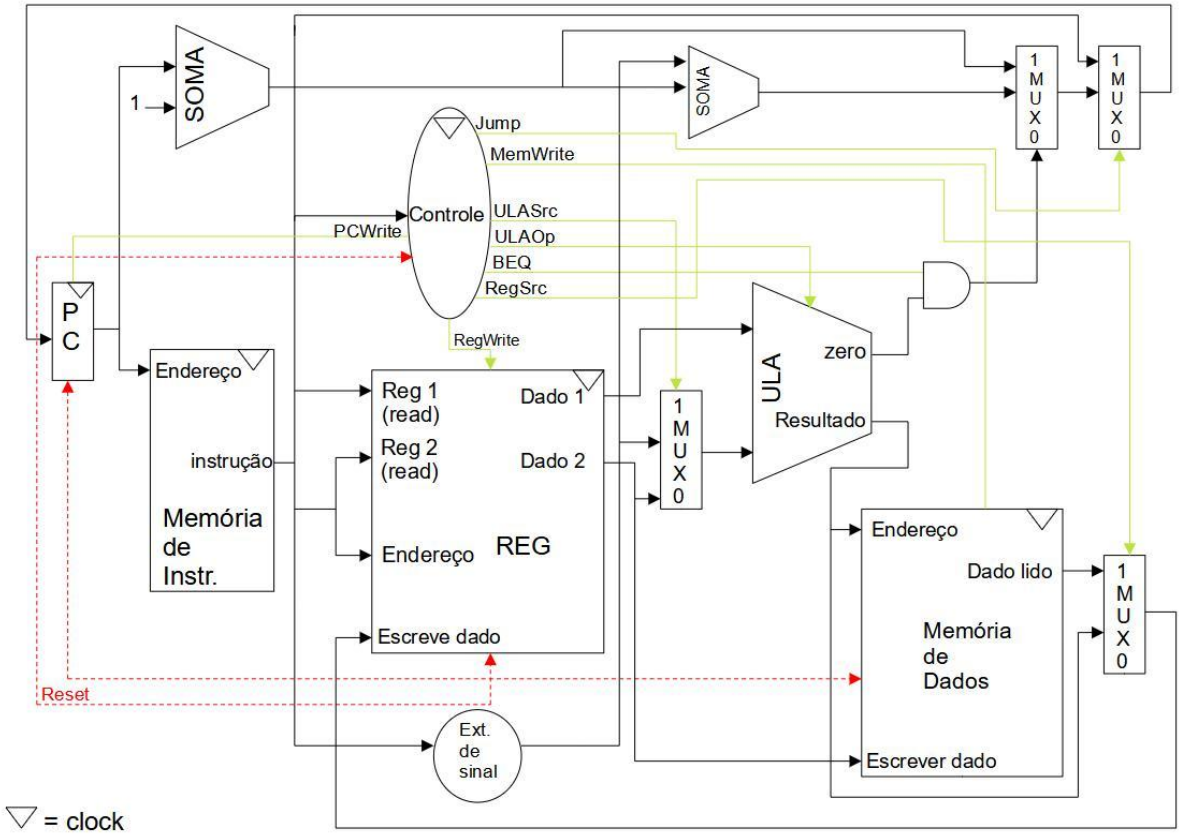
AULA 11: implementação em Verilog dos componentes do nRisc que armazenam estado

Professor: Mateus Felipe Tymburibá Ferreira
Data: 11/08/2021

Aluno: Darmes Araujo Dias

Mudanças no projeto:

Troquei o nome da entrada (Reg1(write)) para endereço e tirei o sinal de controle ReadMem no datapath:



mudança no código do nrisc, porque os jump's e beq's não conseguiam alcançar as instruções:

```
#AULA 7:Projeto de um Processador: implementação do programa-teste no MARS
#Aluno: Darmes Araujo Dias
.data # definindo onde começam os dados do programa
	zero: # Este valor deve ser sempre o endereço 0
	.word 0
	array:
	.word 1,2,3,4,5,6,7,8 # inicializando valores do array
	size:
	.word 8
#Só possuo dois registradores na minha arquitetura, $s1 e $s2
#end do iterator 10
#end da soma 11
#end do valor de array[i] 12
#beq sempre compara os únicos dois registradores
.text # a partir daqui começam as instruções
.globl main
		main:
			la $s2, zero # 00110000
			la $s1, zero # 00100000
		loop:
			la $s1, zero # 00100000
			addi $s1, 10 # 10001010
			lw $s1, 0($s1) # 00000000
			la $s2, size # 00111001
			lw $s2, 0($s2) # 00011000
			beq exit # 10100010
			j continue1 # 11000011
		exit:
				#111XXXXX
		.data
		aux1:
			j loop # 11011000
		continue1:
			la $s2, array # 00110001
			add $s2,$s2,$s1 # 01111000
			lw $s2, 0($s2) # 00011000
			la $s1, zero # 00100000
			addi $s1, 12 # 10001100
			sw $s2, 0($s1) # 01001000
			la $s1, zero # 00100000
			addi $s1, 10 # 10001010
			lw $s1, 0($s1) # 00000000
			j continue2 # 11000010
		aux2:
			j aux1 # 11010101
```

```
        continue2:
        addi $s1, 1 # 10000001
        la $s2, zero # 00110000
        addi $s2, 10 # 10011010
        sw $s1, 0($s2) # 01010000
        la $s2, zero # 00110000
        j continue3 # 11000010
aux3:
        j aux2 # 11011001

continue3:
        addi $s2, 12 # 10011100
        sw $s2, 0($s2) # 01011000
if:
                addi $s2, -2 # 10011110
                la $s1, zero # 00100000
                beq is_even # 10100111
                addi $s1, 1 # 10000001
                beq aux3 # 10111001
                addi $s1,-2 # 10001110
                beq aux3 # 10110111
                j if # 11011001
aux4:
                j aux3 # 11010110

        is_even:
                la $s1, zero # 00100000
        addi $s1, 12 # 10001100
        lw $s1, 0($s1) # 00000000
        la $s2, zero # 00110000
        addi $s2, 11 # 10011011
        lw $s2, 0($s2) # 00011000
        add $s2, $s2,$s1 # 01111000
        la $s1, zero # 00100000
        addi $s1, 11 # 10001011
        sw $s2, 0($s1) # 01001000
        j aux4 # 11010101
```

1) Apresente o código fonte, em Verilog, dos módulos que descrevem os componentes do seu nRisc que armazenam valores.

2) Configure simulações que demonstrem o correto funcionamento de todos os componentes implementados. Apresente o código fonte desse(s) módulo(s) de simulação.

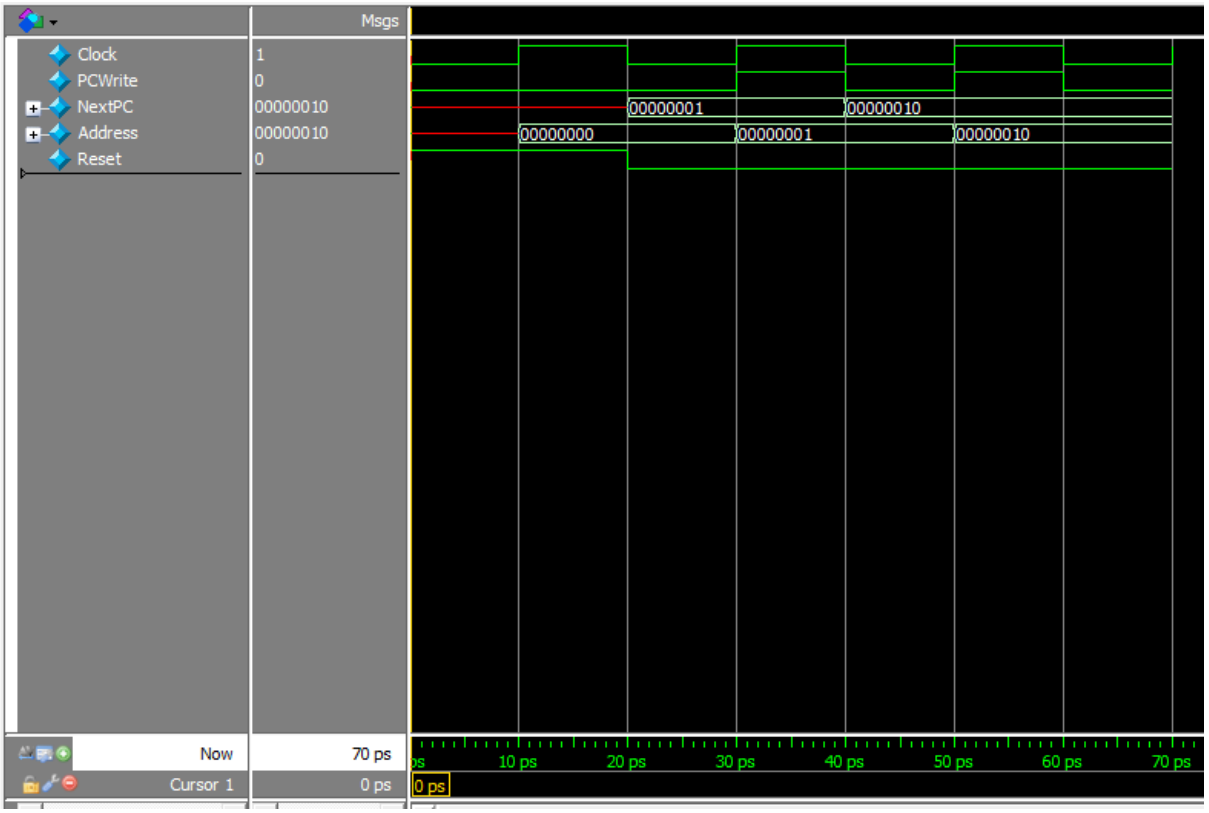
3) Demonstre e explique o funcionamento das simulações, usando fotos da tela (screenshots) da aplicação ModelSim em execução.

➤ PC

```
1   module pc(Clock, PCWrite, Address, NextPC, Reset);
2      input PCWrite, Clock, Reset;
3      input [7:0]NextPC;
4
5      output reg [7:0]Address;
6
7      always @(posedge Clock) begin
8            if(Reset)
9               Address <= 0;
10           else
11              Address <= NextPC;
12
13     end
14
15  endmodule
16
```

```
1   module pc_simulation(a,b);
2      reg PCWrite, Clock, Reset;
3      reg [7:0]NextPC;
4
5      wire [7:0]Address;
6      wire pcwrite, clock, reset;
7      wire [7:0]nextpc;
8
9      input a,b; // só para o quartus me deixar fazer o design do módulo
10
11     assign clock = Clock;
12     assign pcwrite = PCWrite;
13     assign reset = Reset;
14     assign nextpc = NextPC;
15
16     pc pc_aux(clock, pcwrite, Address, nextpc, reset);
17
18     always #10 Clock = ~Clock;
19     initial begin
20        #0 Reset = 1;
21        #0 Clock = 0;
22        #0 PCWrite = 0;
23        #20 Reset = 0;
24        #0 NextPC = 8'b00000001;
25        #10 PCWrite = 1;
26        #10 PCWrite = 0;
27        #0 NextPC = 8'b00000010;
28        #10 PCWrite = 1;
29        #10 PCWrite = 0;
30        #10 $finish;
31     end
32
33  endmodule
34
```

O testbench foi escrito para o teste do módulo PC, ele consiste em resetar meu PC(ele começa no endereço de instrução 0), e depois apontar PC como 1 (2° instrução) e depois ir para o próximo endereço de instrução, passando PC para 2 (3° instrução).

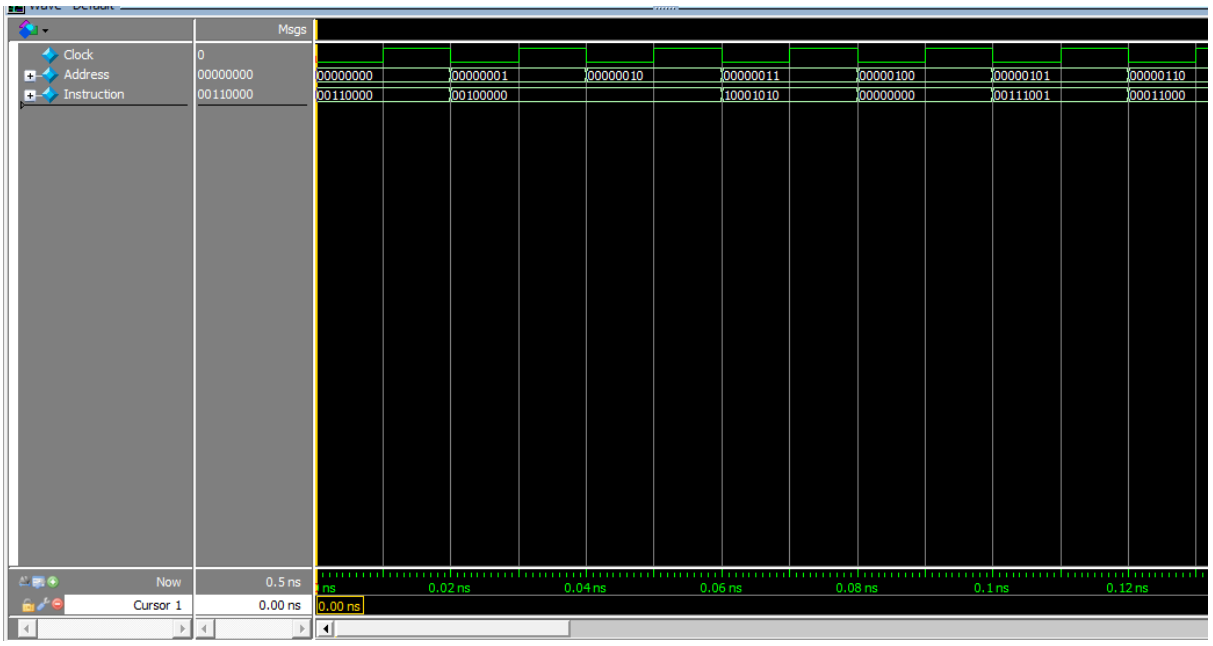


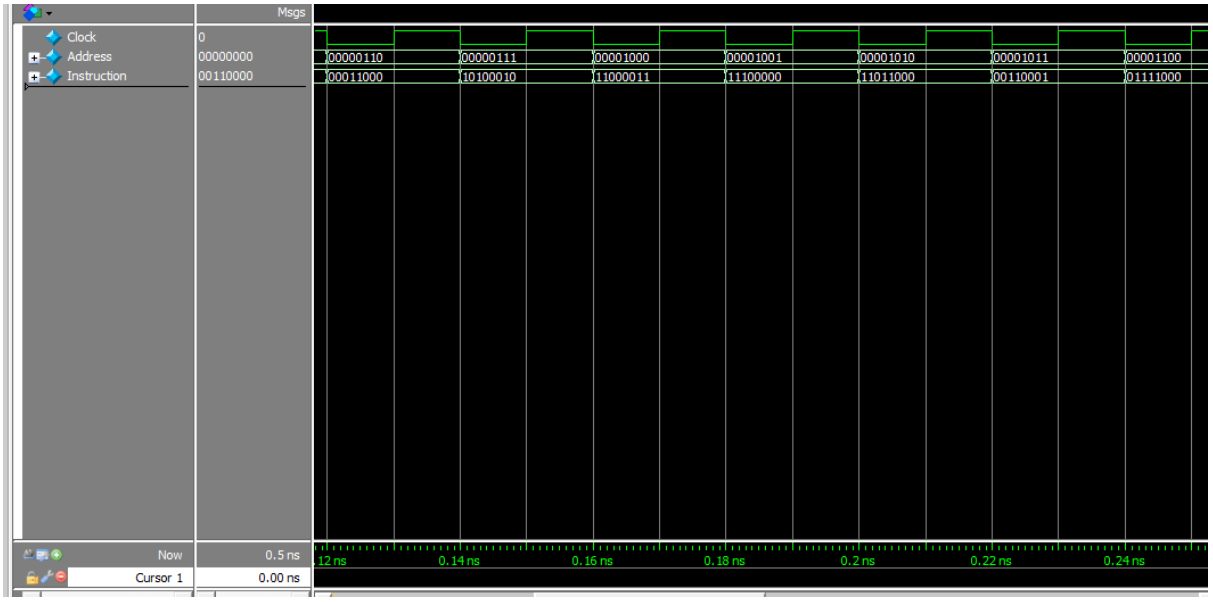
A simulação foi como o esperado.
O wave.do foi salvo na pasta pc_simulation.

➢ Memória de instrução

```
module memory_of_instruction(Clock ,Address, Instruction, Reset, Data_in, WE);
   input Clock, Reset, WE;
   input [7:0]Data_in;
   input [7:0]Address;

   output wire [7:0]Instruction;

   reg [7:0]Memory[63:0];

   assign Instruction = Memory[Address];

   always @(posedge Clock) begin
      if(WE) begin
         Memory[Address] <= Data_in;
      end else begin
         if(Reset) begin
               Memory[Address] <= 0;
         end
      end
   end

   initial begin
      $readmemh("instructionsinhex.hex", Memory);
   end

endmodule
```

```
module memory_of_instruction_simulation(input a, output b, input clock);
   reg Clock, Reset, WE;
   reg [7:0]Address;
   reg [7:0]Data_in;

   //input a,b; //inutil, isso só serve para o quartus me deixar criar o modulo de testbench

   wire [7:0]Instruction;

   //wire clock;
   wire reset,we;
   wire [7:0]address;
   wire [7:0]data_in;

   assign we = WE;
   assign data_in = Data_in;
   assign reset = Reset;
   //assign clock = Clock;
   assign address = Address;

   memory_of_instruction mi(clock,address,Instruction, reset, data_in, we);

   integer i;

   always #10 Clock = ~Clock;
   initial begin
      #0 Clock = 0;
      #0 Address = 0;
      for(i=0;i<64;i=i+1)
         #20 Address = Address + 1;

   end
endmodule
```





Tudo ocorre como o planejado, escrevemos todas as instruções do código nrisc na memória de instruções nas posições corretas.
O wave.do foi salvo na pasta memory_of_instruction_simulation.
E as instruções estão em instructionsinhex.hex também na pasta memory_of_instruction_simulation.

➢ Banco de registradores

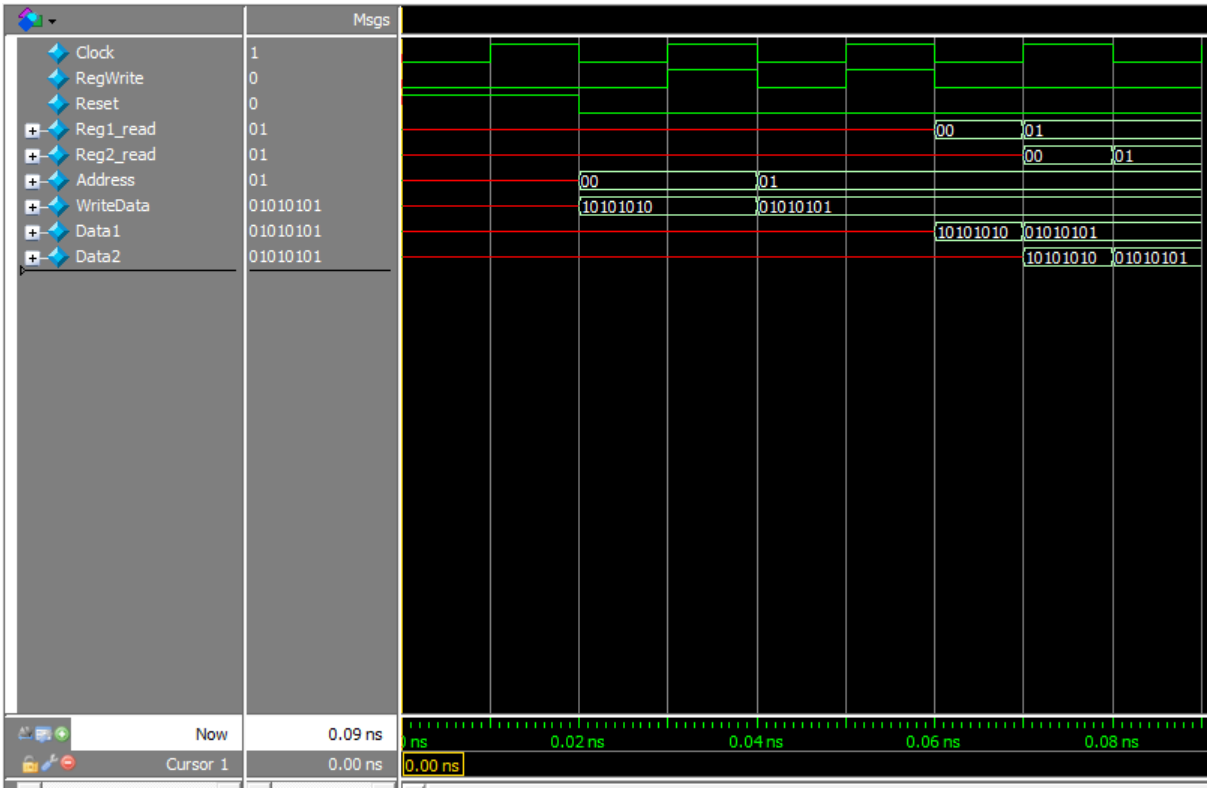
```
1  module registers_bank(Clock, RegWrite,Reg1_read, Reg2_read, Address, WriteData, Data1, Data2, Reset);
2     input Clock, RegWrite, Reset;
3     input [1:0]Reg1_read;
4     input [1:0]Reg2_read;
5     input [1:0]Address; // Reg 1(read) Reg 2(read) Reg 1(write)
6     input [7:0]WriteData; // Escreve dado
7
8     output [7:0]Data1;
9     output [7:0]Data2; // Dado 1 Dado 2
10
11    reg [7:0]Memory[1:0]; // só possuo dois registradores
12
13    assign Data1 = Memory[Reg1_read];
14    assign Data2 = Memory[Reg2_read];
15
16    integer i;
17
18    always @(posedge Clock) begin
19          if(Reset) begin //Quando resetar coloca os regs igual a zero
20             for(i=0;i<2;i=i+1)
21                Memory[i] = 0;
22          end
23          else if(RegWrite)
24             Memory[Address] <= WriteData;
25          else begin //Quando não estiver fazendo nada
26             for(i=0;i<2;i=i+1)
27                Memory[i] <= Memory[i];
28          end
29
30    end
31
32 endmodule
33
```

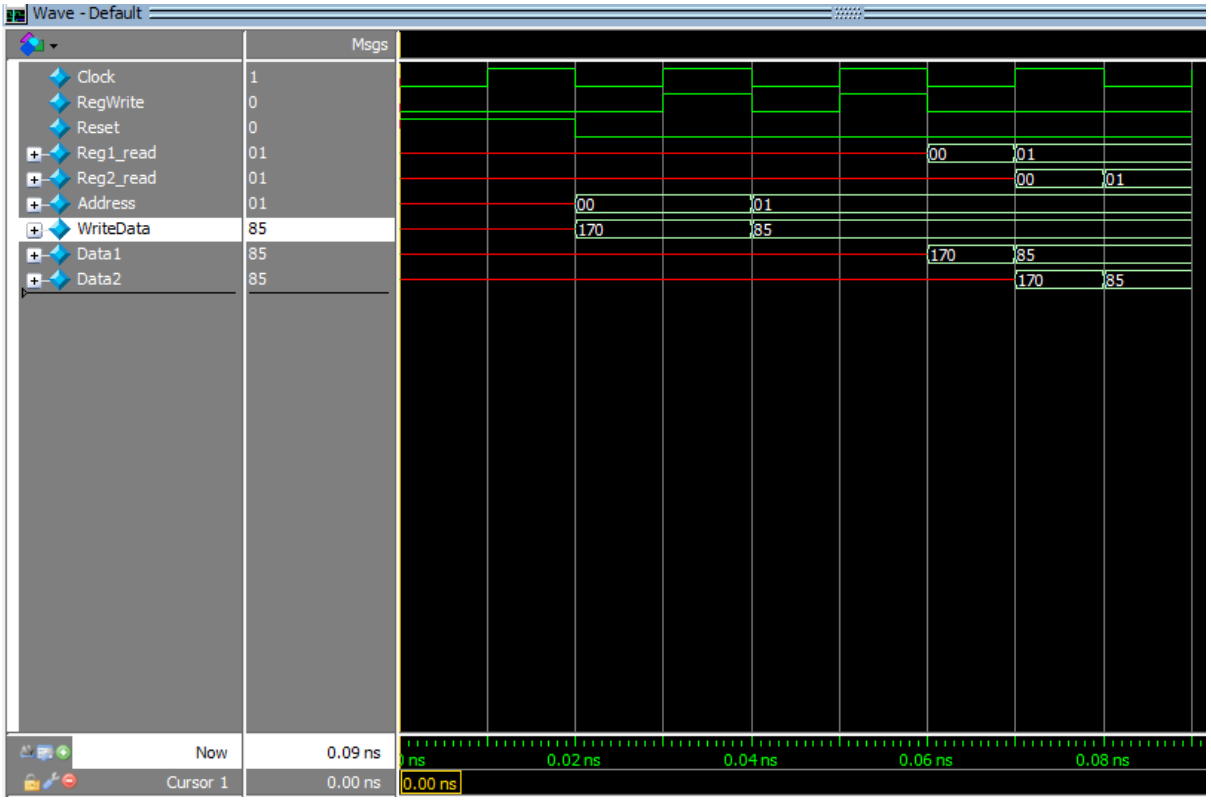
registers_bank_simulation.v

```
1  module registers_bank_simulation(a,b);
2     reg Clock, RegWrite, Reset;
3     reg [1:0]Reg1_read;
4     reg [1:0]Reg2_read;
5     reg [1:0]Address; // Reg 1(read) Reg 2(read) Reg 1(write)
6     reg [7:0]WriteData; // Escreve dado
7
8     wire [7:0]Data1;
9     wire [7:0]Data2; // Dado 1 Dado 2
10    wire clock, regwrite, reset;
11    wire [1:0] reg1_read;
12    wire [1:0]reg2_read;
13    wire [1:0]address;
14    wire [7:0]writedata;
15
16    assign clock = Clock;
17    assign regwrite = RegWrite;
18    assign reset = Reset;
19    assign reg1_read = Reg1_read;
20    assign reg2_read = Reg2_read;
21    assign address = Address;
22    assign writedata = WriteData;
23
24    input a,b;// inutil, só para escrever o testbench e o quartus deixar eu fazer o design do modulo
25    registers_bank rb(clock, regwrite,reg1_read, reg2_read, address, writedata, Data1, Data2, reset);
26    always #10 Clock = ~Clock;
27    initial begin
28       #0 Reset = 1;
29       #0 Clock = 0; // a partir de agora espere 0 e faça isso
30       #0 RegWrite = 0;
31       #20 Reset = 0;
32       #0 Address = 0;
33       #0 WriteData = 8'b10101010;
34       #10 RegWrite = 1;
35       #10 RegWrite = 0;
36       #0 Address = 1;
37       #0 WriteData = 8'b01010101;
38       #10 RegWrite = 1;
39       #10 RegWrite = 0;
40       #0 Reg1_read = 0;
41       #10 Reg1_read = 1;
42       #0 Reg2_read = 0;
43       #10 Reg2_read = 1;
44       #10 $finish;
45    end
46 endmodule
```

Testbench escrito para testar o banco de registradores, ele consiste em resetar o meu banco de registradores, depois ele escreve em WriteData 10101010(170) e após isso Address recebe 0 e RegWrite recebe 1, dessa forma, RegWrite “autoriza” a escrita de

WriteData no registrador 0, e depois RegWrite volta a ser 0. Após isso é escrito 01010101(85) em WriteData, Address recebe 1 e RegWrite recebe 1, dessa maneira, é escrito WriteData no registrador 1, e depois RegWrite volta a ser 0. Depois disso, o testbench escreve nas saídas do banco de registradores esses números(deixando Reg1_read e Reg2_read no HIGH, e depois voltam a ser LOW).





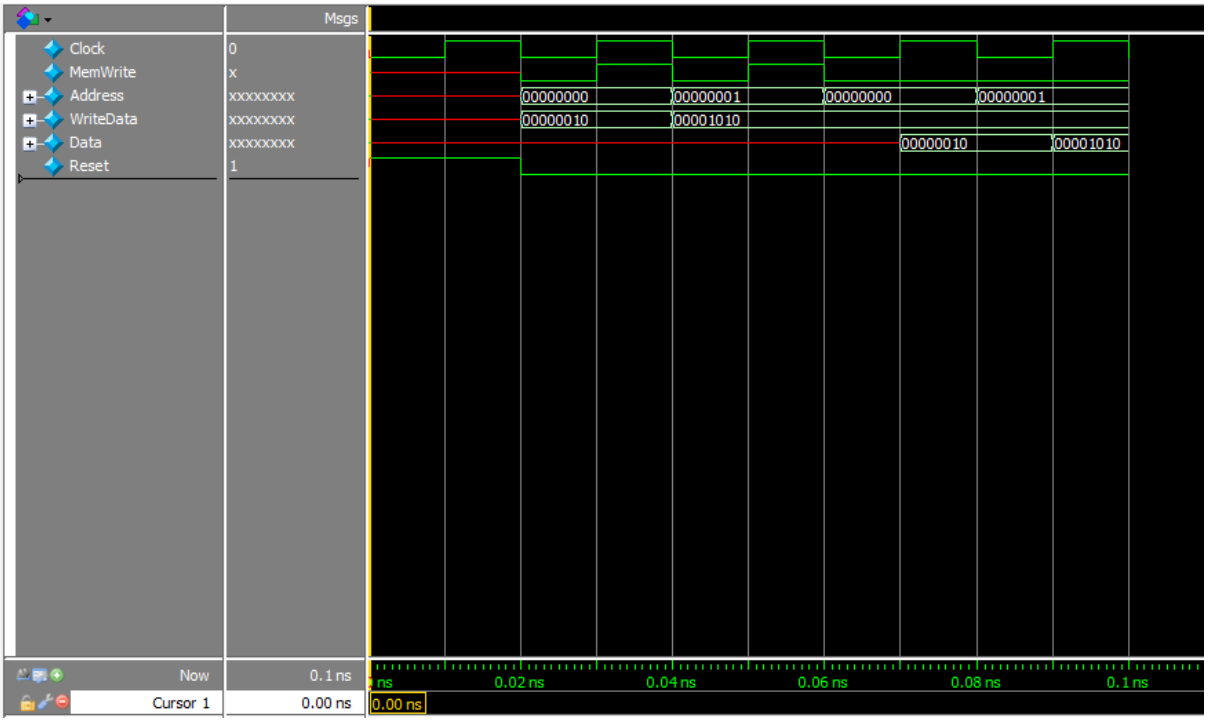
A simulação foi como o esperado.
O wave.do foi salvo na pasta registers_bank_simulation.

➢ Memória de dados

```
1  module data_memory(Clock, Address, WriteData, Data, MemWrite, Reset);
2      input Clock, MemWrite, Reset;
3      input [7:0]Address;
4      input [7:0]WriteData; // Endereço Escrever dado
5
6      output reg[7:0]Data; // Dado lido
7
8      reg [7:0]Memory[0:255];
9
10     integer i;
11
12     always @(posedge Clock) begin
13             if (Reset) begin
14                 Memory[Address] <= 0;
15             end
16             else if(MemWrite)//Escreve na memória
17                 Memory[Address] <= WriteData;
18             else
19                 Data <= Memory[Address];
20     end
21
22 endmodule
23
24
25
26
27
```

```
module data_memory_simulation(a,b);
   reg Clock, MemWrite,  Reset;
   reg [7:0]Address;
   reg [7:0]WriteData; // Endereço Escrever dado

   wire[7:0]Data; // Dado lido
   wire clock, memwrite, reset;
   wire [7:0]address;
   wire [7:0]writedata;

   input a,b; // inutil, estou usando apenas para conseguir fazer o testbench

   assign clock = Clock;
   assign memwrite = MemWrite;

   assign reset = Reset;
   assign address = Address;
   assign writedata = WriteData;

   data_memory dm(clock, address, writedata, Data, memwrite, reset);

   always #10 Clock = ~Clock;
   initial begin
      #0 Reset = 1;
      #0 Clock = 0;
      #20 Reset = 0;
      #0 MemWrite = 0;
      #0 Address = 8'b00000000;
      #0 WriteData = 8'b00000010;
      #10 MemWrite = 1;
      #10 MemWrite = 0;
      #0 Address = 8'b00000001;
      #0 WriteData = 8'b00001010;
      #10 MemWrite = 1;
      #10 MemWrite = 0;
      #0 Address = 8'b00000000;
      #20 Address = 8'b00000001;
      #30 $finish;
   end

endmodule
```

O testbench foi escrito para testar a memória de dados, ele consiste em escrever no 1° endereço (00000000) o número 2(00000010) e escrever no 2° endereço (00000001) o número 10(00001010), e depois colocar a saída da memória de dados com o conteúdo do primeiro endereço de memória e depois do segundo endereço de memória.



A simulação foi como o esperado.

O wave.do foi salvo na pasta data_memory_simulation.