LABORATÓRIO DE ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES I

AULA 12: implementação em Verilog da ULA e da Unidade de Controle do nRisc

Professor: Mateus Felipe Tymburibá Ferreira
Data: 26/08/2021

Aluno: Darmes Araujo Dias

---

1) Apresente o código fonte, em Verilog, dos módulos que implementam a ULA e a Unidade de Controle do seu nRisc.

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.

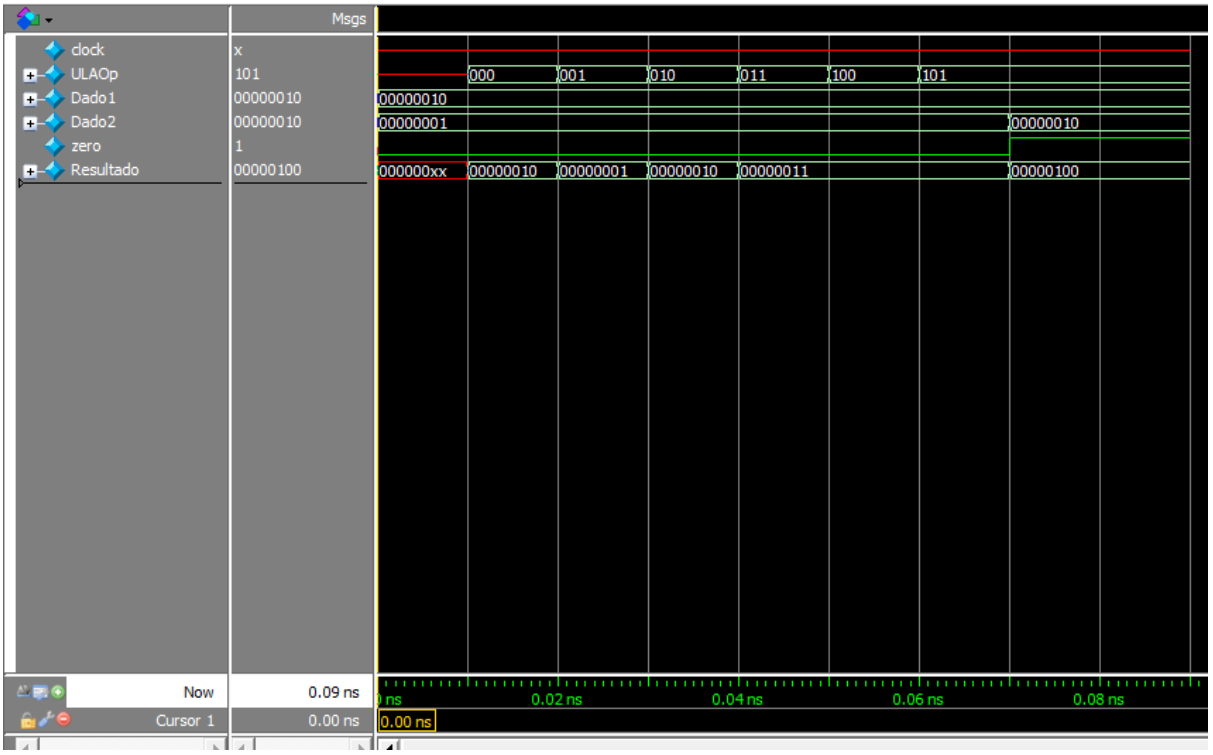
➢ ULA

```
1   module ula(
2   //Dado 2 é a saida do mux, pode ser imediato ou conteudo de um reg
3      input wire [2:0]ULAOp,
4      input wire[7:0]Dado1,
5      input wire[7:0]Dado2,
6      output wire zero,
7      output wire [7:0] Resultado
8   );
9
10
11     assign zero = (Dado1 - Dado2 == 0 & ULAOp == 3'b101) ? 1 : 0;
12
13     assign Resultado =
14        (ULAOp == 3'b000) ? Dado1*Dado2 : // LOAD
15        (ULAOp == 3'b001) ? Dado2: // LA
16        (ULAOp == 3'b010) ? Dado1*Dado2: // STORE
17        (ULAOp == 3'b011 & ULAOp != 3'b101) ? Dado1 + Dado2 : Dado1 + Dado2 ;    // ADD E ADDI
18
19  endmodule
20
```

```
module ula_simulation(input a, output b);
// entrada a e saida b são inúteis, apenas servem para podermos fazer o design do test
   reg [2:0]ULAOp;
   reg [7:0]Dado1;
   reg [7:0]Dado2;
   wire zero;
   wire [7:0] Resultado;

   wire [2:0]ulaop;
   wire [7:0] dado1;
   wire [7:0] dado2;

   assign ulaop = ULAOp;
   assign dado1 = Dado1;
   assign dado2 = Dado2;
   reg clock;
   ula ula1(ulaop,dado1,dado2,zero,Resultado);

   always#10 clock = ~clock;
   initial begin
      #0 Dado1 = 8'b00000010;
      #0 Dado2 = 8'b00000001;
      #10 ULAOp = 3'b000;
      #10 ULAOp = 3'b001;
      #10 ULAOp = 3'b010;
      #10 ULAOp = 3'b011;
      #10 ULAOp = 3'b100;
      #10 ULAOp = 3'b101;
      #10 Dado2 = 8'b00000010;
      #10 ULAOp = 3'b101;
      #10 $finish;
   end
endmodule

```

Observação: eu criei um clock no design do testbench, porque o processador de qualquer forma só irá funcionar na borda positiva do clock.

O testbench foi escrito para simular a funcionalidade da ULA, iniciamos as suas duas entradas com o número 2 (00000010) e 1 (00000001). Com isso, testamos os opcodes das instruções load, load address, stores, add, addi e beq, que são as instruções que necessitam de operações da ULA. Nos opcodes de load, load address e store a ULA calcula o endereço de memória de onde queremos carregar um dado ou queremos guardar o dado e coloca ele na saída Resultado. Já nos opcodes de add e addi, a ULA realiza a soma e coloca o resultado na saída Resultado, sendo possível escrever esse resultado em um registrador ou guardá-lo na memória de dados. E por último, quando testamos o opcode de beq, testamos ele sendo as entradas 2 (00000010) e 1 (00000001), a saída zero fica como 0, e depois testamos com as duas entradas sendo 2 (00000010), e a saída zero fica como 1.



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

➢ Unidade de controle

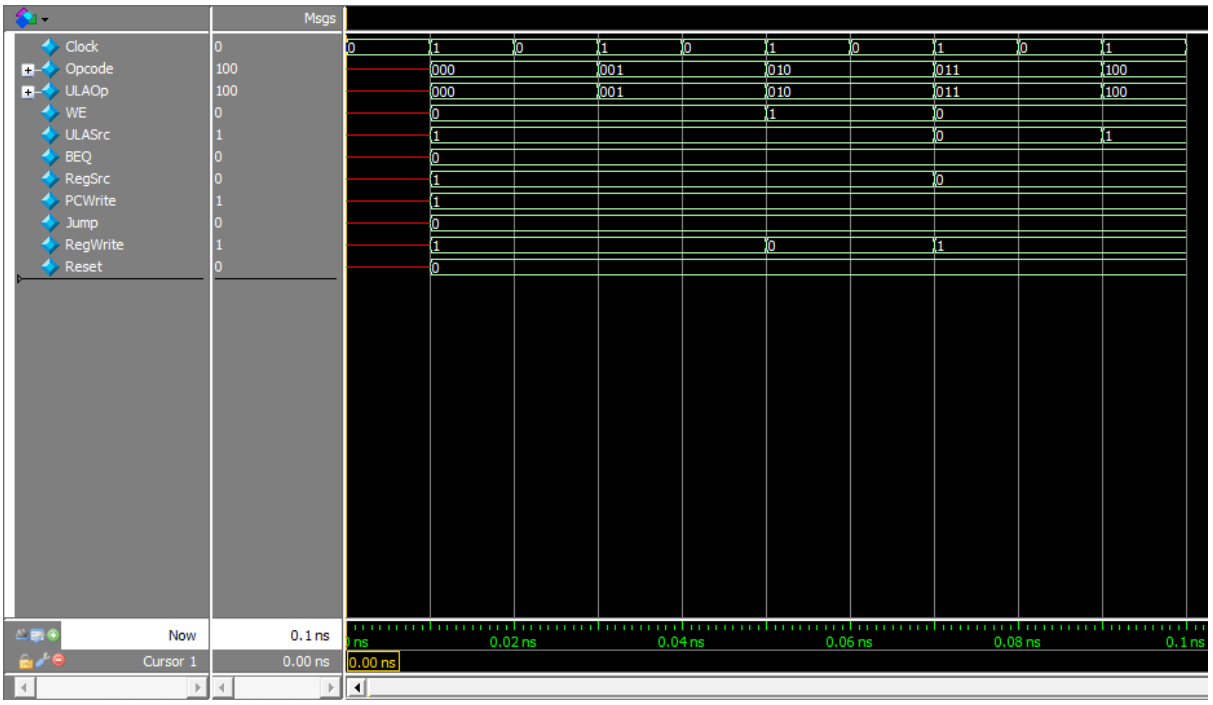
```
module unidade_de_controle(Clock, Jump, Opcode, WE, ULASrc, ULAOp, BEQ, RegSrc, Reset, PCWrite, RegWrite);
   input [2:0] Opcode;
   input Clock;
   output reg [2:0] ULAOp;
   output reg WE, ULASrc, BEQ, RegSrc, PCWrite, Jump, RegWrite, Reset;
   // WE = Write Enable -> MemWrite
   always @(posedge Clock)
      begin
         case(Opcode)
            3'b000: // LOAD
               begin
                  ULAOp <= 3'b000;
                  Jump <= 0;
                  WE <= 0;
                  ULASrc <= 1;
                  BEQ <= 0;
                  RegSrc <= 1;
                  RegWrite <= 1;
                  PCWrite <= 1;
                  Reset <=0;
               end
            3'b001: // LA
               begin
                  ULAOp <= 3'b001;
                  Jump <= 0;
                  WE <= 0;
                  ULASrc <= 1;
                  BEQ <= 0;
                  RegSrc <= 1;
                  RegWrite <= 1;
                  PCWrite <= 1;
               end
            3'b010: // STORE
               begin
                  ULAOp <= 3'b010;
                  Jump <= 0;
                  WE <= 1;
                  ULASrc <= 1;
```

```
39                      BEQ <= 0;
40                      RegSrc <= 1; // X
41                      RegWrite <= 0;
42                      PCWrite <= 1;
43                      Reset <=0;
44                  end
45              3'b011: // ADD
46                  begin
47                      ULAOp <= 3'b011;
48                      Jump <= 0;
49                      WE <= 0;
50                      ULASrc <= 0;
51                      BEQ <= 0;
52                      RegSrc <= 0;
53                      RegWrite <= 1;
54                      PCWrite <= 1;
55                      Reset <=0;
56                  end
57              3'b100: // ADDI
58                  begin
59                      ULAOp <= 3'b100;
60                      Jump <= 0;
61                      WE <= 0;
62                      ULASrc <= 1;
63                      BEQ <= 0;
64                      RegSrc <= 0;
65                      RegWrite <= 1;
66                      PCWrite <= 1;
67                      Reset <=0;
68                  end
69              3'b101: // BEQ
70                  begin
71                      ULAOp <= 3'b101;
72                      Jump <= 0;
73                      WE <= 0;
74                      ULASrc <= 0; // X
75                      BEQ <= 1;
76                      RegSrc <= 0; // X
```

```
77                      RegWrite <= 0;
78                      PCWrite <= 1;
79                      Reset <=0;
80                  end
81              3'b110: // J
82                  begin
83                      ULAOp <= 3'b110;
84                      Jump <= 1;
85                      WE <= 0;
86                      ULASrc <= 0; // X
87                      BEQ <= 1;
88                      RegSrc <= 0; // X
89                      RegWrite <= 0;
90                      PCWrite <= 1;
91                      Reset <=0;
92                  end
93              3'b111: // HALT
94                  begin
95                      ULAOp <= 3'b111;
96                      Jump <= 0;
97                      WE <= 0;
98                      ULASrc <= 0; // X
99                      BEQ <= 0;
100                     RegSrc <= 0;
101                     RegWrite <= 0;
102                     PCWrite <= 0;
103                     Reset <=0;
104                 end
105         endcase
106     end
107
108 endmodule
109
```

```
1  module unidade_de_controle_simulation(input a, output b);
2  //entrada a e saida b são inúteis, só foram criadas para poder programar o design do test
3     reg [2:0] Opcode;
4     reg Clock;
5     wire [2:0] ULAOp;
6     wire WE, ULASrc, BEQ, RegSrc, PCWrite, Jump, RegWrite, Reset;
7     // WE = Write Enable -> MemWrite
8
9     wire [2:0] opcode;
10    wire clock;
11
12    assign opcode = Opcode;
13    assign clock = Clock;
14
15    unidade_de_controle uc(clock,Jump, opcode, WE, ULASrc, ULAOp, BEQ, RegSrc, Reset, PCWrite, RegWrite);
16
17    always #10 Clock = ~Clock;
18    initial begin
19       #0 Clock = 0;
20       #10 Opcode = 3'b000;
21       #20 Opcode = 3'b001;
22       #20 Opcode = 3'b010;
23       #20 Opcode = 3'b011;
24       #20 Opcode = 3'b100;
25       #20 Opcode = 3'b101;
26       #20 Opcode = 3'b110;
27       #20 Opcode = 3'b111;
28       #10 $finish;
29    end
30
31 endmodule
32
```

O testbench foi feito para simular o funcionamento da unidade de controle, que funciona na borda positiva de clock. Ele consiste em receber um opcode e mostrar os sinais da saída da unidade de controle. E comparando com a tabela do relatório 9, está como deveria ser.



Observação: WE = MemWrite

REGSRC -> MEMTOREG

Instruções	Sinais								
	Jump	Mem Write	Read Mem	ULA Src	ULA Op	BEQ	Reg Src	Reg Write	PC Write
load	0	0	1	1	000	0	1	1	1
la	0	0	1	1	001	0	1	1	1
store	0	1	0	1	010	0	X	0	1
add	0	0	0	0	011	0	0	1	1
addi	0	0	0	1	100	0	1	1	1
beq	0	0	1	X	101	1	X	0	1
j	1	0	1	X	110	1	X	0	1
halt	0	0	0	X	111	0	0	0	0

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