

ID: .....

**Desenvolvimento de um sistema computacional  
completo-ProcessorE**

**Ponto de checagem 4: Inclusão da Unidade de  
controle e finalização do sistema  
Relatório apresentado no Laboratório de  
Arquitetura de Computadores**

São José dos Campos - Brasil

Abril de 2017



ID: .....

**Desenvolvimento de um sistema computacional  
completo-ProcessorE**

**Ponto de checagem 4: Inclusão da Unidade de controle e  
finalização do sistema**

**Relatório apresentado no Laboratório de Arquitetura de  
Computadores**

Relatório apresentado à Universidade Federal de São Paulo como parte dos requisitos para aprovação na disciplina de Laboratório de Sistemas Computacionais: Arquitetura e Organização de Computadores.

Docente: Prof. Dr. Tiago de Oliveira

Universidade Federal de São Paulo - UNIFESP

Instituto de Ciência e Tecnologia - Campus São José dos Campos

São José dos Campos - Brasil

Abril de 2017

# Resumo

O projeto consiste em implementar um sistema computacional completo, baseado na arquitetura Mips, utilizando como meios a linguagem de descrição de hardware verilog e FPGA. Nesta etapa a unidade de controle foi interligada a central de processamento e instruções foram executadas a fim de comprovar o funcionamento do processador. Através de simulações por software e no FPGA ficou comprovado o funcionamento o seu funcionamento.

**Palavras-chaves:** CPU. processor. central de processamento.unidade de controle. verilog.

# **Lista de ilustrações**

Figura 1 – Esquematico ProcessorE . . . . .	16
Figura 2 – Simulação de instruções por forma de onda . . . . .	27
Figura 3 – Execução de instruções no FPGA 1-botão amarelo = clock, chaves verdes = entrada, chave laranja = reset, chave rosa = <i>flag</i> de entrada .	29
Figura 4 – Execução de instruções no FPGA 2 . . . . .	29
Figura 5 – Execução de instruções no FPGA 3 . . . . .	30
Figura 6 – Execução de instruções no FPGA 4 . . . . .	30
Figura 7 – Execução de instruções no FPGA 5 . . . . .	31
Figura 8 – Execução de instruções no FPGA 6 . . . . .	31
Figura 9 – Execução de instruções no FPGA 7 . . . . .	31
Figura 10 – Execução de instruções no FPGA 8 . . . . .	32
Figura 11 – Execução de instruções no FPGA 9 . . . . .	32
Figura 12 – Execução de instruções no FPGA 10 . . . . .	33
Figura 13 – Execução de instruções no FPGA 11 . . . . .	33
Figura 14 – Execução de instruções no FPGA 12 . . . . .	33
Figura 15 – Execução de instruções no FPGA 13 . . . . .	34
Figura 16 – Execução de instruções no FPGA 14 . . . . .	34
Figura 17 – Execução de instruções no FPGA 15 . . . . .	34

# **Lista de tabelas**

Tabela 1 – Formato das Instruções . . . . .	15
Tabela 2 – Conjunto de instruções . . . . .	16

# Sumário

1	<b>INTRODUÇÃO</b>	7
2	<b>OBJETIVOS</b>	9
2.1	<b>Geral</b>	9
2.2	<b>Específico</b>	9
3	<b>FUNDAMENTAÇÃO TEÓRICA</b>	11
3.1	<b>Sistemas computacionais</b>	11
3.2	<b><i>Central Processor Unit (CPU)</i></b>	11
3.3	<b>Risc x Cisc</b>	12
3.4	<b>MIPS</b>	12
3.5	<b>Tipos de endereçamento</b>	12
3.6	<b>Verilog</b>	13
3.7	<b>FPGA</b>	13
3.8	<b>Quartus</b>	14
4	<b>DESENVOLVIMENTO</b>	15
4.1	<b>Unidade lógica aritimética</b>	17
4.2	<b>Banco de registradores</b>	18
4.3	<b>Memória de dados</b>	18
4.4	<b>Memória de Instruções</b>	19
4.5	<b><i>Program counter</i></b>	20
4.6	<b>Multiplexador</b>	21
4.7	<b>Extensor de Imediato</b>	22
4.8	<b>Unidade de Controle</b>	23
4.9	<b>IN e OUT</b>	24
4.10	<b>ProcessorE: unidades interligadas</b>	24
5	<b>RESULTADOS OBTIDOS E DISCUSSÕES</b>	27
6	<b>CONSIDERAÇÕES FINAIS</b>	37
	<b>REFERÊNCIAS</b>	39

<b>APÊNDICES</b>	<b>41</b>
<b>APÊNDICE A – UNIDADEC.V</b>	<b>43</b>
<b>APÊNDICE B – DEBOUNCER.V</b>	<b>53</b>
<b>APÊNDICE C – CONVERSOR BIN 32B PARA BCD</b>	<b>55</b>
<b>APÊNDICE D – DISPLAY DE 7 SEGMENTOS</b>	<b>57</b>
<b>APÊNDICE E – EXTIN.V</b>	<b>59</b>

# 1 Introdução

Vivendo na época atual considerada a era da informação, não é difícil encontrar diversas máquinas capazes de tratar grandes quantidades de dados gerados pela interação social, por imagens, sensores, entre outros. Essas máquinas responsáveis pelo armazenamento, processamento e cálculo dos dados são chamadas de computadores. No cerne dos computadores estão os processadores, responsáveis por transformar os dados contidos na máquina em informação para o usuário. Com o objetivo de adquirir mais conhecimento sobre essa tecnologia este projeto mostrará o processo de construção de um sistema computacional, desde seu planejamento a sua implementação([1](#)).



## 2 Objetivos

### 2.1 Geral

Desenvolver um sistema computacional completo que possua as seguintes características:

1. Uma *Central Processor Unit* (CPU).
2. Memória RAM(*Random Access Memory* ).
3. Interface de comunicação externa.
4. Seja implementado em FPGA (*Field-programmable Gate Array*) utilizando a linguagem de descrição de *hardware* verilog e o *software* quartus prime.

### 2.2 Específico

O ponto de checagem 4, descrito no decorrer desse relatório, possui como objetivo interligar a unidade de controle a unidade de processamento do sistema computacional e deixar o processador funcional utilizando FPGA, realizando testes para a comprovação do mesmo.



## 3 Fundamentação Teórica

### 3.1 Sistemas computacionais

Um sistema computacional é um conjunto de dispositivos eletrônicos que são utilizados para processar informações, compostos pela união de hardware e software. Um exemplo de sistema computacional é um celular, que possui funções para tirar fotos, gravar vídeos, sons e comunicação com outros dispositivos, ou seja, ele captura os dados dos sensores e transforma em informação, possibilitando a comunicação com o usuário(2).

Focando no Hardware de um sistema computacional, ele consiste em três principais partes:

1. O processador ou CPU: é a unidade responsável pela execução de instruções requisitadas por um sistema, como cálculos aritméticos, lógicos, operações de entrada e saída e manipulação de dados.
2. Memória principal: é o dispositivo capaz de armazenar dados gerais do sistema. Basicamente o processador acessa a memória para buscar informações e depois de processá-las manda sinais para outras partes do sistema realizando atividades pedidas.
3. Módulo de entrada e saída: responsável pela interface entre a máquina e o usuário, ele facilita a comunicação do programador com o sistema computacional.

### 3.2 Central Processor Unit (CPU)

Responsável por executar as instruções, o processador é constituído das seguintes partes:

- ALU (Unidade Lógica Aritmética): realiza cálculos como soma e subtração e executa comparações lógicas como *and* e *or*.
- Memória de instruções: armazena as instruções que serão executadas.
- Banco de registradores: memória de acesso rápido que armazena os operandos da instrução;
- Unidade de Controle: controla as outras partes do processador para a execução correta das instruções.
- *Program counter*: Armazena o endereço para a próxima instrução a ser executada(3).

### 3.3 Risc x Cisc

Para projetar um processador se tem duas principais arquiteturas a Risc (*Reduced Instruction Set Computer*) e a Cisc (*Complex Instruction Set Computer*). Algumas características marcantes de cada arquitetura podem definir se um processador é Risc ou Cisc, todavia, em algumas implementações pode ser difícil de distinguir se é risc ou cisc devido a misturas de tecnologias. As características de cada arquitetura são listadas a seguir:

RISC:

1. Quantidade reduzida de instruções;
2. Instruções de tamanho fixo;
3. Pouca redundância de instruções;
4. Data path mais simples.

CISC:

1. Grande quantidade de instruções;
2. Instruções de tamanho variável;
3. Alta redundância de instruções;
4. Data path mais complexo e extenso.

Processadores com a arquitetura Risc geralmente são utilizados em video games, roteadores e dispositivos mobile, e processadores com a arquitetura Cisc são muito utilizados em servidores, notebooks e desktops([4](#)).

### 3.4 MIPS

MIPS (*Microprocessor Without Interlocked Pipeline Stages*) é um processador de arquitetura RISC que trabalha de forma monocíclica, ou seja, ele executa uma instrução por ciclo de clock. Devido a isto todas as instruções possuem tempo igual de execução. Como é um processador RISC ele herda algumas de suas características: Quantidade reduzida de instruções, instruções de tamanho fixo, pouca redundância de instruções, data path mais simples([3](#)).

### 3.5 Tipos de endereçamento

Para acessar os dados o processador precisa saber onde esses dados estão localizados, uma forma de saber o caminho desses dados é definindo um tipo de endereçamento, alguns tipos são listados a seguir:

1. Endereçamento por registrador: o operando é o conteúdo de um registrador especificado na instrução.
2. Endereçamento imediato: o conteúdo do operando está incorporado na instrução.
3. Endereçamento indireto a registrador: o operando está no endereço especificado dentro de um registrador repassado pela instrução(5).

## 3.6 Verilog

Verilog é uma linguagem de descrição de hardware(Hardware Description Language - HDL) que permite descrever sistemas digitais, analógicos ou híbridos em vários níveis de abstração(6).

O Verilog difere das outras linguagens pela maneira como é executado. Diferente de uma linguagem procedural um projeto Verilog consiste na separação hierárquica de módulos que contém conexões e registradores. Esses módulos contém blocos que são executados em paralelo, há também a possibilidade de executar processos sequenciais dentro de blocos "begin/end"(7).

Existem dois tipos de blocos processuais em verilog:

- initial: são executados apenas uma vez no início (tempo zero);
- always: são executados sempre, ou quando o argumento de entrada for verdadeiro(6).

## 3.7 FPGA

FPGA ou Field Programmable Gate Array é um tipo de circuito integrado que possui uma lógica programável, pode ser configurado após a sua fabricação, internamente possui um grande arranjo de células lógicas ou blocos lógicos configuráveis contidos em um único circuito integrado. É constituído basicamente de três partes:

1. CLB (Configuration Logical Blocks): circuitos que contém flip-flops e lógicas combinacionais, possibilitando a construção de elementos funcionais lógicos;
2. IOB (Input/Output Block): buffers de entrada e saída que fazem o interfaceamento com os blocos lógicos configuráveis;
3. Switch Matrix (chaves de interconexões): trilhas utilizadas para conectar os blocos de entrada e saída(IOB) com os blocos lógicos configuráveis(CLB)(8);

### 3.8 Quartus

Quartus prime 16.1 é um software de design produzido pela Altera. Ele permite a análise e síntese de linguagem de descrição de hardware e desenhos, permite que o desenvolvedor compile seus projetos, realize análise de timing, simule a reação de um projeto a diferentes estímulos, e configure o dispositivo de destino. O Quartus inclui uma implementação do VHDL e Verilog para descrição de hardware, edição visual de circuitos lógicos, e simulação de formas de onda([9](#)).

## 4 Desenvolvimento

Tomando como base a arquitetura MIPS, o processador deste projeto, chamado de ProcessorE, trabalha de forma monocíclica (uma instrução por ciclo de clock) e possui instruções de tamanho fixo e realiza operações principalmente com registradores, de forma que seu endereçamento é direto por registrador. Possui instruções lógicas e aritméticas, de controle, assim como instruções para manipular a memória e de entrada e saída de dados.

O primeiro passo para o desenvolvimento do ProcessorE foi definir o formato das instruções ([Tabela 1](#)) a ser utilizado. Optou-se por uma padronização no formato das instruções com o objetivo de ter melhor organização no momento da implementação e evitar erros.

Tabela 1 – Formato das Instruções

F1												
Opcode	RD	RS	RT	-	-							
[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]							
F2												
Opcode	RD	RS	IMT									
[31:26]	[25:21]	[20:16]	[15:0]									
F3												
Opcode	RD	endereço/IMT/IO										
[31:26]	[25:21]	[20:0]										
F4												
Opcode	-											
[31:26]	[25:0]											

Fonte: O Autor

Cada instrução possui o tamanho de 32 bits com 6 bits destinados à opcode, o que possibilita a criação de 64 instruções diferentes, tornando possível a expansão para futuras instruções. Dos 32 bits, blocos com 5 bits são destinados ao endereço dos registradores; os 5 bits possibilitam ter acesso aos 32 registradores de uso geral, pois com 5 bits obtemos 32 endereços distintos.

Seguindo como modelo as instruções de alguns processadores já criados, foram escolhidas 25 instruções básicas para serem implementadas. Essas instruções foram classificadas de acordo com sua finalidade e estão descritas na [Tabela 2](#).

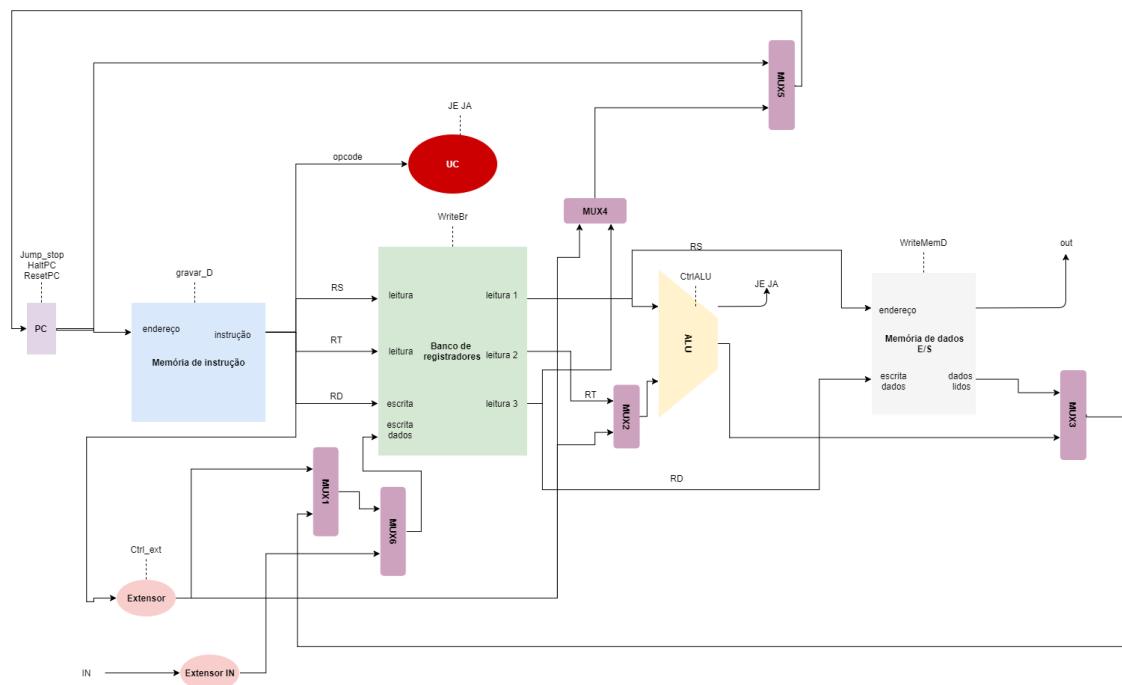
Após definir as instruções, desenhou-se o esquemático do processador ([Figura 1](#)).

Tabela 2 – Conjunto de instruções

Formato	Operação	Opcode Bin	Opcode Dec	Instrução	Tipo
Aritiméticas	$RD \leq RS + RT$	000001	000001	ADD	
	$RD \leq RS + IMT$	000010	000002	ADDI	
	$RD \leq RS - RT$	000011	000003	SUB	
	$RD \leq RS - IMT$	000100	000004	SUBI	
	$RD \leq RS * RT$	000101	000005	MULT	
Lógicas	$RD \leq RS \text{ and } RT$	000110	000006	AND	
	$RD \leq RS \text{ or } RT$	000111	000007	OR	
	$RD \leq RS \text{ xor } RT$	001000	000008	XOR	
	$RD \leq \text{not}(RS)$	001001	000009	NOT	
Deslocamento	$RD \leq RS \gg RT$	001010	000010	SHR	
	$RD \leq RS \ll RT$	001011	000011	SHL	
Movimentação de dados	$RD \leq \text{mem\_raw}[RS]$	001100	000012	LOAD	
	$RD \leq IMT$	001101	000013	LOADI	
	$\text{mem\_raw}[RS] \leq RD$	001110	000014	STORE	
	$RD \leq RS$	001111	000015	MOVE	
Desvios	$PC \leq IMT$	010000	000016	JMPI	
	$PC \leq RD$	010001	000017	JMP	
	$\text{if } RS == RT, PC \leq RD$	010010	000018	JE	
	$\text{if } RS != RT, PC \leq RD$	010011	000019	JNE	
	$\text{if } RS > RT, PC \leq RD$	010100	000020	JA	
Entrada e saída	$\text{if } RS == RT, PC \leq RD$	010101	000021	JNA	
	$RD \leq \text{entrada de dados}$	010110	000022	IN	
Controle	$\text{saída de dados} \leq RD$	010111	000023	OUT	
	nenhuma ação	011000	000024	NOP	
	parar processamento	011001	000025	HALT	

Fonte: O Autor

Figura 1 – Esquematico ProcessorE



Fonte: O autor

Depois de realizar todo o planejamento teórico do ProcessorE seguiu-se para a implementação do mesmo em verilog. Abaixo são mostrados os códigos de todas as unidades implementadas.

## 4.1 Unidade lógica aritimética

A unidade lógica aritmética(ULA) é responsável por realizar três das quatro operações aritméticas básicas- soma, subtração e divisão- e por fazer comparações e deslocamento de bits dos sinais de entrada. No ProcessorE, a ULA também é responsável por setar as *flags* das instruções de *jump*, ou seja, se for executada uma instrução JE a ALU que retornará se o salto deverá ou não ser executado.

O circuito combinacional da ULA possui 2 entradas, uma saída, *flags* para *jumps* e *overflow* de soma e subtração, e uma entrada de controle de 6 bits que define qual cálculo será realizado. O seu código em verilog pode ser visualizado abaixo:

```

1 module ALUnit(in1, in2, ctrlALU, result, of, je, ja);
2
3     input [5:0] ctrlALU;
4     input [31:0] in1, in2;
5     output reg of; // overflow
6     output reg [31:0]result;
7     output reg je, ja; // jump equal / jump above
8     always @(ctrlALU or in1 or in2) begin
9         of = 0;
10
11         case (ctrlALU[5:0])
12             6'b000000: result = in1;
13             6'b000001: begin
14                 result = in1 + in2; //soma
15                 if(in1[31] == 0 && in2[31] == 0 && result[31] == 1) of =
16                     1;
17                 if(in1[31] == 1 && in2[31] == 1 && result[31] == 0) of =
18                     1;
19             end
20             6'b000010: begin
21                 result = in1 - in2; // subtração
22                 if(in1[31] == 0 && in2[31] == 1 && result[31] == 1) of =
23                     1;
24                 if(in1[31] == 1 && in2[31] == 0 && result[31] == 0) of =
25                     1;
26             end
27             6'b000011: result = in1 * in2; // multiplica o
28             6'b000100: result = in1 & in2; // and
29             6'b000101: result = in1 | in2; // or
30             6'b000110: result = in1 ^ in2; // xor
31             6'b000111: result = ~in1; // not
32             6'b001000: result = in1 << in2; // bit shift esquerda
33             6'b001001: result = in1 >> in2; // bit shift direita
34             default: result = 32'b00000000000000000000000000000000;
35         endcase

```

```

35         if(in1 == in2) je <= 1'b1;
36         else je <= 1'b0;
37
38         if($signed(in1) > $signed(in2)) ja <= 1'b1;
39         else ja <= 1'b0;
40
41     end
42
43
44 endmodule

```

## 4.2 Banco de registradores

Conjunto de memórias de acesso rápido utilizados para armazenar os dados a serem calculados.

O banco de registradores do ProcessorE possui um total de 32 registradores de 32 bits, que são gravados ou lidos na subida do clock. Ele possui três entradas para os endereços de três registradores e três saídas que mostram o conteúdo dos mesmos, possui também uma entrada para a escrita de dados e uma *flag* sinalizando se os dados devem ser escritos ou não. Seu código é mostrado abaixo:

```

1 module BReg(endRD, endRS, endRT, clock, dados_Escrita, write, out1, out2, out3);
2     input [4:0] endRD, endRS, endRT;
3     input write, clock;
4     input [31:0] dados_Escrita;
5     output [31:0] out1, out2, out3;
6     //reg [31:0] save;
7     //reg [31:0] save1;
8     //reg [31:0] save2;
9     reg [31:0] registradores [31:0];
10    always @(posedge clock) begin
11        if(write) begin // escrever dados na mem ria
12            registradores[endRD] = dados_Escrita;
13        end
14        //save = endRD;
15        //save1 = endRT;
16        //save2 = endRS;
17    end
18    assign out1 = registradores[endRS];
19    assign out2 = registradores[endRT];
20    assign out3 = registradores[endRD];
21
22 endmodule

```

## 4.3 Memória de dados

A memória de dados tem seu funcionamento similar ao banco de registradores, a diferença é que possui a entrada para apenas um endereço e saída de um endereço, além disso, um registrador é utilizado para armazenar os dados da instrução de saída(OUT).

Código da memória de dados:

```

1 module dad_mem(address,D_escrita, clock, write, dados_lidos, saida, otflag);
2     input [31:0] address;
3     input [31:0] D_escrita;
4     input clock, write, otflag;
5     output [31:0] dados_lidos;
6     reg [31:0]out;
7     output [31:0]saida;
8
9     //reg [31:0] save;
10    reg [31:0] mem_ramD [31:0];
11    always @(posedge clock) begin
12
13        if(write) begin // escrever dados na memoria
14            mem_ramD[address] = D_escrita;
15        end
16        //save = address;
17
18    end
19
20    always @ (negedge clock) begin
21        if(otflag) begin
22            out = D_escrita;
23        end
24    end
25    assign saida = out;
26    assign dados_lidos = mem_ramD[address];
27
28 endmodule

```

## 4.4 Memória de Instruções

A memória de instruções armazena todas as instruções que o processador deverá executar. Ela retorna a instrução de acordo com o endereço fornecido pelo *program counter* e é capaz de armazenar 128 instruções.

Código da memória de instruções:

```

1 module Inst_Mem(PCend, clock,gravar_D,instructionOUT);
2     input [31:0] PCend;
3     input  gravar_D, clock;
4     output [31:0] instructionOUT;
5     //reg [31:0] savePC;
6
7     reg [31:0] mem_ram[127:0];
8     always @(posedge clock) begin
9
10         if(gravar_D == 1) begin // escreve as instruções listadas diretamente na
11             mem_ria
12                 mem_ram[0] = 32'b010110_01001_00000000000000000000000000000000; //RD = IN
13                 mem_ram[1] = 32'b001101_00101_00000000000000000000000000000001; //loadi rd =
14                     5
15                 mem_ram[2] = 32'b000001_00111_01001_00101_000000000000; //soma rs +
16                         rt / in+5
17                 mem_ram[3] = 32'b010111_00111_00000000000000000000000000000000; //out RD
18                             resultado da soma

```

```

15         mem_ram[4] = 32'b010111_01001_000000000000000000000000; //out RD
16             entrada
17
18
19         mem_ram[5] = 32'b0001101_00011_0000000000000000000000001010; //loadi mem
20             [3] = 10 imt = 21bits
21             //mem_ram[1] = 32'b0001101_00101_000000000000000000000000101; //loadi rd
22             = 5
23         mem_ram[6] = 32'b000001_00111_00011_00101_0000000000000000; //soma rs +
24             rt / 10+
25         mem_ram[7] = 32'b0001110_00111_00100_0000000000000000100; // store na
26             ram[RS] o resultado da soma
27         mem_ram[8] = 32'b0001100_01011_00100_0000000000000000101; //LOAD da
28             ram[RS] para o registrador 7
29         mem_ram[9] = 32'b0001111_01111_01011_0000000000000000110; //move RD<=
30             mem[7]
31         mem_ram[10] = 32'b010111_01011_000000000000000000000000111; //out RD
32             registrador com o resultado do load
33         mem_ram[11] = 32'b010111_00101_000000000000000000000000111; //out RD 5
34         mem_ram[12] = 32'b010111_01111_000000000000000000000000111; //out RD
35             registrador do move
36             //mem_ram[9] = 32'b010000_01111_00000000000000000000000010; //JUMPI
37         mem_ram[13] = 32'b0001101_10101_00000000000000000000000010000; //loadi rd =
38             imt = 21bits valor da linha do jump = 16;
39         mem_ram[14] = 32'b0001101_10001_00000000000000000000000010; //loadi rd =
40             imt = 21bits
41         mem_ram[15] = 32'b0001101_11001_00000000000000000000000010; //loadi rd =
42             imt = 21bits
43         mem_ram[16] = 32'b000001_10111_10001_11001_000000000000; //soma rs
44             + rt /
45         mem_ram[17] = 32'b000001_11001_10111_10001_000000000000; //soma rs
46             + rt /
47         mem_ram[18] = 32'b010111_11001_000000000000000000000000111; //out RD
48         mem_ram[19] = 32'b010101_10101_11001_01001_000000000000; //JNA if
49             soma=<entrada pula pra linha 16
50
51         end
52
53         //savePC = PCend;
54     end
55
56     assign instructionOUT = mem_ram[PCend];
57
58 endmodule

```

## 4.5 Program counter

Retorna o endereço da instrução a ser executada. Possui uma entrada, que é o próximo endereço, e duas saídas, uma para o endereço da instrução atual e outra com o endereço da próxima linha. Dependendo da instrução o próximo endereço pode ser mudado.

Código do *program counter*:

```

1 module PC(pcIN, clock, pcOUTat,pcOUTprox,  jump_stop, halt, reset);
2     input [31:0] pcIN;

```

```

3      input  clock, jump_stop, halt, reset;
4      output reg[31:0] pcOUTat, pcOUTprox;
5      always @(posedge clock) begin
6          if(halt == 1) begin
7              end
8          else if(reset == 1) begin
9
10             pcOUTat = 0;
11             pcOUTprox = 32'b00000000000000000000000000000001;
12         end
13         else if(jump_stop == 1) begin
14             pcOUTprox = pcIN + 1;
15         end
16         else if(jump_stop == 0) begin
17             pcOUTat = pcIN;
18             pcOUTprox = pcIN + 1;
19         end
20     end
21
22 endmodule

```

## 4.6 Multiplexador

No desenvolvimento do ProcessorE foram utilizados apenas multiplexadores de 2 entradas. De acordo com as *flags* de saída da unidade de controle cada multiplexador, do total de 6, retorna uma de suas entradas.

Relação do valor de controle com a saída de cada multiplexador:

- Mux1 = 0 retorna o imediato  
Mux1 = 1 retorna o resultado do mux3;
- Mux2 = 0 retorna RT  
Mux2 = 1 retorna o imediato;
- Mux3 = 0 retorna os dados da RAM  
Mux3 = 1 retorna resultado da ALU;
- Mux4 = 0 retorna o RD  
Mux4 = 1 retorna o imediato;
- Mux5 = 0 retorna a saída do PC  
= 1 Mux5 retorna o resultado do mux4;
- Mux6 = 1 coloca o valor de entrada de dados no barramento de gravação do banco de registradores.

Código do multiplexador:

```

1 module mux(in1, in2, saida, c);
2     input [31:0] in1;
3     input [31:0] in2;
4     input c;
5     output reg[31:0] saida;
6
7     always @(*) begin
8
9         case(c)
10            1'b0: saida = in1;
11            1'b1: saida = in2;
12        endcase
13    end
14 endmodule

```

## 4.7 Extensor de Imediato

O extensor de Imediato estende o tamanho do imediato da instrução até 32bits para que ele fique compatível com o tamanho do barramento de dados utilizado no processador. O extensor estende tanto números positivos quanto negativos, e possui um sinal de controle que determina quantos bits ele vai estender, pois o processador tem imediatos de 16 bits e de 21 bits.

Código do Extensor:

```

1 module extensor(in16, in21, ctrl, extendido);
2     input [15:0] in16;
3     input [20:0] in21;
4     input ctrl;
5     output reg [31:0] extendido;
6     always @( * ) begin
7         case(ctrl)
8             1'b1: begin // extende os 16 bits para 32
9                 if(in16[15] == 1)begin
10                     extendido = {16'b1111111111111111,in16};
11                 end else
12                     extendido = {16'b0000000000000000,in16};
13             end
14             1'b0: begin // extende os 21 bits para 32
15                 if(in21[20] == 1)begin
16                     extendido = {16'b11111111111,in21};
17                 end else
18                     extendido = {16'b00000000000,in21};
19             end
20             default: extendido = 32'b00000000000000000000000000000000;
21         endcase
22     end
23 endmodule

```

## 4.8 Unidade de Controle

A unidade de controle comanda todos os sinais de controle do processador e altera esses sinais de acordo com o opcode da instrução executada. A unidade de controle foi implementada como um circuito combinacional. O exemplo para os sinais de controle para a instrução de soma ADD é mostrado abaixo, o código completo se encontra [Apêndice A](#).

```

1  module UnidadeC(reset , opcode , jump_stop , HaltPC , ResetPC , gravar_Dmi , WriteBr , Ctrl_ext , mux1 ,
2      mux2 , ctrlALU , WriteMemD , mux3 , mux4 , mux5 , JE , JA , flagIN , fout , mux6 );
3      output reg jump_stop , HaltPC , ResetPC , gravar_Dmi , WriteBr , Ctrl_ext , WriteMemD , mux1 ,
4          mux2 , mux3 , mux4 , mux5 , fout , mux6 ;
5      output reg [5:0] ctrlALU ;
6      input [5:0] opcode ;
7      input JE , JA , flagIN , reset ; // jump equal / jump above / flag de entrada
8      always @(*) begin
9          if(reset) begin
10              jump_stop = 0;
11              HaltPC = 0;
12              ResetPC = 1;
13              gravar_Dmi = 1;
14              WriteBr = 0;
15              Ctrl_ext = 0;
16              ctrlALU = 0;
17              mux1 = 0;
18              mux2 = 0;
19              mux3 = 0;
20              WriteMemD = 0;
21              mux4 = 0;
22              mux5 = 0;
23              fout = 0;
24              mux6 = 0;
25          end
26          else begin
27              case (opcode[5:0])
28                  6'b000000:;
29                  6'b000001: //ADD
30                  begin
31                      jump_stop = 0;
32                      HaltPC = 0;
33                      ResetPC = 0;
34                      gravar_Dmi = 0;
35                      WriteBr = 1;
36                      Ctrl_ext = 0;
37                      ctrlALU = 6'b000001;
38                      mux1 = 1;
39                      mux2 = 0;
40                      mux3 = 1;
41                      WriteMemD = 0;
42                      mux4 = 0;
43                      mux5 = 0;
44                      fout = 0;
45                      mux6 = 0;
46                  end
47              endcase
48          end
49      end

```

```

50      end
51
52 endmodule

```

## 4.9 IN e OUT

Para a entrada de dados, as chaves do FPGA foram ligadas ao barramento de escrita do banco de registradores através de um multiplexador. Quando uma instrução IN é executada o multiplexador direciona o barramento das chaves para a escrita no banco de registradores, e se a *flag* de entrada estiver alta a unidade de controle passa o sinal para o dado ser gravado no registrador. Como são poucas as chaves do FPGA o sinal delas é estendido para o tamanho de 32 bits, o extensor pode ser visto no [Apêndice E](#).

Para exibir os dados de saída, advindos da memória de dados, no display de 7 segmentos foram utilizados um conversor binário para bcd e um decodificador de bcd para display de 7 segmentos de autoria não própria e fonte desconhecida, que podem ser vistos nos [Apêndice C](#) e [Apêndice D](#) respectivamente.

## 4.10 ProcessorE: unidades interligadas

Após a implementação de todos os componentes do processador fez-se a união deles através de barramentos seguindo o esquemático da [Figura 1](#), mostrada no código abaixo:

```

1 module processorE(clocksis,BC,
2                     //endi,
3                     //instructionOUT,
4                     //ctrlAlu,
5                     flagIN,IN,r,
6                     //      entrada,
7                     /*      out, */
8                     //saida_centena, saida_dezena, saida_sinal,
9                     //saida_unidade,
10                    e_centena, e_dezena, e_sinal, e_unidade);
11
12   wire [31:0]out;
13   input wire [14:0]IN;
14   wire [31:0]inExt;
15   wire [31:0]entrada;
16   wire flagout;
17
18   wire clk;
19   input wire flagIN, r;
20   input wire BC;
21   input wire clocksis;
22
23   wire [31:0]pcin;
24   wire [31:0]endi;
25   wire [31:0]pcprox;
26   wire jump_stop, haltpc, resetpc;
27
28   wire [31:0]instructionOUT;

```

```

27     wire grav_D;
28
29     wire [31:0] dados_escrita;
30     wire writeBR;
31     wire [31:0] RS, RT, RD;
32
33     wire [31:0] imt_ext;
34     wire ctrl_ext;
35     wire [31:0] resultPross;
36     wire mux1;
37
38     wire mux2;
39     wire [31:0] in2_alu;
40
41     wire [5:0] ctrlAlu;
42     wire [31:0] resultAlu;
43     wire of, je, ja;
44
45     wire writeMemD;
46     wire [31:0] dados_lidos;
47     wire mux3;
48     wire mux4;
49     wire mux5;
50     wire [31:0] rmux4;
51     wire mux6;
52     wire [31:0] escreveNoR;
53     output wire[6:0] saida_centena, saida_dezena, saida_sinal, saida_unidade;
54     output wire[6:0] e_centena, e_dezena, e_sinal, e_unidade;
55
56     PC programC(.pcIN(pcIN), .clock(clk), .pcOUTat(endi), .pcOUTprox(pcprox), .
57         jump_stop(jump_stop), .halt(haltpc), .reset(resetpc));
58     Inst_Mem MI(.PCend(endi), .clock(clk), .gravar_D(grav_D), .instructionOUT(
59         instructionOUT));
60     BReg BR(.endRD(instructionOUT[25:21]),
61             .endRS(instructionOUT[20:16]),
62             .endRT(instructionOUT[15:11]),
63             .clock(clk), .dados_Escrita(escreveNoR),
64             .write(writeBR), .out1(RS), .out2(RT), .out3(RD))
65             ;
66
67     extensor ext(.in16(instructionOUT[15:0]), .in21(instructionOUT[20:0]), .ctrl(
68         ctrl_ext), .extendido(imt_ext));
69     mux m1(.in1(imt_ext), .in2(resultPross), .saida(dados_escrita), .c(mux1));
70
71     mux m2(.in1(RT), .in2(imt_ext), .saida(in2_alu), .c(mux2));
72     ALUnit ALU(.in1(RS), .in2(in2_alu), .ctrlALU(ctrlAlu), .result(resultAlu), .of(of)
73         , .je(je), .ja(ja));
74
75     dad_mem MD(.address(RS), .D_escrita(RD), .clock(clk), .write(writeMemD), .
76         dados_lidos(dados_lidos), .saida(out), .otflag(flagout));
77     mux m3(.in1(dados_lidos), .in2(resultAlu), .saida(resultPross), .c(mux3));
78
79     mux m4(.in1(RD), .in2(imt_ext), .saida(rmux4), .c(mux4));
80     mux m5(.in1(pcprox), .in2(rmx4), .saida(pcIN), .c(mux5));
81     mux m6(.in1(dados_escrita), .in2(inExt), .saida(escreveNoR), .c(mux6));
82
83     UnidadeC UC(.reset(r), .opcode(instructionOUT[31:26]), .jump_stop(jump_stop), .
84         HaltPC(haltpc), .ResetPC(resetpc),
85         .gravar_Dmi(grav_D),
86         .WriteBr(writeBR),

```

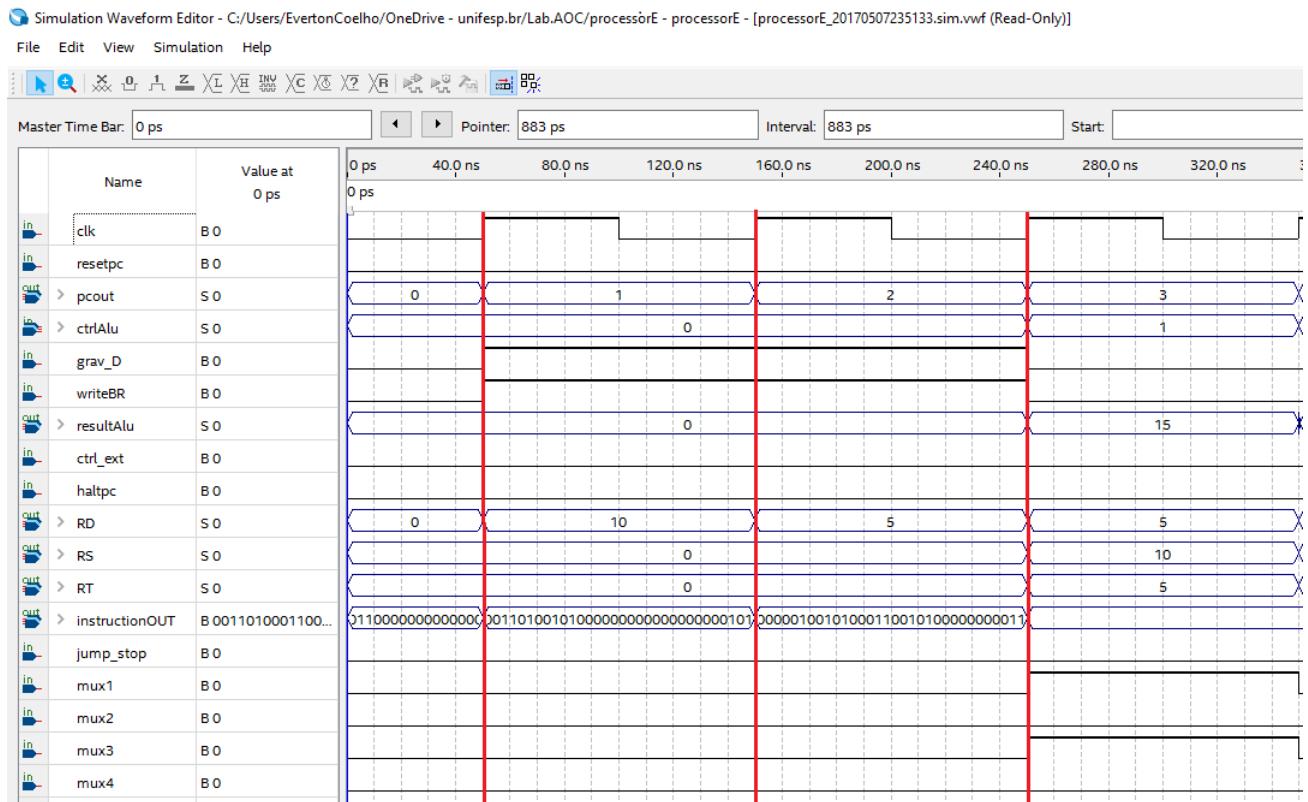
```
80          .Ctrl_ext(ctrl_ext),
81          .mux1(mux1),.mux2(mux2),
82          .ctrlALU(ctrlAlu),
83          .WriteMemD(writeMemD),
84          .mux3(mux3),.mux4(mux4),.mux5(mux5),
85          .JE(je),.JA(ja),
86          .flagIN(flagIN),.fout(flagout),.mux6(mux6));
87
88      extIN ei(.in15(IN), .extendido(inExt));
89
90      BCD saida(.numBCD(out), .saida_centena(saida_centena), .saida_dezena(saida_dezena
91                  ), .saida_sinal(saida_sinal), .saida_unidade(saida_unidade),
92                  .entradaUnidade(entradaUnidade), .entradaDezena(
93                      entradaDezena),.entradaCentena(entradaCentena), .
94                      entradaSinal(entradaSinal));
95
96      BCD entradaIN(.numBCD(inExt), .saida_centena(e_centena), .saida_dezena(e_dezena),
97                  .saida_sinal(e_sinal), .saida_unidade(e_unidade),
98                  .entradaUnidade(entradaUnid), .entradaDezena(entradaDez)
99                  ,.entradaCentena(entradaCent), .entradaSinal(
100                     entradaSin));
100
101      debouncer db(
102          .clk(clocksis), //this is a 50MHz clock provided on FPGA pin PIN_Y2
103          .PB(~BC), //this is the input to be debounced
104          .PB_state(clk) //this is the debounced switch
105      );
106
107  endmodule
```

## 5 Resultados Obtidos e Discussões

Para teste e análise do funcionamento do processador foram feitos vários testes através de simulações por forma de onda no Quartus, uma delas é mostrada a seguir:

*Wave form* obtido após rodar duas instruções de LOADI e uma instrução ADD;

Figura 2 – Simulação de instruções por forma de onda



Fonte: O autor

As duas instruções de LOADI fazem o caminho de passar pela memória de instrução, extender o imediato e salvá-lo no banco de registradores; com a execução desse caminho pode-se afirmar que as unidades que fazem parte dele estão funcionando corretamente.

A instrução de ADD segue o caminho de passar pela memória de instruções, banco de registradores, unidade lógica aritmética e retornar para o banco de registradores; seguindo esse caminho confirma-se que essas unidades estão funcionando, e como as instruções foram executadas acessando a memória de instruções através de um endereço vindo do *program counter* é possível afirmar também que ele está trabalhando corretamente.

Após os testes de formas de onda o processador foi pinado de acordo com as entradas e saídas do FPGA e baixado para o mesmo. As instruções utilizadas para teste

no FPGA são as seguintes:

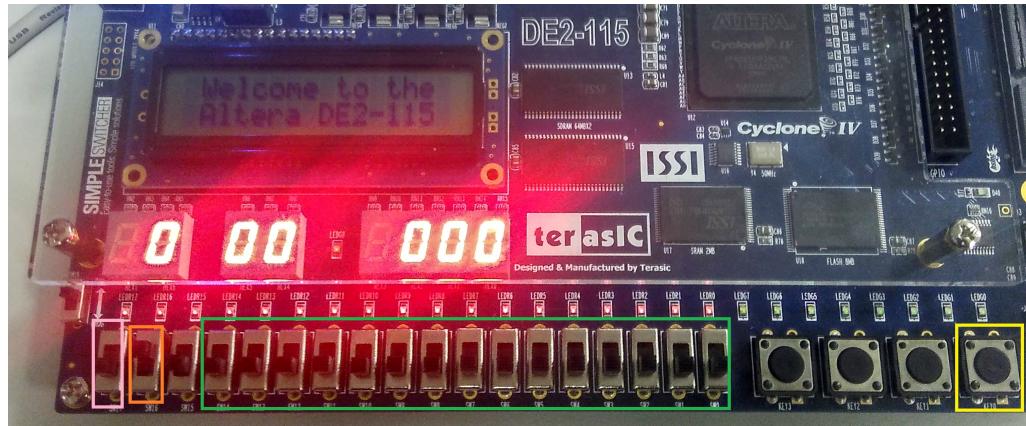
```

1      mem_ram[0] = 32'b010110_01001_000000000000000000000000; //RD = IN
2      mem_ram[1] = 32'b001101_00101_000000000000000000000001; //loadi rd =
3          5
4      mem_ram[2] = 32'b000001_00111_01001_00101_0000000000; //soma rs +
5          rt / in+5
6      mem_ram[3] = 32'b010111_00111_000000000000000000000000; //out RD
7          resultado da soma
8      mem_ram[4] = 32'b010111_01001_000000000000000000000000; //out RD
9          entrada
10     mem_ram[5] = 32'b001101_00011_000000000000000000000001; //loadi mem
11     [3] = 10 imt = 21bits
12     mem_ram[6] = 32'b000001_00111_00011_00101_0000000000; //soma rs +
13     rt / 10+5
14     mem_ram[7] = 32'b001110_00111_00100_0000000000000000100; // store na
15     ram[RS] o resultado da soma
16     mem_ram[8] = 32'b001100_01011_00100_0000000000000000101; //LOAD da
17     ram[RS] para o registrador 7
18     mem_ram[9] = 32'b001111_01111_01011_0000000000000000110; //move RD<=
19     mem[7]
20     mem_ram[10] = 32'b010111_01011_00000000000000000000000111; //out RD
21     registrador com o resultado do load
22     mem_ram[11] = 32'b010111_00101_00000000000000000000000111; //out RD 5
23     mem_ram[12] = 32'b010111_01111_00000000000000000000000111; //out RD
24     registrador do move
25     mem_ram[13] = 32'b001101_10101_000000000000000010000; //loadi rd =
26         imt = 21bits valor da linha do jump = 16;
27     mem_ram[14] = 32'b001101_10001_0000000000000000000000010; //loadi rd =
28         imt = 21bits
29     mem_ram[15] = 32'b001101_11001_0000000000000000000000010; //loadi rd =
30         imt = 21bits
31     mem_ram[16] = 32'b000001_10111_10001_11001_0000000000; //soma rs
32         + rt /
33     mem_ram[17] = 32'b000001_11001_10111_10001_0000000000; //soma rs
34         + rt /
35     mem_ram[18] = 32'b010111_11001_00000000000000000000000111; //out RD
36     mem_ram[19] = 32'b010101_10101_11001_01001_0000000000; //JNA if
37         soma=<entrada pula pra linha 16

```

Iniciando a execução no FPGA, a primeira ação a ser realizada é resetar o processador para que o *program counter* seja zerado e as instruções gravadas na memória. Para fazer isso a chave laranja que é o reset é levantada e o botão de clock apertado uma vez, como mostra a [Figura 3](#).

Figura 3 – Execução de instruções no FPGA 1-botão amarelo = clock, chaves verdes = entrada, chave laranja = reset, chave rosa = *flag* de entrada

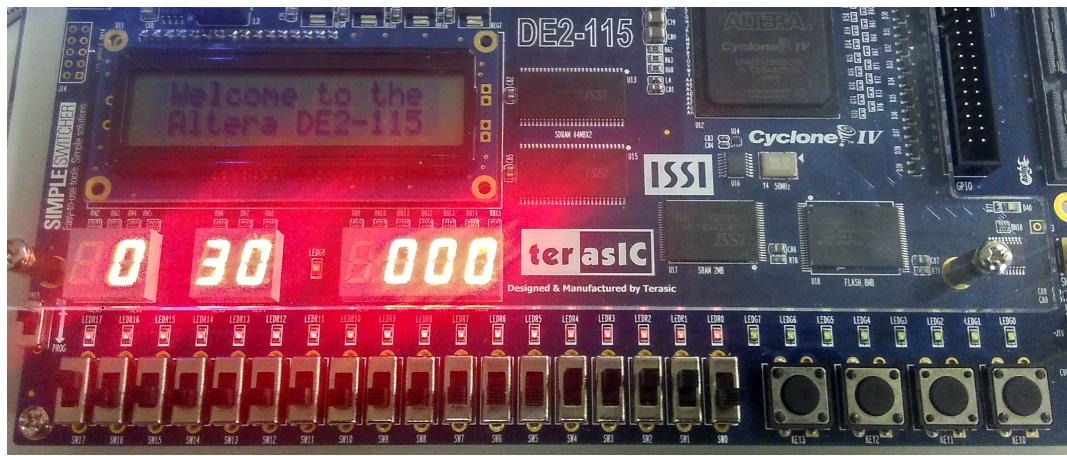


Fonte: O autor

Os quatro *displays* de 7 segmentos da direita são para mostrar os valores das instruções de OUT, e os quatro *displays* da esquerda mostram o valor da entrada de dados.

A primeira instrução a ser realizada é uma instrução de entrada de dados IN. O processador espera até que a *flag* de entrada seja alta para continuar o processamento. No teste o valor de entrada foi 30, visto na Figura 4.

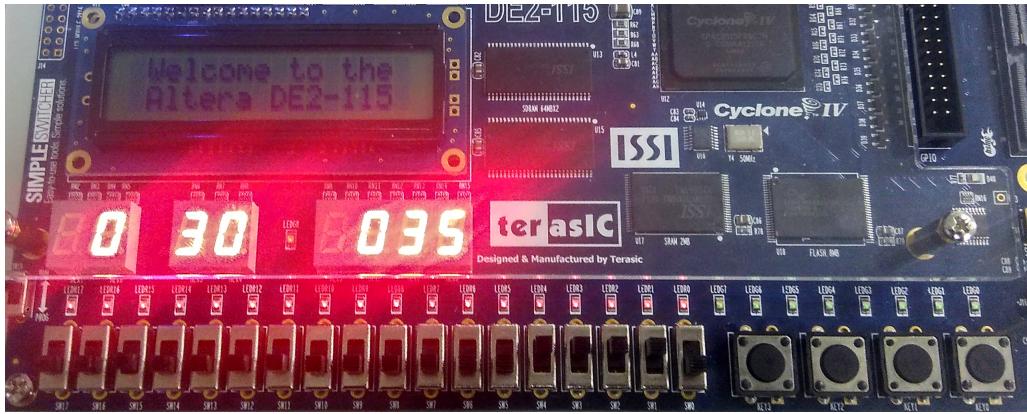
Figura 4 – Execução de instruções no FPGA 2



Fonte: O autor

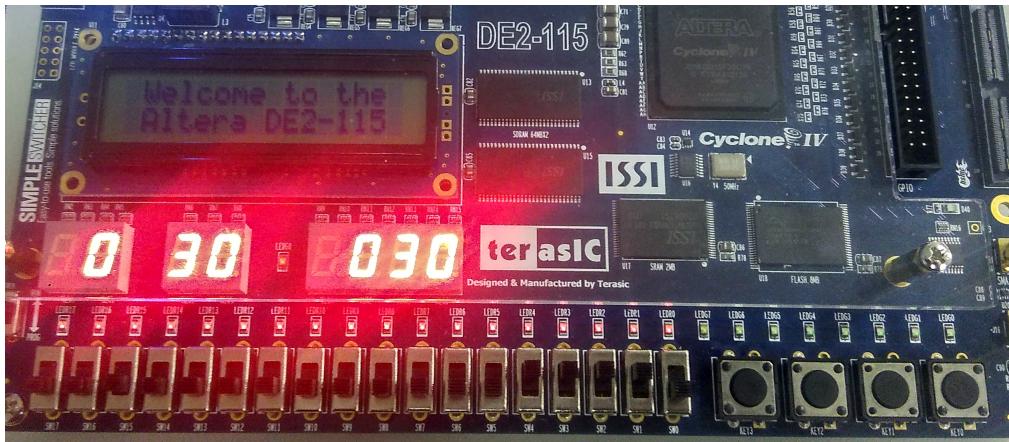
Após a entrada de dados o processador carrega em um registrador o valor 5 usando um LOADI e depois soma com o valor de entrada, o resultado da soma é exibido no *display*(Figura 5), e depois é exibido o valor de entrada(Figura 6).

Figura 5 – Execução de instruções no FPGA 3



Fonte: O autor

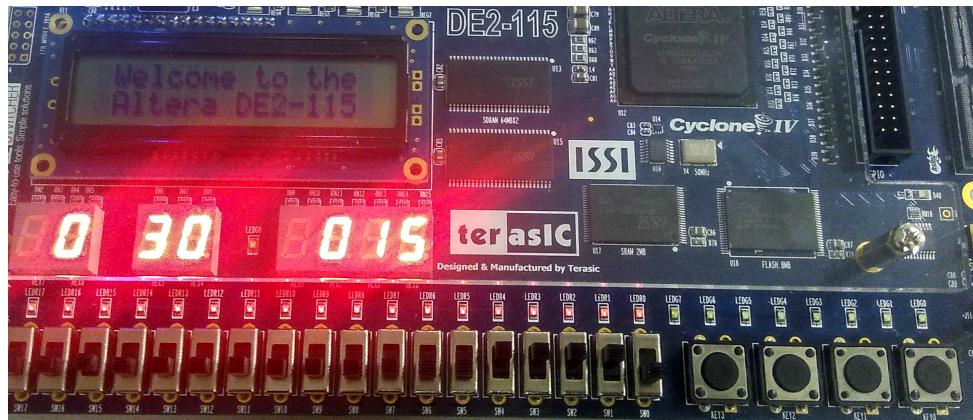
Figura 6 – Execução de instruções no FPGA 4



Fonte: O autor

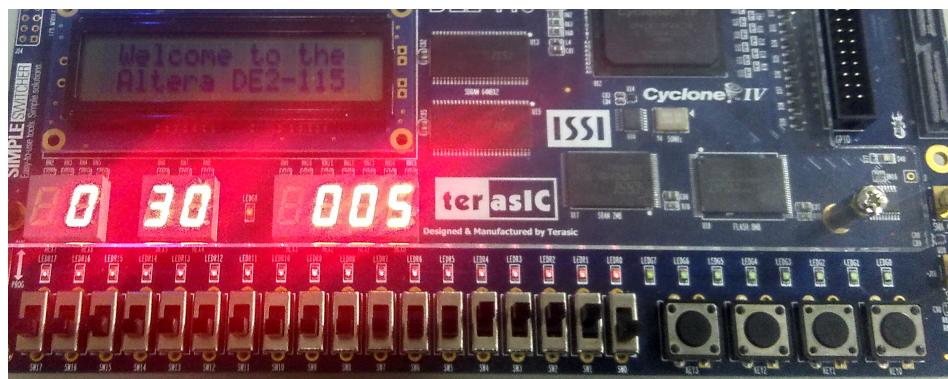
Continuando, o processador realiza outro LOADI e armazena o valor 10 no registrador 3, soma  $10+5$ , armazenado anteriormente ,e salva o resultado no registrador 7. O valor do registrador 7 é armazenado na memória ram com um STORE e depois carregado para o registrador 13 com um LOAD, o valor contido em 13 é movido para o registrador 15. Depois disso são feitos 3 OUTs, o primeiro (Figura 7) com o registrador 13 que armazena o resultado da soma, o segundo (Figura 8) com o valor do registrador 5 que armazena o valor 5, e o terceiro (Figura 9) com o registrador 15 que armazena o valor do registrador 13, ou seja, o resultado da soma também.

Figura 7 – Execução de instruções no FPGA 5



Fonte: O autor

Figura 8 – Execução de instruções no FPGA 6



Fonte: O autor

Figura 9 – Execução de instruções no FPGA 7



Fonte: O autor

Posterior as instruções de OUT são realizadas instruções de LOADI, a primeira armazena no registrador 21 o valor 16, a segunda e a terceira armazenam o valor 2 em

outros dois registradores. os registradores contendo o valor 2 são somados e o resultado da soma é somado novamente com um dos dois registradores, uma instrução de OUT é feita com o resultado dessa soma([Figura 10](#)).

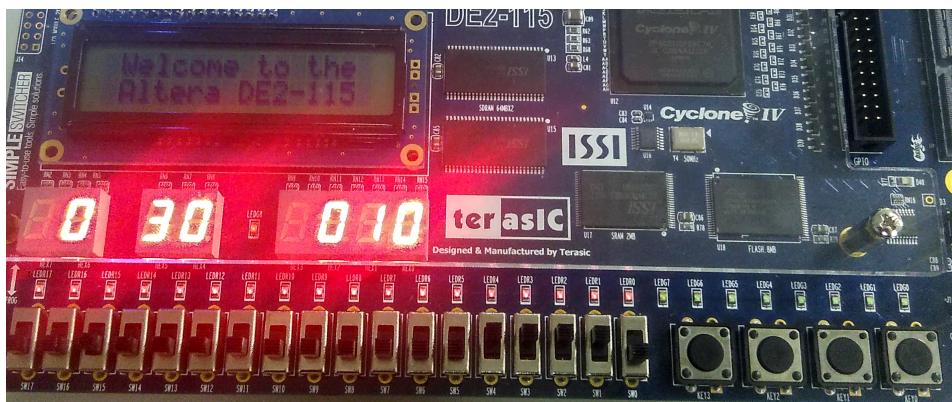
Em seguida a instrução de saída tem-se um salto condicional, *jump not above*. Se o resultado da soma for menor ou igual ao valor entrado no início do programa, a execução volta para linha de valor armazenado no registrador 21, sendo assim, ela volta para linha 16 até que a soma seja maior que 30. A soma cresce de 4 em 4 e para no valor 34, o decorrer dessa execução é mostrado na [Figura 11](#),[Figura 12](#),[Figura 13](#),[Figura 14](#),[Figura 15](#),[Figura 16](#),[Figura 17](#).

Figura 10 – Execução de instruções no FPGA 8



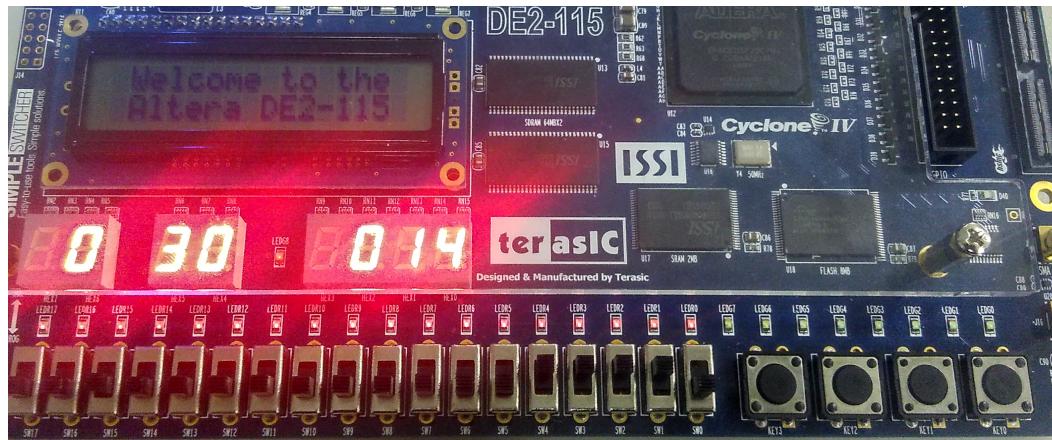
Fonte: O autor

Figura 11 – Execução de instruções no FPGA 9



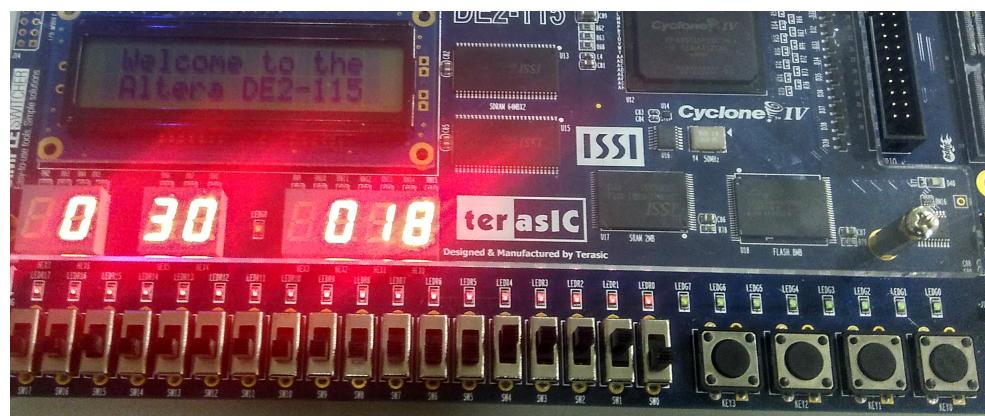
Fonte: O autor

Figura 12 – Execução de instruções no FPGA 10



Fonte: O autor

Figura 13 – Execução de instruções no FPGA 11



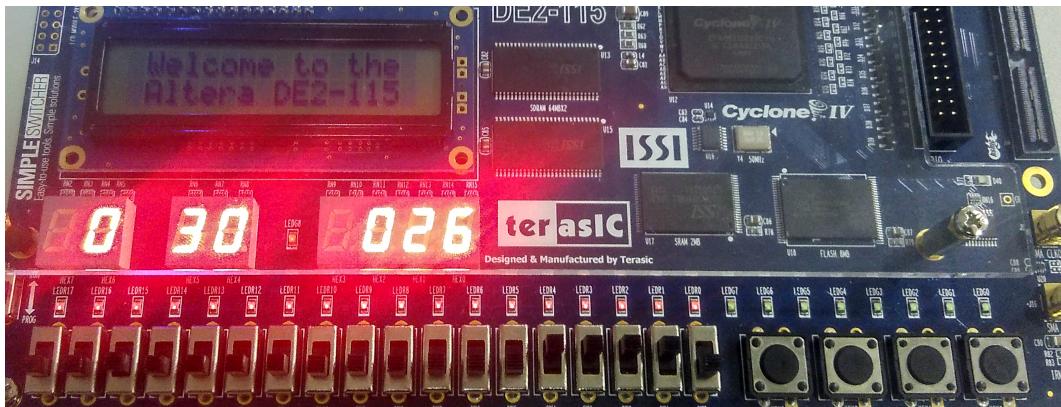
Fonte: O autor

Figura 14 – Execução de instruções no FPGA 12



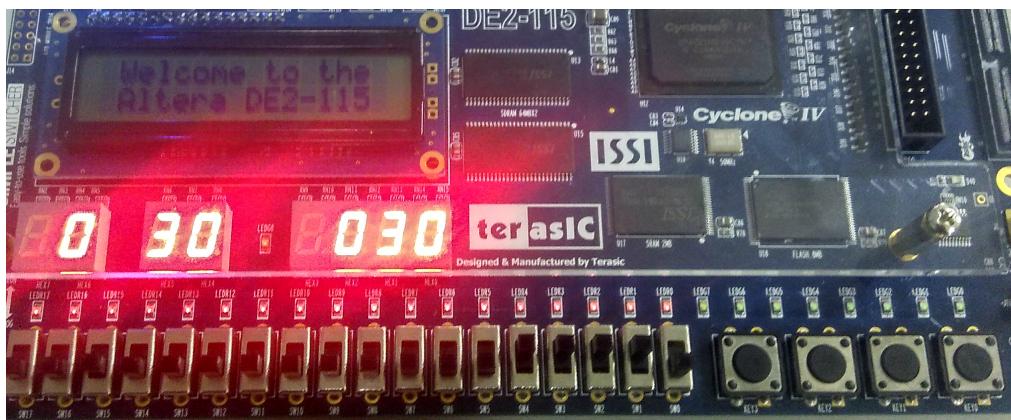
Fonte: O autor

Figura 15 – Execução de instruções no FPGA 13



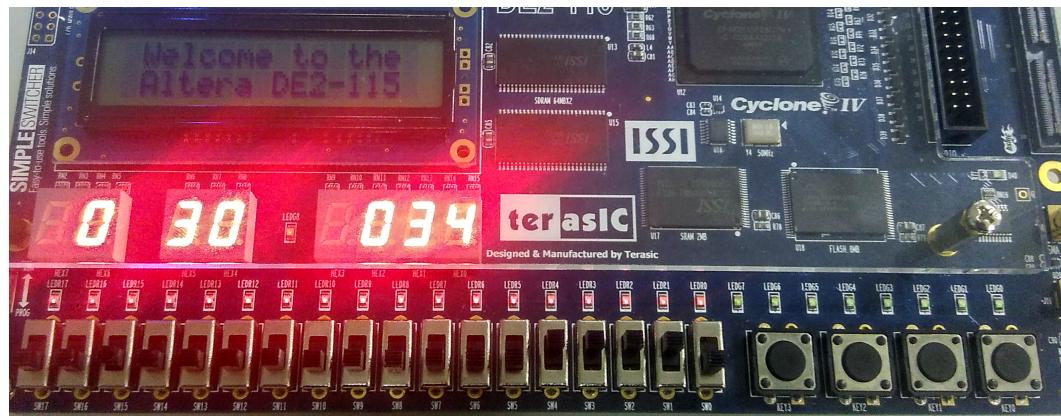
Fonte: O autor

Figura 16 – Execução de instruções no FPGA 14



Fonte: O autor

Figura 17 – Execução de instruções no FPGA 15



Fonte: O autor

No decorrer do programa executado no FPGA foram utilizadas instruções de vários

tipos, garantindo a passagem de dados por todos os módulos do processador. Como o programa foi executado corretamente pode-se afirmar que o processador funcionou com sucesso e cumpriu os objetivos que foram propostos.



## 6 Considerações Finais

O processador criado executou as instruções de forma correta, atendendo as expectativas tanto nas simulações por forma de onda quanto no FPGA. A maior dificuldade na execução do projeto foi conseguir associar as instruções com um esquemático baseado em uma arquitetura pronta, pois, são necessárias algumas modificações no caminho de dados para que as instruções possam ser implementadas. Outra dificuldade foi aprender a linguagem verilog mais a fundo para corrigir erros durante a implementação.

Planos futuros para a melhora do ProcessorE são a inclusão de um módulo de comunicação com Arduino, tanto para entrada e saída de dados quanto para gravar instruções na memória, e aumento da quantidade de instruções, objetivando facilitar a implementação de um compilador.

Neste projeto pôde-se aprender conceitos fundamentais sobre construção, lógica de funcionamento e implementação de um processador. Também foram aplicados conceitos básicos sobre diferenças de arquiteturas para processadores e qual a forma mais simples de implementar uma CPU.



# Referências

- 1 COMPUTADOR. Acessado em 06/04/2017. Disponível em: <<https://pt.wikipedia.org/wiki/Computador>>. Citado na página 7.
- 2 O que é um sistema computacional. Acessado em 05/04/2017. Disponível em: <<https://www.portaleducacao.com.br/conteudo/artigos/informatica/o-que-e-um-sistema-computacional/46697>>. Citado na página 11.
- 3 PATTERSON, D. A.; HENNESSY, J. L. *Computer Organization and Design*. 5th edition. ed. Waltham/MA, EUA: Morgan Kaufmann, 2007. Citado 2 vezes nas páginas 11 e 12.
- 4 CARACTERISTICAS da arquitetura Risq e Cisq. Acessado em 07/04/2017. Disponível em: <[https://pt.wikipedia.org/wiki/RISC#Caracter.C3.ADsticas\\_das\\_Arquiteturas\\_CISC](https://pt.wikipedia.org/wiki/RISC#Caracter.C3.ADsticas_das_Arquiteturas_CISC)>. Citado na página 12.
- 5 ENDEREÇAMENTO de memória. Acessado em 06/04/2017. Disponível em: <<http://usuarios.upf.br/~appel/arquiI/endereca.pdf>>. Citado na página 13.
- 6 INTRODUÇÃO a verilog. Acessado em 05/04/2017. Disponível em: <<http://www.asic-world.com/verilog/intro1.html#Introduction>>. Citado na página 13.
- 7 VERILOG. Acessado em 07/04/2017. Disponível em: <<https://pt.wikipedia.org/wiki/Verilog>>. Citado na página 13.
- 8 FPGA. Acessado em 05/04/2017. Disponível em: <[https://pt.wikipedia.org/wiki/Field-programmable\\_gate\\_array](https://pt.wikipedia.org/wiki/Field-programmable_gate_array)>. Citado na página 13.
- 9 ALTERA quartus. Acessado em 07/04/2017. Disponível em: <[https://en.wikipedia.org/wiki/Altera\\_Quartus](https://en.wikipedia.org/wiki/Altera_Quartus)>. Citado na página 14.



# Apêndices



## APÊNDICE A – UnidadeC.v

Código completo da unidade de controle.

```

1 module UnidadeC(reset, opcode,jump_stop,HaltPC,ResetPC,gravar_Dmi,WriteBr,Ctrl_ext,mux1,
2     mux2,ctrlALU,WriteMemD,mux3,mux4,mux5,JE,JA, flagIN, fout,mux6);
3     output reg jump_stop,HaltPC,ResetPC,gravar_Dmi,WriteBr,Ctrl_ext,WriteMemD,mux1,
4         mux2,mux3,mux4,mux5,fout,mux6;
5     output reg [5:0] ctrlALU;
6     input [5:0] opcode;
7     input JE, JA, flagIN,reset; // jump equal / jump above / flag de entrada
8     always @(*) begin
9         if(reset) begin
10             jump_stop = 0;
11             HaltPC = 0;
12             ResetPC = 1;
13             gravar_Dmi = 1;
14             WriteBr = 0;
15             Ctrl_ext = 0;
16             ctrlALU = 0;
17             mux1 = 0;
18             mux2 = 0;
19             mux3 = 0;
20             WriteMemD = 0;
21             mux4 = 0;
22             mux5 = 0;
23             fout = 0;
24             mux6 = 0;
25         end
26         else begin
27             case (opcode[5:0])
28                 6'b000000:;
29
30                 6'b000001: //ADD
31                 begin
32                     jump_stop = 0;
33                     HaltPC = 0;
34                     ResetPC = 0;
35                     gravar_Dmi = 0;
36                     WriteBr = 1;
37                     Ctrl_ext = 0;
38                     ctrlALU = 6'b000001;
39                     mux1 = 1;
40                     mux2 = 0;
41                     mux3 = 1;
42                     WriteMemD = 0;
43                     mux4 = 0;
44                     mux5 = 0;
45                     fout = 0;
46                     mux6 = 0;
47                 end
48                 6'b000010: //ADDI
49                 begin
50                     jump_stop = 0;
51                     HaltPC = 0;
52                     ResetPC = 0;
53                     gravar_Dmi = 1;
54                     WriteBr = 0;
55                     Ctrl_ext = 1;
56                     ctrlALU = 6'b000010;
57                     mux1 = 0;
58                     mux2 = 1;
59                     mux3 = 0;
60                     WriteMemD = 1;
61                     mux4 = 1;
62                     mux5 = 0;
63                     fout = 1;
64                     mux6 = 0;
65                 end
66             endcase
67         end
68     end
69 endmodule

```

```

51          ResetPC = 0;
52          gravar_Dmi = 0;
53          WriteBr = 1;
54          Ctrl_ext = 1;
55          ctrlALU = 6'b000001;
56          mux1 = 1;
57          mux2 = 1;
58          mux3 = 1;
59          WriteMemD = 0;
60          mux4 = 0;
61          mux5 = 0;
62          fout = 0;
63          mux6 = 0;
64      end
65 6'b000011: //SUB
66 begin
67          jump_stop = 0;
68          HaltPC = 0;
69          ResetPC = 0;
70          gravar_Dmi = 0;
71          WriteBr = 1;
72          Ctrl_ext = 0;
73          ctrlALU = 6'b000010;
74          mux1 = 1;
75          mux2 = 0;
76          mux3 = 1;
77          WriteMemD = 0;
78          mux4 = 0;
79          mux5 = 0;
80          fout = 0;
81          mux6 = 0;
82      end
83 6'b000100: //SUBI
84 begin
85          jump_stop = 0;
86          HaltPC = 0;
87          ResetPC = 0;
88          gravar_Dmi = 0;
89          WriteBr = 1;
90          Ctrl_ext = 1;
91          ctrlALU = 6'b000010;
92          mux1 = 1;
93          mux2 = 1;
94          mux3 = 1;
95          WriteMemD = 0;
96          mux4 = 0;
97          mux5 = 0;
98          fout = 0;
99          mux6 = 0;
100     end
101 6'b000101: //MULT
102 begin
103         jump_stop = 0;
104         HaltPC = 0;
105         ResetPC = 0;
106         gravar_Dmi = 0;
107         WriteBr = 1;
108         Ctrl_ext = 0;
109         ctrlALU = 6'b000011;
110         mux1 = 1;

```

```

111      mux2 = 0;
112      mux3 = 1;
113      WriteMemD = 0;
114      mux4 = 0;
115      mux5 = 0;
116      fout = 0;
117      mux6 = 0;
118  end
119  6'b000110: //AND
120  begin
121      jump_stop = 0;
122      HaltPC = 0;
123      ResetPC = 0;
124      gravar_Dmi = 0;
125      WriteBr = 1;
126      Ctrl_ext = 0;
127      ctrlALU = 6'b000100;
128      mux1 = 1;
129      mux2 = 0;
130      mux3 = 1;
131      WriteMemD = 0;
132      mux4 = 0;
133      mux5 = 0;
134      fout = 0;
135      mux6 = 0;
136  end
137  6'b000111: //OR
138  begin
139      jump_stop = 0;
140      HaltPC = 0;
141      ResetPC = 0;
142      gravar_Dmi = 0;
143      WriteBr = 1;
144      Ctrl_ext = 0;
145      ctrlALU = 6'b000101;
146      mux1 = 1;
147      mux2 = 0;
148      mux3 = 1;
149      WriteMemD = 0;
150      mux4 = 0;
151      mux5 = 0;
152      fout = 0;
153      mux6 = 0;
154  end
155  6'b001000: //XOR
156  begin
157      jump_stop = 0;
158      HaltPC = 0;
159      ResetPC = 0;
160      gravar_Dmi = 0;
161      WriteBr = 1;
162      Ctrl_ext = 0;
163      ctrlALU = 6'b000110;
164      mux1 = 1;
165      mux2 = 0;
166      mux3 = 1;
167      WriteMemD = 0;
168      mux4 = 0;
169      mux5 = 0;
170      fout = 0;

```

```

171                         mux6 = 0;
172                     end
173                     6'b001001: //NOT
174                     begin
175                         jump_stop = 0;
176                         HaltPC = 0;
177                         ResetPC = 0;
178                         gravvar_Dmi = 0;
179                         WriteBr = 1;
180                         Ctrl_ext = 0;
181                         ctrlALU = 6'b000111;
182                         mux1 = 1;
183                         mux2 = 0;
184                         mux3 = 1;
185                         WriteMemD = 0;
186                         mux4 = 0;
187                         mux5 = 0;
188                         fout = 0;
189                         mux6 = 0;
190                     end
191                     6'b001010:begin //SHL
192                         jump_stop = 0;
193                         HaltPC = 0;
194                         ResetPC = 0;
195                         gravvar_Dmi = 0;
196                         WriteBr = 1;
197                         Ctrl_ext = 0;
198                         ctrlALU = 6'b001000;
199                         mux1 = 1;
200                         mux2 = 0;
201                         mux3 = 1;
202                         WriteMemD = 0;
203                         mux4 = 0;
204                         mux5 = 0;
205                         fout = 0;
206                         mux6 = 0;
207                     end
208                     6'b001011:begin //SHR
209                         jump_stop = 0;
210                         HaltPC = 0;
211                         ResetPC = 0;
212                         gravvar_Dmi = 0;
213                         WriteBr = 1;
214                         Ctrl_ext = 0;
215                         ctrlALU = 6'b001001;
216                         mux1 = 1;
217                         mux2 = 0;
218                         mux3 = 1;
219                         WriteMemD = 0;
220                         mux4 = 0;
221                         mux5 = 0;
222                         fout = 0;
223                         mux6 = 0;
224                     end
225                     6'b001100:begin //LOAD
226                         jump_stop = 0;
227                         HaltPC = 0;
228                         ResetPC = 0;
229                         gravvar_Dmi = 0;
230                         WriteBr = 1;

```

```

231           Ctrl_ext = 0;
232           ctrlALU = 0;
233           mux1 = 1;
234           mux2 = 0;
235           mux3 = 0;
236           WriteMemD = 0;
237           mux4 = 0;
238           mux5 = 0;
239           fout = 0;
240           mux6 = 0;
241       end
242   6'b001101:begin //LOADI
243           jump_stop = 0;
244           HaltPC = 0;
245           ResetPC = 0;
246           gravar_Dmi = 0;
247           WriteBr = 1;
248           Ctrl_ext = 0;
249           ctrlALU = 0;
250           mux1 = 0;
251           mux2 = 0;
252           mux3 = 1;
253           WriteMemD = 0;
254           mux4 = 0;
255           mux5 = 0;
256           fout = 0;
257           mux6 = 0;
258       end
259   6'b001110:begin //STORE
260           jump_stop = 0;
261           HaltPC = 0;
262           ResetPC = 0;
263           gravar_Dmi = 0;
264           WriteBr = 0;
265           Ctrl_ext = 0;
266           ctrlALU = 1;
267           mux1 = 1;
268           mux2 = 0;
269           mux3 = 0;
270           WriteMemD = 1;
271           mux4 = 0;
272           mux5 = 0;
273           fout = 0;
274           mux6 = 0;
275       end
276   6'b001111:begin //MOVE
277           jump_stop = 0;
278           HaltPC = 0;
279           ResetPC = 0;
280           gravar_Dmi = 0;
281           WriteBr = 1;
282           Ctrl_ext = 0;
283           ctrlALU = 6'b000000;
284           mux1 = 1;
285           mux2 = 0;
286           mux3 = 1;
287           WriteMemD = 0;
288           mux4 = 0;
289           mux5 = 0;
290           fout = 0;

```

```
291                         mux6 = 0;
292                     end
293             6'b010000:begin // JUMPI
294                 jump_stop = 0;
295                 HaltPC = 0;
296                 ResetPC = 0;
297                 gravar_Dmi = 0;
298                 WriteBr = 0;
299                 Ctrl_ext = 0;
300                 ctrlALU = 1;
301                 mux1 = 1;
302                 mux2 = 0;
303                 mux3 = 1;
304                 WriteMemD = 0;
305                 mux4 = 1;
306                 mux5 = 1;
307                 fout = 0;
308                 mux6 = 0;
309             end
310             6'b010001:begin // JMP
311                 jump_stop = 0;
312                 HaltPC = 0;
313                 ResetPC = 0;
314                 gravar_Dmi = 0;
315                 WriteBr = 0;
316                 Ctrl_ext = 0;
317                 ctrlALU = 1;
318                 mux1 = 1;
319                 mux2 = 0;
320                 mux3 = 1;
321                 WriteMemD = 0;
322                 mux4 = 0;
323                 mux5 = 1;
324                 fout = 0;
325                 mux6 = 0;
326             end
327             6'b010010:begin // JE
328                 if(JE == 1'b1)begin
329                     jump_stop = 0;
330                     HaltPC = 0;
331                     ResetPC = 0;
332                     gravar_Dmi = 0;
333                     WriteBr = 0;
334                     Ctrl_ext = 0;
335                     ctrlALU = 6'b0000010;
336                     mux1 = 1;
337                     mux2 = 0;
338                     mux3 = 1;
339                     WriteMemD = 0;
340                     mux4 = 0;
341                     mux5 = 1;
342                     fout = 0;
343                     mux6 = 0;
344                 end else begin
345                     jump_stop = 0;
346                     HaltPC = 0;
347                     ResetPC = 0;
348                     gravar_Dmi = 0;
349                     WriteBr = 0;
350                     Ctrl_ext = 0;
```

```

351                               ctrlALU = 6'b000010;
352                               mux1 = 0;
353                               mux2 = 0;
354                               mux3 = 0;
355                               WriteMemD = 0;
356                               mux4 = 0;
357                               mux5 = 0;
358                               fout = 0;
359                               mux6 = 0;
360
361                         end
362
363           6'b010011:begin //JNE
364             if(JE == 1'b0)begin
365               jump_stop = 0;
366               HaltPC = 0;
367               ResetPC = 0;
368               gravar_Dmi = 0;
369               WriteBr = 0;
370               Ctrl_ext = 0;
371               ctrlALU = 6'b000010;
372               mux1 = 1;
373               mux2 = 0;
374               mux3 = 1;
375               WriteMemD = 0;
376               mux4 = 0;
377               mux5 = 1;
378               fout = 0;
379               mux6 = 0;
380             end else begin
381               jump_stop = 0;
382               HaltPC = 0;
383               ResetPC = 0;
384               gravar_Dmi = 0;
385               WriteBr = 0;
386               Ctrl_ext = 0;
387               ctrlALU = 0;
388               mux1 = 0;
389               mux2 = 0;
390               mux3 = 0;
391               WriteMemD = 0;
392               mux4 = 0;
393               mux5 = 0;
394               fout = 0;
395               mux6 = 0;
396             end
397
398           6'b010100:begin //JA
399             if(JA == 1)begin
400               jump_stop = 0;
401               HaltPC = 0;
402               ResetPC = 0;
403               gravar_Dmi = 0;
404               WriteBr = 0;
405               Ctrl_ext = 0;
406               ctrlALU = 6'b000010;
407               mux1 = 1;
408               mux2 = 0;
409               mux3 = 1;
410               WriteMemD = 0;
411               mux4 = 0;

```

```

411                         mux5 = 1;
412                         fout = 0;
413                         mux6 = 0;
414                     end else begin
415                         jump_stop = 0;
416                         HaltPC = 0;
417                         ResetPC = 0;
418                         gravar_Dmi = 0;
419                         WriteBr = 0;
420                         Ctrl_ext = 0;
421                         ctrlALU = 0;
422                         mux1 = 0;
423                         mux2 = 0;
424                         mux3 = 0;
425                         WriteMemD = 0;
426                         mux4 = 0;
427                         mux5 = 0;
428                         fout = 0;
429                         mux6 = 0;
430                     end
431                 end
432             6'b010101:begin //JNA
433                 if(JA == 0)begin
434                     jump_stop = 0;
435                     HaltPC = 0;
436                     ResetPC = 0;
437                     gravar_Dmi = 0;
438                     WriteBr = 0;
439                     Ctrl_ext = 0;
440                     ctrlALU = 6'b000010;
441                     mux1 = 1;
442                     mux2 = 0;
443                     mux3 = 1;
444                     WriteMemD = 0;
445                     mux4 = 0;
446                     mux5 = 1;
447                     fout = 0;
448                     mux6 = 0;
449                 end else begin
450                     jump_stop = 0;
451                     HaltPC = 0;
452                     ResetPC = 0;
453                     gravar_Dmi = 0;
454                     WriteBr = 0;
455                     Ctrl_ext = 0;
456                     ctrlALU = 0;
457                     mux1 = 0;
458                     mux2 = 0;
459                     mux3 = 0;
460                     WriteMemD = 0;
461                     mux4 = 0;
462                     mux5 = 0;
463                     fout = 0;
464                     mux6 = 0;
465                 end
466             end
467             6'b010110:begin //IN
468                 if(flagIN == 1)begin
469                     jump_stop = 0;
470                     HaltPC = 0;

```

```

471             ResetPC = 0;
472             gravar_Dmi = 0;
473             WriteBr = 1;
474             Ctrl_ext = 0;
475             ctrlALU = 6'b000010;
476             mux1 = 1;
477             mux2 = 0;
478             mux3 = 0;
479             WriteMemD = 0;
480             mux4 = 0;
481             mux5 = 0;
482             fout = 0;
483             mux6 = 1;
484         end else begin
485             jump_stop = 1;
486             HaltPC = 0;
487             ResetPC = 0;
488             gravar_Dmi = 0;
489             WriteBr = 1;
490             Ctrl_ext = 0;
491             ctrlALU = 0;
492             mux1 = 1;
493             mux2 = 0;
494             mux3 = 0;
495             WriteMemD = 0;
496             mux4 = 0;
497             mux5 = 0;
498             fout = 0;
499             mux6 = 1;
500         end
501     end
502     6'b010111:begin //OUT
503         jump_stop = 0;
504         HaltPC = 0;
505         ResetPC = 0;
506         gravar_Dmi = 0;
507         WriteBr = 0;
508         Ctrl_ext = 0;
509         ctrlALU = 0;
510         mux1 = 1;
511         mux2 = 0;
512         mux3 = 0;
513         WriteMemD = 0;
514         mux4 = 0;
515         mux5 = 0;
516         fout = 1;
517         mux6 = 0;
518     end
519     6'b011000:begin //NOP
520         jump_stop = 0;
521         HaltPC = 0;
522         ResetPC = 0;
523         gravar_Dmi = 0;
524         WriteBr = 0;
525         Ctrl_ext = 0;
526         ctrlALU = 0;
527         mux1 = 0;
528         mux2 = 0;
529         mux3 = 0;
530         WriteMemD = 0;

```

```
531                         mux4 = 0;
532                         mux5 = 0;
533                         fout = 0;
534                         mux6 = 0;
535         end
536     6'b011001:begin //HALT
537             jump_stop = 0;
538             HaltPC = 1;
539             ResetPC = 0;
540             gravar_Dmi = 0;
541             WriteBr = 0;
542             Ctrl_ext = 0;
543             ctrlALU = 0;
544             mux1 = 0;
545             mux2 = 0;
546             mux3 = 0;
547             WriteMemD = 0;
548             mux4 = 0;
549             mux5 = 0;
550             fout = 0;
551             mux6 = 0;
552         end
553     default: begin
554         jump_stop = 0;
555         HaltPC = 0;
556         ResetPC = 0;
557         gravar_Dmi = 0;
558         WriteBr = 0;
559         Ctrl_ext = 0;
560         ctrlALU = 0;
561         mux1 = 0;
562         mux2 = 0;
563         mux3 = 0;
564         WriteMemD = 0;
565         mux4 = 0;
566         mux5 = 0;
567         fout = 0;
568         mux6 = 0;
569     end
570 endcase
571 end
572 end
573
574 endmodule
```

# APÊNDICE B – Debouncer.v

Código do debouncer utilizado no botão de clock.

```

1 module debouncer(
2     input clk, //this is a 50MHz clock provided on FPGA pin PIN_Y2
3     input PB, //this is the input to be debounced
4     output reg PB_state //this is the debounced switch
5 );
6 /*This module debounces the pushbutton PB.
7 *It can be added to your project files and called as is:
8 *DO NOT EDIT THIS MODULE
9 */
10
11 // Synchronize the switch input to the clock
12 reg PB_sync_0;
13 always @(posedge clk) PB_sync_0 <= PB;
14 reg PB_sync_1;
15 always @(posedge clk) PB_sync_1 <= PB_sync_0;
16
17 // Debounce the switch
18 reg [15:0] PB_cnt;
19 always @(posedge clk)
20 if(PB_state==PB_sync_1)
21     PB_cnt <= 0;
22 else
23 begin
24     PB_cnt <= PB_cnt + 1'b1;
25     if(PB_cnt == 16'hffff) PB_state <= ~PB_state;
26 end
27 endmodule

```



# APÊNDICE C – Conversor Bin 32b para BCD

```

1 module BCD (numBCD, saida_centena, saida_dezena, saida_sinal, saida_unidade,
2                                     entradaUnidade, entradaDezena, entradaCentena,
3                                     entradaSinal);
4     input [31:0] numBCD;
5     integer i;
6     reg [9:0] numero;
7     output reg [3:0] entradaUnidade, entradaDezena, entradaCentena, entradaSinal;
8     output [6:0] saida_centena, saida_dezena, saida_sinal, saida_unidade;
9
10    display7seg display_unidade(entradaUnidade, saida_unidade);
11    display7seg display_dezena(entradaDezena, saida_dezena);
12    display7seg display_centena(entradaCentena, saida_centena);
13    display7seg display_sinal(entradaSinal, saida_sinal);
14
15    always @ (numBCD) begin
16        entradaCentena = 4'd0;
17        entradaDezena = 4'd0;
18        entradaUnidade = 4'd0;
19
20        if(numBCD == 32'b11111111111111111111111111111111)begin //display apagado
21            entradaCentena = 4'd11;
22            entradaDezena = 4'd11;
23            entradaUnidade = 4'd11;
24
25        end
26
27        else begin
28            if (numBCD[31] == 1) begin //numBCDero negativo
29                numero = numBCD - 1;
30                numero = ~numero;
31                entradaSinal = 4'd10;
32            end
33            else if (numBCD[31] == 0) begin //numero positivo
34                entradaSinal = 4'd11;
35                numero = numBCD;
36            end
37
38            for (i=9; i>=0; i=i-1) begin
39                if (entradaCentena >= 5) begin
40                    entradaCentena = entradaCentena + 3;
41                end
42                if (entradaDezena >= 5) begin
43                    entradaDezena = entradaDezena +3;
44                end
45                if (entradaUnidade >= 5) begin
46                    entradaUnidade = entradaUnidade +3;
47                end
48                entradaCentena = entradaCentena << 1;
49                entradaCentena[0] = entradaDezena[3];
50                entradaDezena = entradaDezena << 1;
51                entradaDezena[0] = entradaUnidade[3];
52                entradaUnidade = entradaUnidade << 1;
53            end
54        end
55    end

```

```
51                     entradaUnidade[0] = numero[i];  
52             end  
53         end  
54     end  
55 endmodule
```

# APÊNDICE D – Display de 7 segmentos

Código do display de 7 segmentos utilizado dentro do conversor binário para bcd.

```

1 module display7seg (entrada, saida);
2   input [3:0] entrada;
3   output reg [6:0] saida;
4
5   always @ (*) begin
6
7     case (entrada)
8       0:saida=7'b0000001;
9       1:saida=7'b1001111;
10      2:saida=7'b0010010;
11      3:saida=7'b0000110;
12      4:saida=7'b1001100;
13      5:saida=7'b0100100;
14      6:saida=7'b0100000;
15      7:saida=7'b0001111;
16      8:saida=7'b0000000;
17      9:saida=7'b0000100;
18      10:saida=7'b1111110; //sinal de menos
19      default:saida=7'b1111111; //display apagado
20    endcase
21  end
22
23 endmodule

```



# APÊNDICE E – extIN.v

Código do extensor de bits para entrada de 15 bits.

```
1 module extIN(in15, extendido);
2     input [14:0]in15;
3
4     output reg [31:0]extendido;
5     always @( * ) begin
6
7         if(in15[14] == 1)begin
8             extendido = {17'b1111111111111111,in15};
9         end else
10            extendido = {17'b0000000000000000,in15};
11
12     end
13
14 endmodule
```