CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

Disciplina: Laboratório de Arquitetura e Organização de Computadores I, Sexta-feira, 14h50min
Professor: Jeferson Figueiredo Chaves

Aluna: Bruna Castelo Branco Santos
Aluno: Nicola Oliveira Gontijo

Processador

O processador é a parte ativa da placa-mãe, que segue rigorosamente as instruções de um programa. Ele soma e testa números, sinaliza dispositivos de entrada e saída E/S para serem ativados e assim por diante. O processador é conhecido também por CPU, Unidade Central de Processamento e contém dois componentes principais: o caminho de dados e o controle.

- O caminho de dados consiste no componente do processador capaz de realizar operações aritméticas.
- O controle é o componente do processador que comanda o caminho de dados, a memória e os dispositivos E/S de acordo com as instruções do programa.

Unidade de controle (Controle)

Introdução

A unidade de controle é a parte mais complexa da CPU, além de possuir a lógica necessária para realizar a movimentação de dados e instruções do CPU, através dos sinais de controle que emite em instantes de tempo programados, esse dispositivo controla a ação da unidade lógica aritmética (ULA).
Os sinais de controle emitidos pela unidade de controle ocorrem em vários instantes durante o período de realização de um ciclo de instrução e, de modo geral, todos possuem uma duração fixa e igual, originada em um gerador de sinais (Clock).

A unidade de controle recebe as instruções da unidade de E/S, as converte em um formato que pode ser entendido pela ULA, e controla qual etapa do programa está sendo executado.

A unidade de controle executa três ações básicas e pré-programadas pelo próprio fabricante do processador.

As três principais etapas de processamento são:

- Busca: trazer a instrução da memória na posição apontada pelo registrador PC (Program Counter) para uma área de armazenamento dentro da CPU, chamada registrador de instrução (IR).

- Decodificação: Após ser trazida, a instrução precisa ser decodificada. Toda instrução tem uma área chama código de operação (opcode) que determina qual operação aquela instrução realiza; assim, quando os circuitos eletrônicos da CPU “descobrem” o que a instrução deve fazer, isto é chamado de decodificação.

- Execução: Uma vez decodificada, a instrução será executada. Execução pode ser entendida como a aplicação da operação nos operandos. Após a instrução ser executada, o apontador de instruções (PC) é atualizado para o endereço de memória que contém a próxima instrução.

As instruções capazes de serem realizadas no nosso projeto de processador segue o padrão da tabela abaixo:

Operações	Código	Função realizada
mvi	000	RegX ← B
mv	001	RegX ← [RegY]
sub	010	RegZ ← [RegX] - [RegY]
add	011	RegZ ← [RegX] + [RegY]

Tabela 1 - Instruções que o processador suporta

A coluna da esquerda apresenta o nome das instruções, a central o código correspondente da operação e a da direita a função realizada.

- mvi(move immediate) – realiza a passagem de dado, constate de 16 bits, para ser armazenado a um registrador.
- mv(move) – permite que determinado dado seja copiado de um registrador para outro.
- sub(subtração) – realiza a operação de subtração dos valores armazenados nos registradores RegX e RegY e coloca o resultado da operação em um registrador RegZ.
- add(soma) - realiza a operação de soma dos valores armazenados nos registradores RegX e RegY e coloca o resultado da operação em um registrador RegZ.
  Obs: [RegX] e [RegY] fazem referência ao conteúdo armazenado no registrador.

Cada instrução é armazenada em um Registrador IR (Instuction Register), no qual armazena a instrução que está sendo executada. Em função do conteúdo desse registrador, a unidade de controle determina quais os sinais de controle devem ser gerados para executar as operações determinadas pela instrução.

A instrução armazenada no Registrador IR é codificada conforme a tabela 2.

	3 bits	3 bits	3 bits	3 bits	4 bits
mvi	Código da instrução			Endereço do registrador destino	
mv	Código da instrução	Endereço registrador origem		Endereço do registrador destino	
add/sub	Código da instrução	Endereço operando X	Endereço operando Y	Endereço do registrador destino	

Tabela 2 - Formato da instrução

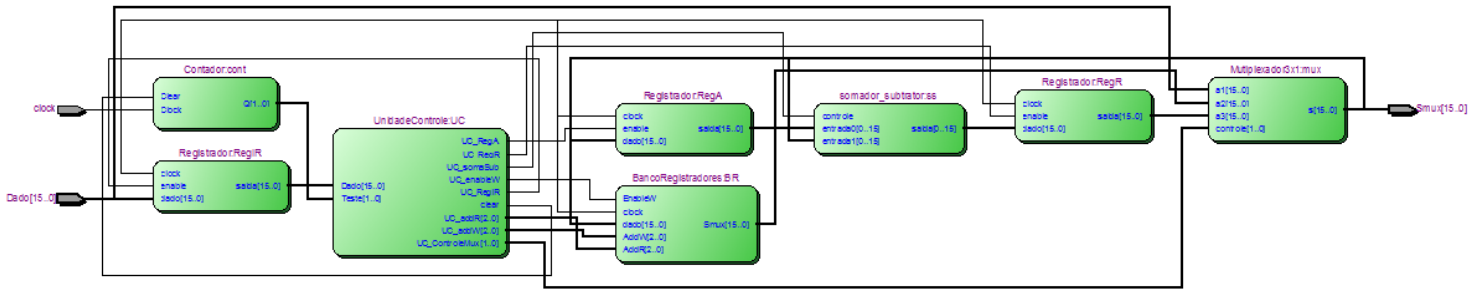

Figura 1 – Technology Viewer tool

Implementação da Unidade de controle:

Para implementação da Unidade de Controle(UC), utilizamos a técnica de máquina de estados finitos.
Uma máquina de estados finito consiste em um conjunto de estados e diretrizes sobre como mudar o estado.
As diretrizes são definidas por uma função de próximo estado, que mapeia o estado atual e as entradas para um novo estado.
Cada estado também especifica um conjunto de saídas ativadas quando a máquina está neste estado.

O conjunto de estados corresponde a todos os valores possíveis do armazenamento interno.
Uma máquina de estados finitos possui um conjunto de estados e de duas funções, chamadas função de próximo estado e a função de saída.

- **Função de próximo estado -** É uma função combinacional que, dadas às entradas e o estado atual, determina o próximo estado.
- **Função de saída -** É uma função que produz um conjunto de saídas a partir do estado atual e suas entradas.

Em nossa máquina de estados finitos, o estado muda junto a um **contador** que é sincronizado junto à entrada Clock.
Sendo que nesse projeto de unidade de controle o contador vai do estado 00 ao estado 11. O contador é resetado sempre que terminada por completo a instrução de operação.

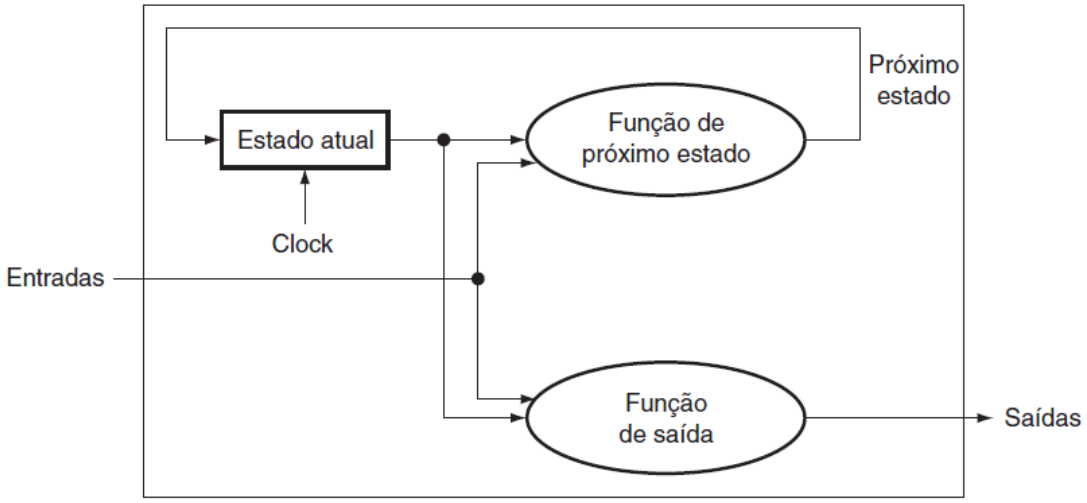

Figura 2 – Diagrama máquina de estados finitos.

Uma máquina de estados finitos é implementada com um registrador de estado que mantém o estado atual e um bloco lógico combinacional para calcular as funções de próximo estado e de saída.

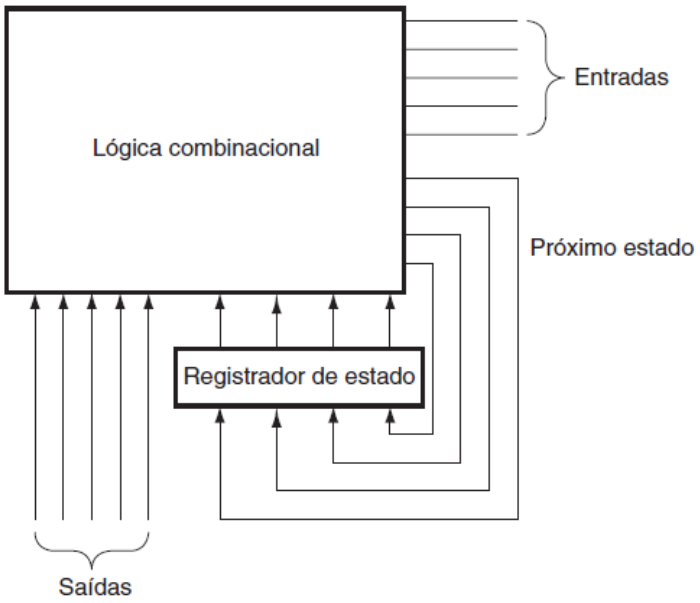

Figura 3 – Máquina de estados finitos

Código Contador:

```
module Contador(Clear, Clock, Q);
input Clear, Clock;
output [1:0] Q;
reg [1:0] Q;
always @(posedge Clock)
if (Clear)
Q <= 2'b0;
else
Q <= Q + 1'b1;
endmodule
```

Simulação circuito Contador:

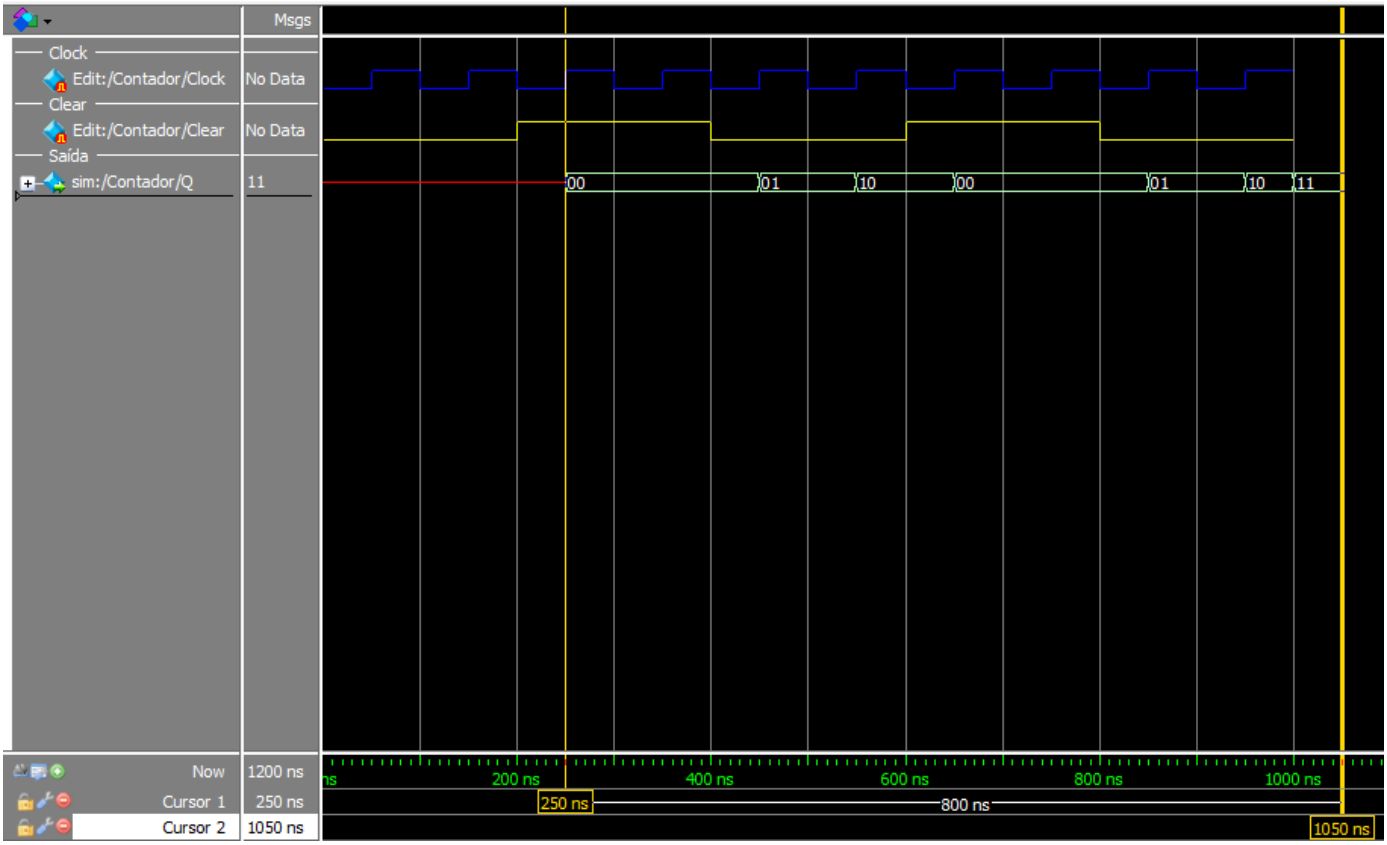

Figura 4 – Simulação ModelSim – Circuito contador

Observações:

- A onda indicada na cor azul corresponde ao clock;
- A onda indicada na cor amarela corresponde ao clear;
- A onda em verde indica saída do circuito contador;
-O pedaço de onda em vermelho representa saída indefinida.

Resultados e análise da simulação:

- O circuito contador implementado é sincronizado a entrada Clock e a cada borda de subida (posedge) sua saída recebe um incremento de 1. Esse circuito possui como uma das entradas o Clear, que tem como função zerar a contagem.

- Análise da simulação de 0ns a 250ns:
  - O valor de saída, onda na cor vermelha, corresponde a uma saída indefinida. Isso ocorre uma vez que o contador não possui um valor do estado anterior (inicial) para sofrer incremento.

- Análise da simulação de 250ns a 1050ns:
  - O valor de saída começa a ser alterado já que o contador recebeu a entrada, com valor zero, correspondente ao Clear que é ativa em 200ns.

- Enquanto o Clear estiver alto (valor 1), a saída do circuito permanece 00 a cada ocorrência de borda de subida do Clock. Estando o Clear baixo (valor 0), a saída começa a sofrer incremento a cada borda de subida do Clock. Zerando novamente apenas quando o Clear voltar a ser alto.

- Análise da simulação a partir de 1050ns:
  - O circuito não apresenta saída, devido a ausência de Clock.

Código Unidade de controle:

```
module UnidadeControle(Dado, Teste, UC_addR, UC_addW, UC_ControleMux, UC_RegA, UC_RegR, UC_somaSub, UC_enableW, UC_RegIR, clear);

input [15:0]Dado;
input [1:0]Teste;

output reg [2:0] UC_addW, UC_addR;
output reg [1:0] UC_ControleMux;
output reg UC_RegA, UC_RegR, UC_somaSub, UC_enableW, UC_RegIR, clear;


wire [2:0] Instr;

//Instrução
assign Instr = Dado[15:13];


always @(Instr or Teste or Dado)
        begin
          UC_RegIR = 1'b1;
                        UC_RegA = 1'b0;
                        UC_RegR = 1'b0;
                        UC_somaSub = 1'b0;
                        UC_enableW = 1'b0;
                        clear = 1'b0;

        //Contador
        case(Teste)

         2'b00:
                begin
                        //Define o estado inicial
                        UC_RegIR = 1'b1;
                        UC_RegA = 1'b0;
                        UC_RegR = 1'b0;
                        UC_somaSub = 1'b0;
                        UC_enableW = 1'b0;
                        clear = 1'b0;
                end

         2'b01:
                case(Instr)
                                3'b000:
                                 begin
                                        UC_RegIR = 1'b0;
                                        UC_ControleMux = 2'b00;
                                        UC_RegA = 1'b0;
                                        UC_RegR = 1'b0;
                                        UC_addW = Dado[6:4];
                                        UC_enableW = 1'b1;
                                        clear = 1'b1;
                                 end
```

```
                        3'b001:
                         begin
                                UC_RegIR = 1'b0;
                                UC_ControleMux = 2'b01;
                                UC_RegA = 1'b0;
                                UC_RegR = 1'b0;
                                UC_addR = Dado[12:10];
                                UC_enableW = 1'b1;
                                UC_addW = Dado[6:4];
                                clear = 1'b1;
                        end

                        3'b010:
                         begin
                                UC_RegIR = 1'b0;
                                UC_addR = Dado[12:10];
                                UC_ControleMux = 2'b01;
                                UC_enableW = 1'b0;
                                UC_RegA = 1'b1;
                                UC_RegR = 1'b0;
                                clear = 1'b0;
                         end
                        3'b011:
                          begin
                                UC_RegIR = 1'b0;
                                UC_ControleMux = 2'b01;
                                UC_addR = Dado[12:10];
                                UC_RegA = 1'b1;
                                UC_RegR = 1'b0;
                                clear = 1'b0;
                          end

        endcase

2'b10:
        case(Instr)
                3'b010:
                begin
                        UC_RegIR = 1'b0;
                        UC_addR = Dado[9:7];
                        UC_ControleMux = 2'b01;
                        UC_enableW = 1'b0;
                        UC_RegA = 1'b0;
                        UC_somaSub = 1'b0;
                        UC_RegR = 1'b1;
                        clear = 1'b0;

                end
                3'b011:
                  begin
                  UC_RegIR = 1'b0;
                        UC_addR = Dado[9:7];
                        UC_ControleMux = 2'b01;
                        UC_enableW = 1'b0;
                        UC_RegA = 1'b0;
                        UC_somaSub = 1'b1;
                        UC_RegR = 1'b1;
                        clear = 1'b0;
                  end
        endcase

2'b11:
```

```
                case(Instr)
                        3'b010:
                        begin
                                UC_RegIR = 1'b0;
                                UC_ControleMux = 2'b10;
                                UC_enableW = 1'b1;
                                UC_RegA = 1'b0;
                                UC_RegR = 1'b0;
                                UC_addW = Dado[6:4];
                                clear = 1'b1;
                        end
                        3'b011:
                         begin
                                UC_RegIR = 1'b0;
                                UC_ControleMux = 2'b10;
                                UC_enableW = 1'b1;
                                UC_RegA = 1'b0;
                                UC_RegR = 1'b0;
                                UC_addW = Dado[6:4];
                                clear = 1'b1;
                         end
                endcase
        endcase
        end

endmodule
```

A simulação da unidade de controle foi realizada no bloco maior, Processador, cujo código do mesmo segue-se abaixo.

```
module Processador(Dado, clock, Smux);

input [15:0]Dado;
input clock;
output [15:0] Smux;

wire clear;
wire [15:0]Sbr_mux, SRegR, Smux, Sss_RegR, SRegA;
wire [15:0]SRegIR_UC;
wire SUC_enableW, SUC_RegA, SUC_RegR, SUC_ss, SUC_RegIR;
wire [2:0]SUC_addW, SUC_addR;
wire [1:0]SUC_cltMux, Teste;

Registrador RegIR(Dado, clock, SUC_RegIR, SRegIR_UC);

Contador cont(clear, clock, Teste);

UnidadeControle UC(SRegIR_UC, Teste, SUC_addR, SUC_addW, SUC_cltMux, SUC_RegA, SUC_RegR, SUC_ss, SUC_enableW,
SUC_RegIR, clear);

Mutiplexador3x1 mux(Dado, Sbr_mux, SRegR, SUC_cltMux, Smux);

Registrador RegA(Smux, clock, SUC_RegA, SRegA);

somador_subtrator ss(SRegA, Smux, SUC_ss, Sss_RegR);

Registrador RegR(Sss_RegR, clock, SUC_RegR, SRegR);

BancoRegistradores BR(Smux, SUC_addW, SUC_addR, SUC_enableW, Sbr_mux, clock);

endmodule
```

Simulação circuito Processador

Será feita a simulação das seguintes instruções:

A = 10;
B = 5;
C = A + B ;
D = A –B;
E = B.

Resultados e análise da simulação:

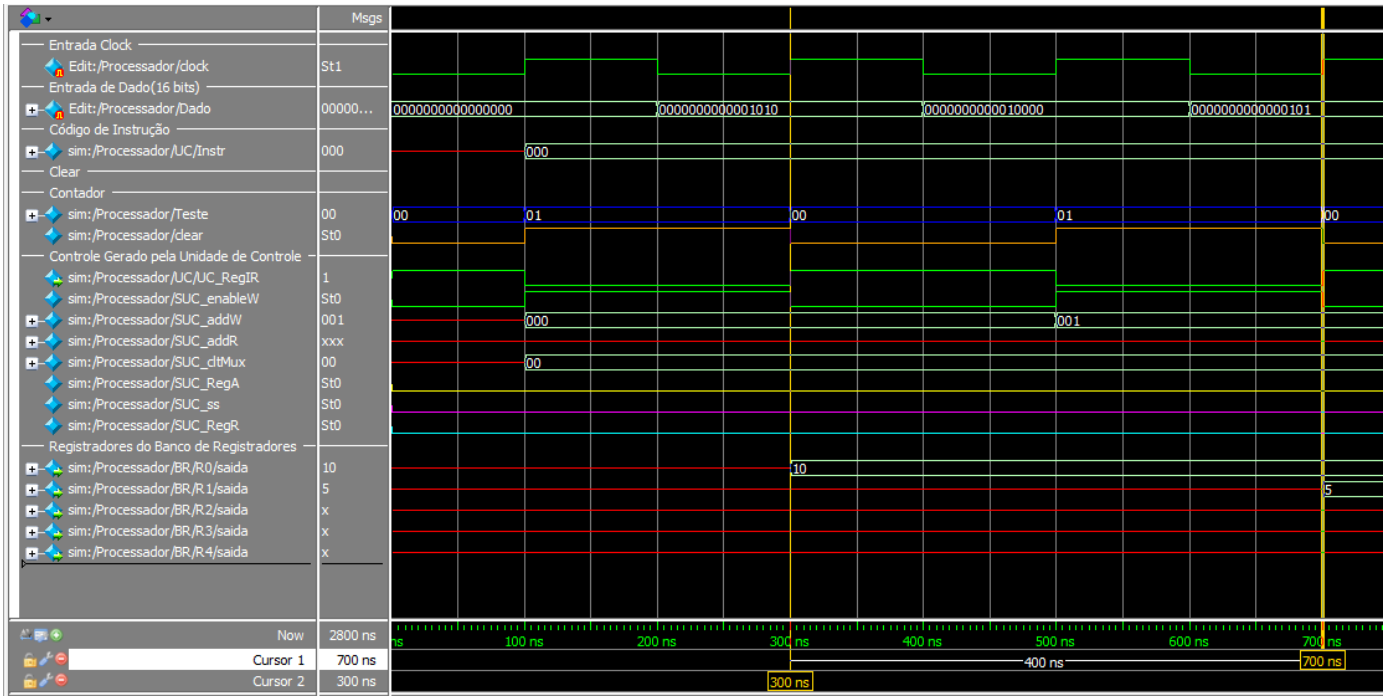

Figura 5 – Simulação ModelSim – Circuito Processador

- O circuito Processador é sincronizado a uma entrada clock com borda de subida (posedge).

- Análise da simulação de 0ns a 300ns:

  - O circuito teve como entrada o dado de 16 bits: 000 000 000 000 0000
    Tal dado é correspondente a instrução mvi(000) que armazenou no registrador 000 do Banco de Registradores o valor de 10, correspondente a entrada de dado 000 000 000 000 1010.
    [Reg000] ← 10.

- Análise da simulação de 300ns a 700ns:
  - O circuito teve como entrada o dado de 16 bits: 000 000 000 001 0000
    Tal dado é correspondente a instrução mvi(000) que armazenou no registrador 001 do Banco de Registradores o valor de 5, correspondente a entrada de dado 000 000 000 000 0101.
    [Reg001] ← 5.

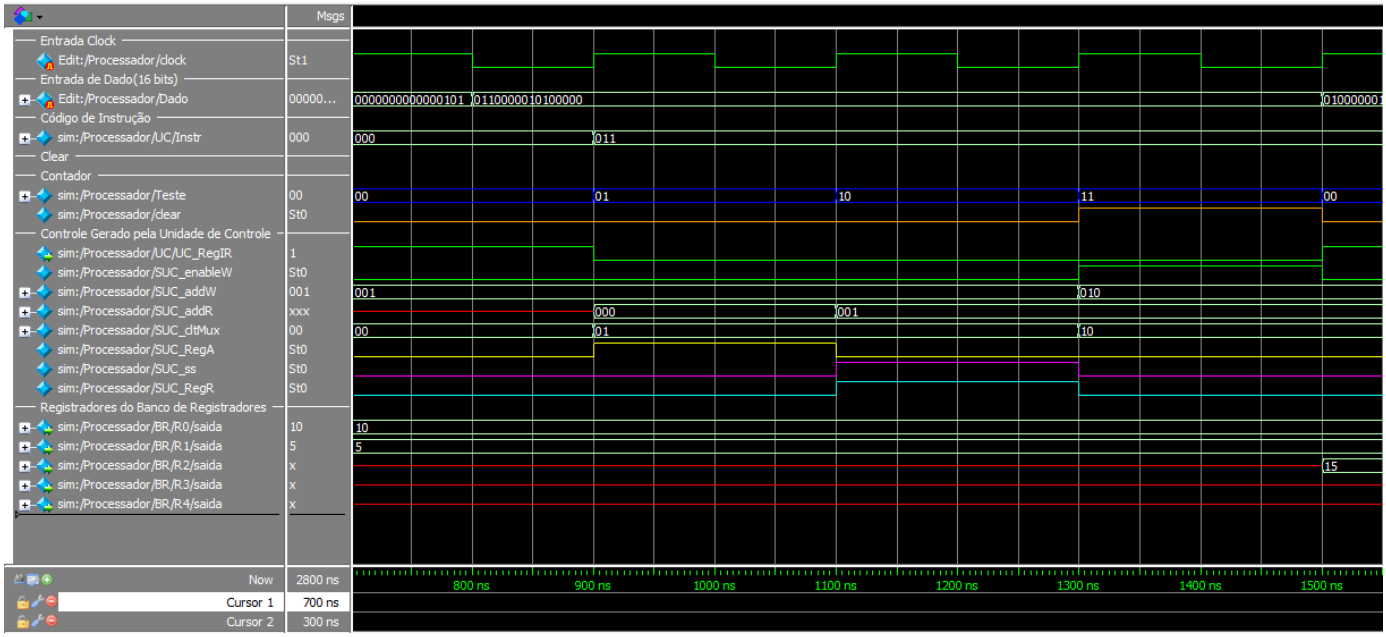

Figura 6 – Simulação ModelSim – Circuito Processador

- Análise da simulação de 700ns a 1500ns:

- O circuito teve como entrada o dado de 16 bits: 011 000 001 010 0000
  Tal dado é correspondente a instrução add(011) que realiza a operação de soma dos conteúdos armazenados nos registrador 000 e 001. Sendo o resultado armazenado em um terceiro registrador cujo endereço 010.
  [Reg000] ← 10;
  [Reg001] ← 5;
  [Reg011] ← [Reg000] + [Reg001].

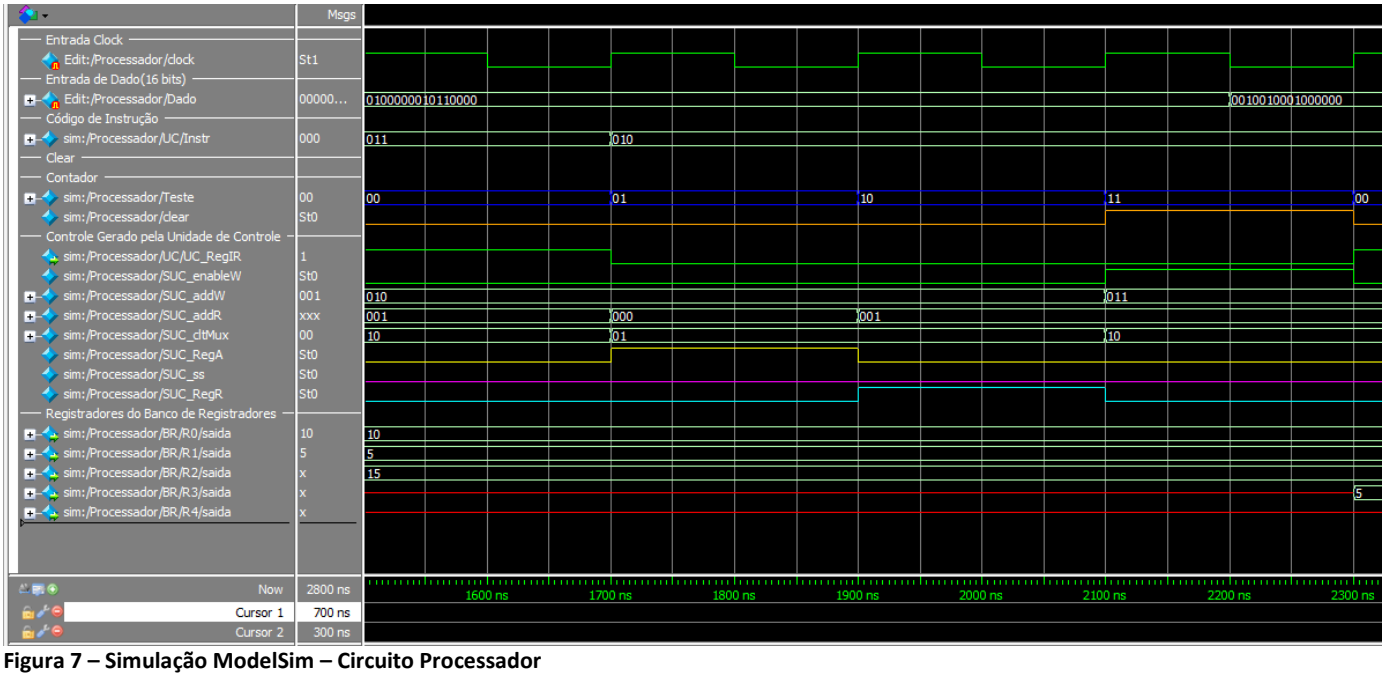

Figura 7 – Simulação ModelSim – Circuito Processador

- Análise da simulação de 1500ns a 2300ns:

- O circuito teve como entrada o dado de 16 bits: 010 000 001 011 0000
  Tal dado é correspondente a instrução sub(010) que realiza a operação de subtração dos conteúdos armazenados nos registrador 000 e 001. Sendo o resultado armazenado em um terceiro registrador cujo endereço 011.
  [Reg000] ← 10;
  [Reg001] ← 5;
  [Reg011] ← [Reg000] - [Reg001].

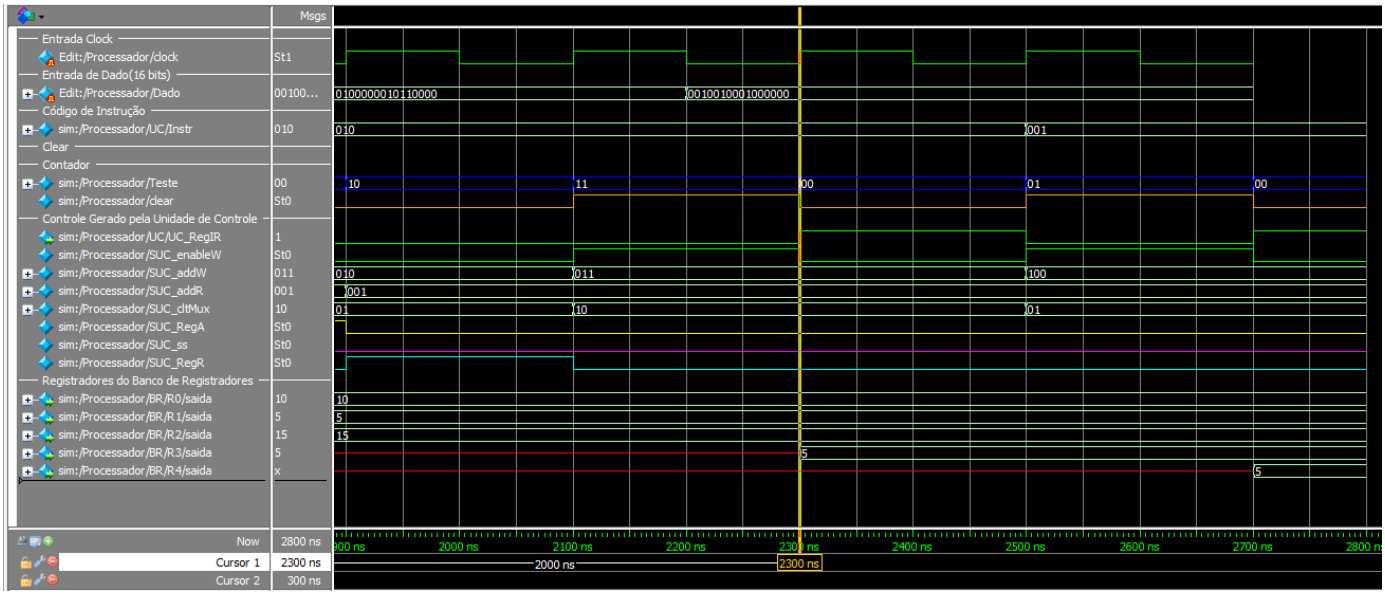

Figura 8 – Simulação ModelSim – Circuito Processador

- Análise da simulação de 2300ns a 2700ns:

  - O circuito teve como entrada o dado de 16 bits: 001 001 000 100 0000
    Tal dado é correspondente a instrução mv(001) que realiza a cópia do dado armazenado no registrador 001 para um outro registrador de endereço 100.
    Reg100 ← [Reg001].

Referências:

- Patterson, D.A e Henessy, J. L., Organização e Projeto de Computadores – A Interface Hardware/Software. Ed. Elsevier, 3ª. Ed, 484 pp, ISBN 139788535215212.
- http://www.asic-world.com/verilog/index.html
- http://www.dcc.unicamp.br/~ducatte/mc722/Material/arq_hp5.pdf
- http://www.dca.ufrn.br/~pablo/FTP/arq_de_comp/apostilha/capitulo5.pdf