



CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

ENGENHARIA DE COMPUTAÇÃO  
LABORATÓRIO DE ARQUITETURA E ORGANIZAÇÃO DE  
COMPUTADORES I

---

## PRÁTICA 10

*Implementação em Verilog dos componentes do nRisc que  
armazenam estado*

---

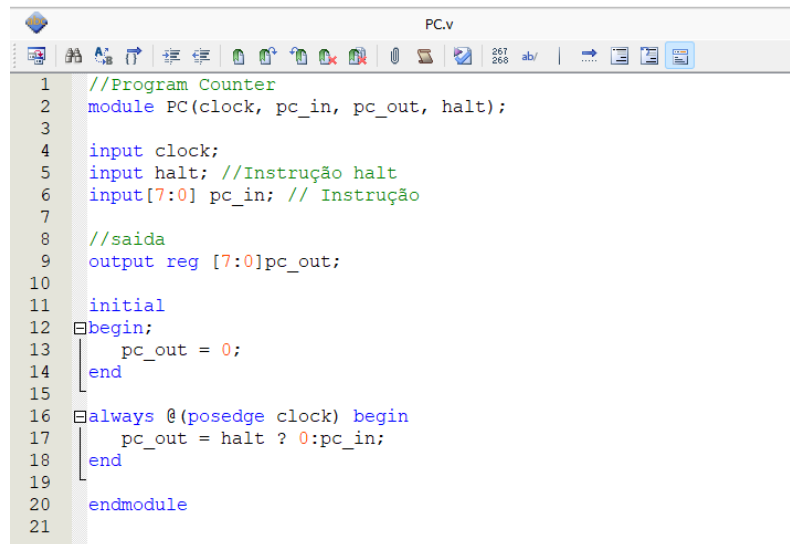
Thamiris Souza Madeira Ferreira – 20193008037

Hugo Barbosa Santana Silva – 20193002946

Belo Horizonte

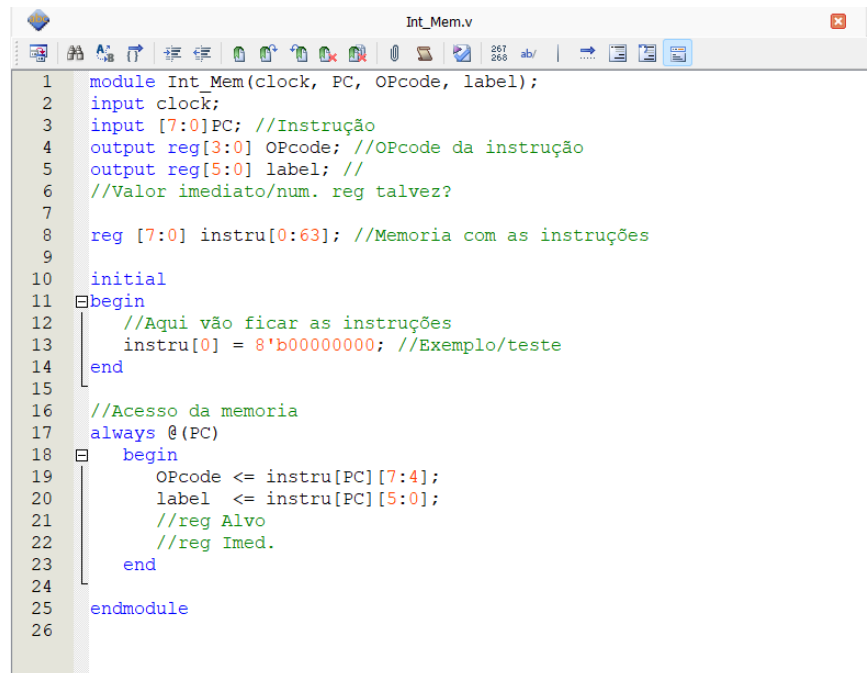
June 26, 2022

# 1 Código fonte



```
1 //Program Counter
2 module PC(clock, pc_in, pc_out, halt);
3
4 input clock;
5 input halt; //Instrução halt
6 input[7:0] pc_in; // Instrução
7
8 //saida
9 output reg [7:0]pc_out;
10
11 initial
12 begin;
13     pc_out = 0;
14 end
15
16 always @(posedge clock) begin
17     pc_out = halt ? 0:pc_in;
18 end
19
20 endmodule
21
```

Figure 1: Program Counter - PC



```
1 module Int_Mem(clock, PC, OPcode, label);
2 input clock;
3 input [7:0]PC; //Instrução
4 output reg[3:0] OPcode; //OPcode da instrução
5 output reg[5:0] label; //
6 //Valor imediato/num. reg talvez?
7
8 reg [7:0] instru[0:63]; //Memoria com as instruções
9
10 initial
11 begin
12     //Aqui vão ficar as instruções
13     instru[0] = 8'b00000000; //Exemplo/teste
14 end
15
16 //Acesso da memoria
17 always @(PC)
18 begin
19     OPcode <= instru[PC][7:4];
20     label <= instru[PC][5:0];
21     //reg Alvo
22     //reg Imed.
23 end
24
25 endmodule
26
```

Figure 2: Memória de instruções

```

21 module Mem_Dados(clock, endereco, MemRead, MemWrite, dado_in, dado_out);
22
23 input clock, MemRead, MemWrite;
24 input [7:0]endereco, dado_in;
25 output [7:0]dado_out;
26
27 reg [7:0] dado [15:0];
28
29 //assign dado_out = line;
30
31 //Carrega dados na memoria
32 initial
33 begin
34     dado[0] = 8'b00000000; //t1 = 1;
35     dado[1] = 8'b00000001; //t2 = 1;
36     dado[2] = 8'b00000010; //sum
37 end
38
39 always @(posedge clock)//Escrita
40
41 begin
42 if(MemWrite)
43     memdata[endereco] = dado_in;
44 end
45
46 //Leitura de dados?
47
48 endmodule
49
50

```

Figure 3: Memória de dados

```

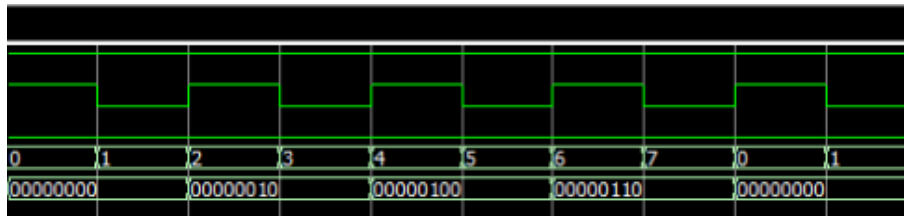
1 module regis(Reg1, Reg2, RegWrite, DataWrite, Reg1Write, Data1, Data2, clock);
2
3 input Reg1, Reg2, Reg1Write;
4 input [7:0] DataWrite;
5 input RegWrite, clock;
6
7 output [7:0] Data1, Data2;
8 reg [7:0] bank [3:0];
9
10 assign Data1 = bank[Reg1];
11 assign Data2 = bank[Reg2];
12
13 always
14 begin
15     @ (posedge clock)
16     if(RegWrite)
17         bank[Reg1Write] <= DataWrite;
18     end
19 end
20
21 endmodule
22

```

Figure 4: Banco de registradores

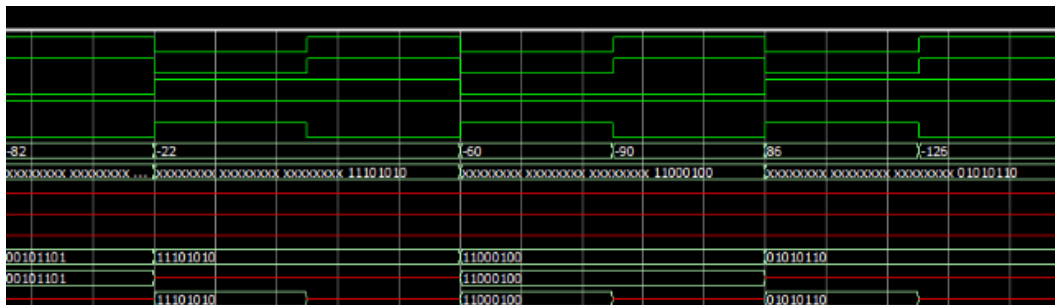
## 2 Simulação dos módulos implementados

Demonstramos o correto funcionamento dos módulos implementados e o código fonte dos módulos de simulação:



Teste de PC

São recebidas quatro entradas, sendo que para as três primeiras foram criados valores testes para que o programa pudesse gerar a saída. A entrada foi gerada a partir do cálculo do PC anterior. E por fim, a saída PC é resultado do valor recebido na entrada.



Teste do banco de registradores

Os registrador Reg1 e Reg1write estão com o mesmo sinal, porque são os mesmo registrador, o RegWrite está sempre ativo para conseguirmos enxergar que ele sempre escreve no banco de dados o DataWrite em cada registrador apropriado. Por fim, isso ocorre sempre que há uma borda de subida do clock.

## 3 Obstáculos encontrados

Houve um erro durante a simulação e não foi possível realizá-la, não sabemos exatamente o que é esse erro. Tentamos contornar esse problema utilizando waves, mas sem sucesso. O erro apresentado foi:

```
# Error loading design
# Error: Error loading design
#      Pausing macro execution
# MACRO ./Mem_Dados_run_msim_rtl_verilog.do PAUSED at line 11
VSIM(paused)>
```

Esse erro se apresentou também no relatório 9, mas conseguimos contornar utilizando a função *include* "nome\_do\_arquivo.v". Porém, nesta prática essa solução se mostrou ineficaz. Procuraremos outra solução ao longo da prática seguinte.

Os resultados esperados seriam:

A memória de instruções recebe um binário, que representa qual instrução será realizada, e a saída significa a instrução que será realizada naquele ciclo. Já na memória de dados, a simulação funciona a partir de "MemRead" e "MemWrite", de modo que quando o primeiro está ligado o processador irá ler a memória, enquanto que se o segundo estiver ligado, a memória será lida e escrita na variável "dato\_in".