**Testbench dos componentes do nRisc:**

### *1) Controle:*

```
`timescale 1ns / 1ps

module tb_Controle;
    reg Op2, Op1, Op0, funct, reset;
    wire ALUSrc, MemToReg, RegWrite, MemRead, MemWrite;
    wire Branch, Jump, NextOp, End, ALUOp1, ALUOp0;

    Controle uut (
        .Op2(Op2), .Op1(Op1), .Op0(Op0), .funct(funct), .reset(reset),
        .ALUSrc(ALUSrc), .MemToReg(MemToReg), .RegWrite(RegWrite),
        .MemRead(MemRead), .MemWrite(MemWrite), .Branch(Branch),
        .Jump(Jump), .NextOp(NextOp), .End(End),
        .ALUOp1(ALUOp1), .ALUOp0(ALUOp0)
    );

    initial begin
        // Cabeçalho formatado
        $display(" Op2 Op1 Op0 | funct | reset | Sinais de Controle");

        // Monitor simplificado sem tempo
        $monitor("  %b   %b   %b  |   %b   |   %b   | %b | %b | %b | %b |
%b | %b | %b | %b | %b | %b | %b",
            Op2, Op1, Op0, funct, reset,
            ALUSrc, MemToReg, RegWrite, MemRead, MemWrite,
            Branch, Jump, NextOp, End, ALUOp1, ALUOp0);

        // Casos de teste
        // Reset ativo
        reset = 1; {Op2, Op1, Op0} = 3'b000; funct = 0; #10;

        // Reset inativo
        reset = 0; #10;

        // Teste das operações
        // ADD e SUB
        {Op2, Op1, Op0} = 3'b000; funct = 0; #10; // ADD
        {Op2, Op1, Op0} = 3'b000; funct = 1; #10; // SUB

        // ADDI
        {Op2, Op1, Op0} = 3'b001; funct = 0; #10;
        {Op2, Op1, Op0} = 3'b001; funct = 1; #10;

        // MULT
        {Op2, Op1, Op0} = 3'b010; funct = 0; #10;
        {Op2, Op1, Op0} = 3'b010; funct = 1; #10;
```

```
        // LD e ST
        {Op2, Op1, Op0} = 3'b011; funct = 0; #10; // LD
        {Op2, Op1, Op0} = 3'b011; funct = 1; #10; // ST

        // JR
        {Op2, Op1, Op0} = 3'b100; funct = 0; #10;
        {Op2, Op1, Op0} = 3'b100; funct = 1; #10;

        // BEQ
        {Op2, Op1, Op0} = 3'b110; funct = 0; #10;
        {Op2, Op1, Op0} = 3'b110; funct = 1; #10;

        // SLT e HLT
        {Op2, Op1, Op0} = 3'b111; funct = 0; #10;
        {Op2, Op1, Op0} = 3'b111; funct = 1; #10; // HLT

        // Teste adicional com reset durante operação
        reset = 1; #10;
        reset = 0; #10;
        $finish;
    end
endmodule
```

| Op2 | Op1 | Op0 | funct | reset | Sinais de Controle | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

### *2) ULA:*

```
`timescale 1ns / 1ps

module ULA_tb;

    // Inputs
    reg [7:0] A;
    reg [7:0] B;
    reg [7:0] reEntrada;
    reg ALUOp1;
    reg ALUOp0;
    reg funct;
    reg RegWrite;

    // Outputs
    wire [7:0] S;
    wire [7:0] reSaida;
    wire zero;
    wire overflow;

    // Instantiate the Unit Under Test (UUT)
    Ula uut (
        .A(A),
        .B(B),
        .reEntrada(reEntrada),
        .ALUOp1(ALUOp1),
        .ALUOp0(ALUOp0),
        .funct(funct),
        .RegWrite(RegWrite),
        .S(S),
        .reSaida(reSaida),
        .zero(zero),
        .overflow(overflow)
    );

    initial begin
        // Initialize Inputs
        A = 0;
        B = 0;
        reEntrada = 0;
        ALUOp1 = 0;
        ALUOp0 = 0;
        funct = 0;
        RegWrite = 0;

        // Wait 100 ns for global reset to finish
        #100;
```

```
        // Test Case 1: ADD operation (ALUOp=00, funct=0, RegWrite=1)
        $display("\n1: add (ALUOp=00, funct=0, RegWrite=1)");
        ALUOp1 = 0; ALUOp0 = 0; funct = 0; RegWrite = 1;
        A = 8'd10; B = 8'd20; reEntrada = 8'hAA;
        #10;
        $display(" A=%d | B=%d | S=%d | zero=%b | reEntrada=%h |
reSaida=%h | overflow=%b", A, B, S, zero, reEntrada, reSaida, overflow);

        // Test Case 2: SUB operation (ALUOp=00, funct=1, RegWrite=1)
        $display("2: sub (ALUOp=00, funct=1, RegWrite=1)");
        ALUOp1 = 0; ALUOp0 = 0; funct = 1; RegWrite = 1;
        A = 8'd30; B = 8'd15; reEntrada = 8'hBB;
        #10;
        $display(" A=%d | B=%d | S=%d | zero=%b | reEntrada=%h |
reSaida=%h | overflow=%b", A, B, S, zero, reEntrada, reSaida, overflow);

        // Test Case 3: RESULT operation (equal values, ALUOp=00,
RegWrite=0)
        $display("3: result (ALUOp=00, RegWrite=0)(equal values)");
        ALUOp1 = 0; ALUOp0 = 0; RegWrite = 0;
        A = 8'd25; B = 8'hCC; reEntrada = 8'hCC;
        #10;
        $display(" A=%d | B=%d | S=%d | zero=%b | reEntrada=%h |
reSaida=%h | overflow=%b", A, B, S, zero, reEntrada, reSaida, overflow);

        // Test Case 4: RESULT operation (different values, ALUOp=00,
RegWrite=0)
        $display("4: result (ALUOp=00, RegWrite=0)(different values)");
        ALUOp1 = 0; ALUOp0 = 0; RegWrite = 0;
        A = 8'd30; B = 8'd25; reEntrada = 8'hDD;
        #10;
        $display(" A=%d | B=%d | S=%d | zero=%b | reEntrada=%h |
reSaida=%h | overflow=%b", A, B, S, zero, reEntrada, reSaida, overflow);

        // Test Case 5: ADDI operation (ALUOp=01)
        $display("5: addi (ALUOp=01)");
        ALUOp1 = 0; ALUOp0 = 1; RegWrite = 1;
        A = 8'd50; B = 8'd25; reEntrada = 8'hEE;
        #10;
        $display(" A=%d | B=%d | S=%d | zero=%b | reEntrada=%h |
reSaida=%h | overflow=%b", A, B, S, zero, reEntrada, reSaida, overflow);

        // Test Case 6: MULT operation (ALUOp=10)
        $display("6: mult (ALUOp=10)");
        ALUOp1 = 1; ALUOp0 = 0; RegWrite = 1;
        A = 8'd10; B = 8'd5; reEntrada = 8'hFF;
        #10;
        $display(" A=%d | B=%d | S=%d | zero=%b | reEntrada=%h |
reSaida=%h | overflow=%b", A, B, S, zero, reEntrada, reSaida, overflow);
```

```
        // Test Case 7: BEQ operation (equal values, ALUOp=11, funct=0)
        $display("7: beq (ALUOp=11, funct=0)(equal values)");
        ALUOp1 = 1; ALUOp0 = 1; funct = 0; RegWrite = 0;
        A = 8'd40; B = 8'd40; reEntrada = 8'h11;
        #10;
      $display("  A=%d | B=%d | S=%d | zero=%b | reEntrada=%h |
reSaida=%h | overflow=%b", A, B, S, zero, reEntrada, reSaida, overflow);

        // Test Case 8: BEQ operation (different values, ALUOp=11,
funct=0)
        $display("8: beq (ALUOp=11, funct=0)(different values)");
        ALUOp1 = 1; ALUOp0 = 1; funct = 0; RegWrite = 0;
        A = 8'd40; B = 8'd41; reEntrada = 8'h22;
        #10;
        $display(" A=%d | B=%d | S=%d | zero=%b | reEntrada=%h |
reSaida=%h | overflow=%b", A, B, S, zero, reEntrada, reSaida, overflow);

        // Test Case 9: SLT operation (A < B, ALUOp=11, funct=1)
        $display("9: slt (ALUOp=11, funct=1)(A < B)");
        ALUOp1 = 1; ALUOp0 = 1; funct = 1; RegWrite = 0;
        A = 8'd10; B = 8'd20; reEntrada = 8'h33;
        #10;
        $display(" A=%d | B=%d | S=%d | zero=%b | reEntrada=%h |
reSaida=%h | overflow=%b", A, B, S, zero, reEntrada, reSaida, overflow);

        // Test Case 10: SLT operation (A >= B, ALUOp=11, funct=1)
        $display("10: slt (ALUOp=11, funct=1)(A >= B)");
        ALUOp1 = 1; ALUOp0 = 1; funct = 1; RegWrite = 0;
        A = 8'd30; B = 8'd20; reEntrada = 8'h44;
        #10;
        $display(" A=%d | B=%d | S=%d | zero=%b | reEntrada=%h |
reSaida=%h | overflow=%b", A, B, S, zero, reEntrada, reSaida, overflow);

        // Test Case 11: Overflow in ADD
        $display("11: Overflow add");
        ALUOp1 = 0; ALUOp0 = 0; funct = 0; RegWrite = 1;
        A = 8'd255; B = 8'd1; reEntrada = 8'h55;
        #10;
        $display(" A=%d | B=%d | S=%d | zero=%b | reEntrada=%h |
reSaida=%h | overflow=%b", A, B, S, zero, reEntrada, reSaida, overflow);

        // Test Case 12: Overflow in MULT
        $display("12: Overflow mult");
        ALUOp1 = 1; ALUOp0 = 0; RegWrite = 1;
        A = 8'd128; B = 8'd2; reEntrada = 8'h66;
        #10;
        $display(" A=%d | B=%d | S=%d | zero=%b | reEntrada=%h |
reSaida=%h | overflow=%b", A, B, S, zero, reEntrada, reSaida, overflow);
```

```
        $finish;
    end
endmodule
1: add (ALUOp=00, funct=0, RegWrite=1)
        A= 10 | B= 20 | S= 30 | zero=0 | reEntrada=aa | reSaida=aa | overflow=0
2: sub (ALUOp=00, funct=1, RegWrite=1)
        A= 30 | B= 15 | S= 15 | zero=0 | reEntrada=bb | reSaida=bb | overflow=1
3: result (ALUOp=00, RegWrite=0)(equal values)
        A= 25 | B=204 | S=  0 | zero=1 | reEntrada=cc | reSaida=fc | overflow=1
4: result (ALUOp=00, RegWrite=0)(different values)
        A= 30 | B= 25 | S=  0 | zero=0 | reEntrada=dd | reSaida=fc | overflow=0
5: addi (ALUOp=01)
        A= 50 | B= 25 | S= 75 | zero=0 | reEntrada=ee | reSaida=ee | overflow=0
6: mult (ALUOp=10)
        A= 10 | B=  5 | S= 50 | zero=0 | reEntrada=ff | reSaida=ff | overflow=0
7: beq (ALUOp=11, funct=0)(equal values)
        A= 40 | B= 40 | S=  0 | zero=0 | reEntrada=11 | reSaida=01 | overflow=1
8: beq (ALUOp=11, funct=0)(different values)
        A= 40 | B= 41 | S=  0 | zero=0 | reEntrada=22 | reSaida=00 | overflow=0
9: slt (ALUOp=11, funct=1)(A < B)
        A= 10 | B= 20 | S=  0 | zero=0 | reEntrada=33 | reSaida=01 | overflow=0
10: slt (ALUOp=11, funct=1)(A >= B)
        A= 30 | B= 20 | S=  0 | zero=0 | reEntrada=44 | reSaida=00 | overflow=1
11: Overflow add
        A=255 | B=  1 | S=  0 | zero=0 | reEntrada=55 | reSaida=55 | overflow=1
12: Overflow mult
        A=128 | B=  2 | S=  0 | zero=0 | reEntrada=66 | reSaida=66 | overflow=1
```

### *3) Memória de dados:*

```
`timescale 1ns / 1ps

module MemoriaData_tb;

    // Inputs
    reg [7:0] endereco;
    reg [7:0] dadoEscrever;
    reg MemRead;
    reg MemWrite;
    reg Clock;

    // Outputs
    wire [7:0] dadoLido;

    // Instantiate the Unit Under Test (UUT)
    MemoriaData uut (
        .endereco(endereco),
        .dadoEscrever(dadoEscrever),
        .MemRead(MemRead),
        .MemWrite(MemWrite),
        .Clock(Clock),
        .dadoLido(dadoLido)
    );

    // Clock generation
    initial begin
        Clock = 0;
        forever #10 Clock = ~Clock;
    end

    initial begin
        // Initialize Inputs
        endereco = 0;
        dadoEscrever = 0;
        MemRead = 0;
        MemWrite = 0;

        // Wait for global reset
        #20;

        // Test 1: Leitura dos dados iniciais
        $display("1: Leitura dos dados iniciais");
        MemRead = 1;
        for (integer i = 0; i < 6; i = i + 1) begin
            endereco = i;
            #20;
            $display("Endereço %d: Dado lido = %b (%s)",
```

```
            i, dadoLido, dadoLido);
end
MemRead = 0;
#20;

// Test 2: Escrita e leitura de novos dados
$display("2: Escrita e leitura de novos dados");
MemWrite = 1;
endereco = 6;
dadoEscrever = 8'b01010111; // 'W'
#20;

endereco = 7;
dadoEscrever = 8'b01001111; // 'O'
#20;

endereco = 8;
dadoEscrever = 8'b01010010; // 'R'
#20;

endereco = 9;
dadoEscrever = 8'b01001100; // 'L'
#20;

endereco = 10;
dadoEscrever = 8'b01000100; // 'D'
#20;

MemWrite = 0;
MemRead = 1;
#20;

$display("Verificando dados escritos:");
for (integer i = 0; i < 11; i = i + 1) begin
    endereco = i;
    #20;
    $display("Endereço %d: Dado lido = %b (%s)",
             i, dadoLido, dadoLido);
end
MemRead = 0;
#20;


// Test 3: Sobreescrita
$display("3: Sobreescrita");
MemWrite = 1;
endereco = 0;
dadoEscrever = 8'b01010111; // 'W'
#20;
```

```
        endereco = 1;
        dadoEscrever = 8'b01001111; // 'O'
        #20;

        endereco = 2;
        dadoEscrever = 8'b01010010; // 'R'
        #20;

        endereco = 3;
        dadoEscrever = 8'b01001100; // 'L'
        #20;

        endereco = 4;
        dadoEscrever = 8'b01000100; // 'D'
        #20;

        MemWrite = 0;
        MemRead = 1;
        #20;

        $display("Verificando dados escritos:");
        for (integer i = 0; i < 11; i = i + 1) begin
            endereco = i;
            #20;
            $display("Endereço %d: Dado lido = %b (%s)",
                     i, dadoLido, dadoLido);
        end
        MemRead = 0;
        #20;

        // Test 4: Verificação de operações simultâneas
        $display("4: Verificação de operações simultâneas");
        endereco = 20;
        dadoEscrever = 8'b01011010; // 'Z'
        MemWrite = 1;
        MemRead = 1; // Leitura e escrita ao mesmo tempo
        #20;
        $display("Tentativa de leitura durante escrita - Dado lido = %b",
dadoLido);

        MemWrite = 0;
        #20;
        $display("Leitura após escrita - Dado lido = %b", dadoLido);
        $finish;
    end

endmodule
```

```
1: Leitura dos dados iniciais
Endereço           0: Dado lido = 01001000 (H)
Endereço           1: Dado lido = 01000101 (E)
Endereço           2: Dado lido = 01001100 (L)
Endereço           3: Dado lido = 01001100 (L)
Endereço           4: Dado lido = 01001111 (O)
Endereço           5: Dado lido = 00000000 ( )
2: Escrita e leitura de novos dados
Verificando dados escritos:
Endereço           0: Dado lido = 01001000 (H)
Endereço           1: Dado lido = 01000101 (E)
Endereço           2: Dado lido = 01001100 (L)
Endereço           3: Dado lido = 01001100 (L)
Endereço           4: Dado lido = 01001111 (O)
Endereço           5: Dado lido = 00000000 ( )
Endereço           6: Dado lido = 01010111 (W)
Endereço           7: Dado lido = 01001111 (O)
Endereço           8: Dado lido = 01010010 (R)
Endereço           9: Dado lido = 01001100 (L)
Endereço          10: Dado lido = 01000100 (D)
3: Sobreescrita
Verificando dados escritos:
Endereço           0: Dado lido = 01010111 (W)
Endereço           1: Dado lido = 01001111 (O)
Endereço           2: Dado lido = 01010010 (R)
Endereço           3: Dado lido = 01001100 (L)
Endereço           4: Dado lido = 01000100 (D)
Endereço           5: Dado lido = 00000000 ( )
Endereço           6: Dado lido = 01010111 (W)
Endereço           7: Dado lido = 01001111 (O)
Endereço           8: Dado lido = 01010010 (R)
Endereço           9: Dado lido = 01001100 (L)
Endereço          10: Dado lido = 01000100 (D)
4: Verificação de operações simultâneas
Tentativa de leitura durante escrita - Dado lido = 01011010
Leitura após escrita - Dado lido = 01011010
```

### *4) Memória de Instruções:*

```
`timescale 1ns / 1ps

module MemoriaInstrucoes_tb;
    // Inputs
    reg [7:0] endereco;

    // Outputs
    wire OpCode2, OpCode1, OpCode0, funct;
    wire [1:0] reg1, reg2;
    wire [2:0] imediato;

    // Instantiate the Unit Under Test (UUT)
    MemoriaInstrucoes uut (
        .endereco(endereco),
        .OpCode2(OpCode2),
        .OpCode1(OpCode1),
        .OpCode0(OpCode0),
        .funct(funct),
        .reg1(reg1),
        .reg2(reg2),
        .imediato(imediato)
    );

    initial begin
        // Initialize Inputs
        endereco = 0;

        // Monitor changes
        $monitor("Endereço=%d, OpCode=%b%b%b, Reg1=%b, Reg2=%b, Funct=%b,
Imediato=%b",
                 endereco, OpCode2, OpCode1, OpCode0, reg1, reg2, funct,
imediato);

        // Test sequence
        #10 endereco = 0;   // sub $c2, $c2
        #10 endereco = 4;   // addi $c0, 3
        #10 endereco = 10;  // ld $c1, $c2
        #10 endereco = 12;  // beq $c1, $c0
        #10 endereco = 16;  // addi $c3, 3
        #10 endereco = 19;  // result $c3, 1
        #10 endereco = 30;  // mult $c0, $c3
        #10 endereco = 33;  // slt $c1, $c0
        #10 endereco = 42;  // nop
        #10 endereco = 54;  // addi $c0, 3
        #10 endereco = 60;  // slt $c1, $c0
        #10 endereco = 78;  // st $c1, $c2
        #10 endereco = 81;  // jr $c0
```

```
        #10 endereco = 83;  // hlt
        #10
        $finish;
    end
endmodule
```

```
Endereço=  0, OpCode=000, Reg1=10, Reg2=10, Funct=1, Imediato=101
Endereço=  4, OpCode=001, Reg1=00, Reg2=01, Funct=1, Imediato=011
Endereço= 10, OpCode=011, Reg1=01, Reg2=10, Funct=0, Imediato=100
Endereço= 12, OpCode=101, Reg1=01, Reg2=00, Funct=0, Imediato=000
Endereço= 16, OpCode=010, Reg1=11, Reg2=11, Funct=0, Imediato=110
Endereço= 19, OpCode=110, Reg1=11, Reg2=00, Funct=1, Imediato=001
Endereço= 30, OpCode=010, Reg1=00, Reg2=11, Funct=0, Imediato=110
Endereço= 33, OpCode=101, Reg1=01, Reg2=00, Funct=1, Imediato=001
Endereço= 42, OpCode=111, Reg1=00, Reg2=00, Funct=0, Imediato=000
Endereço= 54, OpCode=001, Reg1=00, Reg2=01, Funct=1, Imediato=011
Endereço= 60, OpCode=101, Reg1=01, Reg2=00, Funct=1, Imediato=001
Endereço= 78, OpCode=011, Reg1=01, Reg2=10, Funct=1, Imediato=101
Endereço= 81, OpCode=100, Reg1=00, Reg2=00, Funct=0, Imediato=000
Endereço= 83, OpCode=111, Reg1=00, Reg2=00, Funct=1, Imediato=001
```

### *5) Banco de registradores:*

```
module tb_BancoDeRegistradores;
    // Entradas
    reg [7:0] reEntrada;
    reg Clock, Reset, RegWrite;
    reg [1:0] Reg1, Reg2;
    reg [7:0] dadoEscrever;

    // Saídas
    wire [7:0] dado1, dado2, reSaida;

    // Instância do módulo testado
    BancoDeRegistradores uut (
        .reEntrada(reEntrada),
        .Clock(Clock),
        .Reset(Reset),
        .RegWrite(RegWrite),
        .Reg1(Reg1),
        .Reg2(Reg2),
        .dadoEscrever(dadoEscrever),
        .dado1(dado1),
        .dado2(dado2),
        .reSaida(reSaida)
    );

    // Gerador de clock (período de 10 unidades de tempo)
    always #5 Clock = ~Clock;

    initial begin
        // Inicialização
        Clock = 0;
        Reset = 0;
        RegWrite = 0;
        reEntrada = 8'h00;
        dadoEscrever = 8'h00;
        Reg1 = 2'b00;
        Reg2 = 2'b00;

        // Monitoramento
        $display("Time |\tClk\tRst\tRW\tR1\tR2\tDadoW\tReIn\t|
Dado1\tDado2\tReOut");
        $monitor("%3t  |\t%b\t%b\t%b\t%d\t%d\t%h\t%h\t| %h\t%h\t%h",
                $time, Clock, Reset, RegWrite, Reg1, Reg2,
                dadoEscrever, reEntrada, dado1, dado2, reSaida);

        // 1. Teste de Reset
        $display("1: Teste de Reset");
        Reset = 1; #10;
```

```
        Reset = 0; #10;

        // 2. Teste de escrita em todos registradores
        $display("2: Teste de escrita");
        RegWrite = 1;
        reEntrada = 8'h11; Reg1 = 2'd0; dadoEscrever = 8'hAA; #10;
        reEntrada = 8'h22; Reg1 = 2'd1; dadoEscrever = 8'hBB; #10;
        reEntrada = 8'h33; Reg1 = 2'd2; dadoEscrever = 8'hCC; #10;
        reEntrada = 8'h44; Reg1 = 2'd3; dadoEscrever = 8'hDD; #10;
        RegWrite = 0; #10;

        // 3. Teste de leitura de todos os pares
        $display("3: Teste de leitura");
        reEntrada = 8'h55;
        Reg1 = 2'd0; Reg2 = 2'd1; #10;
        Reg1 = 2'd2; Reg2 = 2'd3; #10;
        Reg1 = 2'd0; Reg2 = 2'd3; #10;
        Reg1 = 2'd1; Reg2 = 2'd2; #10;

        // 4. Teste escrita sem RegWrite
        $display("4: Teste escrita sem RegWrite");
        reEntrada = 8'h66;
        RegWrite = 0; Reg1 = 2'd0; dadoEscrever = 8'hFF; #10;
        Reg1 = 2'd0; Reg2 = 2'd0; #10; // Verifica se não escreveu

        // 5. Teste atualização de reEntrada
        $display("5: Teste reEntrada");
        reEntrada = 8'h77; #10;
        reEntrada = 8'h88; #10;

        // 6. Reset final
        $display("6: Reset final");
        Reset = 1; #10;
        Reset = 0; #10;

        $finish;
    end
endmodule
```

| Time | Clk | Rst | RW | R1 | R2 | DadoW | ReIn | Dado1 | Dado2 | ReOut |
|---|---|---|---|---|---|---|---|---|---|---|
| 1: Teste de Reset | | | | | | | | | | |
| 0 | 0 | 1 | 0 | 0 | 0 | 00 | 00 | 00 | 00 | fc |
| 5 | 1 | 1 | 0 | 0 | 0 | 00 | 00 | 00 | 00 | fc |
| 10 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 | 00 | fc |
| 15 | 1 | 0 | 0 | 0 | 0 | 00 | 00 | 00 | 00 | 00 |
| 2: Teste de escrita | | | | | | | | | | |
| 20 | 0 | 0 | 1 | 0 | 0 | aa | 11 | 00 | 00 | 00 |
| 25 | 1 | 0 | 1 | 0 | 0 | aa | 11 | aa | aa | 11 |
| 30 | 0 | 0 | 1 | 1 | 0 | bb | 22 | 00 | aa | 11 |
| 35 | 1 | 0 | 1 | 1 | 0 | bb | 22 | bb | aa | 22 |
| 40 | 0 | 0 | 1 | 2 | 0 | cc | 33 | 00 | aa | 22 |
| 45 | 1 | 0 | 1 | 2 | 0 | cc | 33 | cc | aa | 33 |
| 50 | 0 | 0 | 1 | 3 | 0 | dd | 44 | 00 | aa | 33 |
| 55 | 1 | 0 | 1 | 3 | 0 | dd | 44 | dd | aa | 44 |
| 60 | 0 | 0 | 0 | 3 | 0 | dd | 44 | dd | aa | 44 |
| 65 | 1 | 0 | 0 | 3 | 0 | dd | 44 | dd | aa | 44 |
| 3: Teste de leitura | | | | | | | | | | |
| 70 | 0 | 0 | 0 | 0 | 1 | dd | 55 | aa | bb | 44 |
| 75 | 1 | 0 | 0 | 0 | 1 | dd | 55 | aa | bb | 55 |
| 80 | 0 | 0 | 0 | 2 | 3 | dd | 55 | cc | dd | 55 |
| 80 | 0 | 0 | 0 | 2 | 3 | dd | 55 | cc | dd | 55 |
| 85 | 1 | 0 | 0 | 2 | 3 | dd | 55 | cc | dd | 55 |
| 90 | 0 | 0 | 0 | 0 | 3 | dd | 55 | aa | dd | 55 |
| 95 | 1 | 0 | 0 | 0 | 3 | dd | 55 | aa | dd | 55 |
| 100 | 0 | 0 | 0 | 1 | 2 | dd | 55 | bb | cc | 55 |
| 105 | 1 | 0 | 0 | 1 | 2 | dd | 55 | bb | cc | 55 |
| 4: Teste escrita sem RegWrite | | | | | | | | | | |
| 110 | 0 | 0 | 0 | 0 | 2 | ff | 66 | aa | cc | 55 |
| 115 | 1 | 0 | 0 | 0 | 2 | ff | 66 | aa | cc | 66 |
| 120 | 0 | 0 | 0 | 0 | 0 | ff | 66 | aa | aa | 66 |
| 125 | 1 | 0 | 0 | 0 | 0 | ff | 66 | aa | aa | 66 |
| 5: Teste reEntrada | | | | | | | | | | |
| 130 | 0 | 0 | 0 | 0 | 0 | ff | 77 | aa | aa | 66 |
| 135 | 1 | 0 | 0 | 0 | 0 | ff | 77 | aa | aa | 77 |
| 140 | 0 | 0 | 0 | 0 | 0 | ff | 88 | aa | aa | 77 |
| 145 | 1 | 0 | 0 | 0 | 0 | ff | 88 | aa | aa | 88 |
| 6: Reset final | | | | | | | | | | |
| 150 | 0 | 1 | 0 | 0 | 0 | ff | 88 | 00 | 00 | fc |
| 155 | 1 | 1 | 0 | 0 | 0 | ff | 88 | 00 | 00 | fc |
| 160 | 0 | 0 | 0 | 0 | 0 | ff | 88 | 00 | 00 | fc |
| 165 | 1 | 0 | 0 | 0 | 0 | ff | 88 | 00 | 00 | 88 |

### *6) PC:*

```
module tb_Pc;
    reg [7:0] endereco;
    reg End, Reset, Clock;
    wire [7:0] enderecoSaida;

    Pc uut (.endereco(endereco), .End(End), .Reset(Reset), .Clock(Clock),
.enderecoSaida(enderecoSaida));

    always #5 Clock = ~Clock;

    initial begin
        Clock = 0;
      $monitor("Time=%2t | Clock=%b | Reset=%b | End=%b | endereco=%d |
enderecoSaida=%d",
                 $time, Clock, Reset, End, endereco, enderecoSaida);

        // Teste 1: Reset inicial
        Reset = 1; endereco = 8'd5; End = 0; #10;
        Reset = 0; #10;

        // Teste 2: Atualiza endereço normalmente
        endereco = 8'd10; End = 0; #10;

        // Teste 3: End = 1 → PC não muda
        endereco = 8'd20; End = 1; #10;
        endereco = 8'd30; #10;

        // Teste 4: Reset no final
        Reset = 1; #10;
        Reset = 0; #10;

        $finish;
    end
endmodule
```

```
Time= 0 | Clock=0 | Reset=1 | End=0 | endereco=  5 | enderecoSaida=  0
Time= 5 | Clock=1 | Reset=1 | End=0 | endereco=  5 | enderecoSaida=  0
Time=10 | Clock=0 | Reset=0 | End=0 | endereco=  5 | enderecoSaida=  0
Time=15 | Clock=1 | Reset=0 | End=0 | endereco=  5 | enderecoSaida=  5
Time=20 | Clock=0 | Reset=0 | End=0 | endereco= 10 | enderecoSaida=  5
Time=25 | Clock=1 | Reset=0 | End=0 | endereco= 10 | enderecoSaida= 10
Time=30 | Clock=0 | Reset=0 | End=1 | endereco= 20 | enderecoSaida= 10
Time=35 | Clock=1 | Reset=0 | End=1 | endereco= 20 | enderecoSaida= 10
Time=40 | Clock=0 | Reset=0 | End=1 | endereco= 30 | enderecoSaida= 10
Time=45 | Clock=1 | Reset=0 | End=1 | endereco= 30 | enderecoSaida= 10
Time=50 | Clock=0 | Reset=1 | End=1 | endereco= 30 | enderecoSaida=  0
Time=55 | Clock=1 | Reset=1 | End=1 | endereco= 30 | enderecoSaida=  0
Time=60 | Clock=0 | Reset=0 | End=1 | endereco= 30 | enderecoSaida=  0
Time=65 | Clock=1 | Reset=0 | End=1 | endereco= 30 | enderecoSaida=  0
```

**Testbench do nRisc:**

```
module TestBench;

reg Clock, Reset;
wire [7:0] PC_value;
wire [7:0] Instruction;
wire [7:0] Reg0, Reg1, Reg2, Reg3, RegRE;
wire [7:0] MemData0, MemData1, MemData2, MemData3, MemData4;
wire MemRead, MemWrite, RegWrite;

// Instância do processador
Processador proc(
    .Clock(Clock),
    .Reset(Reset)
);

// Acessando sinais internos do processador
assign PC_value = proc.enderecoPC;
assign Instruction = proc.memInst_.memoria[proc.enderecoPC];
assign Reg0 = proc.bancoReg_.registradores[0];
assign Reg1 = proc.bancoReg_.registradores[1];
assign Reg2 = proc.bancoReg_.registradores[2];
assign Reg3 = proc.bancoReg_.registradores[3];
assign RegRE = proc.bancoReg_.reSaida;
assign MemRead = proc.MemRead;
assign MemWrite = proc.MemWrite;
assign RegWrite = proc.RegWrite;


// Acessando sinais de controle e status
wire ALUOp1 = proc.ALUOp1;
wire ALUOp0 = proc.ALUOp0;
wire zero = proc.zero;
wire Branch = proc.Branch;
wire result = proc.ula_.result;

// Acessando memória de dados
assign MemData0 = proc.memData_.memoria[0];
assign MemData1 = proc.memData_.memoria[1];
assign MemData2 = proc.memData_.memoria[2];
assign MemData3 = proc.memData_.memoria[3];
assign MemData4 = proc.memData_.memoria[4];

// Gerador de clock
initial begin
    Clock = 0;
    forever #5 Clock = ~Clock;
end
```

```
// Tarefa para decodificar instruções
function [8*20:1] decode_instruction;
    input [7:0] instr;
    begin
        casez(instr[7:5])
            3'b000: begin // sub ou add
                if(instr[0] == 0)
                    decode_instruction = "add";
                else
                    decode_instruction = "sub";
                decode_instruction = {decode_instruction, " $c",
8'(instr[4:3]+48), ", $c", 8'(instr[2:1]+48)};
            end
            3'b001: begin // addi
                decode_instruction = {"addi $c", 8'(instr[4:3]+48), ", ",
8'(instr[2:0])};
            end
            3'b010: begin // mult
                decode_instruction = {"mult $c", 8'(instr[4:3]+48), ",
$c", 8'(instr[2:1]+48)};
            end
            3'b011: begin // ld ou st
                if(instr[0] == 0)
                    decode_instruction = "ld";
                else
                    decode_instruction = "st";
                decode_instruction = {decode_instruction, " $c",
8'(instr[4:3]+48), ", $c", 8'(instr[2:1]+48)};
            end
            3'b100: begin // jr
                decode_instruction = {"jr $c", 8'(instr[4:3]+48)};
            end
            3'b101: begin // beq ou slt
                if(instr[0] == 0)
                    decode_instruction = "beq";
                else
                    decode_instruction = "slt";
                decode_instruction = {decode_instruction, " $c",
8'(instr[4:3]+48), ", $c", 8'(instr[2:1]+48)};
            end
            3'b110: begin // result
                decode_instruction = {"result $c", 8'(instr[4:3]+48), ",
", 8'(instr[0])};
            end
            3'b111: begin // nop ou hlt
                if(instr[0] == 0)
                    decode_instruction = "nop";
                else
```

```
                    decode_instruction = "hlt";
            end
        endcase
    end
endfunction

// Tarefa para mostrar estado da memória de dados
task show_data_memory;
    input [7:0] addr_start;
    input [7:0] addr_end;
    integer i;
    begin
        $display("Memoria de dados [%0d:%0d]:", addr_start, addr_end);
        for(i = addr_start; i <= addr_end; i = i + 1) begin
            $display("  [%0d] = %8b (%0d / '%c')",
                     i,
                     proc.memData_.memoria[i],
                     proc.memData_.memoria[i],
                     proc.memData_.memoria[i],
                     (proc.memData_.memoria[i] >= 32 &&
proc.memData_.memoria[i] <= 126) ? proc.memData_.memoria[i] : " ");
        end
    end
endtask

// Execução do teste
initial begin
    $display("Cifra = 5");
    // Mostrar conteúdo inicial da memória de dados
    $display("1: Estado Inicial");
    show_data_memory(0, 4);

    // Reset inicial
    Reset = 1;
    #10 Reset = 0;

    $display("2: Execucao do Programa");
    $display("PC  | Instrucao | Assembly      |  c0     c1     c2     c3
re");

    // Monitorar execução
    forever begin
        // Esperar próximo ciclo de clock
        @(posedge Clock);

        // Mostrar estado atual
        $display("%3d | %8b  | %-14s| %3d    %3d    %3d    %3d    %3h",
                 PC_value,
                 Instruction,
```

```
                decode_instruction(Instruction),
                Reg0, Reg1, Reg2, Reg3, RegRE);

        // Parar quando encontrar instrução HLT
        if (Instruction == 8'b11100001) begin
           $display("3: Programa Finalizado");
        // Mostrar conteúdo final da memória de dados
        show_data_memory(0, 4);

        $finish;
        end
    end
end

endmodule
```

```
Cifra = 5
1: Estado Inicial
Memoria de dados [0:4]:
  [0] = 01001000 (72 / 'H') 72
  [1] = 01000101 (69 / 'E') 69
  [2] = 01001100 (76 / 'L') 76
  [3] = 01001100 (76 / 'L') 76
  [4] = 01001111 (79 / 'O') 79
2: Execucao do Programa
PC  | Instrucao | Assembly      |  c0    c1    c2    c3    re
  0 | 00010101  | sub $c2, $c2  |   0     0     0     0    fc
  1 | 00000001  | sub $c0, $c0  |   0     0     0     0    fc
  2 | 00011111  | sub $c3, $c3  |   0     0     0     0    fc
  3 | 11100000  | nop           |   0     0     0     0    fc
  4 | 00100011  | addi $c0, □   |   0     0     0     0    fc
  5 | 00100010  | addi $c0, □   |   3     0     0     0    fc
  6 | 00100000  | addi $c0,     |   5     0     0     0    fc
  7 | 00100000  | addi $c0,     |   5     0     0     0    fc
  8 | 00100000  | addi $c0,     |   5     0     0     0    fc
  9 | 11100000  | nop           |   5     0     0     0    fc
 10 | 01101100  | ld $c1, $c2   |   5     0     0     0    fc
 11 | 00001000  | add $c1, $c0  |   5    72     0     0    fc
 12 | 10101000  | beq $c1, $c0  |   5    77     0     0    fc
 13 | 00000001  | sub $c0, $c0  |   5    77     0     0    00
 14 | 11100000  | nop           |   0    77     0     0    00
 15 | 00111011  | addi $c3, □   |   0    77     0     0    00
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 16 | 01011110 | mult $c3, $c3 | 0 | 77 | 0 | 3 | 00 |
| 17 | 01011110 | mult $c3, $c3 | 0 | 77 | 0 | 9 | 00 |
| 18 | 00111001 | addi $c3, □ | 0 | 77 | 0 | 81 | 00 |
| 19 | 11011001 | result $c3, □ | 0 | 77 | 0 | 82 | 00 |
| 20 | 00000001 | sub $c0, $c0 | 0 | 77 | 0 | 82 | fc |
| 21 | 00011111 | sub $c3, $c3 | 0 | 77 | 0 | 82 | fc |
| 22 | 11100000 | nop | 0 | 77 | 0 | 0 | fc |
| 23 | 00100011 | addi $c0, □ | 0 | 77 | 0 | 0 | fc |
| 24 | 00100011 | addi $c0, □ | 3 | 77 | 0 | 0 | fc |
| 25 | 00100011 | addi $c0, □ | 6 | 77 | 0 | 0 | fc |
| 26 | 00100001 | addi $c0, □ | 9 | 77 | 0 | 0 | fc |
| 27 | 00111011 | addi $c3, □ | 10 | 77 | 0 | 0 | fc |
| 28 | 00111011 | addi $c3, □ | 10 | 77 | 0 | 3 | fc |
| 29 | 00111011 | addi $c3, □ | 10 | 77 | 0 | 6 | fc |
| 30 | 01000110 | mult $c0, $c3 | 10 | 77 | 0 | 9 | fc |
| 31 | 00100001 | addi $c0, □ | 90 | 77 | 0 | 9 | fc |
| 32 | 00011111 | sub $c3, $c3 | 91 | 77 | 0 | 9 | fc |
| 33 | 10101001 | slt $c1, $c0 | 91 | 77 | 0 | 0 | fc |
| 34 | 00000001 | sub $c0, $c0 | 91 | 77 | 0 | 0 | 01 |
| 35 | 00011111 | sub $c3, $c3 | 0 | 77 | 0 | 0 | 01 |
| 36 | 00111011 | addi $c3, □ | 0 | 77 | 0 | 0 | 01 |
| 37 | 00000110 | add $c0, $c3 | 0 | 77 | 0 | 3 | 01 |
| 38 | 01011000 | mult $c3, $c0 | 3 | 77 | 0 | 3 | 01 |
| 39 | 01011000 | mult $c3, $c0 | 3 | 77 | 0 | 9 | 01 |
| 40 | 00011110 | add $c3, $c3 | 3 | 77 | 0 | 27 | 01 |
| 41 | 11011001 | result $c3, □ | 3 | 77 | 0 | 54 | 01 |
| 54 | 00100011 | addi $c0, □ | 3 | 77 | 0 | 54 | fc |
| 55 | 00100001 | addi $c0, □ | 6 | 77 | 0 | 54 | fc |
| 56 | 01000000 | mult $c0, $c0 | 7 | 77 | 0 | 54 | fc |
| 57 | 01000000 | mult $c0, $c0 | 49 | 77 | 0 | 54 | fc |
| 58 | 00100001 | addi $c0, □ | 97 | 77 | 0 | 54 | fc |
| 59 | 00011111 | sub $c3, $c3 | 98 | 77 | 0 | 54 | fc |
| 60 | 10101001 | slt $c1, $c0 | 98 | 77 | 0 | 0 | fc |
| 61 | 00000001 | sub $c0, $c0 | 98 | 77 | 0 | 0 | 01 |
| 62 | 11100000 | nop | 0 | 77 | 0 | 0 | 01 |
| 63 | 00111011 | addi $c3, □ | 0 | 77 | 0 | 0 | 01 |
| 64 | 01011110 | mult $c3, $c3 | 0 | 77 | 0 | 3 | 01 |
| 65 | 01011110 | mult $c3, $c3 | 0 | 77 | 0 | 9 | 01 |
| 66 | 00111101 | addi $c3, □ | 0 | 77 | 0 | 81 | 01 |
| 67 | 11011001 | result $c3, □ | 0 | 77 | 0 | 78 | 01 |
| 78 | 01101101 | st $c1, $c2 | 0 | 77 | 0 | 78 | fc |
| 79 | 00110001 | addi $c2, □ | 0 | 77 | 0 | 78 | fc |
| 80 | 00100001 | addi $c0, □ | 0 | 77 | 1 | 78 | fc |
| 81 | 10000000 | jr $c0 | 1 | 77 | 1 | 78 | fc |
| 1 | 00000001 | sub $c0, $c0 | 1 | 77 | 1 | 78 | fc |
| 2 | 00011111 | sub $c3, $c3 | 0 | 77 | 1 | 78 | fc |
| 3 | 11100000 | nop | 0 | 77 | 1 | 0 | fc |
| 4 | 00100011 | addi $c0, □ | 0 | 77 | 1 | 0 | fc |
| 5 | 00100010 | addi $c0, □ | 3 | 77 | 1 | 0 | fc |
| 6 | 00100000 | addi $c0, | 5 | 77 | 1 | 0 | fc |
| 7 | 00100000 | addi $c0, | 5 | 77 | 1 | 0 | fc |
| 8 | 00100000 | addi $c0, | 5 | 77 | 1 | 0 | fc |
| 9 | 11100000 | nop | 5 | 77 | 1 | 0 | fc |
| 10 | 01101100 | ld $c1, $c2 | 5 | 77 | 1 | 0 | fc |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 11 | 00001000 | add $c1, $c0 | 5 | 69 | 1 | 0 | fc |
| 12 | 10101000 | beq $c1, $c0 | 5 | 74 | 1 | 0 | fc |
| 13 | 00000001 | sub $c0, $c0 | 5 | 74 | 1 | 0 | 00 |
| 14 | 11100000 | nop | 0 | 74 | 1 | 0 | 00 |
| 15 | 00111011 | addi $c3, □ | 0 | 74 | 1 | 0 | 00 |
| 16 | 01011110 | mult $c3, $c3 | 0 | 74 | 1 | 3 | 00 |
| 17 | 01011110 | mult $c3, $c3 | 0 | 74 | 1 | 9 | 00 |
| 18 | 00111001 | addi $c3, □ | 0 | 74 | 1 | 81 | 00 |
| 19 | 11011001 | result $c3, □ | 0 | 74 | 1 | 82 | 00 |
| 20 | 00000001 | sub $c0, $c0 | 0 | 74 | 1 | 82 | fc |
| 21 | 00011111 | sub $c3, $c3 | 0 | 74 | 1 | 82 | fc |
| 22 | 11100000 | nop | 0 | 74 | 1 | 0 | fc |
| 23 | 00100011 | addi $c0, □ | 0 | 74 | 1 | 0 | fc |
| 24 | 00100011 | addi $c0, □ | 3 | 74 | 1 | 0 | fc |
| 25 | 00100011 | addi $c0, □ | 6 | 74 | 1 | 0 | fc |
| 26 | 00100001 | addi $c0, □ | 9 | 74 | 1 | 0 | fc |
| 27 | 00111011 | addi $c3, □ | 10 | 74 | 1 | 0 | fc |
| 28 | 00111011 | addi $c3, □ | 10 | 74 | 1 | 3 | fc |
| 29 | 00111011 | addi $c3, □ | 10 | 74 | 1 | 6 | fc |
| 30 | 01000110 | mult $c0, $c3 | 10 | 74 | 1 | 9 | fc |
| 31 | 00100001 | addi $c0, □ | 90 | 74 | 1 | 9 | fc |
| 32 | 00011111 | sub $c3, $c3 | 91 | 74 | 1 | 9 | fc |
| 33 | 10101001 | slt $c1, $c0 | 91 | 74 | 1 | 0 | fc |
| 34 | 00000001 | sub $c0, $c0 | 91 | 74 | 1 | 0 | 01 |
| 35 | 00011111 | sub $c3, $c3 | 0 | 74 | 1 | 0 | 01 |
| 36 | 00111011 | addi $c3, □ | 0 | 74 | 1 | 0 | 01 |
| 37 | 00000110 | add $c0, $c3 | 0 | 74 | 1 | 3 | 01 |
| 38 | 01011000 | mult $c3, $c0 | 3 | 74 | 1 | 3 | 01 |
| 39 | 01011000 | mult $c3, $c0 | 3 | 74 | 1 | 9 | 01 |
| 40 | 00011110 | add $c3, $c3 | 3 | 74 | 1 | 27 | 01 |
| 41 | 11011001 | result $c3, □ | 3 | 74 | 1 | 54 | 01 |
| 54 | 00100011 | addi $c0, □ | 3 | 74 | 1 | 54 | fc |
| 55 | 00100001 | addi $c0, □ | 6 | 74 | 1 | 54 | fc |
| 56 | 01000000 | mult $c0, $c0 | 7 | 74 | 1 | 54 | fc |
| 57 | 01000000 | mult $c0, $c0 | 49 | 74 | 1 | 54 | fc |
| 58 | 00100001 | addi $c0, □ | 97 | 74 | 1 | 54 | fc |
| 59 | 00011111 | sub $c3, $c3 | 98 | 74 | 1 | 54 | fc |
| 60 | 10101001 | slt $c1, $c0 | 98 | 74 | 1 | 0 | fc |
| 61 | 00000001 | sub $c0, $c0 | 98 | 74 | 1 | 0 | 01 |
| 62 | 11100000 | nop | 0 | 74 | 1 | 0 | 01 |
| 63 | 00111011 | addi $c3, □ | 0 | 74 | 1 | 0 | 01 |
| 64 | 01011110 | mult $c3, $c3 | 0 | 74 | 1 | 3 | 01 |
| 65 | 01011110 | mult $c3, $c3 | 0 | 74 | 1 | 9 | 01 |
| 66 | 00111101 | addi $c3, □ | 0 | 74 | 1 | 81 | 01 |
| 67 | 11011001 | result $c3, □ | 0 | 74 | 1 | 78 | 01 |
| 78 | 01101101 | st $c1, $c2 | 0 | 74 | 1 | 78 | fc |
| 79 | 00110001 | addi $c2, □ | 0 | 74 | 1 | 78 | fc |
| 80 | 00100001 | addi $c0, □ | 0 | 74 | 2 | 78 | fc |
| 81 | 10000000 | jr $c0 | 1 | 74 | 2 | 78 | fc |
| 1 | 00000001 | sub $c0, $c0 | 1 | 74 | 2 | 78 | fc |
| 2 | 00011111 | sub $c3, $c3 | 0 | 74 | 2 | 78 | fc |
| 3 | 11100000 | nop | 0 | 74 | 2 | 0 | fc |
| 4 | 00100011 | addi $c0, □ | 0 | 74 | 2 | 0 | fc |
| 5 | 00100010 | addi $c0, □ | 3 | 74 | 2 | 0 | fc |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | 00100000 | addi $c0, | 5 | 74 | 2 | 0 | fc |
| 7 | 00100000 | addi $c0, | 5 | 74 | 2 | 0 | fc |
| 8 | 00100000 | addi $c0, | 5 | 74 | 2 | 0 | fc |
| 9 | 11100000 | nop | 5 | 74 | 2 | 0 | fc |
| 10 | 01101100 | ld $c1, $c2 | 5 | 74 | 2 | 0 | fc |
| 11 | 00001000 | add $c1, $c0 | 5 | 76 | 2 | 0 | fc |
| 12 | 10101000 | beq $c1, $c0 | 5 | 81 | 2 | 0 | fc |
| 13 | 00000001 | sub $c0, $c0 | 5 | 81 | 2 | 0 | 00 |
| 14 | 11100000 | nop | 0 | 81 | 2 | 0 | 00 |
| 15 | 00111011 | addi $c3, □ | 0 | 81 | 2 | 0 | 00 |
| 16 | 01011110 | mult $c3, $c3 | 0 | 81 | 2 | 3 | 00 |
| 17 | 01011110 | mult $c3, $c3 | 0 | 81 | 2 | 9 | 00 |
| 18 | 00111001 | addi $c3, □ | 0 | 81 | 2 | 81 | 00 |
| 19 | 11011001 | result $c3, □ | 0 | 81 | 2 | 82 | 00 |
| 20 | 00000001 | sub $c0, $c0 | 0 | 81 | 2 | 82 | fc |
| 21 | 00011111 | sub $c3, $c3 | 0 | 81 | 2 | 82 | fc |
| 22 | 11100000 | nop | 0 | 81 | 2 | 0 | fc |
| 23 | 00100011 | addi $c0, □ | 0 | 81 | 2 | 0 | fc |
| 24 | 00100011 | addi $c0, □ | 3 | 81 | 2 | 0 | fc |
| 25 | 00100011 | addi $c0, □ | 6 | 81 | 2 | 0 | fc |
| 26 | 00100001 | addi $c0, □ | 9 | 81 | 2 | 0 | fc |
| 27 | 00111011 | addi $c3, □ | 10 | 81 | 2 | 0 | fc |
| 28 | 00111011 | addi $c3, □ | 10 | 81 | 2 | 3 | fc |
| 29 | 00111011 | addi $c3, □ | 10 | 81 | 2 | 6 | fc |
| 30 | 01000110 | mult $c0, $c3 | 10 | 81 | 2 | 9 | fc |
| 31 | 00100001 | addi $c0, □ | 90 | 81 | 2 | 9 | fc |
| 32 | 00011111 | sub $c3, $c3 | 91 | 81 | 2 | 9 | fc |
| 33 | 10101001 | slt $c1, $c0 | 91 | 81 | 2 | 0 | fc |
| 34 | 00000001 | sub $c0, $c0 | 91 | 81 | 2 | 0 | 01 |
| 35 | 00011111 | sub $c3, $c3 | 0 | 81 | 2 | 0 | 01 |
| 36 | 00111011 | addi $c3, □ | 0 | 81 | 2 | 0 | 01 |
| 37 | 00000110 | add $c0, $c3 | 0 | 81 | 2 | 3 | 01 |
| 38 | 01011000 | mult $c3, $c0 | 3 | 81 | 2 | 3 | 01 |
| 39 | 01011000 | mult $c3, $c0 | 3 | 81 | 2 | 9 | 01 |
| 40 | 00011110 | add $c3, $c3 | 3 | 81 | 2 | 27 | 01 |
| 41 | 11011001 | result $c3, □ | 3 | 81 | 2 | 54 | 01 |
| 54 | 00100011 | addi $c0, □ | 3 | 81 | 2 | 54 | fc |
| 55 | 00100001 | addi $c0, □ | 6 | 81 | 2 | 54 | fc |
| 56 | 01000000 | mult $c0, $c0 | 7 | 81 | 2 | 54 | fc |
| 57 | 01000000 | mult $c0, $c0 | 49 | 81 | 2 | 54 | fc |
| 58 | 00100001 | addi $c0, □ | 97 | 81 | 2 | 54 | fc |
| 59 | 00011111 | sub $c3, $c3 | 98 | 81 | 2 | 54 | fc |
| 60 | 10101001 | slt $c1, $c0 | 98 | 81 | 2 | 0 | fc |
| 61 | 00000001 | sub $c0, $c0 | 98 | 81 | 2 | 0 | 01 |
| 62 | 11100000 | nop | 0 | 81 | 2 | 0 | 01 |
| 63 | 00111011 | addi $c3, □ | 0 | 81 | 2 | 0 | 01 |
| 64 | 01011110 | mult $c3, $c3 | 0 | 81 | 2 | 3 | 01 |
| 65 | 01011110 | mult $c3, $c3 | 0 | 81 | 2 | 9 | 01 |
| 66 | 00111101 | addi $c3, □ | 0 | 81 | 2 | 81 | 01 |
| 67 | 11011001 | result $c3, □ | 0 | 81 | 2 | 78 | 01 |
| 78 | 01101101 | st $c1, $c2 | 0 | 81 | 2 | 78 | fc |
| 79 | 00110001 | addi $c2, □ | 0 | 81 | 2 | 78 | fc |
| 80 | 00100001 | addi $c0, □ | 0 | 81 | 3 | 78 | fc |
| 81 | 10000000 | jr $c0 | 1 | 81 | 3 | 78 | fc |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 00000001 | sub $c0, $c0 | 1 | 81 | 3 | 78 | fc |
| 2 | 00011111 | sub $c3, $c3 | 0 | 81 | 3 | 78 | fc |
| 3 | 11100000 | nop | 0 | 81 | 3 | 0 | fc |
| 4 | 00100011 | addi $c0, □ | 0 | 81 | 3 | 0 | fc |
| 5 | 00100010 | addi $c0, □ | 3 | 81 | 3 | 0 | fc |
| 6 | 00100000 | addi $c0, | 5 | 81 | 3 | 0 | fc |
| 7 | 00100000 | addi $c0, | 5 | 81 | 3 | 0 | fc |
| 8 | 00100000 | addi $c0, | 5 | 81 | 3 | 0 | fc |
| 9 | 11100000 | nop | 5 | 81 | 3 | 0 | fc |
| 10 | 01101100 | ld $c1, $c2 | 5 | 81 | 3 | 0 | fc |
| 11 | 00001000 | add $c1, $c0 | 5 | 76 | 3 | 0 | fc |
| 12 | 10101000 | beq $c1, $c0 | 5 | 81 | 3 | 0 | fc |
| 13 | 00000001 | sub $c0, $c0 | 5 | 81 | 3 | 0 | 00 |
| 14 | 11100000 | nop | 0 | 81 | 3 | 0 | 00 |
| 15 | 00111011 | addi $c3, □ | 0 | 81 | 3 | 0 | 00 |
| 16 | 01011110 | mult $c3, $c3 | 0 | 81 | 3 | 3 | 00 |
| 17 | 01011110 | mult $c3, $c3 | 0 | 81 | 3 | 9 | 00 |
| 18 | 00111001 | addi $c3, □ | 0 | 81 | 3 | 81 | 00 |
| 19 | 11011001 | result $c3, □ | 0 | 81 | 3 | 82 | 00 |
| 20 | 00000001 | sub $c0, $c0 | 0 | 81 | 3 | 82 | fc |
| 21 | 00011111 | sub $c3, $c3 | 0 | 81 | 3 | 82 | fc |
| 22 | 11100000 | nop | 0 | 81 | 3 | 0 | fc |
| 23 | 00100011 | addi $c0, □ | 0 | 81 | 3 | 0 | fc |
| 24 | 00100011 | addi $c0, □ | 3 | 81 | 3 | 0 | fc |
| 25 | 00100011 | addi $c0, □ | 6 | 81 | 3 | 0 | fc |
| 26 | 00100001 | addi $c0, □ | 9 | 81 | 3 | 0 | fc |
| 27 | 00111011 | addi $c3, □ | 10 | 81 | 3 | 0 | fc |
| 28 | 00111011 | addi $c3, □ | 10 | 81 | 3 | 3 | fc |
| 29 | 00111011 | addi $c3, □ | 10 | 81 | 3 | 6 | fc |
| 30 | 01000110 | mult $c0, $c3 | 10 | 81 | 3 | 9 | fc |
| 31 | 00100001 | addi $c0, □ | 90 | 81 | 3 | 9 | fc |
| 32 | 00011111 | sub $c3, $c3 | 91 | 81 | 3 | 9 | fc |
| 33 | 10101001 | slt $c1, $c0 | 91 | 81 | 3 | 0 | fc |
| 34 | 00000001 | sub $c0, $c0 | 91 | 81 | 3 | 0 | 01 |
| 35 | 00011111 | sub $c3, $c3 | 0 | 81 | 3 | 0 | 01 |
| 36 | 00111011 | addi $c3, □ | 0 | 81 | 3 | 0 | 01 |
| 37 | 00000110 | add $c0, $c3 | 0 | 81 | 3 | 3 | 01 |
| 38 | 01011000 | mult $c3, $c0 | 3 | 81 | 3 | 3 | 01 |
| 39 | 01011000 | mult $c3, $c0 | 3 | 81 | 3 | 9 | 01 |
| 40 | 00011110 | add $c3, $c3 | 3 | 81 | 3 | 27 | 01 |
| 41 | 11011001 | result $c3, □ | 3 | 81 | 3 | 54 | 01 |
| 54 | 00100011 | addi $c0, □ | 3 | 81 | 3 | 54 | fc |
| 55 | 00100001 | addi $c0, □ | 6 | 81 | 3 | 54 | fc |
| 56 | 01000000 | mult $c0, $c0 | 7 | 81 | 3 | 54 | fc |
| 57 | 01000000 | mult $c0, $c0 | 49 | 81 | 3 | 54 | fc |
| 58 | 00100001 | addi $c0, □ | 97 | 81 | 3 | 54 | fc |
| 59 | 00011111 | sub $c3, $c3 | 98 | 81 | 3 | 54 | fc |
| 60 | 10101001 | slt $c1, $c0 | 98 | 81 | 3 | 0 | fc |
| 61 | 00000001 | sub $c0, $c0 | 98 | 81 | 3 | 0 | 01 |
| 62 | 11100000 | nop | 0 | 81 | 3 | 0 | 01 |
| 63 | 00111011 | addi $c3, □ | 0 | 81 | 3 | 0 | 01 |
| 64 | 01011110 | mult $c3, $c3 | 0 | 81 | 3 | 3 | 01 |
| 65 | 01011110 | mult $c3, $c3 | 0 | 81 | 3 | 9 | 01 |
| 66 | 00111101 | addi $c3, □ | 0 | 81 | 3 | 81 | 01 |
| 67 | 11011001 | result $c3, □ | 0 | 81 | 3 | 78 | 01 |
| 78 | 01101101 | st $c1, $c2 | 0 | 81 | 3 | 78 | fc |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 79 | 00110001 | addi $c2, □ | 0 | 81 | 3 | 78 | fc |
| 80 | 00100001 | addi $c0, □ | 0 | 81 | 4 | 78 | fc |
| 81 | 10000000 | jr $c0 | 1 | 81 | 4 | 78 | fc |
| 1 | 00000001 | sub $c0, $c0 | 1 | 81 | 4 | 78 | fc |
| 2 | 00011111 | sub $c3, $c3 | 0 | 81 | 4 | 78 | fc |
| 3 | 11100000 | nop | 0 | 81 | 4 | 0 | fc |
| 4 | 00100011 | addi $c0, □ | 0 | 81 | 4 | 0 | fc |
| 5 | 00100010 | addi $c0, □ | 3 | 81 | 4 | 0 | fc |
| 6 | 00100000 | addi $c0, | 5 | 81 | 4 | 0 | fc |
| 7 | 00100000 | addi $c0, | 5 | 81 | 4 | 0 | fc |
| 8 | 00100000 | addi $c0, | 5 | 81 | 4 | 0 | fc |
| 9 | 11100000 | nop | 5 | 81 | 4 | 0 | fc |
| 10 | 01101100 | ld $c1, $c2 | 5 | 81 | 4 | 0 | fc |
| 11 | 00001000 | add $c1, $c0 | 5 | 79 | 4 | 0 | fc |
| 12 | 10101000 | beq $c1, $c0 | 5 | 84 | 4 | 0 | fc |
| 13 | 00000001 | sub $c0, $c0 | 5 | 84 | 4 | 0 | 00 |
| 14 | 11100000 | nop | 0 | 84 | 4 | 0 | 00 |
| 15 | 00111011 | addi $c3, □ | 0 | 84 | 4 | 0 | 00 |
| 16 | 01011110 | mult $c3, $c3 | 0 | 84 | 4 | 3 | 00 |
| 17 | 01011110 | mult $c3, $c3 | 0 | 84 | 4 | 9 | 00 |
| 18 | 00111001 | addi $c3, □ | 0 | 84 | 4 | 81 | 00 |
| 19 | 11011001 | result $c3, □ | 0 | 84 | 4 | 82 | 00 |
| 20 | 00000001 | sub $c0, $c0 | 0 | 84 | 4 | 82 | fc |
| 21 | 00011111 | sub $c3, $c3 | 0 | 84 | 4 | 82 | fc |
| 22 | 11100000 | nop | 0 | 84 | 4 | 0 | fc |
| 23 | 00100011 | addi $c0, □ | 0 | 84 | 4 | 0 | fc |
| 24 | 00100011 | addi $c0, □ | 3 | 84 | 4 | 0 | fc |
| 25 | 00100011 | addi $c0, □ | 6 | 84 | 4 | 0 | fc |
| 26 | 00100001 | addi $c0, □ | 9 | 84 | 4 | 0 | fc |
| 27 | 00111011 | addi $c3, □ | 10 | 84 | 4 | 0 | fc |
| 28 | 00111011 | addi $c3, □ | 10 | 84 | 4 | 3 | fc |
| 29 | 00111011 | addi $c3, □ | 10 | 84 | 4 | 6 | fc |
| 30 | 01000110 | mult $c0, $c3 | 10 | 84 | 4 | 9 | fc |
| 31 | 00100001 | addi $c0, □ | 90 | 84 | 4 | 9 | fc |
| 32 | 00011111 | sub $c3, $c3 | 91 | 84 | 4 | 9 | fc |
| 33 | 10101001 | slt $c1, $c0 | 91 | 84 | 4 | 0 | fc |
| 34 | 00000001 | sub $c0, $c0 | 91 | 84 | 4 | 0 | 01 |
| 35 | 00011111 | sub $c3, $c3 | 0 | 84 | 4 | 0 | 01 |
| 36 | 00111011 | addi $c3, □ | 0 | 84 | 4 | 0 | 01 |
| 37 | 00000110 | add $c0, $c3 | 0 | 84 | 4 | 3 | 01 |
| 38 | 01011000 | mult $c3, $c0 | 3 | 84 | 4 | 3 | 01 |
| 39 | 01011000 | mult $c3, $c0 | 3 | 84 | 4 | 9 | 01 |
| 40 | 00011110 | add $c3, $c3 | 3 | 84 | 4 | 27 | 01 |
| 41 | 11011001 | result $c3, □ | 3 | 84 | 4 | 54 | 01 |
| 54 | 00100011 | addi $c0, □ | 3 | 84 | 4 | 54 | fc |
| 55 | 00100001 | addi $c0, □ | 6 | 84 | 4 | 54 | fc |
| 56 | 01000000 | mult $c0, $c0 | 7 | 84 | 4 | 54 | fc |
| 57 | 01000000 | mult $c0, $c0 | 49 | 84 | 4 | 54 | fc |
| 58 | 00100001 | addi $c0, □ | 97 | 84 | 4 | 54 | fc |
| 59 | 00011111 | sub $c3, $c3 | 98 | 84 | 4 | 54 | fc |
| 60 | 10101001 | slt $c1, $c0 | 98 | 84 | 4 | 0 | fc |
| 61 | 00000001 | sub $c0, $c0 | 98 | 84 | 4 | 0 | 01 |
| 62 | 11100000 | nop | 0 | 84 | 4 | 0 | 01 |
| 63 | 00111011 | addi $c3, □ | 0 | 84 | 4 | 0 | 01 |
| 64 | 01011110 | mult $c3, $c3 | 0 | 84 | 4 | 3 | 01 |
| 65 | 01011110 | mult $c3, $c3 | 0 | 84 | 4 | 9 | 01 |

```
 66 | 00111101 | addi $c3, □   |  0   84   4   81   01
 67 | 11011001 | result $c3, □ |  0   84   4   78   01
 78 | 01101101 | st $c1, $c2   |  0   84   4   78   fc
 79 | 00110001 | addi $c2, □   |  0   84   4   78   fc
 80 | 00100001 | addi $c0, □   |  0   84   5   78   fc
 81 | 10000000 | jr $c0        |  1   84   5   78   fc
  1 | 00000001 | sub $c0, $c0  |  1   84   5   78   fc
  2 | 00011111 | sub $c3, $c3  |  0   84   5   78   fc
  3 | 11100000 | nop           |  0   84   5    0   fc
  4 | 00100011 | addi $c0, □   |  0   84   5    0   fc
  5 | 00100010 | addi $c0, □   |  3   84   5    0   fc
  6 | 00100000 | addi $c0,     |  5   84   5    0   fc
  7 | 00100000 | addi $c0,     |  5   84   5    0   fc
  8 | 00100000 | addi $c0,     |  5   84   5    0   fc
  9 | 11100000 | nop           |  5   84   5    0   fc
 10 | 01101100 | ld $c1, $c2   |  5   84   5    0   fc
 11 | 00001000 | add $c1, $c0  |  5    0   5    0   fc
 12 | 10101000 | beq $c1, $c0  |  5    5   5    0   fc
 13 | 00000001 | sub $c0, $c0  |  5    5   5    0   01
 14 | 11100000 | nop           |  0    5   5    0   01
 15 | 00111011 | addi $c3, □   |  0    5   5    0   01
 16 | 01011110 | mult $c3, $c3 |  0    5   5    3   01
 17 | 01011110 | mult $c3, $c3 |  0    5   5    9   01
 18 | 00111001 | addi $c3, □   |  0    5   5   81   01
 19 | 11011001 | result $c3, □ |  0    5   5   82   01
 82 | 11100000 | nop           |  0    5   5   82   fc
 83 | 11100001 | hlt           |  0    5   5   82   fc
3: Programa Finalizado
Memoria de dados [0:4]:
  [0] = 01001101 (77 / 'M') 77
  [1] = 01001010 (74 / 'J') 74
  [2] = 01010001 (81 / 'Q') 81
  [3] = 01010001 (81 / 'Q') 81
  [4] = 01010100 (84 / 'T') 84
```

Outros casos de teste:

1)

```
Cifra = 15
1: Estado Inicial
Memoria de dados [0:5]:
  [0] = 01000010 (66 / 'B') 66
  [1] = 01000001 (65 / 'A') 65
  [2] = 01010100 (84 / 'T') 84
  [3] = 01000001 (65 / 'A') 65
  [4] = 01010100 (84 / 'T') 84
  [5] = 01000001 (65 / 'A') 65
3: Programa Finalizado
Memoria de dados [0:5]:
  [0] = 01010001 (81 / 'Q') 81
  [1] = 01010000 (80 / 'P') 80
  [2] = 01001001 (73 / 'I') 73
  [3] = 01010000 (80 / 'P') 80
  [4] = 01001001 (73 / 'I') 73
  [5] = 01010000 (80 / 'P') 80
```

2)

```
Cifra = -15
1: Estado Inicial
Memoria de dados [0:8]:
  [0] = 01000111 (71 / 'G') 71
  [1] = 01000001 (65 / 'A') 65
  [2] = 01001100 (76 / 'L') 76
  [3] = 01001111 (79 / 'O') 79
  [4] = 01000110 (70 / 'F') 70
  [5] = 01010010 (82 / 'R') 82
  [6] = 01001001 (73 / 'I') 73
  [7] = 01010100 (84 / 'T') 84
  [8] = 01001111 (79 / 'O') 79
3: Programa Finalizado
Memoria de dados [0:8]:
  [0] = 01010010 (82 / 'R') 82
  [1] = 01001100 (76 / 'L') 76
  [2] = 01010111 (87 / 'W') 87
  [3] = 01011010 (90 / 'Z') 90
  [4] = 01010001 (81 / 'Q') 81
  [5] = 01000011 (67 / 'C') 67
  [6] = 01010100 (84 / 'T') 84
  [7] = 01000101 (69 / 'E') 69
  [8] = 01011010 (90 / 'Z') 90
```

Todos os testbenchs foram realizados no site https://www.edaplayground.com/.