LABORATÓRIO DE ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES I

AULA 14: interligação dos módulos em Verilog do nRisc

Professor: Mateus Felipe Tymburibá Ferreira
Data: 09/09/2021

Aluno: Darmes Araujo Dias

Alterações:

Como minha unidade de controle não funcionava como uma máquina de estados e a maneira como ela estava não estava dando certo(a instrução só funcionava depois da borda positiva, ou seja, estava difícil sincronizar) e não havia maneira de trocar as saídas para wire e fazer assign, eu troquei todas as saídas por uma saída só, de 10 bits, que contém todas as informações necessárias, e que são preenchidas através de um ternário.

```
1   module unidade_de_controle(Opcode, decoded_instruction);
2      input wire[2:0] Opcode;
3      output wire [9:0] decoded_instruction;
4                                          //ulaop[2:0], jump, memwrite, ulasrc, beq, regsrc, regwrite,pcwrite
5      assign decoded_instruction = (Opcode==3'b000) ? 10'b0000010011 :
6                                   (Opcode==3'b001) ? 10'b0010010111 :
7                                   (Opcode==3'b010) ? 10'b0100110101 :
8                                   (Opcode==3'b011) ? 10'b0110000111 :
9                                   (Opcode==3'b100) ? 10'b1000010111 :
10                                  (Opcode==3'b101) ? 10'b1011001001 :
11                                  (Opcode==3'b110) ? 10'b1101001001 :
12                                  (Opcode==3'b111) ? 10'b1110000000 : 10'b1111111111;
13
14     // WE = Write Enable -> MemWrite
15  endmodule
16
```

Retirei os componentes multiplexadores feitos por mim, resolvi usar um assign com ternário, pois estavam dando problema na hora de sincronizar.

1) Apresente o código fonte, em Verilog, do módulo que descreve a interligação dos componentes do seu nRisc.

2) Apresente o código fonte, em Verilog, do módulo que descreve a simulação do funcionamento do seu processador.

3) Demonstre e explique o funcionamento da simulação do programa definido por você no início do projeto usando fotos da tela (screenshots) da aplicação ModelSim em execução.

```
module processador_nrisc(input wire clock, input wire reset, output wire [7:0] instruction_out);
   wire pcwrite, jump, memwrite, ulasrc, beq, regsrc, regwrite_;
   wire [2:0] ulaop;
   wire [7:0] address;
   wire [7:0] instruction;
   wire [7:0] dummy_data_in;

   assign dummy_data_in = 8'b0;

   wire rd, rs, rs2;
   wire [7:0] escrevedado;
   wire [7:0] data1;
   wire [7:0] data2;

   wire [2:0]opcode;
   wire [7:0]dado2;
   wire [7:0] immediate;
   wire zero;
   wire [7:0] resultado;
   wire [7:0] dadolido;
   wire[7:0] address_mais_immediate;
   wire[7:0] address_mais_um;
   //wire [7:0] resultadoMUX1;
   wire [7:0] resultadoMUX2;
   wire beq_and_zero;
   wire [9:0] decoded_instruction;
   assign ulaop = decoded_instruction[9:7];
   assign jump = decoded_instruction[6];
   assign memwrite = decoded_instruction[5];
   assign ulasrc = decoded_instruction[4];
   assign beq = decoded_instruction[3];
   assign regsrc = decoded_instruction[2];
   assign regwrite_ = decoded_instruction[1];
   assign pcwrite = decoded_instruction[0];


   assign rd = instruction[4];
   assign rs = (opcode == 3'b101) ? 1'b0 : instruction[3]; // forçar que os rs sejam diferentes no beq
   assign rs2 = (opcode==3'b011) ? instruction[2] : (opcode==3'b101) ? 1'b1 : 1'b0;
   assign opcode = instruction[7:5];
   assign address_mais_immediate = address + immediate;
   assign address_mais_um = address+1'b1;
   assign beq_and_zero = beq & zero;
   assign instruction_out = instruction; // evitar warning 21074

   pc PC(clock, pcwrite, address , resultadoMUX2 ,reset); // nextPC ou resultado MUX2

   memory_of_instruction MEM_OF_INSTRUCTION(clock, address, instruction, 0, dummy_data_in, 0); // WE é 0 porque eu nunca vou escrever, já vai estar escrito

   registers_bank REG_BANK(clock, regwrite_, rs, rs2, rd, escrevedado, data1, data2, reset);

   extensor_de_sinal EXT_DE_SINAL(instruction, immediate);

   mux MUX_ULA(immediate,data2,ulasrc,dado2);
   ula ULA(ulaop, data1,dado2, zero, resultado);

   data_memory DATA_MEMORY(clock, resultado, data2, dadolido, memwrite, reset);
   mux MUX_DATA_MEM(resultado, dadolido, regsrc, escrevedado);

   unidade_de_controle UNIDADE_DE_CONTROLE(opcode, decoded_instruction);

   //mux MUX1(address_mais_immediate, address_mais_um, jump, resultadoMUX1);
   assign resultadoMUX2 = (jump==1'b1 & opcode == 3'b110) ? address_mais_immediate :
                          (jump==1'b1 & beq_and_zero == 1'b1 & opcode == 3'b101) ? address_mais_immediate : address_mais_um;
   //mux MUX2(resultadoMUX1, address_mais_um, beq_and_zero, resultadoMUX2);


endmodule
```

Observação: na memória de instruções eu deixo write enable (WE) e o reset sempre como zero, pois eu sempre inicializo minha lista de instruções na memória de

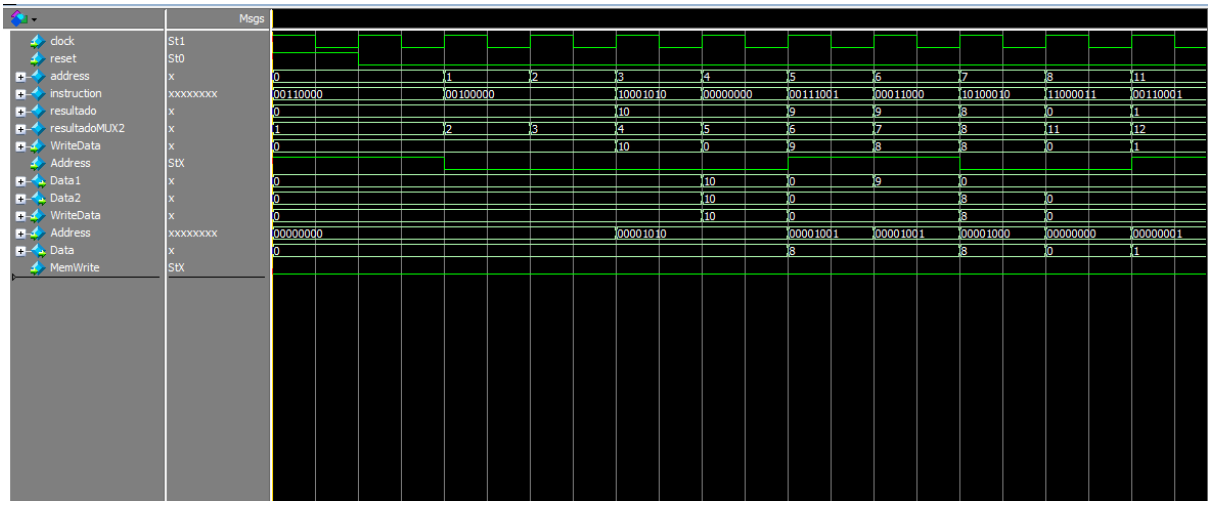
instruções, ou seja, não é necessário escrever nela em momento algum e nem reseta-la, e o dummy_data_in é apenas para preencher o parâmetro, porque não vai ser usado.

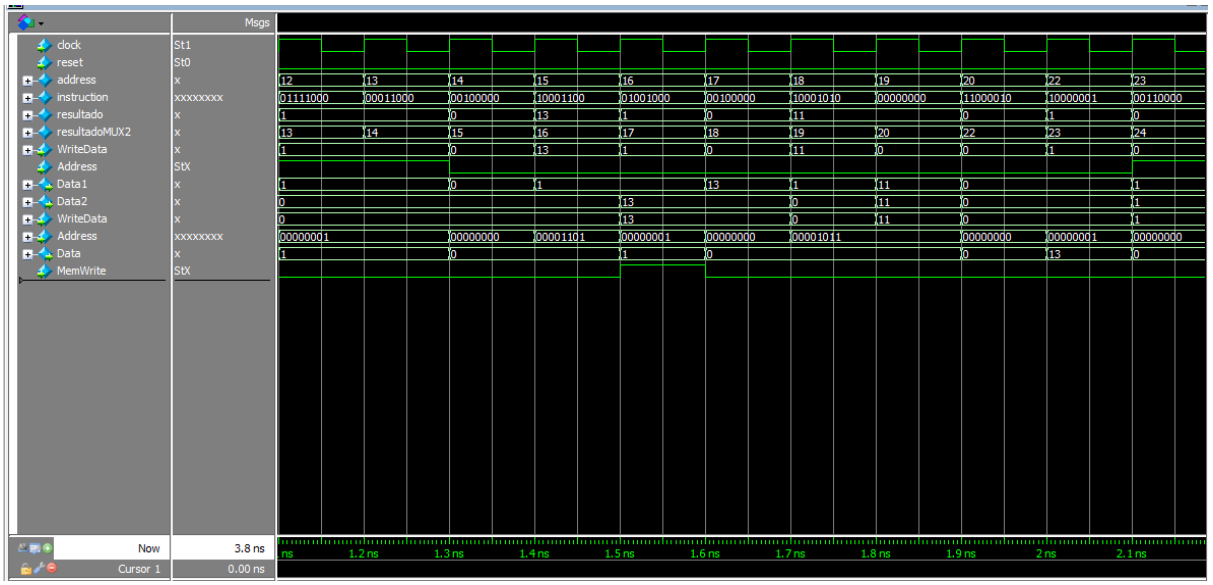
Eu não criei um testbench para este projeto, pois considerando que eu inicializo a memória de instruções com as instruções do meu programa, eu não necessariamente precisava de fazer o testbench para conseguir simular, e o testbench me tomaria mais tempo do que eu possuía. Eu fiz a simulação a mão e disponibilizei abaixo.

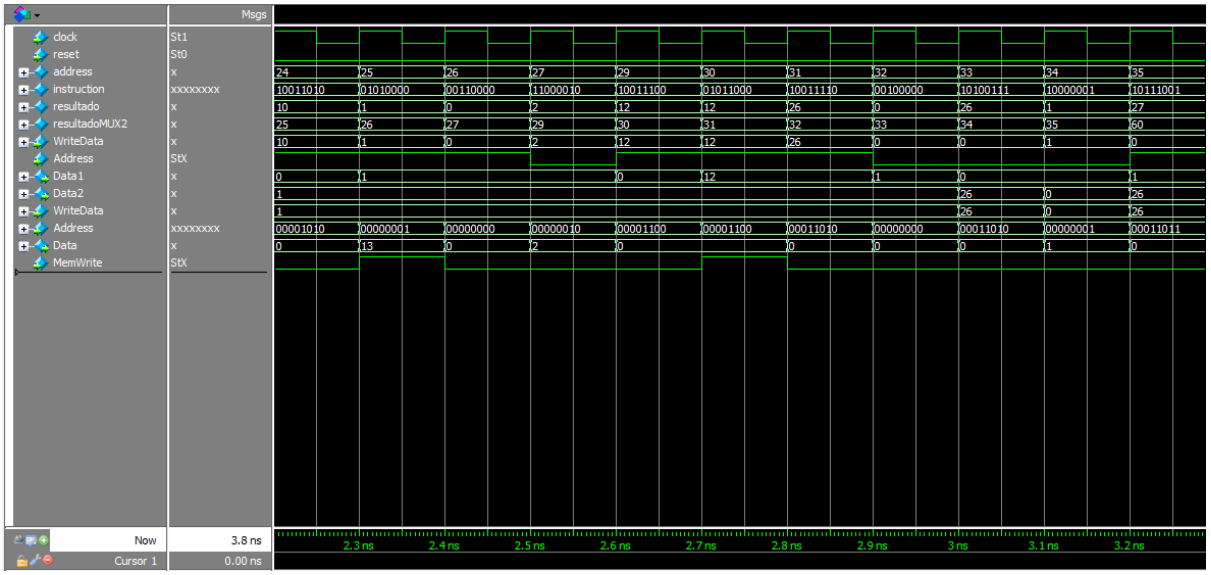
Por linha de comando do simulador modelsim-altera:

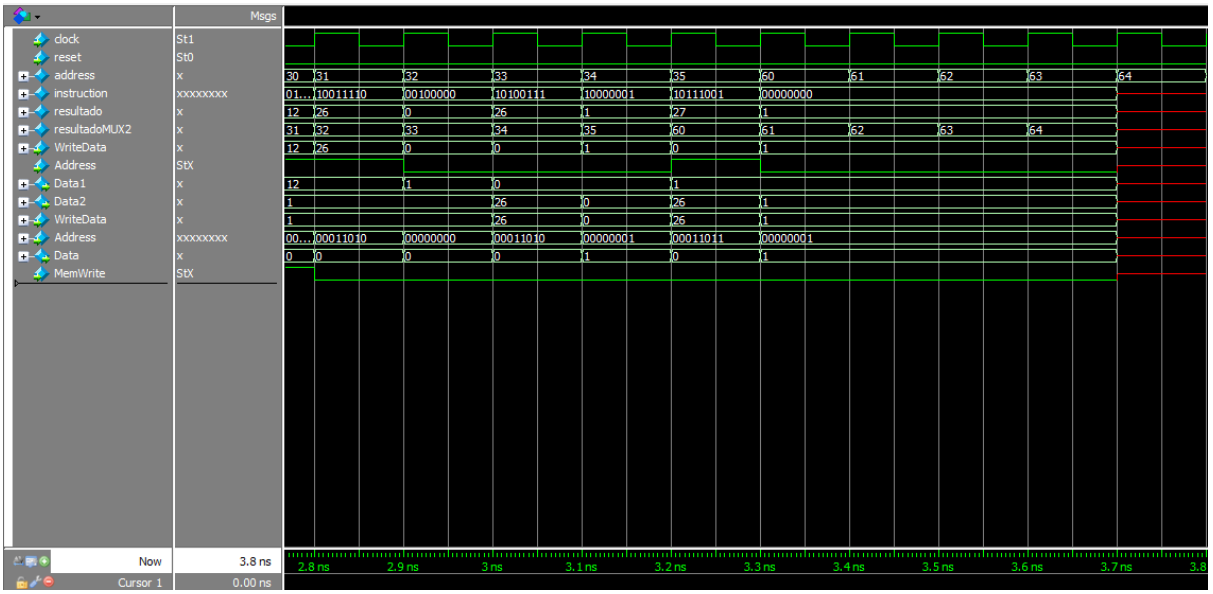
```
add wave -position end  sim:/processador_nrisc/clock
add wave -position end  sim:/processador_nrisc/reset
add wave -position end  sim:/processador_nrisc/address
add wave -position end  sim:/processador_nrisc/instruction
add wave -position end  sim:/processador_nrisc/resultado
add wave -position end  sim:/processador_nrisc/resultadoMUX2
add wave -position end  sim:/processador_nrisc/REG_BANK/WriteData
add wave -position end  sim:/processador_nrisc/REG_BANK/Data1
add wave -position end  sim:/processador_nrisc/REG_BANK/Data2
add wave -position 7  sim:/processador_nrisc/REG_BANK/Address
add wave -position end  sim:/processador_nrisc/DATA_MEMORY/Address
add wave -position 10  sim:/processador_nrisc/DATA_MEMORY/WriteData
add wave -position end  sim:/processador_nrisc/DATA_MEMORY/Data
add wave -position end  sim:/processador_nrisc/DATA_MEMORY/MemWrite
force -freeze sim:/processador_nrisc/clock 1 0, 0 {50 ps} -r 100
force -freeze sim:/processador_nrisc/reset 1 0
run
force -freeze sim:/processador_nrisc/reset 0 0
run
```

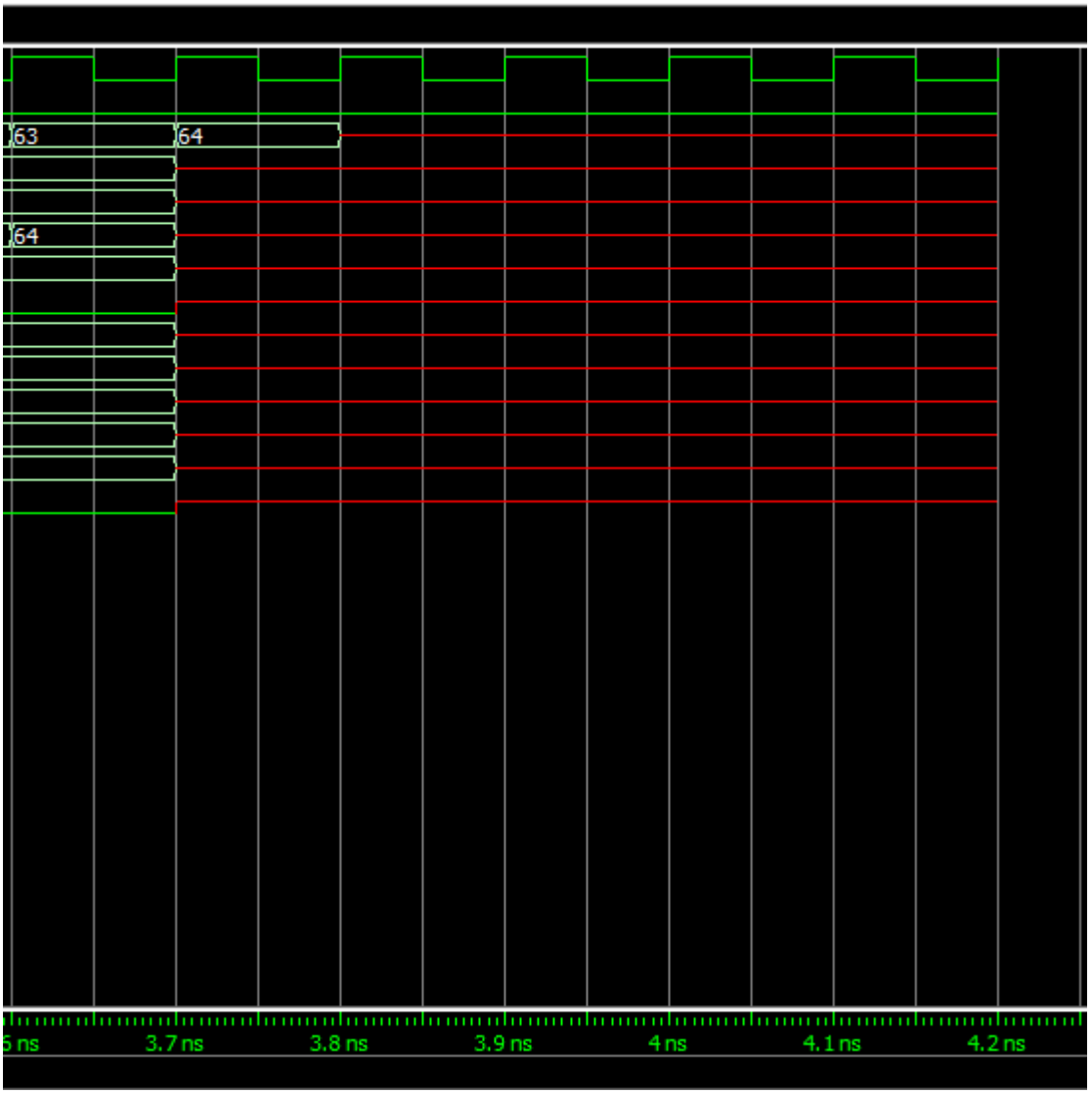
É notório que a inicialização de instruções foi um sucesso, elas estão corretas e ordem.
Nota-se também que nas primeiras instruções ocorreu tudo como deveria ser, os loads não conseguem retornar resultado no mesmo ciclo então retornam no próximo, os saltos ocorrem como deveriam, os stores ocorrem como deveriam. Porém, não consegui fazer saltos negativos, ou seja, retornar o programa para outros passos, e também a soma não dá certo, mas não consegui entender o porquê disso, pois está a frente dos problemas que tenho que enfrentar para compreender.


Msgs
clock
reset
address
instruction
resultado
resultadoMUX2
WriteData
Address
Data1
Data2
WriteData
Address
Data
MemWrite


Msgs
clock
reset
address
instruction
resultado
resultadoMUX2
WriteData
Address
Data1
Data2
WriteData
Address
Data
MemWrite
Now
3.8 ns
Cursor 1
0.00 ns
1.2 ns
1.3 ns
1.4 ns
1.5 ns
1.6 ns
1.7 ns
1.8 ns
1.9 ns
2 ns
2.1 ns


Msgs
clock
reset
address
instruction
resultado
resultadoMUX2
WriteData
Address
Data1
Data2
WriteData
Address
Data
MemWrite
Now
3.8 ns
Cursor 1
0.00 ns
2.3 ns
2.4 ns
2.5 ns
2.6 ns
2.7 ns
2.8 ns
2.9 ns
3 ns
3.1 ns
3.2 ns





A simulação mostrou que o processador não se comportou como esperávamos.