PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS – CAMPUS I

ENGENHARIA DE COMPUTAÇÃO

ARQUITETURA DE COMPUTADORES

Carlos Henrique Vieira Marques – 18720367

Giuliano Marques Sanfins – 17142837

João Henrique Pereira –18712919

Pedro Zacarias –17144874

PROJETO 2

IMPLEMENTAÇÃO DE UM PROCESSADOR SIMPLIFICADO EM VHDL

CAMPINAS

Outubro de 2019

SUMÁRIO

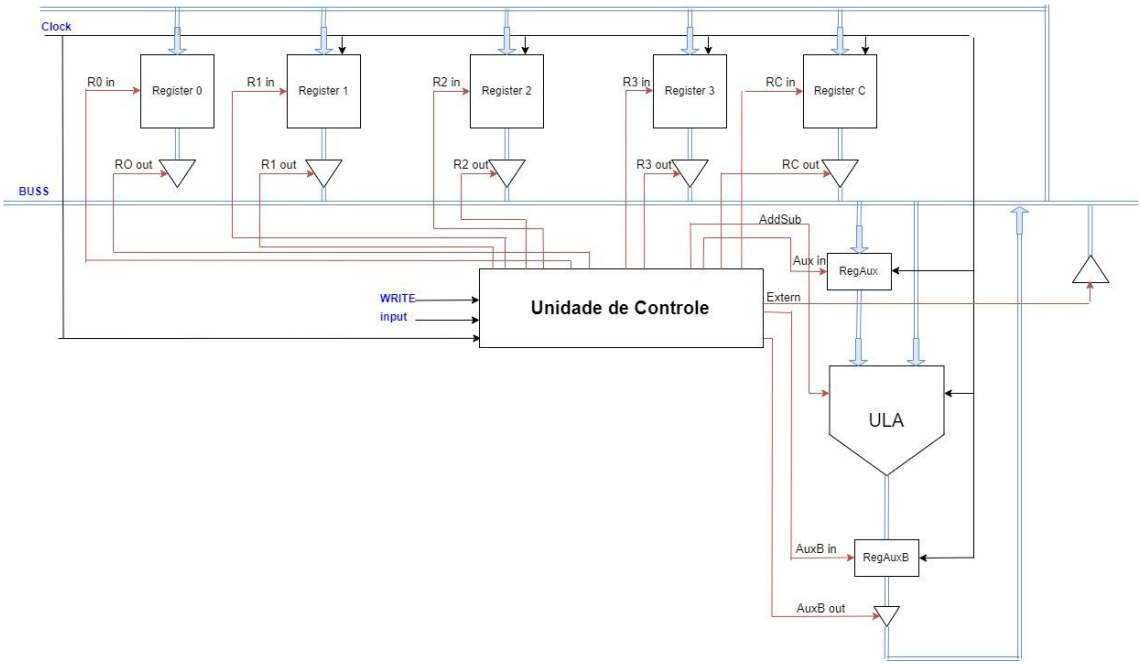
1. DESCRIÇÃO E TOPOLOGIA DA CPU

O respectivo projeto tem como objetivo desenvolver e simular uma CPU experimental em vhdl que execute as seguintes instruções:

Instrução	Significado	Descrição
MOV Ri,Rj	Ri <- Rj	Move
MOV Ri,Imed	Ri <- Imed	Move Immediate
XCHG Ri,Rj	Ri <- Rj e Rj <- Ri	Exchange
ADD Ri,Rj	Ri <- Ri + Rj	Add
ADDI Ri,Imed	Ri <- Ri + Imed	Add Immediate
SUB Ri,Rj	Ri <- Ri - Rj	Subtract
SUBI Ri,Imed	Ri <- Ri - Imed	Subtract Immediate
AND Ri,Rj	Ri <- Ri & Rj	And
ANDI Ri,Imed	Ri <- Ri & Imed	And Immediate
OR Ri,Rj	Ri <- Ri | Rj	Or
ORI Ri,Imed	Ri <- Ri | Imed	Or Immediate

(Figura 1: Instruções da CPU)

A CPU, mais especificamente a Unidade de Controle, aguarda uma instrução, que se validada aciona os registradores e a ULA através de sinais de controle, podendo escrever e ler valores dos registradores transmitindo-os através de barramentos e realizar operações lógicas e aritméticas de acordo com a instrução especificada.



(Figura 2: Topologia da CPU)

2. ESPECIFICAÇÃO

2.1. REGISTRADORES (QUANTIDADE, ENDEREÇO E TAMANHO)

No projeto foram utilizados quatro registradores principais e três registradores auxiliares. Cada registrador tem capacidade de armazenar 16bits e seus endereços são enumerados de 00b à 11b. Os registradores auxiliares nomeados por A, B, C não são referenciados nas instruções e têm a função de armazenar valores temporários, sendo A registrador que armazena uma das entradas da ULA, B a saída e C utilizado para a troca na instrução `XCHG`.

2.2. FORMATO DAS INSTRUÇÕES (OPCODE)

A instrução recebida tem um total de 16 bits. O formato de opcode utilizado foi o de 4 bits, devido ao fato de haver 11 instruções básicas, sendo elas de 0000 até 1010, que ocupam as posições de bits 15-12. Os endereços dos registradores ocupam 4 bits ao todo, ocupando a posição 11–8 na instrução. Os últimos 4 bits, ocupando os bits 7 - 0, serão preenchidos com valores 0 por não serem utilizados nas instruções que não possuem valores imediatos, caso seja utilizado uma instrução contendo valores imediatos os últimos 4 bits serão preenchidos com os valores necessários.

(Tabela 1: Instruções e Opcode)

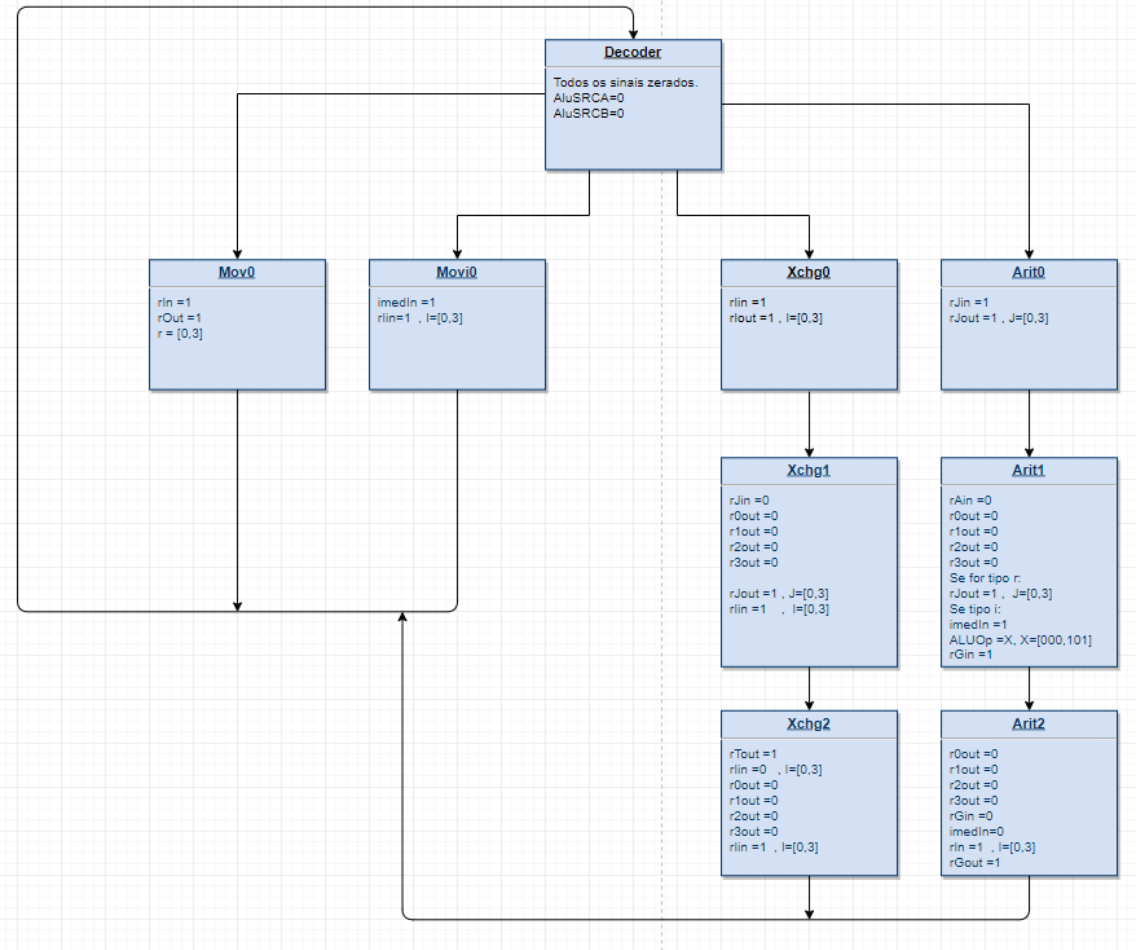
Instrução	**Opcode**	**Tipo de Instrução**
ADD	0000	Tipo R
SUB	0010	Tipo R
AND	0100	Tipo R
OR	0110	Tipo R
ADDI	0001	Tipo I
SUBI	0011	Tipo I
ANDI	0101	Tipo I
ORI	0111	Tipo I
MOV	1000	Tipo R
MOVI	1001	Tipo I
XCHG	1010	Tipo R

OPCODE – 4 bits	REGISTRADORES – 4 bits	IMEDIATO – 12 bits

(Tabela 2: Modelo de Instrução)

2.3. UC (UNIDADE DE CONTROLE)

O diagrama de estados da figura 3 representa a maneira como a unidade de controle opera:



(Figura 3: Máquina de Estados)

Sinais em bits	Definição
Clock	Usado para sistematizar ações, sendo gerado por oscilador externo à CPU.
Registradores: • R[0-3]in • R[0-3]out	R com sufixo in libera registrador para leitura quando sinal igual a 1. R com sufixo out libera valores do registrador para o bus quando sinal igual a 1.
imedin	Quando sinal de Imediato for igual a 1 será utilizado um buffer tri-state que controla a entrada de dados imediatos.
ALUOp	Sinais que indicam para a ULA qual operação deve ser feita. De acordo com a tabela de Instruções e Opcode.

(Tabela 3: Sinais e Definição)

3. RESULTADOS

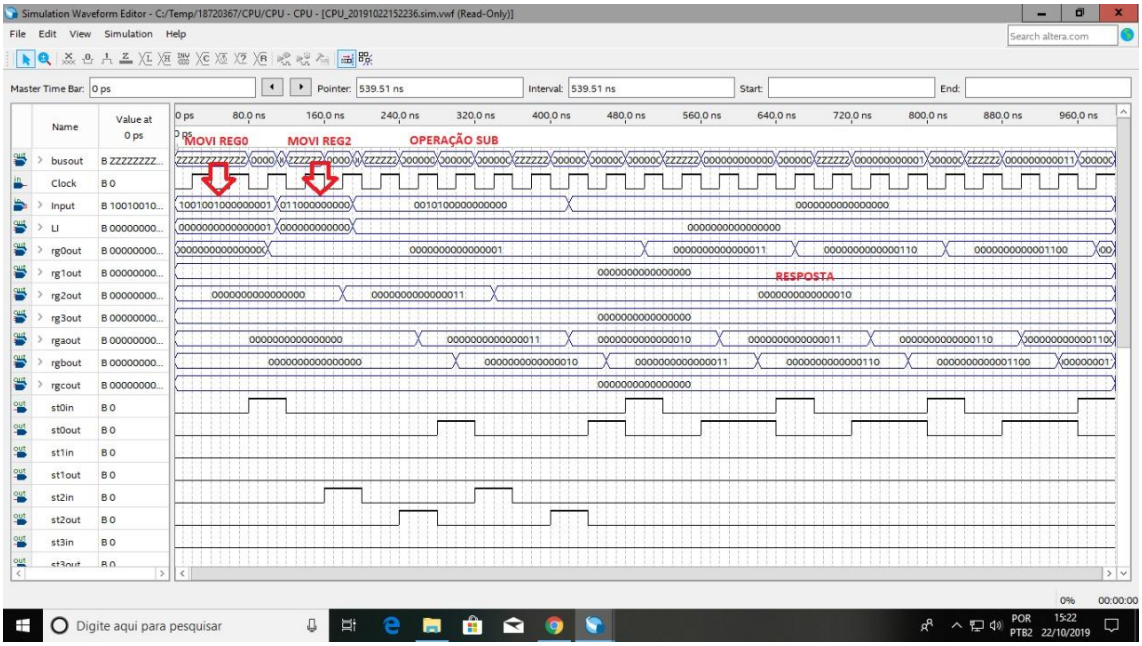
3.1. TESTES REALIZADOS

As seguintes instruções foram simuladas:

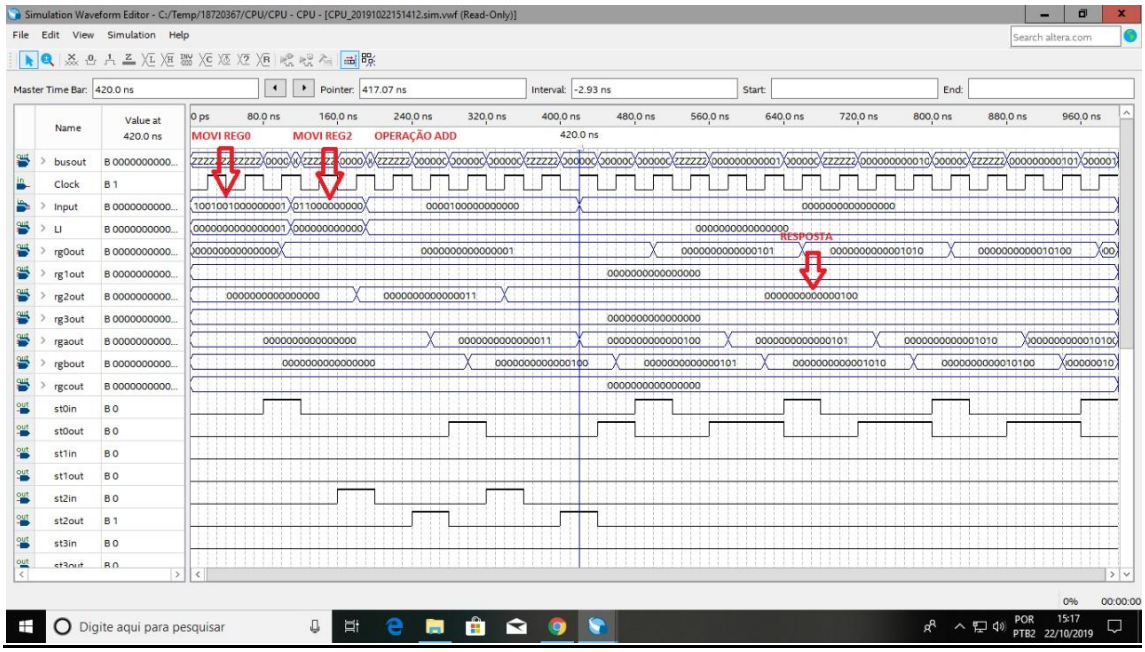
Arquivo	Instrução	Descrição
Waveform.vwf	00100000000000000	sub
Waveform2.vwf	00001000000000000	add
Waveform3.vwf	01101000000000000	Or
Waveform4.vwf	01001000000000000	And
Waveform5.vwf	10101000000000000	Xchg
Waveform6.vwf	00011000000000100	Addi
Waveform7.vwf	00111000000000001	Subi
Waveform8.vwf	0101100000000010	Andi
Waveform9.vwf	01111000000000100	Ori
Waveform10.vwf	10001000000000000	Mov

(Tabela 4: Instruções simuladas)

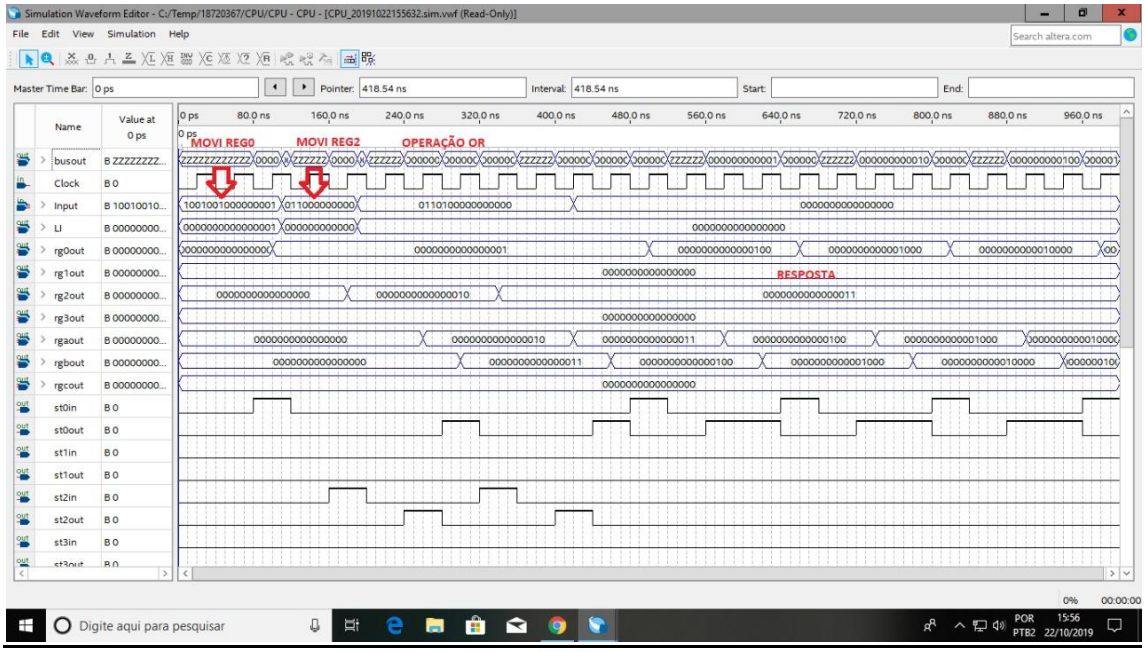
As instruções do tipo movi não estão presentes na tabela pois precedem todas as instruções testadas, entretanto é possível visualizá-las nas figuras abaixo:



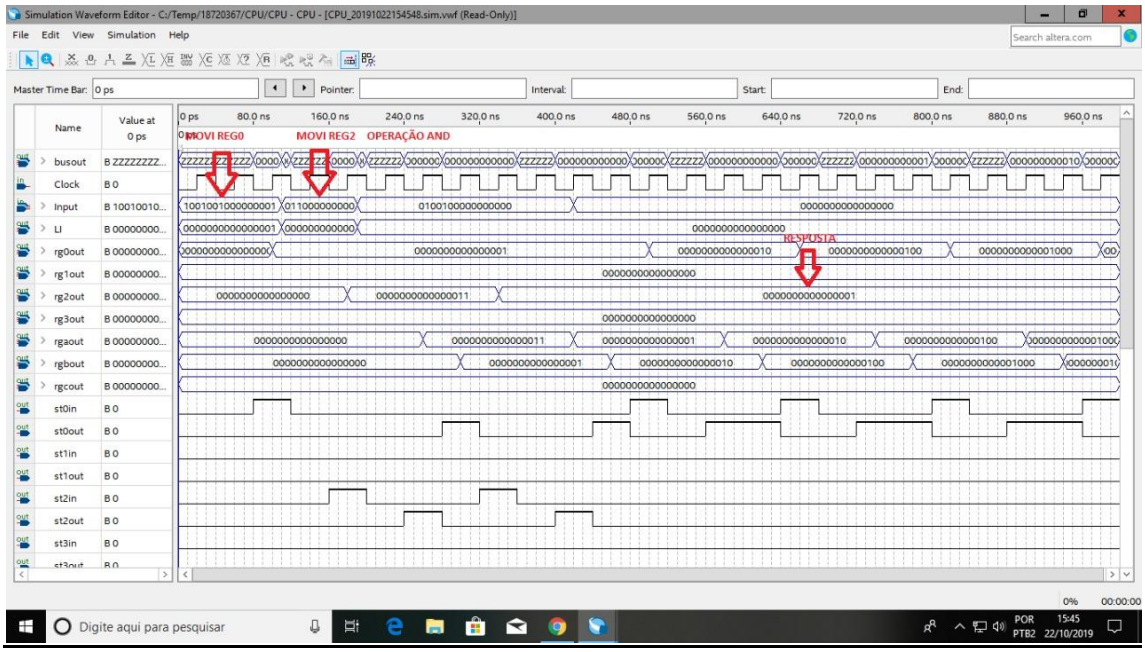
(Figura 4.1 – Sub)



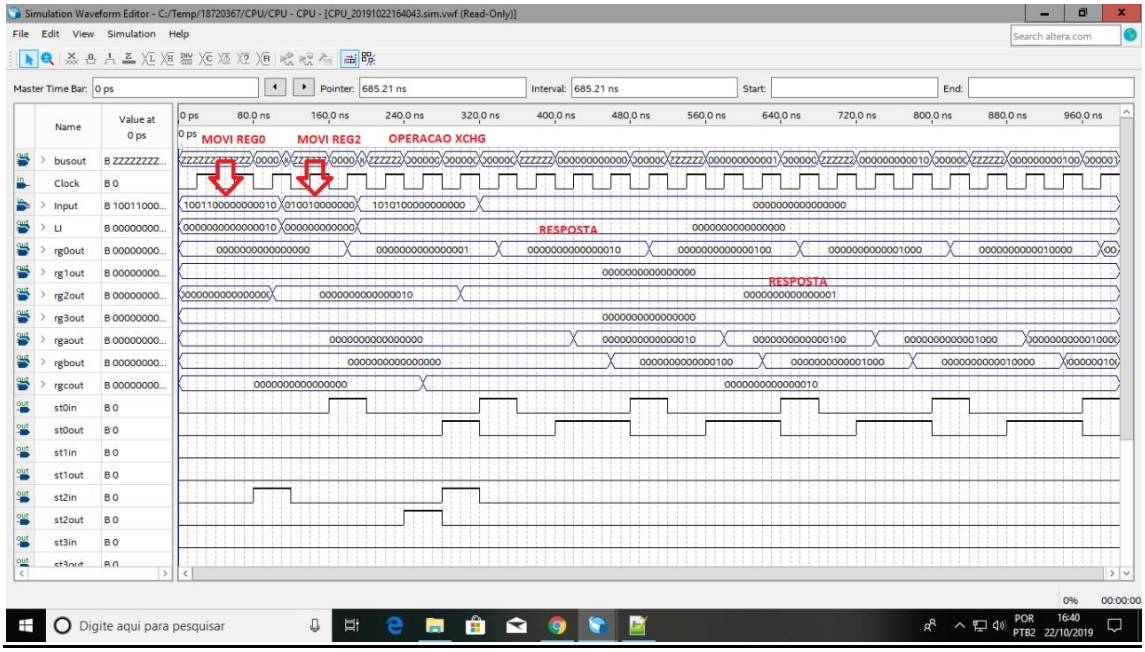
(Figura 4.2 – add)



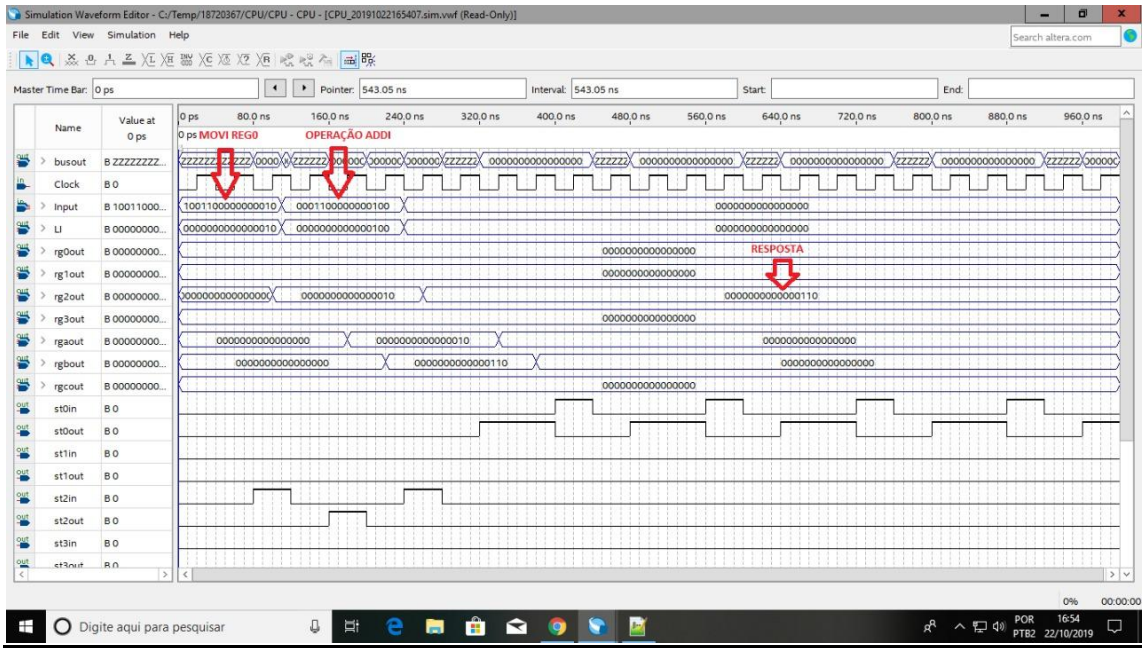
(Figura 4.3 – or)



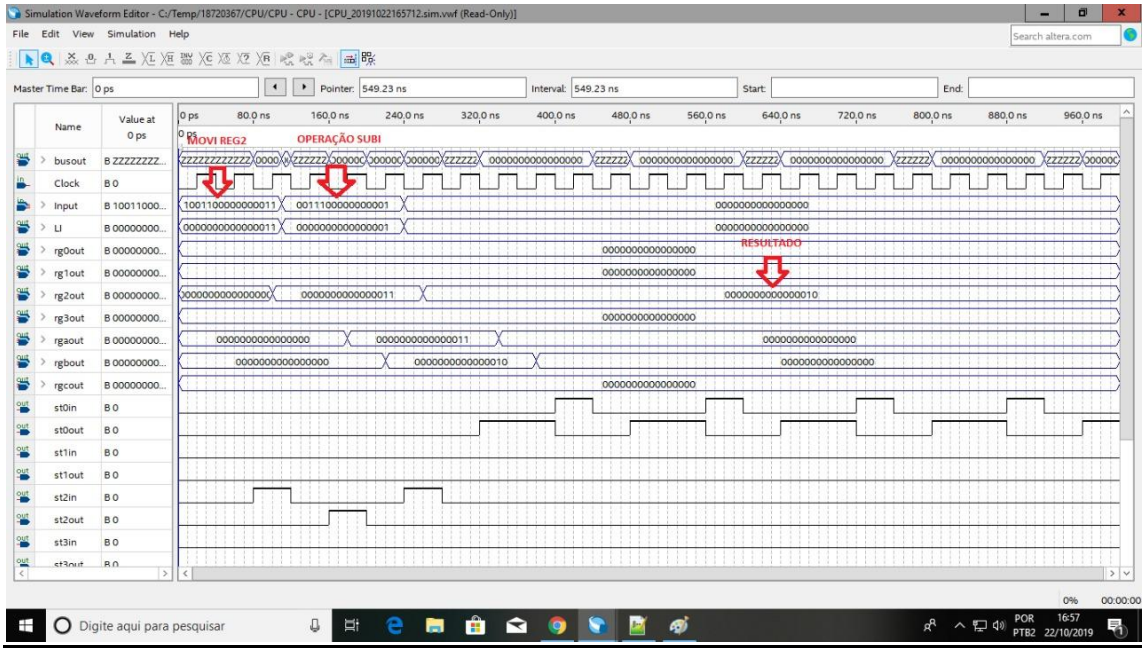
(Figura 4.4 – and)



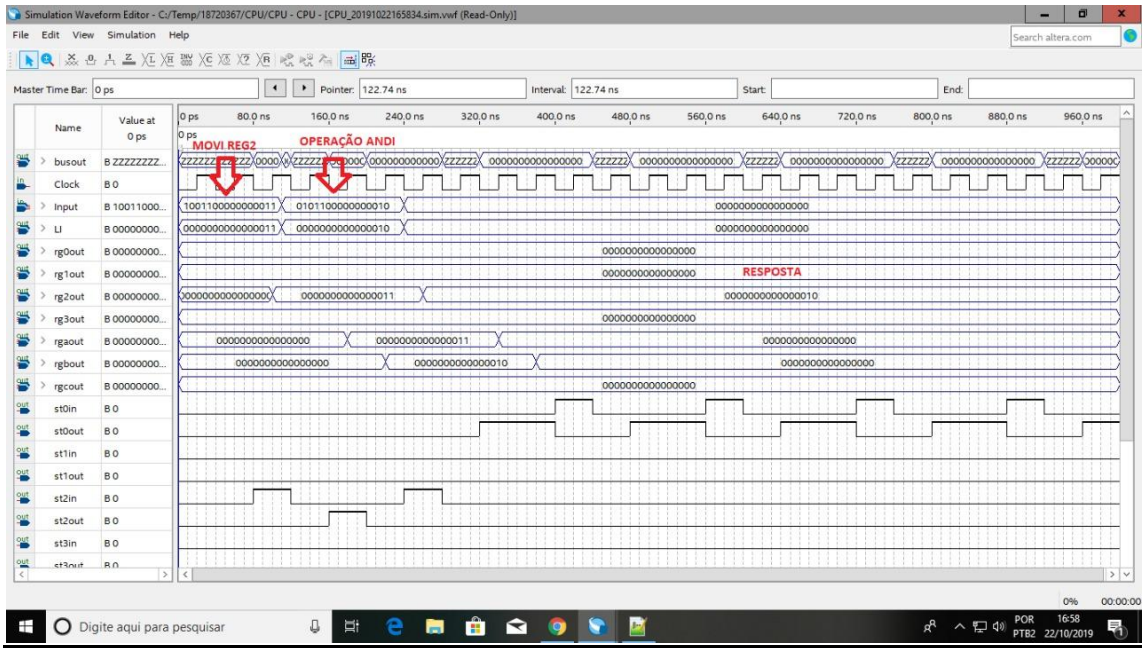
(Figura 4.5 – xchg)



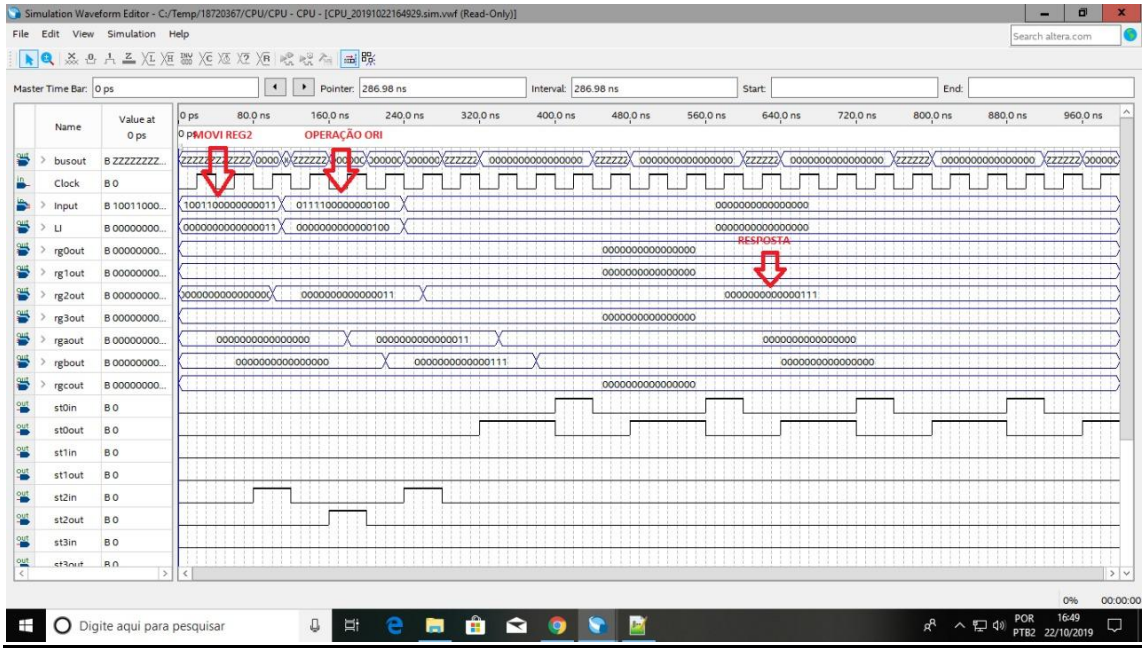
(Figura 4.6 – addi)



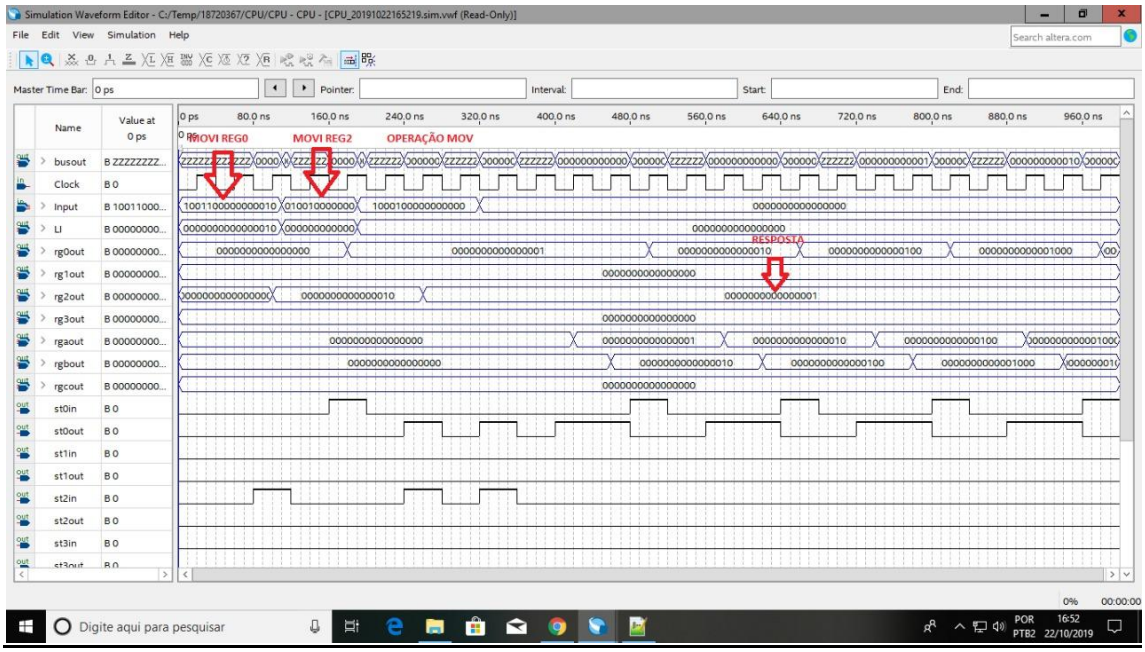
(Figura 4.7 – Subi)



(Figura 4.8 – andi)



(Figura 4.9 – ori)



(Figura 4.10 – mov)

3.2. RESULTADOS E DISCUSSÃO

Os resultados obtidos foram satisfatórios dado que, todos as operações simuladas obtiveram êxito e correspondem às abordagens teóricas vistas em sala de aula.

4. BIBLIOGRAFIA

BROWN, Stephen; VRANESIC, Svonko. Secção 7.14, Design Examples. Fundaments of Digital Logic with VHDL Design. McGraw-Hill Education, 2008.p.438-468.

5. ANEXOS

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;
ENTITY ULA IS
        PORT ( aluop : IN STD_LOGIC_VECTOR(0 TO 2) ; --aluop
                        A : IN STD_LOGIC_VECTOR(0 TO 15) ;--operandos
                        B: IN STD_LOGIC_VECTOR(0 TO 15) ;
                        SAIDA : OUT STD_LOGIC_VECTOR(0 TO 15) ) ; --saida
END ULA ;

ARCHITECTURE Behavior OF ULA IS
BEGIN
        PROCESS ( aluop, A, B )
        BEGIN
                CASE aluop IS
                        WHEN "000"=>
                                SAIDA <=A + B; --SOMA
                        WHEN "001"=>
                                SAIDA <=A - B; --SUBTRAÇÃO
                        WHEN "010"=>
                                SAIDA <=A AND B; --AND
                        WHEN "011"=>
                                SAIDA <=A OR B; --OR
                        WHEN others => SAIDA <= "0000000000000000";

                END CASE ;
        END PROCESS ;
END Behavior ;
```

(Figura 5.1 – ULA)

```
LIBRARY ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;


ENTITY register16bits is
        PORT(input: in std_logic_vector(0 to 15);
                        clock: in STD_LOGIC;
                        reset: in STD_LOGIC;
                        enable: in STD_LOGIC;
                        output: out STD_LOGIC_VECTOR(0 TO 15));

        END register16bits;

        ARCHITECTURE Behavior OF register16bits IS
        BEGIN
                PROCESS(clock,reset)
                BEGIN
                                if (clock'EVENT AND clock = '1' and enable = '1') THEN

                                                output <= input;

                                END if;
                                if (reset = '1') THEN

                                                output <= "0000000000000000";

                                                END if;

                END PROCESS;
        END Behavior;
```

(Figura 5.2 – Registrador)

```
LIBRARY ieee;
Use ieee.std_logic_1164.all;


PACKAGE COMPONENTES is


                component register16bits IS
                port (input: in std_logic_vector(0 to 15);
                        clock: in std_logic;
                        reset: in std_logic;
                        enable: in std_logic;
                        output: out std_logic_vector(0 to 15));
        end component;

        component tristate IS
                port (input: in std_logic_vector(0 to 15);
                        ctrl: in std_logic;
                        output: out std_logic_vector(0 to 15));
        end component;

        component ULA IS
                PORT ( aluop : IN STD_LOGIC_VECTOR(0 TO 2) ; --aluop
                        A, B : IN STD_LOGIC_VECTOR(0 TO 15) ;--operandos
                        SAIDA : OUT STD_LOGIC_VECTOR(0 TO 15) ) ; --saida
        end component;

        component UC IS
                port ( Instrucao: in std_logic_vector(0 to 15);
                                Clock: in std_logic;
            reg0out, reg0in: out std_logic;
            reg1out, reg1in: out std_logic;
            reg2out, reg2in: out std_logic;
            reg3out, reg3in: out std_logic;
            regAUX: out std_logic;
            auxBin, auxBout: out std_logic;
                        auxCin, auxCout: out std_logic;
                        LIin: out std_logic;
                        Aluop: out std_logic_vector(0 to 2));
        end component;

END COMPONENTES;
```

(Figura 5.3 –Componentes)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

ENTITY tristate IS
        PORT (input: in std_logic_vector(0 to 15);
                        ctrl: in std_logic;
                        output: out std_logic_vector(0 to 15));
END tristate;

ARCHITECTURE buf OF tristate IS
BEGIN
                PROCESS(ctrl, input)
                BEGIN
                IF (ctrl = '1') THEN
                        output <= input;
                ELSE
                        output <= "ZZZZZZZZZZZZZZZZ";
                END IF;
        END PROCESS;
END buf;
```

(Figura 5.4 – Tri-state)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;
USE work.COMPONENTES.all;

ENTITY CPU IS
		PORT(
		Clock :IN STD_LOGIC;
		Input : IN STD_LOGIC_VECTOR (0 to 15);
		busout: out std_logic_vector(15 DOWNTO 0);

		rg0out,rg1out,rg2out,rg3out,rgaout,rgbout,rgcout: out std_logic_vector(0 to 15);

		st0in,st0out,st1in,st1out,st2in,st2out,st3in,st3out,stAin,stBin,stBout,stCin,stCout: out std_logic;

		ulashow: out std_logic_vector(0 to 2);
		LI: out std_logic_vector(0 to 15));

	END CPU;


	ARCHITECTURE Behavior OF CPU IS


		SIGNAL BUSS: STD_LOGIC_VECTOR(15 DOWNTO 0);
		SIGNAL ALUOP: STD_LOGIC_VECTOR(0 TO 2);
		SIGNAL Reset: STD_LOGIC;

		SIGNAL R0in,R0out: STD_LOGIC;
		SIGNAL R1in,R1out: STD_LOGIC;
		SIGNAL R2in,R2out: STD_LOGIC;
		SIGNAL R3in,R3out: STD_LOGIC;
		SIGNAL RAUXin: STD_LOGIC;
		SIGNAL RBin,RBout: STD_LOGIC;
		SIGNAL RCin,RCout: STD_LOGIC;


		SIGNAL reg0: STD_LOGIC_VECTOR (0 TO 15);
		SIGNAL reg1: STD_LOGIC_VECTOR (0 TO 15);
		SIGNAL reg2: STD_LOGIC_VECTOR (0 TO 15);
		SIGNAL reg3: STD_LOGIC_VECTOR (0 TO 15);
		SIGNAL RAUXout: STD_LOGIC_VECTOR (0 TO 15);
		SIGNAL ULAout: STD_LOGIC_VECTOR (0 TO 15);
		SIGNAL RBtoTRS: STD_LOGIC_VECTOR (0 TO 15);
		SIGNAL RCtoTRS: STD_LOGIC_VECTOR (0 TO 15);
```

(Figura 5.51 – CPU)

```
		SIGNAL RAUXout: STD_LOGIC_VECTOR (0 TO 15);
		SIGNAL ULAout: STD_LOGIC_VECTOR (0 TO 15);
		SIGNAL RBtoTRS: STD_LOGIC_VECTOR (0 TO 15);
		SIGNAL RCtoTRS: STD_LOGIC_VECTOR (0 TO 15);

		SIGNAL Imediato: STD_LOGIC;
		SIGNAL Imedcont: STD_LOGIC_VECTOR (0 TO 15);

		BEGIN

	rg0out <= reg0;
	rg1out <= reg1;
	rg2out <= reg2;
	rg3out <= reg3;
	rgaout <= RAUXout;
	rgbout <= RBtoTRS;
	rgcout <= RCtoTRS;

	st0out<=R0out;
	st1out <= R1out;
	st2out <= R2out;
	st3out <= R3out;
	st0in <=R0in;
	st1in<=R1in;
	st2in <= R2in;
	st3in <= R3in;

	stAin <= RAUXin;
	stBin <= RBin;
	stBout <= RBout;
	stCin <= RCin;
	stCout <= RCout;

	ulashow <= ALUOP;
	LI <= imedcont;
	busout <= BUSS;

	process (input)
		begin

		if (input(8) = '0') then
			imedcont <= "00000000" & Input(8 to 15);

		else
```

(Figura 5.52 – CPU)

```
                        imedcont <= "11111111" & Input(8 to 15);

                end if;

        end process;

                imed: tristate port map (imedcont,Imediato,BUSS);

                --FLUXO DO REGISTRADOR 0
                regs0:  register16bits port map(BUSS,Clock,Reset,R0in,reg0);
                tri0: tristate port map(reg0,R0out,BUSS);

                --FLUXO DO REGISTRADOR 1
                regs1:  register16bits port map(BUSS,Clock,Reset,R1in,reg1);
                tri1:   tristate port map(reg1,R1out,BUSS);

                --FLUXO DO REGISTRADOR 2
                regs2:  register16bits port map(BUSS,Clock,Reset,R2in,reg2);
                tri2: tristate port map (reg2,R2out,BUSS);

                --FLUXO DO REGISTRADOR 3
                regs3:  register16bits port map (BUSS,Clock,Reset,R3in,reg3);
                tri3:   tristate port map (reg3,R3out,BUSS);

                --Fluxo do Registrador AUX
                regsAUX: register16bits port map(BUSS,Clock,Reset,RAUXin,RAUXout);

                --Fluxo do Registrador AUX 2(B)
                regsAux2: register16bits port map(ULAout,Clock,Reset,RBin,RBtoTRS);
                triB: tristate port map (RBtoTRS,RBout,BUSS);

                --Fluxo do registrador AUX 3(C)
                regsAux3: register16bits port map(BUSS,Clock,Reset,RCin,RCtoTRS);
                triC: tristate port map (RCtoTRS,RCout,BUSS);



                --ULA
                alu: ULA port map(ALUOP,RAUXout,buss,ULAout);

                --Unidade de controle
                controlunit: UC port map(Input,Clock,R0out,R0in,R1out,R1in,R2out,R2in,R3out,R3in,RAUXin,RBin,RBout,RCin,RCout,Imediato,ALUOP);


END Behavior;
```

(Figura 5.53 – CPU)

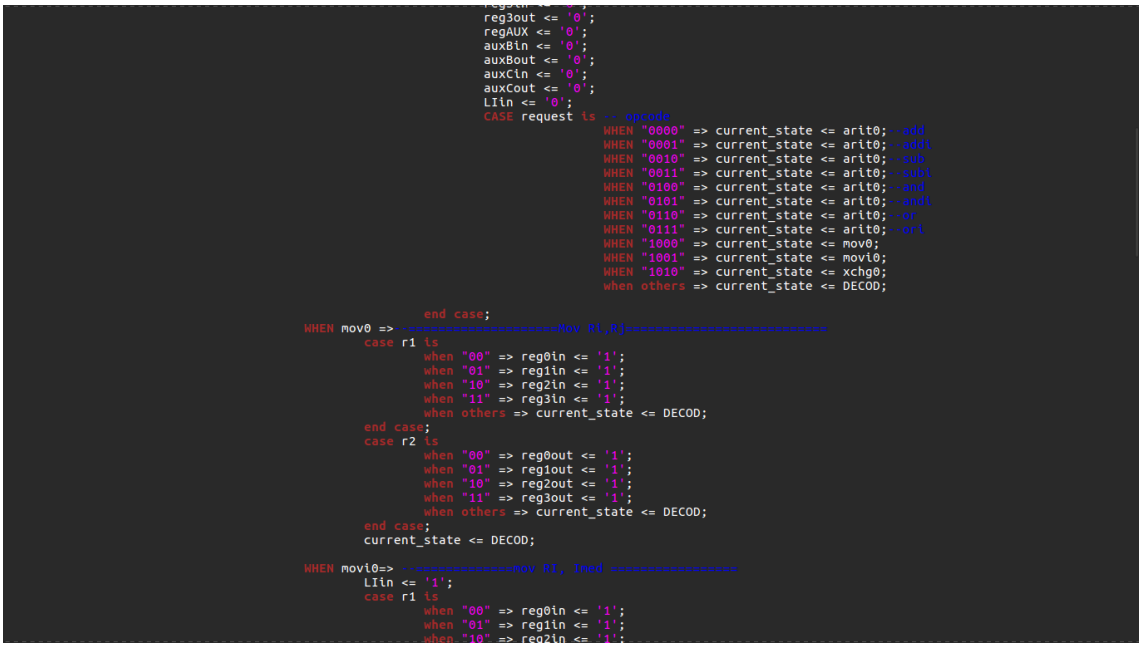
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

ENTITY UC IS
        PORT (  Instrucao: in std_logic_vector(0 to 15);
                                Clock: in std_logic;
            reg0out, reg0in: out std_logic;
            reg1out, reg1in: out std_logic;
            reg2out, reg2in: out std_logic;
            reg3out, reg3in: out std_logic;
            regAUX: out std_logic;
            auxBin, auxBout: out std_logic;
                        auxCin, auxCout: out std_logic;
                        LIin: out std_logic;
                        Aluop: out std_logic_vector(0 to 2));


END UC;

ARCHITECTURE Behavior OF UC IS
        TYPE State_type IS (DECOD, mov0, movi0, xchg0, xchg1, xchg2, arit0, arit1, arit2); -- defindo os Estados
        SIGNAL current_state : State_type := DECOD;
    SIGNAL request: std_logic_vector(0 to 3);
        signal r1, r2: std_logic_vector(0 to 1);
BEGIN
        request <= Instrucao(0 to 3);
        r1 <= Instrucao(4 to 5);
        r2 <= Instrucao(6 to 7);
        PROCESS (Clock)
        BEGIN
                if(Clock'EVENT AND Clock = '0')THEN
            CASE current_state IS
                                                WHEN DECOD =>
                                                                reg0in <= '0';
                                                                reg0out <= '0';
                                                                reg1in <= '0';
                                                                reg1out <= '0';
                                                                reg2in <= '0';
                                                                reg2out <= '0';
                                                                reg3in <= '0';
                                                                reg3out <= '0';
                                                                regAUX <= '0';
                                                                auxBin <= '0';
                                                                auxBout <= '0';
```

(Figura 5.61 –Unidade de Controle)

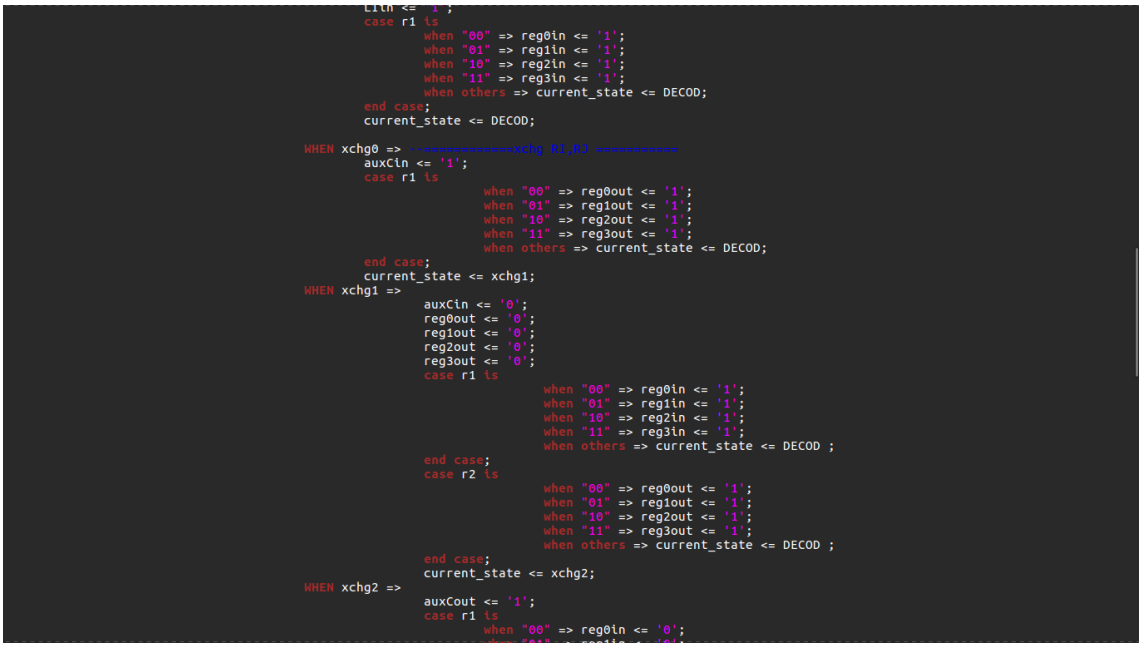


```
                                reg3out <= '0';
                                regAUX <= '0';
                                auxBin <= '0';
                                auxBout <= '0';
                                auxCin <= '0';
                                auxCout <= '0';
                                LIin <= '0';
                                CASE request is -- opcode
                                        WHEN "0000" => current_state <= arit0; --add
                                        WHEN "0001" => current_state <= arit0; --addi
                                        WHEN "0010" => current_state <= arit0; --sub
                                        WHEN "0011" => current_state <= arit0; --subi
                                        WHEN "0100" => current_state <= arit0; --and
                                        WHEN "0101" => current_state <= arit0; --andi
                                        WHEN "0110" => current_state <= arit0; --or
                                        WHEN "0111" => current_state <= arit0; --ori
                                        WHEN "1000" => current_state <= mov0;
                                        WHEN "1001" => current_state <= movi0;
                                        WHEN "1010" => current_state <= xchg0;
                                        when others => current_state <= DECOD;

                        end case;
        WHEN mov0 => --=====================Mov Ri,Rj=========================
                case r1 is
                        when "00" => reg0in <= '1';
                        when "01" => reg1in <= '1';
                        when "10" => reg2in <= '1';
                        when "11" => reg3in <= '1';
                        when others => current_state <= DECOD;
                end case;
                case r2 is
                        when "00" => reg0out <= '1';
                        when "01" => reg1out <= '1';
                        when "10" => reg2out <= '1';
                        when "11" => reg3out <= '1';
                        when others => current_state <= DECOD;
                end case;
                current_state <= DECOD;

        WHEN movi0=> --==============Mov Ri, Imed ================
                LIin <= '1';
                case r1 is
                        when "00" => reg0in <= '1';
                        when "01" => reg1in <= '1';
                        when "10" => reg2in <= '1';
```

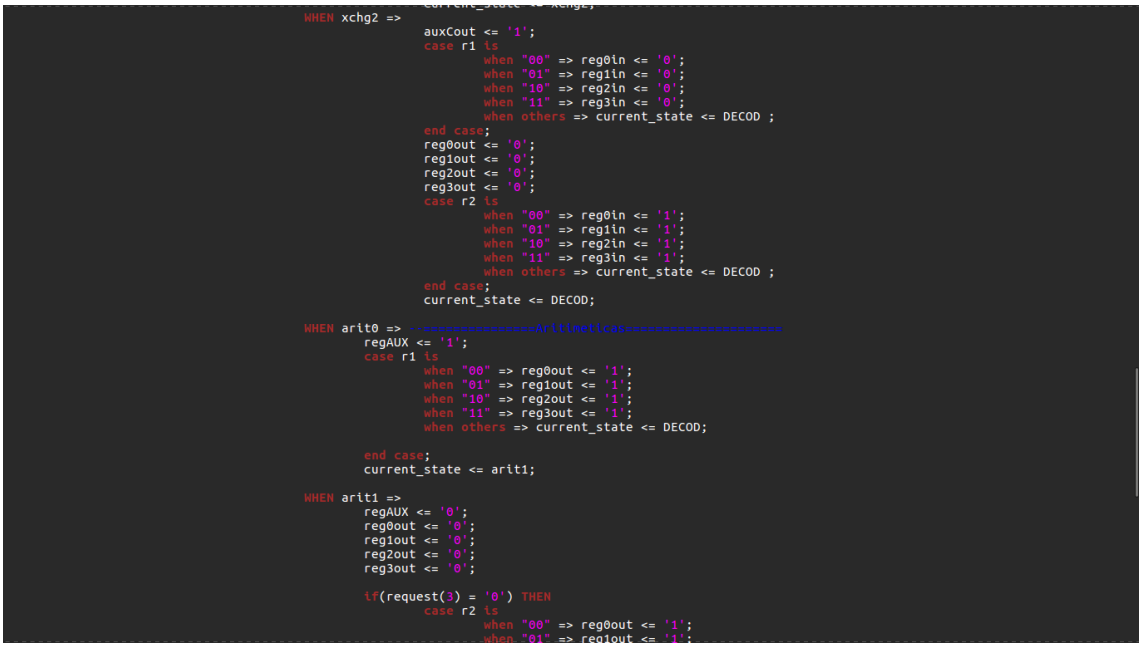
(Figura 5.62 –Unidade de Controle)



```
                LIin <= '1';
                case r1 is
                        when "00" => reg0in <= '1';
                        when "01" => reg1in <= '1';
                        when "10" => reg2in <= '1';
                        when "11" => reg3in <= '1';
                        when others => current_state <= DECOD;
                end case;
                current_state <= DECOD;

        WHEN xchg0 => --============xchg Ri,Rj ==========
                auxCin <= '1';
                case r1 is
                                when "00" => reg0out <= '1';
                                when "01" => reg1out <= '1';
                                when "10" => reg2out <= '1';
                                when "11" => reg3out <= '1';
                                when others => current_state <= DECOD;
                end case;
                current_state <= xchg1;
        WHEN xchg1 =>
                        auxCin <= '0';
                        reg0out <= '0';
                        reg1out <= '0';
                        reg2out <= '0';
                        reg3out <= '0';
                        case r1 is
                                        when "00" => reg0in <= '1';
                                        when "01" => reg1in <= '1';
                                        when "10" => reg2in <= '1';
                                        when "11" => reg3in <= '1';
                                        when others => current_state <= DECOD ;
                        end case;
                        case r2 is
                                        when "00" => reg0out <= '1';
                                        when "01" => reg1out <= '1';
                                        when "10" => reg2out <= '1';
                                        when "11" => reg3out <= '1';
                                        when others => current_state <= DECOD ;
                        end case;
                        current_state <= xchg2;
        WHEN xchg2 =>
                        auxCout <= '1';
                        case r1 is
                                when "00" => reg0in <= '0';
```

(Figura 5.63 –Unidade de Controle)



```
WHEN xchg2 =>
                auxCout <= '1';
                case r1 is
                        when "00" => reg0in <= '0';
                        when "01" => reg1in <= '0';
                        when "10" => reg2in <= '0';
                        when "11" => reg3in <= '0';
                        when others => current_state <= DECOD ;
                end case;
                reg0out <= '0';
                reg1out <= '0';
                reg2out <= '0';
                reg3out <= '0';
                case r2 is
                        when "00" => reg0in <= '1';
                        when "01" => reg1in <= '1';
                        when "10" => reg2in <= '1';
                        when "11" => reg3in <= '1';
                        when others => current_state <= DECOD ;
                end case;
                current_state <= DECOD;

WHEN arit0 => --================Aritmetica====================
        regAUX <= '1';
        case r1 is
                when "00" => reg0out <= '1';
                when "01" => reg1out <= '1';
                when "10" => reg2out <= '1';
                when "11" => reg3out <= '1';
                when others => current_state <= DECOD;

        end case;
        current_state <= arit1;

WHEN arit1 =>
        regAUX <= '0';
        reg0out <= '0';
        reg1out <= '0';
        reg2out <= '0';
        reg3out <= '0';

        if(request(3) = '0') THEN
                case r2 is
                        when "00" => reg0out <= '1';
                        when "01" => reg1out <= '1';
```

(Figura 5.64 –Unidade de Controle)

```
        if(request(3) = '0') THEN
                case r2 is
                        when "00" => reg0out <= '1';
                        when "01" => reg1out <= '1';
                        when "10" => reg2out <= '1';
                        when "11" => reg3out <= '1';
                        when others => current_state <= DECOD;
                end case;
        else

                LIin <= '1';

        end if;
        case request is
                        when "0000" => ALUOp <= "000";
                        when "0001" => ALUOp <= "000";
                        when "0010" => ALUOp <= "001";
                        when "0011" => ALUOp <= "001";
                        when "0100" => ALUOp <= "010";
                        when "0101" => ALUOp <= "010";
                        when "0110" => ALUOp <= "011";
                        when "0111" => ALUOp <= "011";
                        when others => ALUOp <= "ZZZ";
                --ULA <= "0000"
        end case;
        auxBin <= '1';
        current_state <= arit2;

WHEN arit2 =>
        reg0out <= '0';
        reg1out <= '0';
        reg2out <= '0';
        reg3out <= '0';
        auxBin <= '0';
        LIin <= '0';
        case r1 is
                        when "00" => reg0in <= '1';
                        when "01" => reg1in <= '1';
                        when "10" => reg2in <= '1';
                        when "11" => reg3in <= '1';
                        when others => current_state <= DECOD;
        end case;
        auxBout <= '1';
        current_state <= DECOD;
```

(Figura 5.65 –Unidade de Controle)

```
                        WHEN arit2 =>
                                reg0out <= '0';
                                reg1out <= '0';
                                reg2out <= '0';
                                reg3out <= '0';
                                auxBin <= '0';
                                LIin <= '0';
                                case r1 is
                                                when "00" => reg0in <= '1';
                                                when "01" => reg1in <= '1';
                                                when "10" => reg2in <= '1';
                                                when "11" => reg3in <= '1';
                                                when others => current_state <= DECOD;
                                end case;
                                auxBout <= '1';
                                current_state <= DECOD;

                                WHEN others =>
                                        current_state <= DECOD;

                END case;
        END IF;
END PROCESS;
END Behavior;
```

(Figura 5.67 –Unidade de Controle)