UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
CONCEPÇÃO ESTRUTURADA DE CIRCUITOS INTEGRADOS

PROJETO MIPS – ENTREGA 8

THIAGO ALVES DE ARAUJO

2016019787

Sumário

1 Tabela de figuras

2 DataPath

Após desenvolvermos todos os modulos necessarios para o mips32, vamos junta-los para construir todo o caminho de dados. Abaixo podemos ver os diagramas de blocos de cada modulo separado e os fios que fazem suas conexoes (os fios destacados em laranja são os sinais da unidade de controle)

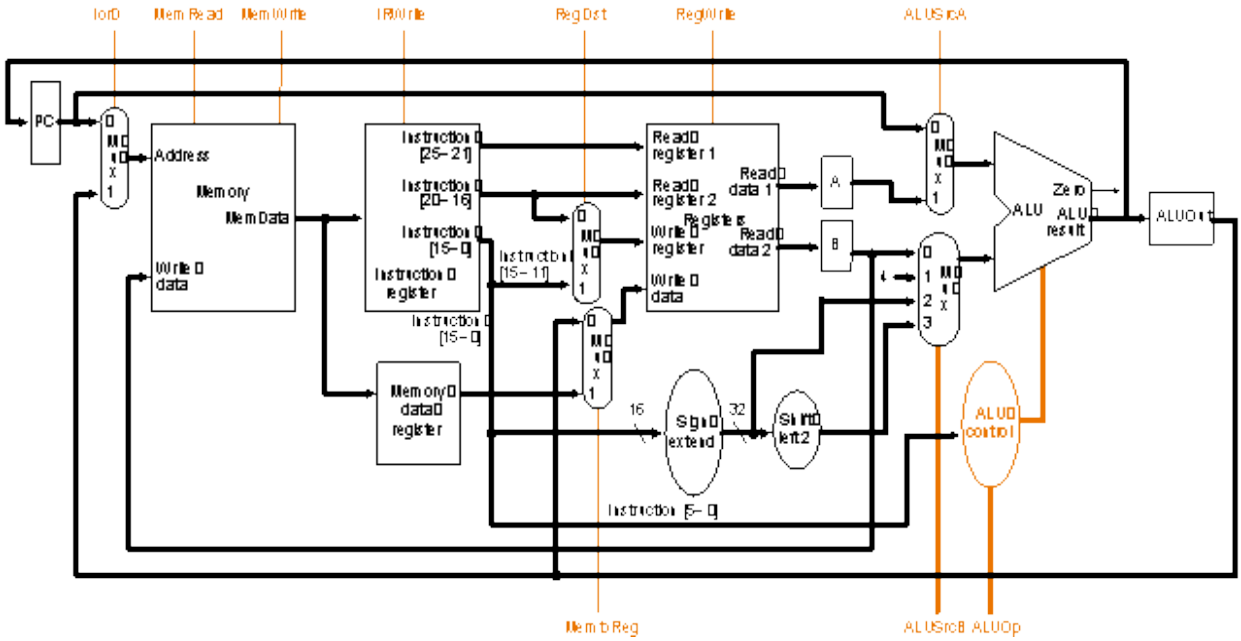

Figura 1 Datapath - Diagrama de blocos

2.1 Goldem model

Abaixo podemos ver um trecho do código utilizado para simular todo o caminho de dados e gerar os *goldem vectors*

.

```
void DataPath(const vector<string> &RD, const vector<string> &clk, const vector<string> &reset, const vector<string> &IorD,
              const vector<string> &IRWrite, const vector<string> &RegDst, const vector<string> &MemtoReg,
              const vector<string> &RegWrite, const vector<string> &AluSrcA, const vector<string> &Branch,
              const vector<string> &PCWrite, const vector<string> &AluSrcB, const vector<string> &PCSrc,
              const vector<string> &AluControl){

    vector<string> writeFile, outInst, outData, aluOut, outPC, registers;
    writeFile.resize(RD.size());
    outInst.resize(RD.size());
    outData.resize(RD.size());
    aluOut.resize(RD.size());
    outPC.resize(RD.size());
    registers.resize(32);

    string A1, A2, A3, muxA3_1, muxA3_2, muxWD3_1, muxWD3_2, WD3, SrcA, SrcB, outSignExtend, shifterOutExtend;
    string zero, ovf, aluResult, sAnd, enPC, jumpPC, inPC, adr;
    pair<string, string> outA_B;
```

Figura 2 Datapath - Goldem Model

2.2 Goldem vector

A partir do goldem model mostrado anteriormente geramos os seguintes vetores de teste. Foram criados 32 vetores com cada um possuindo 114 bits que serão descritos adiante.

```
0_1_0_0_0_0_0_0_00_000_0_0_00_00000000000000000000000000000000
_00000000000000000000000000000000_
00000000000000000000000000000000_0

1_1_0_0_0_0_0_0_00_000_0_0_00_00000000000000000000000000000000
_00000000000000000000000000000000_
00000000000000000000000000000000_0

0_0_0_1_0_0_0_0_01_010_0_1_00_00100000000000010000000000000101
_00000000000000000000000000000000_
00000000000000000000000000000000_0

1_0_0_1_0_0_0_0_01_010_0_1_00_00100000000000010000000000000101
_00000000000000000000000000000100_
00000000000000000000000000000000_0

0_0_0_0_0_0_0_0_11_010_0_0_00_00100000000000010000000000000101
_00000000000000000000000000000100_
00000000000000000000000000000000_0

1_0_0_0_0_0_0_0_11_010_0_0_00_00100000000000010000000000000101
_00000000000000000000000000000100_
00000000000000000000000000000000_0
```

Figura 3 Datapath - Goldem vectors

2.3 Testbench

Abaixo podemos ver o código em SystemVerilog responsável por testar o Datapath. O testbench segue o padrão dos códigos desenvolvidos anteriormente.

```
`timescale 1ns/10ps

module Datapath_tb;

   logic clk, reset;
   logic ck, reset_;
   int counter, errors, aux_error;
   logic IorD, RegDest, MemtoReg, IRWrite, RegWrite, ALUSrcA, Branch, PCWrite;
   logic[1:0] ALUSrcB, PCSrc;
   logic[2:0] ALUControl;
   logic[31:0] RD;
   logic[31:0] Adr;
   logic[31:0] WD;
   logic overflow;
   logic[31:0] Adr_expected;
   logic[31:0] WD_expected;
   logic overflow_expected;
   logic [113:0] testvectors[32];

   Datapath duv(.ck(ck), .reset_(reset), .IorD(IorD), .RegDest(RegDest), .MemtoReg(MemtoReg), .IRWrite(IRWrite),
             .RegWrite(RegWrite), .ALUSrcA(ALUSrcA), .ALUSrcB(ALUSrcB), .ALUControl(ALUControl), .PCSrc(PCSrc),
             .Branch(Branch), .PCWrite(PCWrite), .WD(WD), .Adr(Adr), .RD(RD), .overflow(overflow));
   initial begin
      $display("Iniciando Testbench");
      $readmemb("Datapath.tv",testvectors);
      counter=0; errors=0;
      reset = 1;
      #16;
      reset = 0;
   end

   always begin
      clk=1; #11;
      clk=0; #5;
   end
  always @(posedge clk)begin
     if(~reset)begin
        //Atribui valores do vetor nas entradas do DUT e nos valores esperados
        {ck, reset_, IorD, IRWrite, RegDest, MemtoReg, RegWrite, ALUSrcA, ALUSrcB, ALUControl,
        Branch, PCWrite, PCSrc, RD, Adr_expected, WD_expected, overflow_expected} = testvectors[counter];
     end
  end

  always @(negedge clk)begin
     if(~reset)begin

        aux_error = errors;

        assert (Adr === Adr_expected)
        else
        begin
           $display(" Adr = %b, %b expected", Adr, Adr_expected);
           errors = errors + 1;
        end

        assert (WD === WD_expected)
        else
        begin
           $display(" WD = %b, %b expected", WD, WD_expected);
           errors = errors + 1;
        end

        assert (overflow === overflow_expected)
        else
        begin
           $display("overflow = %b, %b expected", overflow, overflow_expected);
           errors = errors + 1;
        end

        if(aux_error === errors)begin
           $display("| ck | reset | IorD | RegDest | MemtoReg | IRWrite | RegWrite | ALUSrcA | Branch | PCWrite | ALUSrcB | PCSrc | ALUControl
           $display("| %b  |   %b   |  %b  |    %b    |     %b     |    %b    |     %b    |    %b   |    %b   |   %b    |   %b  |  %b   |
           $display("|              RD                |              Adr                |              WD                | Overflow |");
           $display("|              RD                |              Adr                |              WD                | Overflow |");
           $display("| %b | %b | %b |    %b     | OK", RD, Adr, WD, overflow);
           $display(" ");
        end
        else begin
           $display("| ck | reset | IorD | RegDest | MemtoReg | IRWrite | RegWrite | ALUSrcA | Branch | PCWrite | ALUSrcB | PCSrc | ALUControl |");
           $display("| %b  |   %b   |  %b  |    %b    |     %b     |    %b    |     %b    |    %b   |    %b   |   %b    |   %b  |  %b   |   %b
           $display("|              RD                |              Adr                |              WD                | Overflow |");
           $display("| %b | %b | %b |    %b     | ERRO", RD, Adr, WD, overflow);
           $display(" ");
        end
        counter++;

        if(counter == $size(testvectors))begin

           $display("Testes Efetuados  = %0d", counter);
           $display("Erros Encontrados = %0d", errors);
           $stop;
        end
     end
  end

endmodule
```

Figura 4 Datapath – Testbench

2.4 Modelo duv

Para o Datapath temos as seguintes entradas: ck, reset_, IorD, RegDest, MemtoReg, IRWrite, RegWrite, ALUSrcA, ALUSrcB, ALUControl, PCSrc, Branch, PCWrite, WD. Destes, apenas o WD (Write Data) não é gerado pela unidade de controle. Este dado é inserido para simular a escrita na memória. Os sinais de saída são: Adr (Address) que é resultado da saída do primeiro mux2

após o PC, RD que é a saída da memória que entra no banco de registradores e por fim o overflow (sinal da ula). Abaixo podemos ver como o modulo do Datapath ficou. Baseando-se no diagrama de blocos apresentado anteriormente, os módulos foram instanciados seguindo a sequencia da esquerda para a direita da imagem.

```
module Datapath(ck, reset_, IorD, RegDest, MemtoReg, IRWrite, RegWrite, ALUSrcA, ALUSrcB, ALUControl, PCSrc, Branch, PCWrite, WD, Adr, RD, overflow);

	input logic ck, reset_;
	input logic IorD, RegDest, MemtoReg, IRWrite, RegWrite, ALUSrcA, Branch, PCWrite;
	input logic[1:0] ALUSrcB, PCSrc;
	input logic[2:0] ALUControl;
	input logic[31:0] RD;
	output logic[31:0] Adr;
	output logic[31:0] WD;
	output logic overflow;

	logic zero, outputAND_PC, PCEn;
	logic[31:0] inputPC, outputPC, outputRegInstr, outputRegData, outputRegA, outputRegB, WD3, RD1, RD2;
	logic[31:0] SrcA, SrcB, outputSignalExtension, outputShifter2, ALUOut, ALUResult;
	logic[4:0] outputMux5;
	logic[31:0] inputShifter2_2, outputShifterJump, jumpAddr;

	and andPC(outputAND_PC, zero, Branch);
	or orPC(PCEn, outputAND_PC, PCWrite);

	//Flopenr de 32 bits
	registradorEnReset PC(ck, reset_, PCEn, inputPC, outputPC);

	//Mux2 de 32 bits
	mux2 mux2_1(outputPC, ALUOut, IorD, Adr);

	//Flopenr de 32 bits
	registradorEnReset RegInstr(ck, reset_, IRWrite, RD, outputRegInstr);

	//Flopr de  32 bits
	registradorReset RegData(ck, reset_, RD, outputRegData);

	//Mux2 de 5 bits
	mux2_5bits mux2_5bits(outputRegInstr[20:16], outputRegInstr[15:11], RegDest, outputMux5);

	//Mux2 de 32 bits
	mux2 mux2_2(ALUOut, outputRegData, MemtoReg, WD3);

	bancoRegistradores bancoRegistradores(outputRegInstr[25:21], outputRegInstr[20:16], outputMux5, WD3, RegWrite, ck, reset_, RD1, RD2);

	//Flopr de 32 bits
	registradorReset RegA(ck, reset_, RD1, outputRegA);
	registradorReset RegB(ck, reset_, RD2, outputRegB);

	assign WD = outputRegB;

	//Mux2 de 32 bits
	mux2 mux2_3(outputPC, outputRegA, ALUSrcA, SrcA);

	//Extensor de sinal
	signalExtension signalExtension(outputRegInstr[15:0], outputSignalExtension);

	//Deslocador
	shifter2 shifter2_1(outputSignalExtension, outputShifter2);

	//Mux4 de 32 bits
	mux4 mux4_1(outputRegB, 32'b00000000000000000000000000000100, outputSignalExtension, outputShifter2, ALUSrcB, SrcB);

	//ULA de 32 bits
	ula32bits ula(ALUControl, SrcA, SrcB, ALUResult, overflow, zero);

	//Flopr de 32 bits
	registradorReset RegULA(ck, reset_, ALUResult, ALUOut);

	assign inputShifter2_2[25:0] = outputRegInstr[25:0];
	assign inputShifter2_2[31:26] = 6'b000000;

	//Deslocador
	shifter2 shifter2_2(inputShifter2_2, outputShifterJump);

	assign jumpAddr = {outputPC[31:28], outputShifterJump[27:0]};

	//Mux4 de 32 bits
	mux4 mux4_2(.d0(ALUResult), .d1(ALUOut), .d2(jumpAddr), .s(PCSrc), .out(inputPC));

endmodule
```

Figura 5 Datapath - Modelo DUV

2.5 Simulação RTL

Com o modulo pronto, foi iniciado a simulação RTL. Como podemos ver, o modulo está se comportando como o esperado e nenhum erro foi encontrado.

```
# | ck | reset | IorD | RegDest | MemtoReg | IRWrite | RegWrite | ALUSrcA | Branch | PCWrite | ALUSrcB | PCSrc | ALUControl |
# | 1  |   0   |  0   |    0    |    0     |    0    |    1     |    1    |   0    |    0    |   10    |  00   |    010     |
# |                RD                |                Adr                |                WD                | Overflow |
# | 00000000000000000000000000000011 | 00000000000000000000000000001100 | 00000000000000000000000000000000 |    0     | OK
#
# | ck | reset | IorD | RegDest | MemtoReg | IRWrite | RegWrite | ALUSrcA | Branch | PCWrite | ALUSrcB | PCSrc | ALUControl |
# | 0  |   0   |  0   |    0    |    0     |    1    |    0     |    0    |   0    |    1    |   01    |  00   |    010     |
# |                RD                |                Adr                |                WD                | Overflow |
# | 00001000000000000000000000000010 | 00000000000000000000000000001100 | 00000000000000000000000000000000 |    0     | OK
#
# | ck | reset | IorD | RegDest | MemtoReg | IRWrite | RegWrite | ALUSrcA | Branch | PCWrite | ALUSrcB | PCSrc | ALUControl |
# | 1  |   0   |  0   |    0    |    0     |    1    |    0     |    0    |   0    |    1    |   01    |  00   |    010     |
# |                RD                |                Adr                |                WD                | Overflow |
# | 00001000000000000000000000000010 | 00000000000000000000000000010000 | 00000000000000000000000000000010 |    0     | OK
#
# | ck | reset | IorD | RegDest | MemtoReg | IRWrite | RegWrite | ALUSrcA | Branch | PCWrite | ALUSrcB | PCSrc | ALUControl |
# | 0  |   0   |  0   |    0    |    0     |    0    |    0     |    0    |   0    |    0    |   11    |  00   |    010     |
# |                RD                |                Adr                |                WD                | Overflow |
# | 00001000000000000000000000000010 | 00000000000000000000000000010000 | 00000000000000000000000000000010 |    0     | OK
#
# | ck | reset | IorD | RegDest | MemtoReg | IRWrite | RegWrite | ALUSrcA | Branch | PCWrite | ALUSrcB | PCSrc | ALUControl |
# | 1  |   0   |  0   |    0    |    0     |    0    |    0     |    0    |   0    |    0    |   11    |  00   |    010     |
# |                RD                |                Adr                |                WD                | Overflow |
# | 00001000000000000000000000000010 | 00000000000000000000000000010000 | 00000000000000000000000000000000 |    0     | OK
#
# Testes Efetuados  = 32
# Erros Encontrados = 0
```

Figura 6 Datapath - Transcript da simulação RTL

Também podemos ver o *RTL view* gerado a partir do modulo testado.

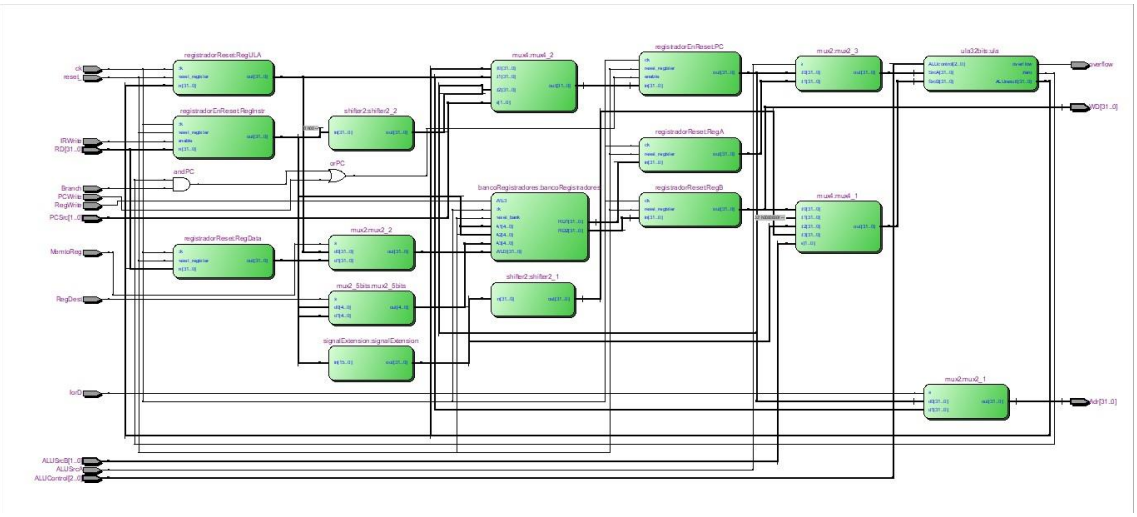

Figura 7 Datapath - RTL view

2.6 Simulação *Gate Level*

Com a simulação RTL feita, vamos testar o modulo na simulação Gate level para verificar o tempo mínimo de funcionamento levando em conta os atrasos, etc. Iniciando o clock com #9, podemos ver que 3 erros foram encontrados

```
# | ck | reset | IorD | RegDest | MemtoReg | IRWrite | RegWrite | ALUSrcA | Branch | PCWrite | ALUSrcB | PCSrc | ALUControl |
# | 1  |   0   |  0   |    0    |    0     |    1    |    0     |    0    |   0    |    1    |   01    |  00   |    010     |
# |                RD                |                Adr                |                WD                | Overflow |
# | 00001000000000000000000000000010 | 00000000000000000000000000010000 | 00000000000000000000000000000010 |    0     | OK
#
# | ck | reset | IorD | RegDest | MemtoReg | IRWrite | RegWrite | ALUSrcA | Branch | PCWrite | ALUSrcB | PCSrc | ALUControl |
# | 0  |   0   |  0   |    0    |    0     |    0    |    0     |    0    |   0    |    0    |   11    |  00   |    010     |
# |                RD                |                Adr                |                WD                | Overflow |
# | 00001000000000000000000000000010 | 00000000000000000000000000010000 | 00000000000000000000000000000010 |    0     | OK
#
# | ck | reset | IorD | RegDest | MemtoReg | IRWrite | RegWrite | ALUSrcA | Branch | PCWrite | ALUSrcB | PCSrc | ALUControl |
# | 1  |   0   |  0   |    0    |    0     |    0    |    0     |    0    |   0    |    0    |   11    |  00   |    010     |
# |                RD                |                Adr                |                WD                | Overflow |
# | 00001000000000000000000000000010 | 00000000000000000000000000010000 | 00000000000000000000000000000000 |    0     | OK
#
# Testes Efetuados  = 32
# Erros Encontrados = 3
```

Figura 8 Datapath - clk #9

Aumentando o tempo para #10, o número de erros cai para apenas um.

```
# | ck | reset | IorD | RegDest | MemtoReg | IRWrite | RegWrite | ALUSrcA | Branch | PCWrite | ALUSrcB | PCSrc | ALUControl |
# | 0  |   0   |  0   |    0    |    0     |    0    |    0     |    0    |   0    |    0    |   11    |  00   |    010     |
# |                RD                |                Adr                |                WD                | Overflow |
# | 00001000000000000000000000000010 | 00000000000000000000000000010000 | 00000000000000000000000000000010 |    0     | OK
#
# | ck | reset | IorD | RegDest | MemtoReg | IRWrite | RegWrite | ALUSrcA | Branch | PCWrite | ALUSrcB | PCSrc | ALUControl |
# | 1  |   0   |  0   |    0    |    0     |    0    |    0     |    0    |   0    |    0    |   11    |  00   |    010     |
# |                RD                |                Adr                |                WD                | Overflow |
# | 00001000000000000000000000000010 | 00000000000000000000000000010000 | 00000000000000000000000000000000 |    0     | OK
#
# Testes Efetuados  = 32
# Erros Encontrados = 1
```

Figura 9 Datapath - clk #10

Por fim, com o clock em #11 podemos ver que o modulo se comporta perfeitamente.

```
# | ck | reset | IorD | RegDest | MemtoReg | IRWrite | RegWrite | ALUSrcA | Branch | PCWrite | ALUSrcB | PCSrc | ALUControl |
# | 1  |   0   |  0   |    0    |    0     |    1    |    0     |    0    |   0    |    1    |   01    |  00   |    010     |
# |                RD                |                Adr                |                WD                | Overflow |
# | 00001000000000000000000000000010 | 00000000000000000000000000010000 | 00000000000000000000000000000010 |    0     | OK
#
# | ck | reset | IorD | RegDest | MemtoReg | IRWrite | RegWrite | ALUSrcA | Branch | PCWrite | ALUSrcB | PCSrc | ALUControl |
# | 0  |   0   |  0   |    0    |    0     |    0    |    0     |    0    |   0    |    0    |   11    |  00   |    010     |
# |                RD                |                Adr                |                WD                | Overflow |
# | 00001000000000000000000000000010 | 00000000000000000000000000010000 | 00000000000000000000000000000010 |    0     | OK
#
# | ck | reset | IorD | RegDest | MemtoReg | IRWrite | RegWrite | ALUSrcA | Branch | PCWrite | ALUSrcB | PCSrc | ALUControl |
# | 1  |   0   |  0   |    0    |    0     |    0    |    0     |    0    |   0    |    0    |   11    |  00   |    010     |
# |                RD                |                Adr                |                WD                | Overflow |
# | 00001000000000000000000000000010 | 00000000000000000000000000010000 | 00000000000000000000000000000000 |    0     | OK
#
# Testes Efetuados  = 32
# Erros Encontrados = 0
```

Figura 10 Datapath - clk #11

Abaixo podemos ver os sinais de entrada e saída do Datapath.

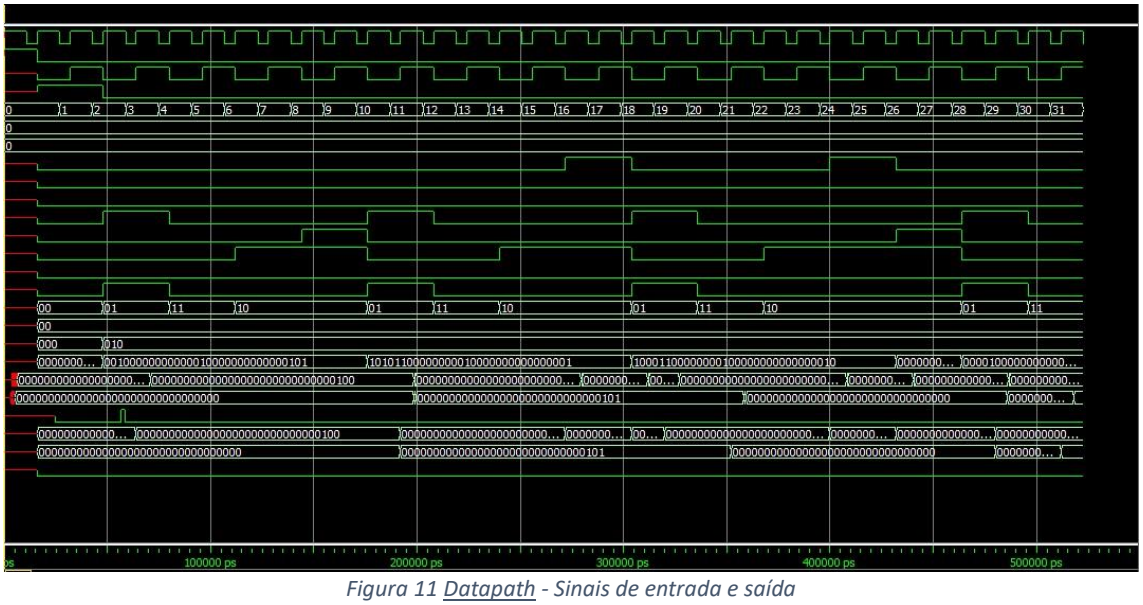

Figura 11 Datapath - Sinais de entrada e saída