Prática Nº11 - SEL0606

Alunos (Bancada 5):
Vitor Alexandre Garcia Vaz - 14611432
Gabriel Dezejácomo Maruschi - 14571525

Sumário

Lista de Imagens

1 Introdução

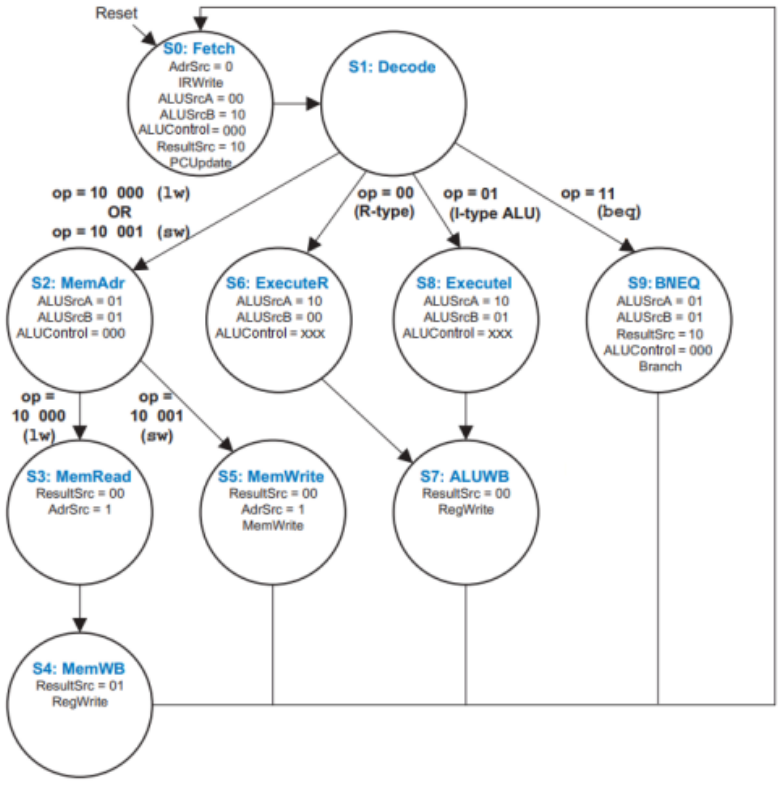
1.1 Objetivos

Nesta prática do Laboratório de Sistemas Digitais, implementamos uma Unidade de Controle para a arquitetura RV16Cm (RISC-V de 16 bits com conjunto Compacto de instruções modificadas) cuja funcionalidade é baseada no conceito de Máquinas de estado de finitos (FSM). Utilizamos a linguaguem VHDL no software Quartus, e testamos no kit DE10-LITE (MAX 10 10M50DAF484C7G) bem como no software ModelSim.

1.2 Máquina de estados finitos

As máquinas de estados finitos são circuitos sequenciais nos quais estímulos podem gerar resultados diferentes a depender do estado em que se encontra o circuito. Esta sempre estará em apenas um dos estados definidos pelo projeto, o qual guarda informações sobre estados passados.
A representação da FSM pode ser feita com o diagrama de estados: Os nós representam os estados enquanto os valores das setas são os valores de entradas que geram transição.
No diagrama abaixo, utilizado para modelagem da unidade de controle desenvolvida, as entradas não apresentadas nas setas são *don't cares* para as respectivas transições. Além disso, estão representados também dentro dos nós o valor das saídas geradas pelos estados.

Figure 1: Diagrama de estados



Fonte: Instrução da prática 11

1.3 Unidade de controle

A Unidade de Controle desenvolvida baseia-se na arquitetura RV16Cm (RISC-V de 16 bits com conjunto Compacto de instruções modificadas) e gera sinais de controle a partir dos valores de entrada e do estado anterior. Seu funcionamento é baseado no diagrama de estados da figura 1. Os sinais que compõe a máquina são dados a seguir:

Sinais de entrada

- *clk*: clock;
- *clr*: clear;
- *zero*: flag zero;
- *op/funct3*: Código da operação.

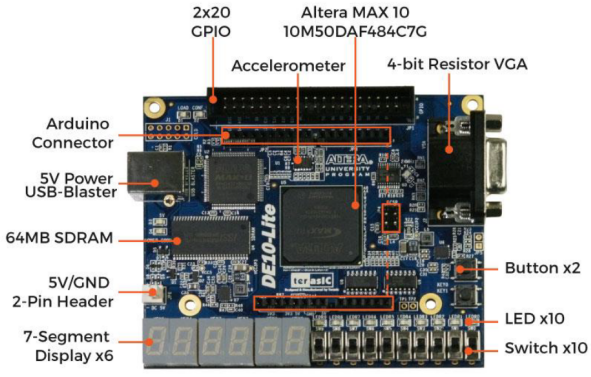
Sinais de saída

- *PCWrite*: Escrita no registrador PC (Program Counter);
- *AdrSrc*: Seleção do endereço de busca na memória;
- *MemWrite*: Flag de escrita na memória;
- *IRWrite*: Flag de escrita do código da instrução em um registrador auxiliar;
- *ResultSrc*: Seleção de dados processados ou lidos na memória;
- *ALUControl*: Seleção da operação da Unidade Lógica Aritmética;
- *ALUSrcA/B*: Seletores das entradas para a ULA;
- *ImmSrc*: Seletor de operação do módulo *Extent* (opera com valores imediatos);
- *RegWrite*: Flag de escrita no banco de registradores;

2 Materiais e métodos

O código utilizado para desenvolver a máquina de estado finitos (FSM) foi escrito em VHDL e compilado no Quartus. A fim de testá-los no DE10-LITE, desenvolvemos também um interface cujo objetivo foi redirecionar os sinais dos pinos para os módulos e vice-versa. Utilizamos também o software ModelSim da Intel para visualizar as ondas de entrada e saída da FSM.

Figure 2: Kit DE10-LITE



Fonte: DE10-Lite User Manual

2.1 Interface

A interface, codificada como é mostrado abaixo (3), utiliza as chaves SW(0) e SW(1) para **op**, SW(2) a SW(4) para **funct3** e SW(5) para **zero**. KEY(0) e KEY(1) foram usados para **clk** e **clr**, respectivamente. Os sinais de saída foram direcionados para os LEDS e para o módulo decoficador do display de 7 segmentos.

Figure 3: Código da interface 1

```
-- Version: 1.1
-- Date: 07/11/2024
-- Owners: Gabriel D. Maruschi
--            Vitor Garcia Vaz

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY DE10_LITE_FSM IS
    PORT(
        SW : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
        KEY : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        LEDR : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
        HEX0 : OUT STD_LOGIC_VECTOR(7 downto 0);
        HEX1 : OUT STD_LOGIC_VECTOR(7 downto 0);
        HEX2 : OUT STD_LOGIC_VECTOR(7 downto 0);
        HEX3 : OUT STD_LOGIC_VECTOR(7 downto 0);
        HEX4 : OUT STD_LOGIC_VECTOR(7 downto 0)
    );
END DE10_LITE_FSM;


ARCHITECTURE estrutural OF DE10_LITE_FSM IS

    -- Vetores de sinais de controle internos
    signal intResultSrc : std_logic_vector(1 downto 0);
    signal intALUControl : std_logic_vector(2 downto 0);
    signal intALUSrcB : std_logic_vector(1 downto 0);
    signal intALUSrcA : std_logic_vector(1 downto 0);
    signal intImmSrc : std_logic_vector(1 downto 0);

    -- Vetores de sinais de controle internos extendidos para 4 bits
    signal extResultSrc : std_logic_vector(3 downto 0);
    signal extALUControl : std_logic_vector(3 downto 0);
    signal extALUSrcB : std_logic_vector(3 downto 0);
    signal extALUSrcA : std_logic_vector(3 downto 0);
    signal extImmSrc : std_logic_vector(3 downto 0);
```

Fonte: os autores

Figure 4: Código da interface 2

```
BEGIN

    -- Instanciação do módulo FSM
    FSM: entity work.FSM
        PORT MAP (
            clk => not KEY(0),
            clr => KEY(1),
            zero => SW(5),
            op => SW(1 downto 0),
            funct3 => SW(4 downto 2),
            PCWrite => LEDR(0),
            AdrSrc => LEDR(1),
            MemWrite => LEDR(2),
            IRWrite => LEDR(3),
            ResultSrc => intResultSrc(1 downto 0),
            ALUControl => intALUControl(2 downto 0),
            ALUSrcB => intALUSrcB(1 downto 0),
            ALUSrcA => intALUSrcA(1 downto 0),
            ImmSrc => intImmSrc(1 downto 0),
            RegWrite => LEDR(4)
        );

    -- Extensão dos sinais de controle para 4 bits
    extResultSrc <= "00" & intResultSrc;
    extALUControl <= '0' & intALUControl;
    extALUSrcB <= "00" & intALUSrcB;
    extALUSrcA <= "00" & intALUSrcA;
    extImmSrc <= "00" & intImmSrc;

    -- Displays de 7 segmentos
    display1: entity work.hex27seg
        PORT MAP (
            hexa => extResultSrc,
            segments => HEX0
        );

    display2: entity work.hex27seg
        PORT MAP (
            hexa => extALUControl,
            segments => HEX1
        );

    display3: entity work.hex27seg
        PORT MAP (
            hexa => extALUSrcB,
            segments => HEX2
        );

    display4: entity work.hex27seg
        PORT MAP (
            hexa => extALUSrcA,
            segments => HEX3
        );

    display5: entity work.hex27seg
        PORT MAP (
            hexa => extImmSrc,
            segments => HEX4
        );

END estrutural;
```

Fonte: os autores

2.2 Máquina de Estados Finitos

A FSM foi desenvolvida em VHDL e utiliza estruturas de **PROCESS** para avaliar mudanças concorrentes nas entradas, bem como os respectivos estados a fim de definir as saídas e o próximo estado.

Figure 5: Máquina de Estados Finitos (1)

```vhdl
-- Version: 1.1
-- Date: 07/11/2024
-- Owners: Gabriel D. Maruschi
--            Vitor Garcia Vaz

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY FSM is
    PORT(

        -- Entradas
        clk     : in std_logic;
        clr     : in std_logic;
        zero    : in std_logic;
        op      : in std_logic_vector(1 downto 0);
        funct3  : in std_logic_vector(2 downto 0);

        -- Saídas (Sinais de controle)
        PCWrite     : out std_logic;
        AdrSrc      : out std_logic;
        MemWrite    : out std_logic;
        IRWrite     : out std_logic;
        ResultSrc   : out std_logic_vector(1 downto 0);
        ALUControl  : out std_logic_vector(2 downto 0);
        ALUSrcB     : out std_logic_vector(1 downto 0);
        ALUSrcA     : out std_logic_vector(1 downto 0);
        ImmSrc      : out std_logic_vector(1 downto 0);
        RegWrite    : out std_logic
    );

END FSM;


ARCHITECTURE estrutural OF FSM IS

    -- Declaração dos estados s0 a s9
    type state_type is (s0, s1, s2, s3, s4, s5, s6, s7, s8, s9);

    -- Estado atual e próximo estado
    signal state, next_state : state_type; --  tem que ser do mesmo tipo que os estados definidos

BEGIN

    -- Lógica de controle do próximo estado (verifica reset e transição positiva de clock)
    PROCESS(clk, clr)
    BEGIN

        if clr = '1' then
            state <= s0;

        elsif clk'event and clk = '1' then
            state <= next_state;

        end if;

    END PROCESS;
```

Fonte: os autores

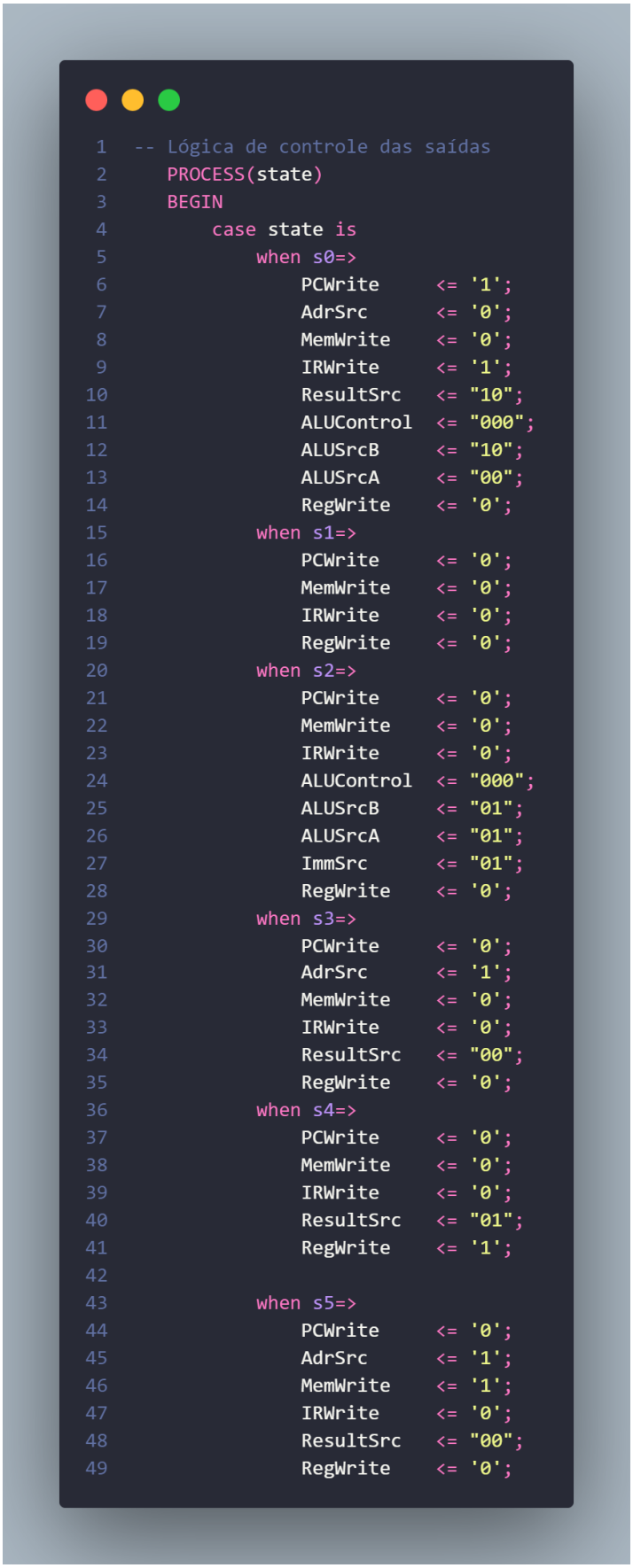
Na construção da arquitetura, definiu-se o tipo de dado **state_type** cujos exemplares foram **state** e **next_state**. A função deste dado definido foi encapsular para cada estado do projeto os sinais de saída referentes, facilitando na escrita, leitura do código e simulação.

Figure 6: Máquina de Estados Finitos (2)

```
1
2  -- Lógica de controle do próximo estado
3  PROCESS(state)
4  BEGIN
5      case state is
6          when s0=>
7              next_state <= s1;
8          when s1=>
9              if op = "00" then
10                 next_state <= s6;
11
12             elsif op = "01" then
13                 next_state <= s8;
14
15             elsif op = "10" then
16                 next_state <= s2;
17
18             elsif op = "11" then
19                 next_state <= s9;
20
21             end if;
22         when s2=>
23             if funct3(0) = '0' then
24                 next_state <= s3;
25
26             else
27                 next_state <= s5;
28
29             end if;
30         when s3=>
31             next_state <= s4;
32         when s4=>
33             next_state <= s0;
34         when s5=>
35             next_state <= s0;
36         when s6=>
37             next_state <= s7;
38         when s7=>
39             next_state <= s0;
40         when s8=>
41             next_state <= s7;
42         when s9=>
43             next_state <= s0;
44     end case;
45
46 END PROCESS;
```

Fonte: os autores

Figure 7: Máquina de Estados Finitos (3)



```
-- Lógica de controle das saídas
   PROCESS(state)
   BEGIN
       case state is
           when s0=>
               PCWrite     <= '1';
               AdrSrc      <= '0';
               MemWrite    <= '0';
               IRWrite     <= '1';
               ResultSrc   <= "10";
               ALUControl  <= "000";
               ALUSrcB     <= "10";
               ALUSrcA     <= "00";
               RegWrite    <= '0';
           when s1=>
               PCWrite     <= '0';
               MemWrite    <= '0';
               IRWrite     <= '0';
               RegWrite    <= '0';
           when s2=>
               PCWrite     <= '0';
               MemWrite    <= '0';
               IRWrite     <= '0';
               ALUControl  <= "000";
               ALUSrcB     <= "01";
               ALUSrcA     <= "01";
               ImmSrc      <= "01";
               RegWrite    <= '0';
           when s3=>
               PCWrite     <= '0';
               AdrSrc      <= '1';
               MemWrite    <= '0';
               IRWrite     <= '0';
               ResultSrc   <= "00";
               RegWrite    <= '0';
           when s4=>
               PCWrite     <= '0';
               MemWrite    <= '0';
               IRWrite     <= '0';
               ResultSrc   <= "01";
               RegWrite    <= '1';

           when s5=>
               PCWrite     <= '0';
               AdrSrc      <= '1';
               MemWrite    <= '1';
               IRWrite     <= '0';
               ResultSrc   <= "00";
               RegWrite    <= '0';
```

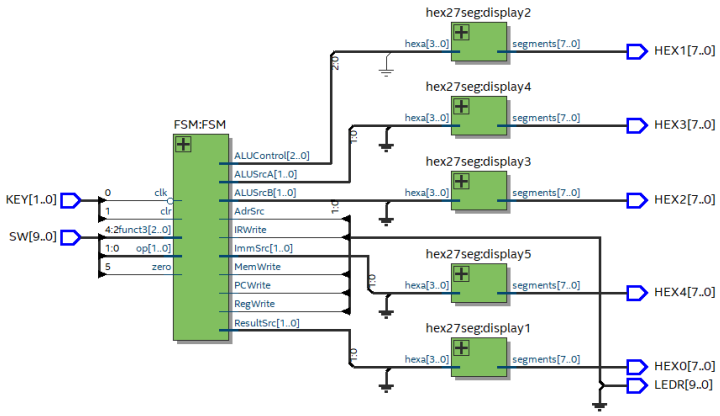
Fonte: os autores

3 Conclusão

3.1 Cicuito RTL

A partir do código em VHDL feito em laborratório, e usando a ferramenta de síntese disponibilizada pelo quartus, o seguinte circuito, referente à interface entre a máquina de estados da unidade de controle e os pinos da FPGA, foi obtido:

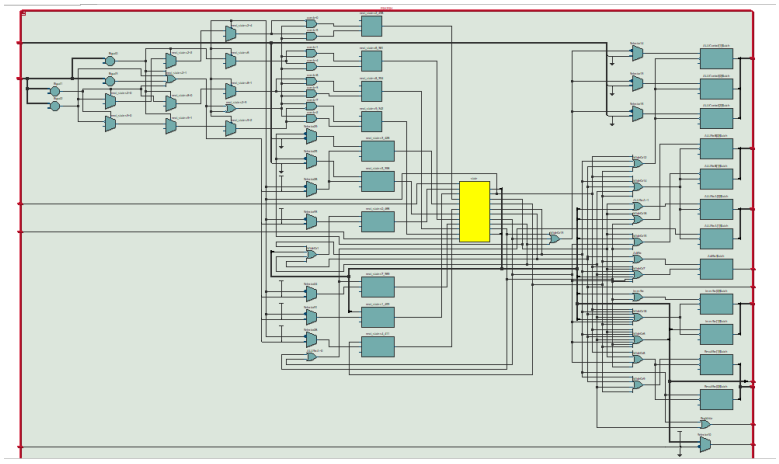
Figure 8: Visuzalização do circuito referente à interface



Fonte: os autores

E para a máquina de estados em si, obtemos o seguinte circuito:

Figure 9: Visualização do circuito referente à Unidade de Controle

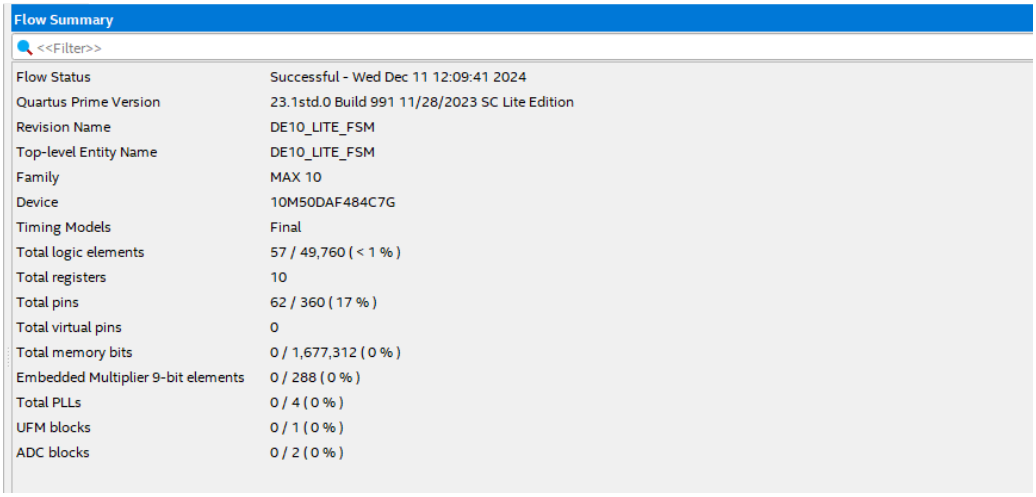


Fonte: os autores

3.2 Número de células lógicas

Ademais, por meio do resumo do funcionamento e constituição do circuito descrito em VHDL no software (fig10), chegamos à conclusão de que o circuito sintetizado apresentou um uso de 10 registradores, 57 elementos lógicos e 60 pinos do dispositivo reconfigurável (FPGA).

Figure 10: Resumo de funcionamento



Flow Summary

<<Filter>>

Flow Status	Successful - Wed Dec 11 12:09:41 2024
Quartus Prime Version	23.1std.0 Build 991 11/28/2023 SC Lite Edition
Revision Name	DE10_LITE_FSM
Top-level Entity Name	DE10_LITE_FSM
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	57 / 49,760 (< 1 %)
Total registers	10
Total pins	62 / 360 (17 %)
Total virtual pins	0
Total memory bits	0 / 1,677,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

Fonte: os autores

3.3 Funcionamento da máquina de estados no Modelsim

Os testes de funcionamento do circuito da unidade de controle foi feito através do software de simulção de circuitos digitais Modelsim. Nesse sentido, foram feitas 4 simulações: uma para as instruções tipo R e I; uma para a instrução BNEQ; uma para as instruções tipo L; e a última para as isntruções do tipo S.

Com isso, tanto para elaboração da descrição do hardware quanto para a conferência do funcionamento correto do circuito, tomamos como base a tabela de estados e saídas da fig.11, a qual foi elaborada conforme às especificações da arquitetura RISC de 16 bits proposta nas instruções dessa prática.

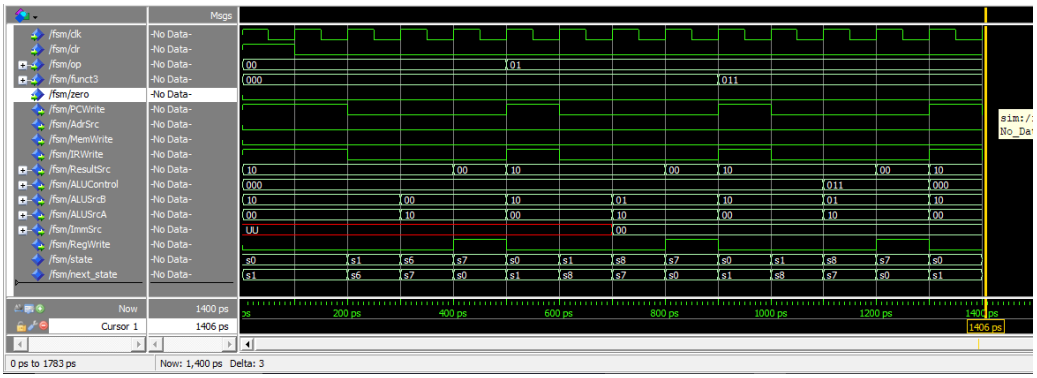
Figure 11: Tabela de estados (e saídas) da unidade de controle

Sinal / Estado	S0: Fetch	S1 : Decode	S2: MemAdr	S3: MemRead	S4: MemWB	S5: MemWrite	S6: ExecuteR	S7: ALUWB	S8: ExecuteI	S9: BNEQZ
PCWrite	1	0	0	0	0	0	0	0	0	/zero
AdrSrc	0	x	x	1	x	1	x	x	x	x
MemWrite	0	0	0	0	0	1	0	0	0	0
IRWrite	1	0	0	0	0	0	0	0	0	0
ResultSrc[1:0]	10_2	xx	xx	00_2	01_2	00_2	xx	00_2	xx	10
ALUControl[2:0]	000_2	xxx	000_2	xxx	xxx	xxx	funct3	funct3	funct3	000_2
ALUSrcB[1:0]	10_2	xx	01_2	xx	xx	xx	00_2	xx	01_2	01_2
ALUSrcA[1:0]	00_2	xx	01_2	xx	xx	xx	10_2	xx	10_2	01_2
ImmSrc[1:0]	xx	xx	01_2	xx	xx	xx	xx	xx	00_2	10_2
RegWrite	0	0	0	0	1	0	0	1	0	0

Fonte: os autores

INSTRUÇÕES TIPO R/I

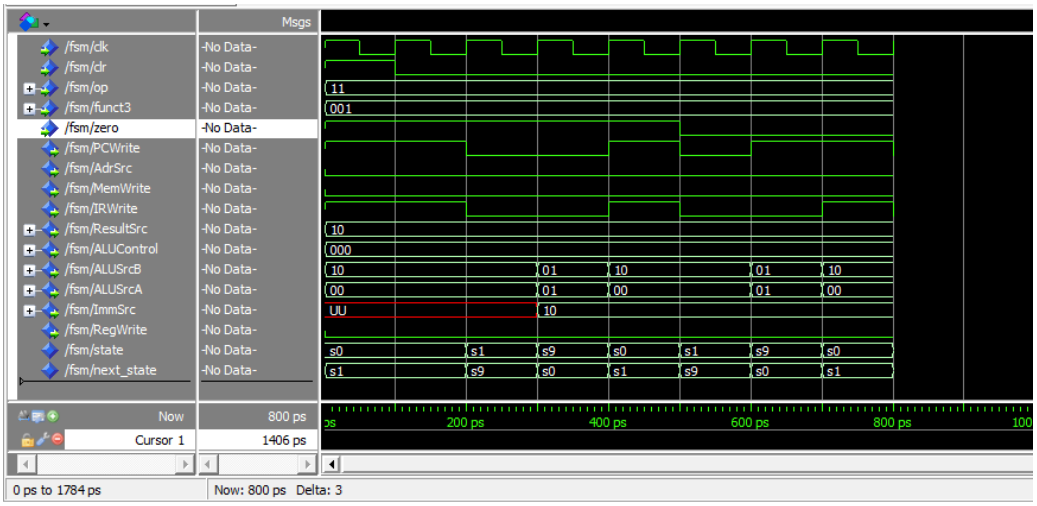
Figure 12: Simulação das instruções tipo R/I



Fonte: os autores

INSTRUÇÃO BNEQ

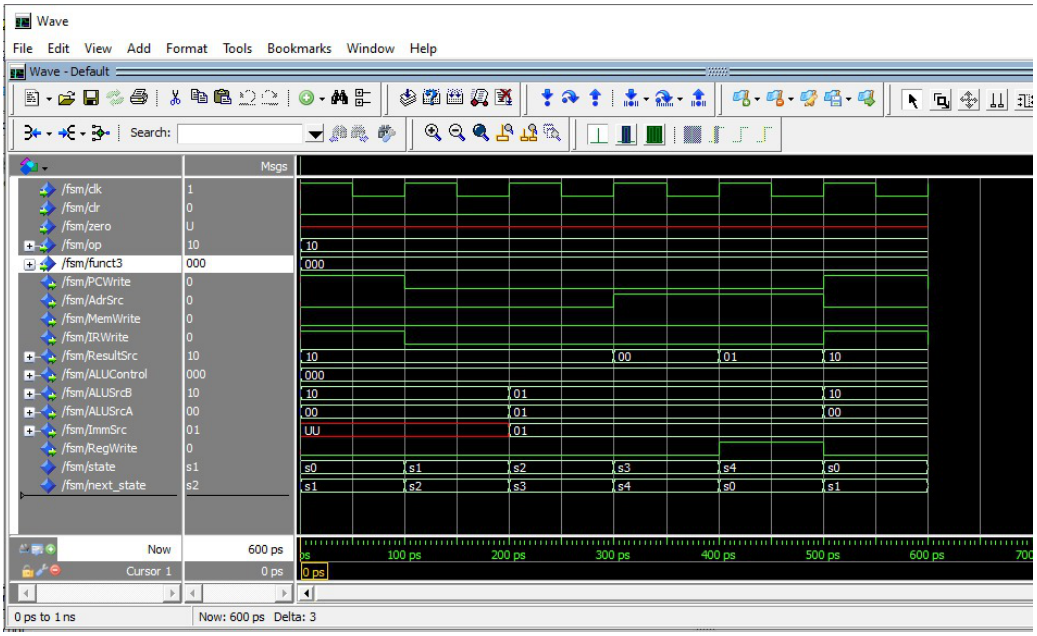
Figure 13: Simulação da instrução BNEQ



Fonte: os autores

INSTRUÇÃO TIPO L

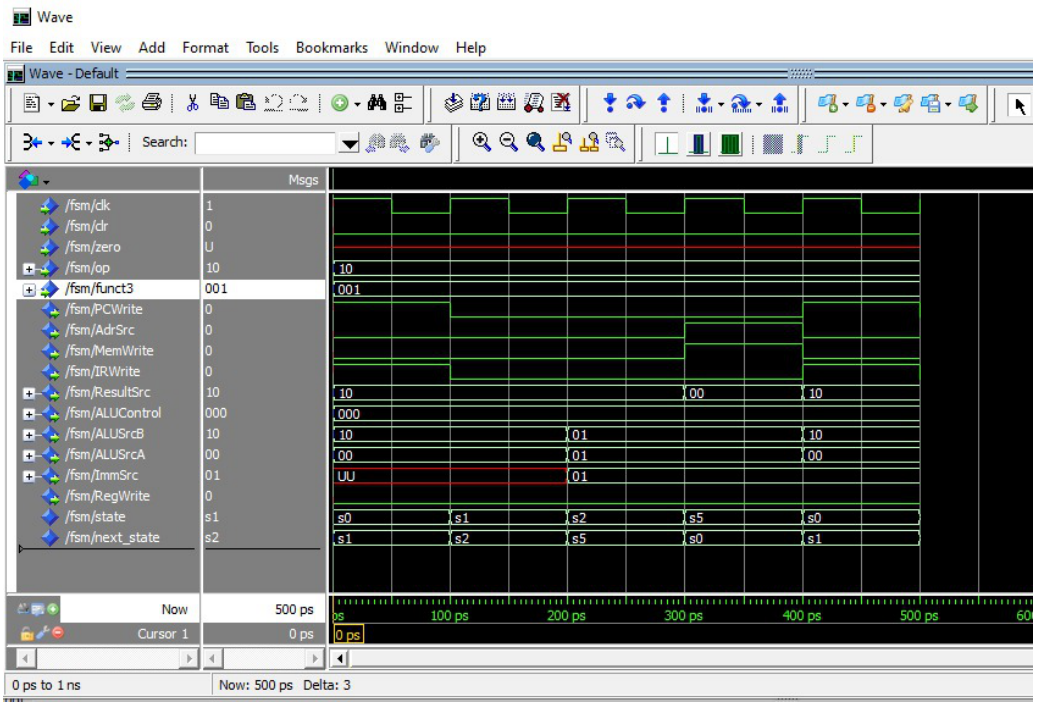
Figure 14: Simulação de isntrução do tipo L



Fonte: os autores

INSTRUÇÃO TIPO S

Figure 15: Simulaçãoo de instrução do tipo S



Fonte: os autores

3.4 Resultados

Por fim , em todos os testes, os estados de cada instrução - indicada pelas entradas op e funct3 - consecutiram no próximo estado esperado e em saídas esperadas, conforme a tabela de estados e saídas da fig.11 e , dessa forma, houve êxito na implementação da máquina de estados.