Laboratório de Arquitetura e Organização de Computadores

Implementação de um processador utilizando uma placa FPGA DE2-115

Gabriel Kenji de Almeida

Universidade Federal do Estado de São Paulo

January 16, 2020

Schedule

Considerações Iniciais

Introdução

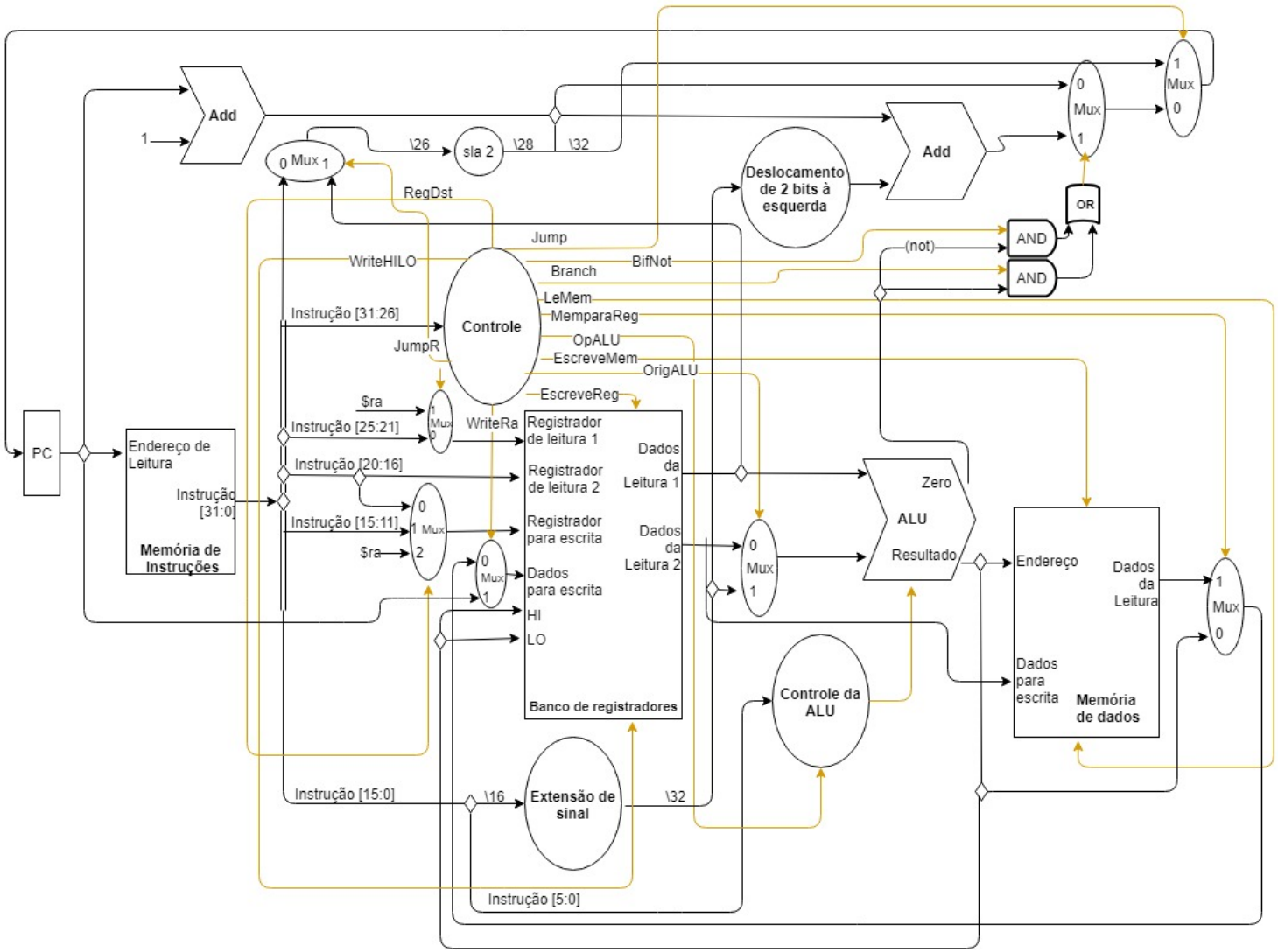
- O rápido avanço da tecnologia na última década certamente elevou a importância da eletrônica digital em um patamar nunca visto antes. É fato que a última geração da humanidade possui, em todo nível social e econômico, podendo até ser de modo indireto,uma certa dependência ou até comodismo proporcionados pela eletrônica digital.
- Um dos componentes presentes na maioria dos sistemas digitais mais complexos é o processador, que poderia ser entendido como sendo o "cérebro" do sistema, local onde todas as operações e tomadas de decisões são realizadas.
- Sendo assim, o profissional atuante em áreas que se relacionam com a eletrônica digital deve dominar com maestria uma variedade diversificada de conceitos teóricos e práticos sobre sistemas digitais, incluindo conceitos relacionados a processadores.

Objetivos

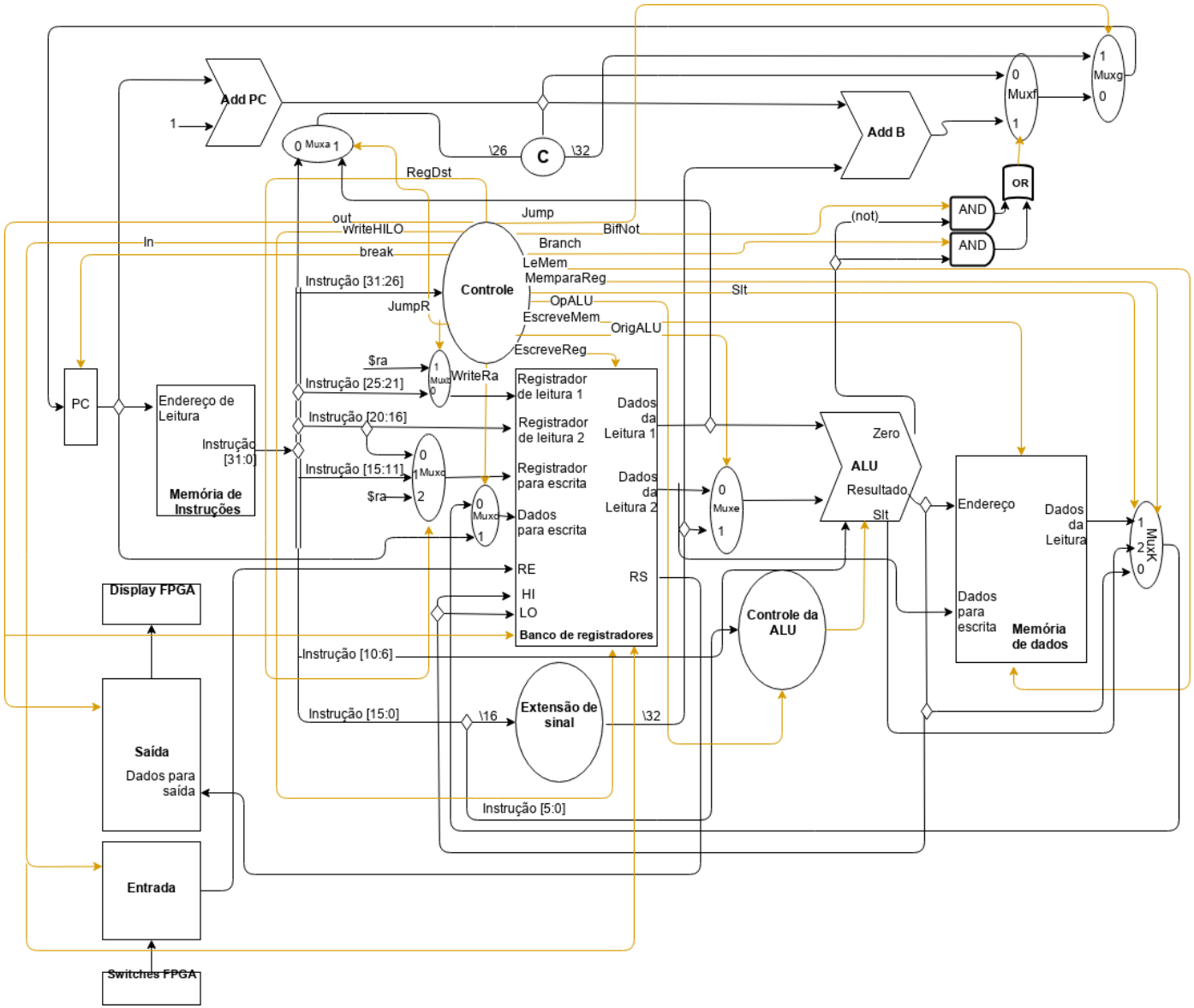
- Este projeto tem como objetivo principal a elaboração e o desenvolvimento de um processador funcional a partir de uma arquitetura customizada baseada no MIPS, que possua a capacidade de executar os algoritmos mais básicos da literatura de lógica de programação. A implementação ocorreu por meio do uso da linguagem de descrição de hardware Verilog, no ambiente do software Quartus II. O processador será simulado no dispositivo FPGA (2) DE2-115 Cyclone IV.

Caminho de Dados

Primeira versão do caminho de dados



Versão atual do caminho de dados



Sinais de Controle

Houve a inclusão de alguns sinais novos em relação ao conjunto de sinais adotados anteriormente.

- **RegDst**:
- **EscreveReg**:
- **OrigALU**;
- **LeMem**;
- **EscreveMem**;
- **MemparaReg**;
- **OpALU**;
- **WriteRa**;
- **BifNot**;
- **Branch**;
- **Jump**;
- **JumpR**;
- **In**;
- **Out**;
- **Break**.

Unidade de Controle

Pelo fato do processador desenvolvido adotar a mesma abordagem de um processador MIPS monociclo, a unidade de controle consiste basicamente em um circuito combinacional decodificador, cuja entrada é o campo "Opcode" da instrução executada, e as saídas são os sinais controle.

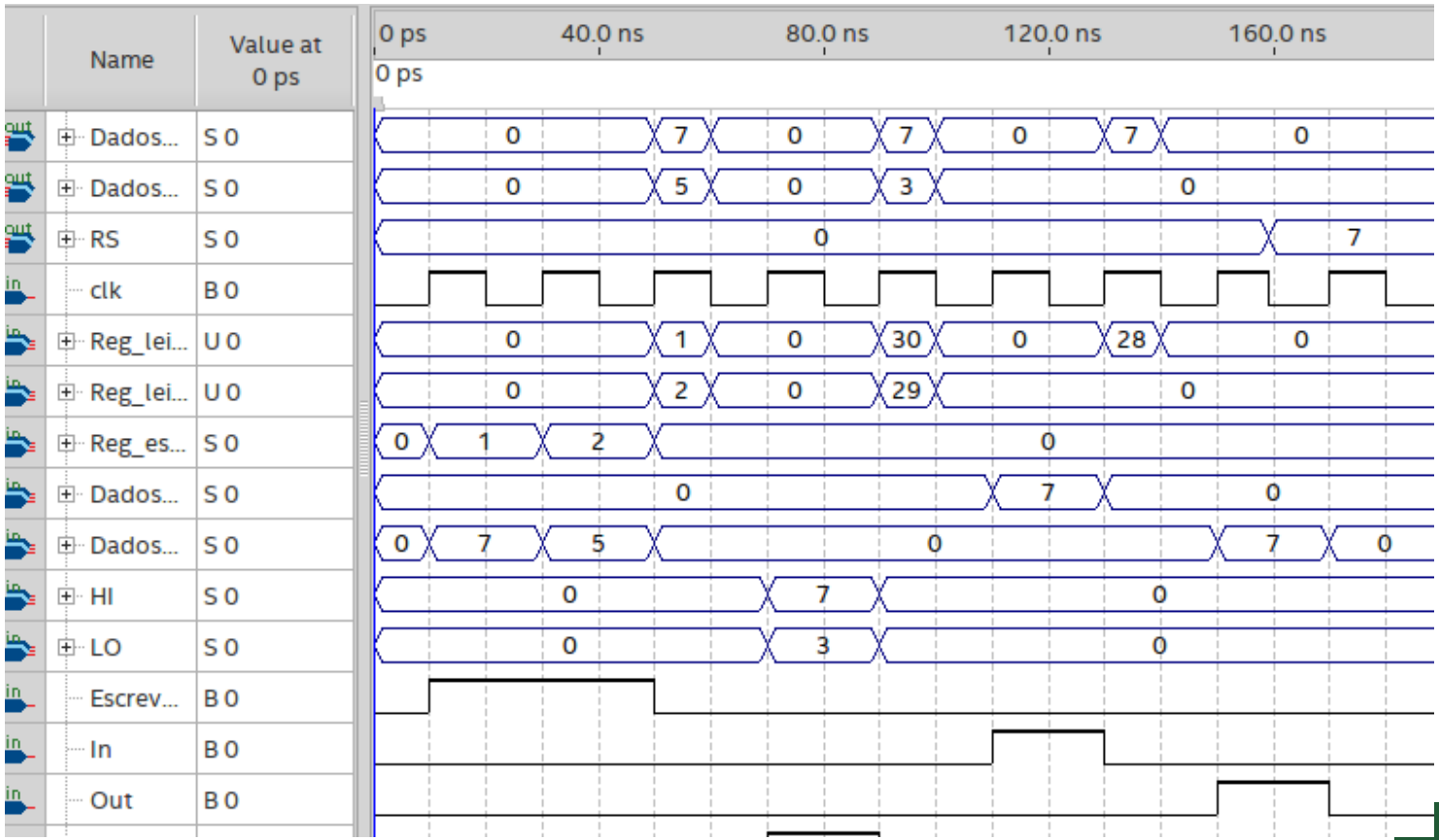
```
module unidade_Controle(Opcode, WriteHILO, EscreveReg, OrigALU, LeMem, EscreveMem, MemparaReg,
WriteRa, BifNot, Slt, Jump, JumpR, Out, In, Break, Branch, OpALU, RegDst);

    input [5:0] Opcode;

    output reg WriteHILO, EscreveReg, OrigALU, LeMem, EscreveMem, MemparaReg,
    WriteRa, BifNot, Slt, Jump, JumpR, Out, In, Break, Branch;

    output reg [1:0] OpALU, RegDst;

    always @(*)
    begin

       case(Opcode)

          6'b000001:
             begin
                WriteHILO = 0;
                EscreveReg = 1;
                OrigALU = 0;
                LeMem = 0;
                EscreveMem = 0;
                MemparaReg = 0;
                WriteRa = 0;
                BifNot = 0;
                Slt = 0;
                Jump = 0;
                JumpR = 0;
                Out = 0;
                In = 0;
                Break = 0;
                Branch = 0;
                OpALU = 2'b00;
                RegDst = 2'b01;
             end
```

Unidade de Controle da ULA

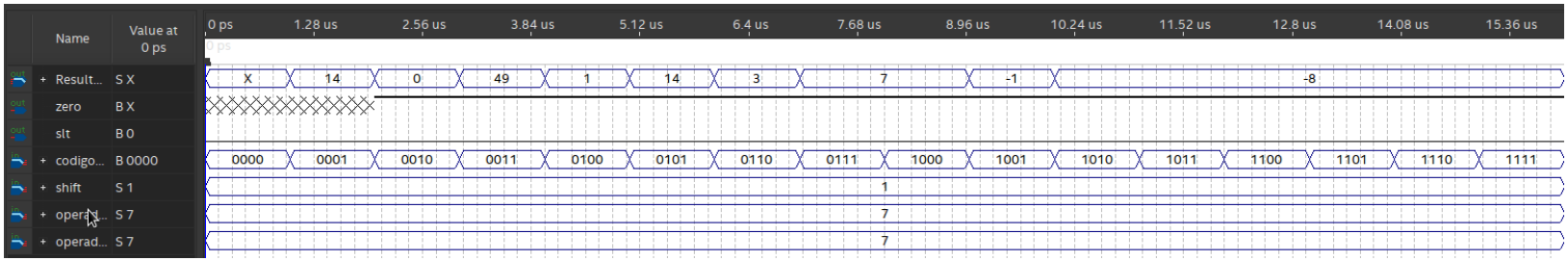
```
1  module controle_ALU (OpAlu, Opcode, controle_ULA);
2
3     input [1:0] OpAlu;
4     input [5:0] Opcode;
5
6     output reg [3:0] controle_ULA;
7
8     always @(*)
9     begin
10       if (OpAlu == 2'b00) controle_ULA = 4'b0001;
11       if (OpAlu == 2'b01) controle_ULA = 4'b0010;
12
13       if (OpAlu == 2'b10) begin
14
15          case (Opcode)
16             6'b000101: controle_ULA = 4'b0011;
17             6'b000111: controle_ULA = 4'b0100;
18             6'b001011: controle_ULA = 4'b0110;
19             6'b001100: controle_ULA = 4'b0101;
20             6'b001101: controle_ULA = 4'b0111;
21             6'b001110: controle_ULA = 4'b0111;
22             6'b001111: controle_ULA = 4'b1000;
23             6'b010000: controle_ULA = 4'b1000;
24             6'b010001: controle_ULA = 4'b1001;
25             6'b010010: controle_ULA = 4'b1001;
26             6'b010011: controle_ULA = 4'b1010;
27             6'b010110: controle_ULA = 4'b1011;
28             default: controle_ULA = 4'b0000;
29          endcase
30       end
31
32    end
33
34 endmodule
```

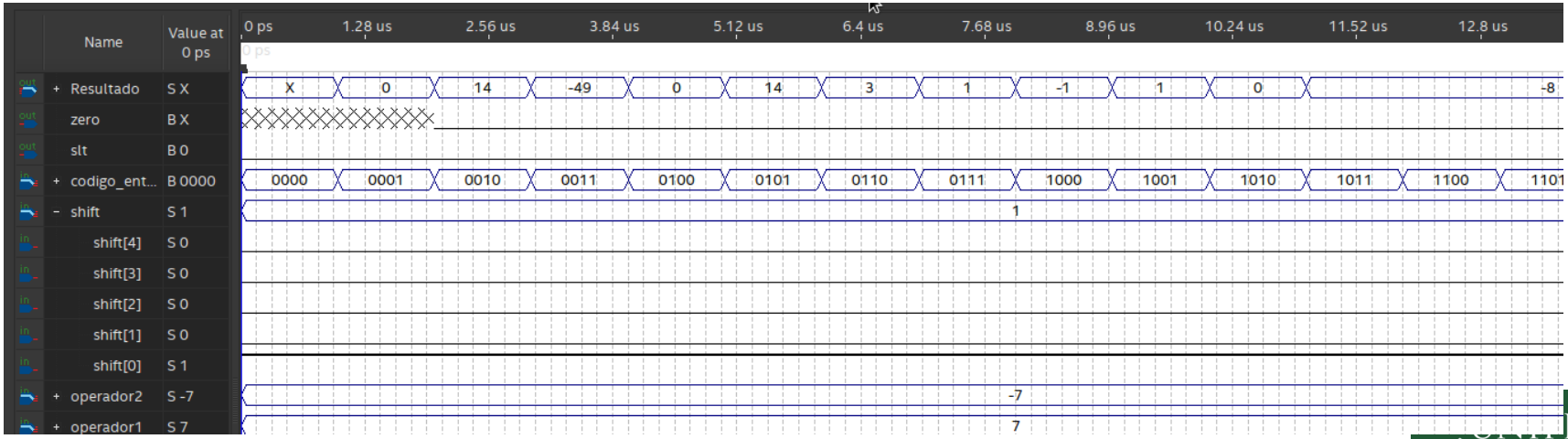
Simulações

Simulação do banco de registradores

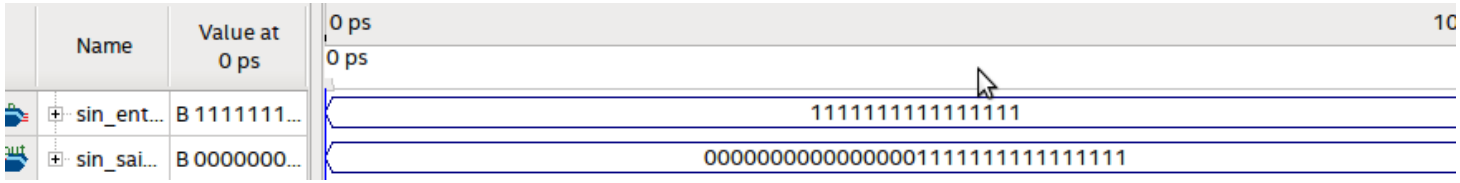


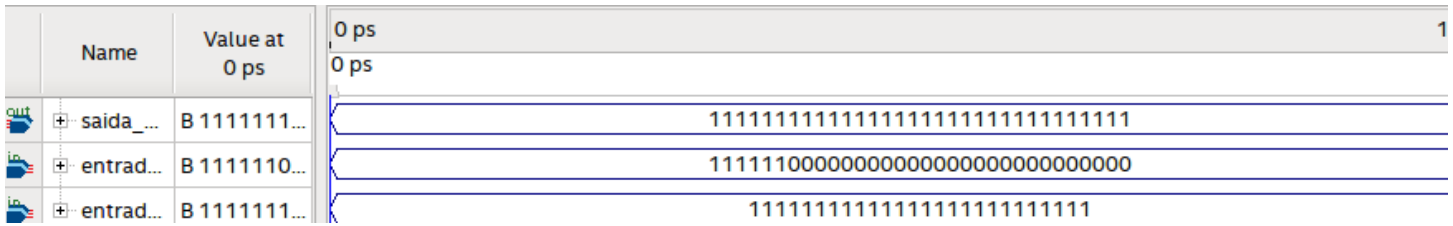
Simulações da ULA



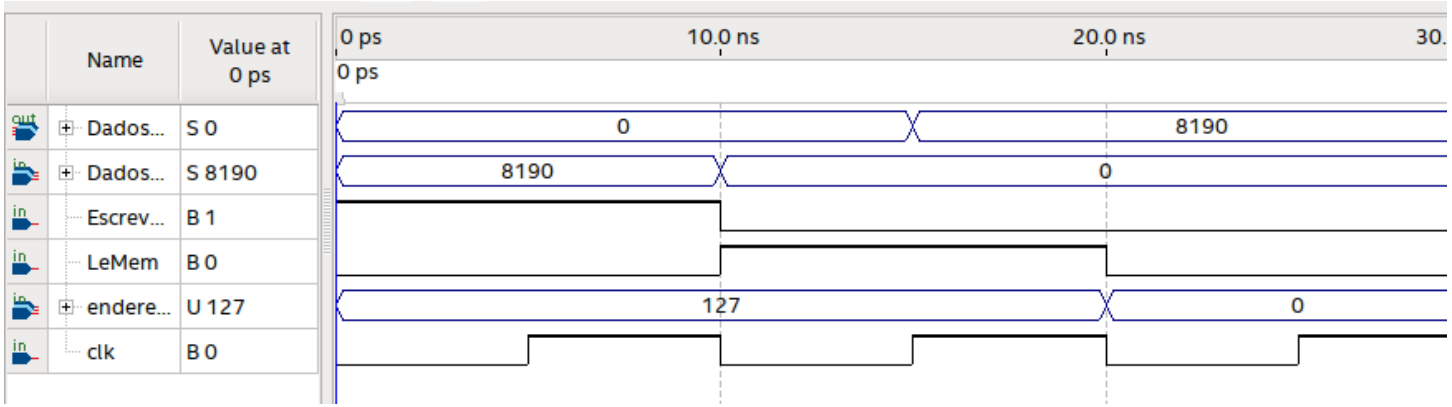


Simulações do concatenador e do extensor

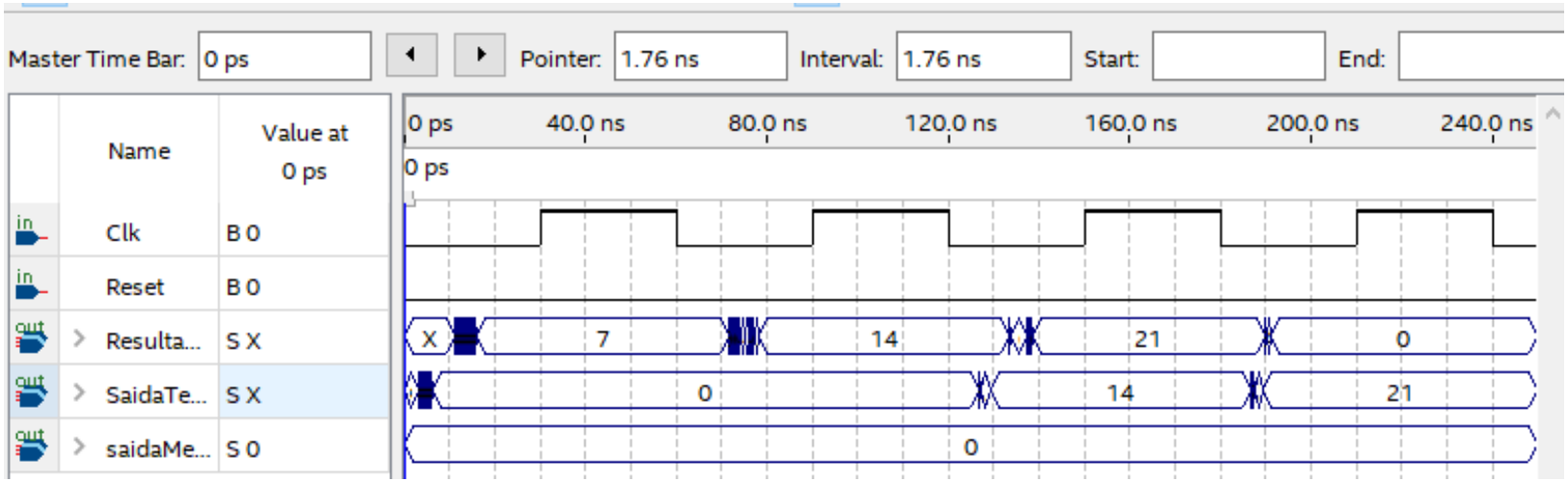




Simulação da memória de dados



Simulação de instruções de adição



Considerações Finais

A principal dificuldade encontrada nesta etapa do projeto foi com a manipulação do *software* utilizado, que apresenta vários *bugs* e inconsistências, tanto no momento de compilação como no momento de simulação. Mas, com algumas adaptações para contornar esses problemas, foi possível concluir a proposta do Ponto de Checagem 3 de modo satisfatório. As próximas etapas na finalização do projeto incluem:

- Simulação de cada instrução individualmente e correção de possíveis complicações;
- Projeto de um divisor de frequência para possibilitar que o processador funcione com o *clock* nativo do fpga sem maiores complicações;
- Projeto de um circuito decodificador de binário para BCD, e de um circuito de BCD para o display de 7 segmentos, para que a saída seja exibida adequadamente;
- Implementação de algoritmos básicos utilizando o conjunto de instruções.

Apêndices

Conjunto de Instruções

Instruções de operações aritméticas:

- add;
- addi;
- sub;
- subi;
- mult;
- div.

Instruções de transferência de informação:

- lw;
- sw.

Instruções de operações lógicas e comparações:

- and;
- andi;
- or;
- ori;
- xor;
- xori;
- nor;
- slt;
- slti;
- not.

Instruções de deslocamento:

- sra;
- sla.

Instruções de salto e desvio:

- j;
- jr;
- beq;
- bne;
- jal

Instruções de parada, entrada e saída:

- In;
- Out;
- break.

Formatos das instruções

Instruções do tipo R					
Opcode	R1	R2	Rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Instruções do tipo I			
Opcode	R1	Rd	Imediato
6 bits	5 bits	5 bits	16 bits

Instruções do tipo J	
Opcode	Endereço
6 bits	26 bits

Modos de endereçamento

- **Imediato**, onde o operando é uma constante. Será utilizado para instruções do tipo I de operações lógicas e aritméticas. (addi)
- **Por Registrador**, onde as constantes da instrução carregam um endereço para um registrador. Será utilizado para todas as instruções do tipo R. (add)
- **Por deslocamento**, onde o operando é um endereço de memória estabelecido a partir da soma de um registrador e uma constante na instrução. Será utilizado para instruções de transferência de informação do tipo I. (lw)
- **Relativo a PC**, usado para saltos do tipo I, é semelhante ao endereçamento por deslocamento, mas o registrador é o PC. (beq)
- **Pseudo Direto**, usado para instruções do tipo J, o endereço do desvio é deslocado 2 bits à esquerda e concatenado com os 4 bits mais significativos do PC. (j)

Conjunto de Registradores

- Registradores High(HI) e Low(LO).
- 10 registradores temporários.
- 10 registradores de uso geral.
- 5 registradores que armazenam argumentos para funções.
- 2 registradores que armazenam resultados das funções
- Um registrador de endereço ($ra), utilizado para armazenar o endereço de retorno da instrução **jal**.
- Um registrador de pilha, que armazena o endereço do topo de uma pilha na memória.
- Um registrador para armazernar o endereço da próxima instrução a ser executada. Este registrador não está no banco de registradores. No diagrama em bloco, ele é representado pelo módulo com o rótulo "PC" (Program Counter).
- Um registrador de saída;
- Um registrador de entrada;

Código desenvolvido do Banco de Registradores

```
module banco_Registradores (Reg_leitura1, Reg_leitura2, Reg_escrita, Dados_entrada, Dados_escrita, Dados_leitura1,
Dados_leitura2, RS, EscreveReg, WriteHILO, In, ResultadoHILO, clk, Out, saidaTeste);

  parameter RA = 5'b11111;
  parameter HI = 5'b11110;
  parameter LO = 5'b11101;
  parameter RE = 5'b11100;
  parameter ZERO = 5'b11010;

  input [4:0] Reg_leitura1;
  input [4:0] Reg_leitura2;
  input [31:0] Dados_escrita;
  input [31:0] Dados_entrada;
  input clk;
  input EscreveReg, WriteHILO, Out, In;
  input [63:0] ResultadoHILO;
  input [4:0] Reg_escrita;

  output wire [31:0] Dados_leitura1;
  output wire [31:0] Dados_leitura2;
  output reg [31:0] RS;
  output reg [31:0] saidaTeste; /// Usado somente para testar o processador


  reg [31:0] registradores_banco [31:0];

  assign Dados_leitura1 = registradores_banco[Reg_leitura1]; // LEITURA DOS REGISTRADORES
  assign Dados_leitura2 = registradores_banco[Reg_leitura2];

  always @ (negedge clk) // ESCRITA NOS REGISTRADORES
  begin
     registradores_banco[ZERO] = 0;

     if(WriteHILO)
        begin
        registradores_banco[HI] = ResultadoHILO[63:32];
        registradores_banco[LO] = ResultadoHILO[31:0];
        end
     if(EscreveReg) registradores_banco[Reg_escrita] = Dados_escrita;
     if (Out) RS = Dados_escrita;
     if (In) registradores_banco[RE] = Dados_entrada;
     saidaTeste = registradores_banco[LO];
  end
  endmodule
```

Código desenvolvido do Concatenador e do Extensor de Sinal

```
1  module concatenador (entrada_AddPC, entrada_jump, saida_endJump);
2     input [31:0] entrada_AddPC;
3     input [25:0] entrada_jump;
4
5     output wire [31:0] saida_endJump;
6     assign saida_endJump = { entrada_AddPC [31:26] , entrada_jump };
7
8  endmodule
9
```

```
1   module extensor_sinal (sin_entrada, sin_saida);
2
3   input [15:0] sin_entrada;
4
5   output wire [31:0] sin_saida;
6   reg [15:0] extend = 16'b0000000000000000;
7
8   assign sin_saida = {extend, sin_entrada};
9
10  endmodule
11
```

Código desenvolvido para a Unidade de Controle da ULA

```
1  module controle_ALU (OpAlu, Opcode, controle_ULA);
2
3      input [1:0] OpAlu;
4      input [5:0] Opcode;
5
6      output reg [3:0] controle_ULA;
7
8      always @(*)
9      begin
10         if (OpAlu == 2'b00) controle_ULA = 4'b0001;
11         if (OpAlu == 2'b01) controle_ULA = 4'b0010;
12
13         if (OpAlu == 2'b10) begin
14
15             case (Opcode)
16                 6'b000101: controle_ULA = 4'b0011;
17                 6'b000111: controle_ULA = 4'b0100;
18                 6'b001011: controle_ULA = 4'b0110;
19                 6'b001100: controle_ULA = 4'b0101;
20                 6'b001101: controle_ULA = 4'b0111;
21                 6'b001110: controle_ULA = 4'b0111;
22                 6'b001111: controle_ULA = 4'b1000;
23                 6'b010000: controle_ULA = 4'b1000;
24                 6'b010001: controle_ULA = 4'b1001;
25                 6'b010010: controle_ULA = 4'b1001;
26                 6'b010011: controle_ULA = 4'b1010;
27                 6'b010110: controle_ULA = 4'b1011;
28                 default: controle_ULA = 4'b0000;
29             endcase
30         end
31
32     end
33
34 endmodule
```

Código desenvolvido para a Memória de Dados e de Instruções

```
module mem_Dados (clk, endereco, Dados_escrita, Dados_leitura, EscreveMem, LeMem, saidaMem1);

input clk, EscreveMem, LeMem;
input [31:0] Dados_escrita;
input [31:0] endereco;
output reg [31:0] Dados_leitura, saidaMem1;

reg [31:0] memoria [255:0];

always @ (posedge clk)
begin

   if (EscreveMem) memoria[endereco] <= Dados_escrita;
   if (LeMem) Dados_leitura <= memoria[endereco];
   saidaMem1 = memoria[1];

end

endmodule
```

```
module mem_Instrucao(endereco_leitura, clk, instrucao);

   input [31:0]endereco_leitura;
   input clk;
   integer programa = 1;

   reg [31:0]mem_instrucao[255:0];

   output [31:0] instrucao;

   always @(posedge clk)
   begin
      if(programa == 1)
         begin
////////////////////INSTRUCOES DO PROGRAMA/////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////////////////////////
         end
    programa = 0;
   end
   assign instrucao = mem_instrucao[endereco_leitura];

endmodule
```

Amostra de código da interligação de todos os módulos

```
1   module processador (Clk, Chaves, saida_Display, Reset, SaidaTeste, Resultado_ULA, saidaMemDados);
2
3      input Clk, Reset;
4      input [9:0] Chaves;
5      output wire [31:0] saida_Display, SaidaTeste, saidaMemDados; //// ULTIMA VARIAVEL PARA TESTE, RETIRAR DEPOIS
6
7
8      //SINAIS DE CONTROLE/////////////////////////////////////////////////////////
9      wire WriteHILOwire, EscreveRegwire, OrigALUwire,
10     LeMemwire, EscreveMemwire, MemparaRegwire, WriteRawire, BifNotwire, Sltwire,
11     Jumpwire, JumpRwire, Outwire, Inwire, Breakwire, Branchwire;
12
13     wire [1:0] OpALUwire, RegDstwire;
14
15     wire[3:0] controleALU;
16
17     ////////////////////////////////////////
18
19     output wire [63:0] Resultado_ULA;
20
21     wire [31:0] Saida_addPC, saida_moduloEntrada, saida_leitura2, saida_RS,
22     saida_Concatenador, saida_AddB, saida_MemData, saidaPC, saida_Muxg, saida_Muxf,
23     Instrucao, saida_Extensor, saida_Muxe, saida_leitura1, saida_Muxk, saida_Muxd;
24
25     wire [4:0] saida_Muxc;
26
27     wire [25:0] saida_Muxa;
28
29     wire [4:0] saida_Muxb;
30
31     wire slt_ula, zero_ula, saida_branch_logic;
```

```
unidade_Controle UndControle(
      .Opcode(Instrucao[31:26]),
      .WriteHILO(WriteHILOwire),
      .EscreveReg(EscreveRegwire),
      .OrigALU(OrigALUwire),
      .LeMem(LeMemwire),
      .EscreveMem(EscreveMemwire),
      .MemparaReg(MemparaRegwire),
      .WriteRa(WriteRawire),
      .BifNot(BifNotwire),
      .Slt(Sltwire),
      .Jump(Jumpwire),
      .JumpR(JumpRwire),
      .Out(Outwire),
      .In(Inwire),
      .Break(Breakwire),
      .Branch(Branchwire),
      .OpALU(OpALUwire),
      .RegDst(RegDstwire)

);
controle_ALU UndContALU(
      .OpAlu(OpALUwire),
      .Opcode(Instrucao[31:26]),
      .controle_ULA(controleALU)

);


Program_counter pc(
      .breakk(Breakwire),
      .entrada_PC(saida_Muxg),
      .saida_PC(saidaPC),
      .clk(Clk),
      .reset(Reset)

);
```