Pontifícia Universidade Católica de Campinas – PUC CAMPINAS

Projeto 2 – CPU

Cesar Marrote Manzano	RA: 18051755
Fabrício Silva Cardoso	RA: 18023481
Matheus Henrique Moretti	RA:18082974
Pedro Ignacio Trevisan	RA:18016568

ÍNDICE

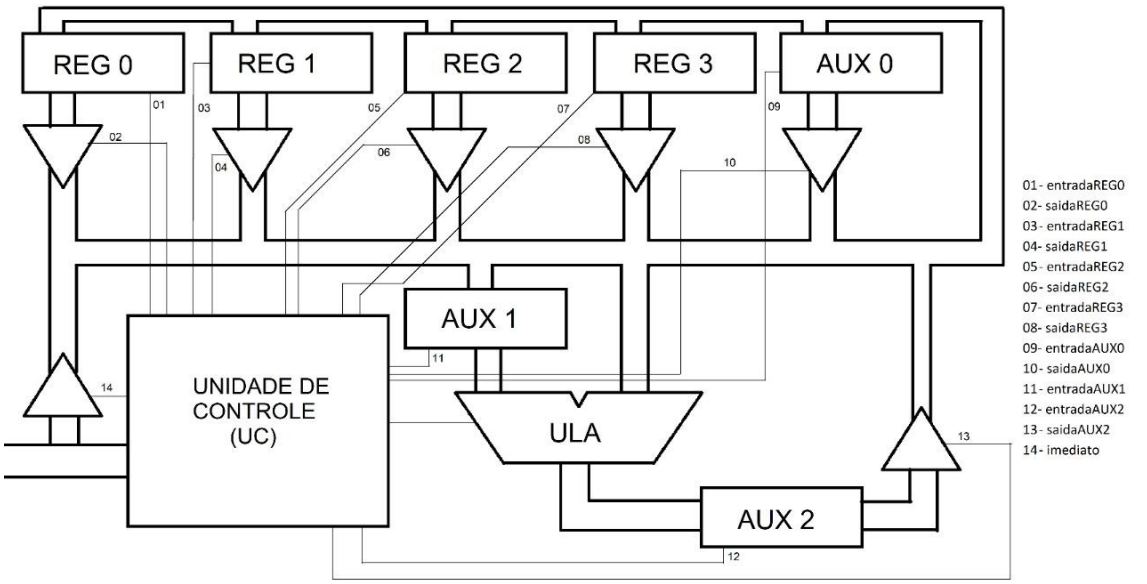
1. DESCRIÇÃO TEXTUAL DO PROJETO COM A TOPOLOGIA DA CPU

O segundo projeto teve como objetivo o desenvolvimento de uma CPU simples, que executa as instruções abaixo:

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

Inicialmente, a CPU estará esperando um código válido para que se inicie as operações. Caso exista a necessidade de se fazer uma operação com um imediato, temos um sinal que controla a entrada do dado para o barramento. O barramento (de 8 bits), possui dados circulando para que seja possível realizar as operações necessárias. Os sinais de controle necessitam estar sincronizados com as operações, para não aconteça a perca de nenhum dado, pois usam sempre o mesmo barramento.

Topologia da CPU



2. ESPECIFICAÇÃO

2.1. REGISTRADORES (QUANTIDADE, ENDEREÇO E TAMANHO)

Utilizamos quatro registradores principais e três auxiliares. O registrador principal tem como função armazenar os valores para que seja feita as operações. Um dos auxiliares é usado para a instrução XCHG e os outros dois são usados da seguinte forma: um para armazenar o valor que entrará na ULA e outro que conterá o valor da operação da ULA. Todos os 7 registradores possuem tamanho de 8 bits.

2.2 FORMATO DAS INSTRUÇÕES (OPCODE)

INSTRUÇÃO	OPCODE	FORMATO
MOV	0000	-
MOVI	0001	-
XCHG	0010	-
ADDI	0011	TIPO I
SUBI	0100	TIPO I
ANDI	0101	TIPO I
ORI	0110	TIPO I
ADD	0111	TIPO R
SUB	1000	TIPO R
AND	1001	TIPO R
OR	1010	TIPO R

Para podermos testar o programa, adotamos as instruções com 16 bits. Os bits das instruções estão organizados da seguinte forma:

- De 15-12 representam as instruções que serão realizadas, ou seja, irá conter os opcodes;

- De 11-10 representam o registrador destino;

- De 9-8 representam algum registrador temporário;

- De 7-0 representam os imediatos.

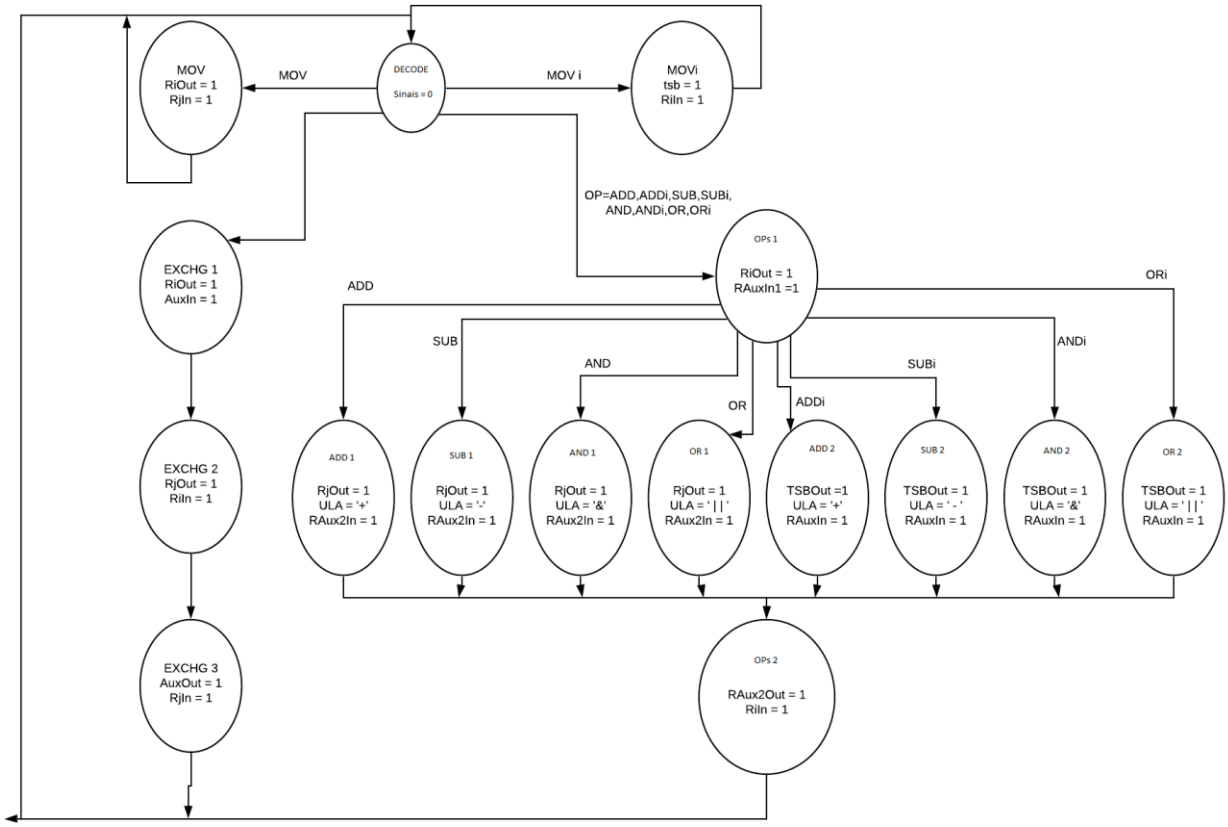
2.3. UNIDADE DE CONTROLE: DIAGRAMA E TABELA DE ESTADOS, SINAIS E SEUS SIGNIFICADOS

A tabela abaixo mostra as operações realizadas pela unidade de controle (UC). Cada operação possui o seu respectivo código e quando há códigos inválidos, a UC fica no estado de waiting, ou seja, esperando um código válido para realizar a instrução.

ESTADO	CÓDIGO
MOV	0000

MOVI	0001
XCHG	0010
OPERAÇÕES ARITMETICAS	0011 até 1010
WAITING	Códigos Inválidos

O diagrama de estados abaixo representa como será projetada a UC:



3. RESULTADOS

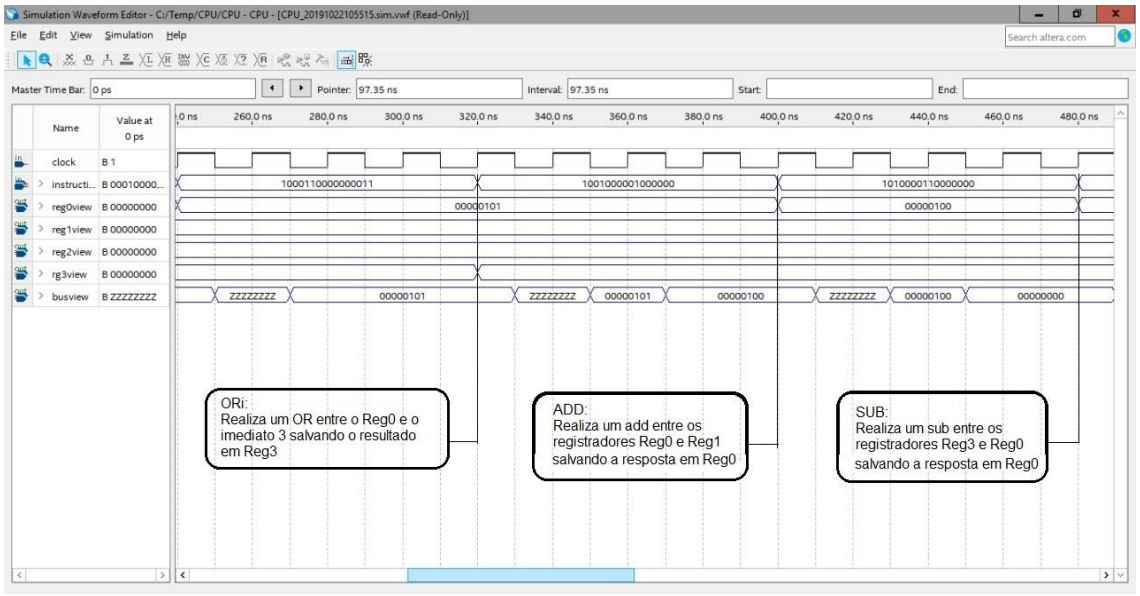
3.1. DESCRIÇÃO DOS TESTES REALIZADOS

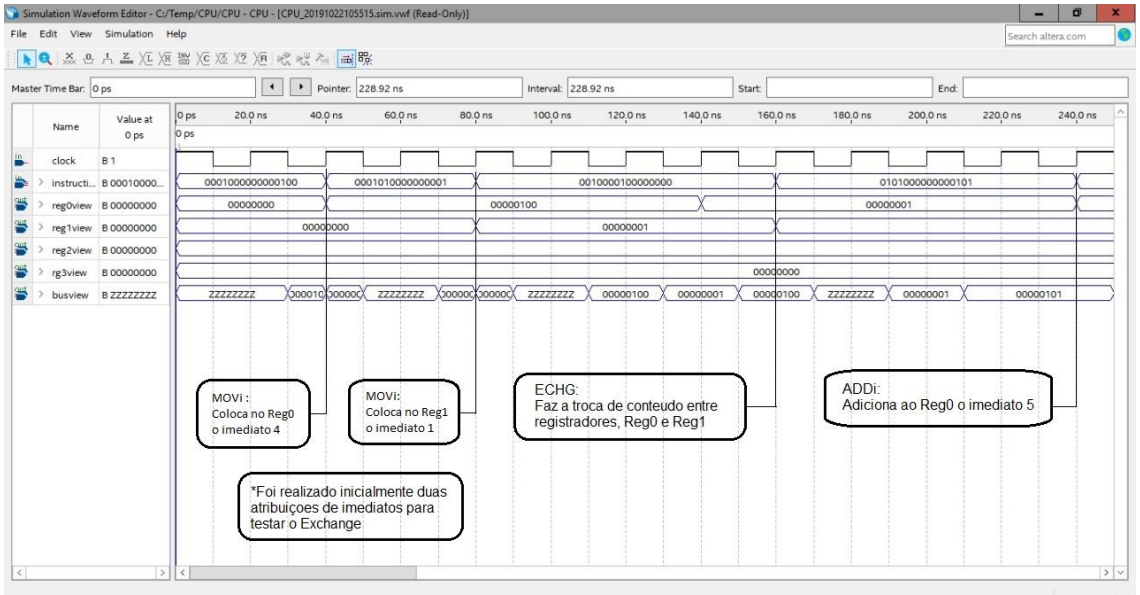
Os testes foram realizados na ordem dos opcodes.

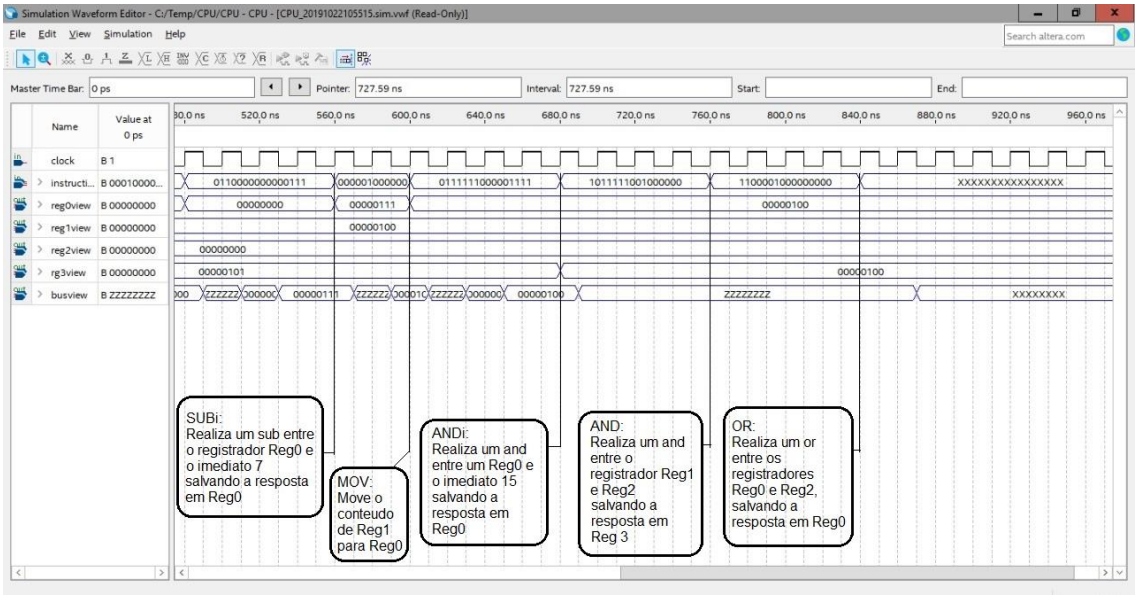
INSTRUÇÃO	OPERAÇÃO
0001000000000100	**MOVI:** REG0 <− 4
0001000000000001	**MOVI:** REG1 <− 1
0010000010000000	**XCHG:** REG0 <−> REG1
0101000000000101	**ADDI:** REG0 <− REG0 ADDI 5
1000110000000011	**ORI:** REG3 <− REG0 ORI 3
1001000001000000	**ADD:** REG0 <− REG0 ADD REG1
1010000011000000	**SUB:** REG0 <− REG3 SUB REG0
0110000000000111	**SUBI:** REG0 <− REG0 SUBI 7
0000000100000000	**MOV:** REG0 <− REG1
0111111000001111	**ANDI:** REG0 <− REG0 ANDI 15

1011111001000000	**AND:** REG3 <− REG2 AND REG1
1100001000000000	**OR:** REG0 <− REG2 OR REG0

As imagens abaixo representam os testes realizados







3.2. RESULTADOS E DISCUSSÃO

Todos os testes realizados estão de acordo com o esperado, portanto concluímos que o projeto teve resultado satisfatório.

4. BIBLIOGRAFIA E ANEXO

- Brown S., Vranesic S. “Fundamentals of digital Logic with VHDL Design”

4.1. ANEXO

Código produzido:

ULA.vhd

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

entity ULA is
	port (registrador1: in std_logic_vector(0 to 7);
			registrador2: in std_logic_vector(0 to 7);
			operacao:     in std_logic_vector(0 to 1);
			saidaULA: out std_logic_vector(0 to 7));
end ULA;

architecture ALU of ULA is
begin
	process(registrador1, registrador2, operacao)
	begin
		case operacao is
			when "00" => saidaULA <= registrador1 + registrador2;
			when "01" => saidaULA <= registrador1 - registrador2;
			when "10" => saidaULA <= registrador1 and registrador2;
			when "11" => saidaULA <= registrador1 or registrador2;
			when others => saidaULA <= "00000000";
		end case;
	end process;
end ALU;
```

UnidadeControle.vhd

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

entity UnidadeControle is
	port (instrucao: in std_logic_vector(0 to 15);
		clock: in std_logic;
		entradaReg0, saidaReg0: out std_logic;
		entradaReg1, saidaReg1: out std_logic;
		entradaReg2, saidaReg2: out std_logic;
		entradaReg3, saidaReg3: out std_logic;
		aux1: out std_logic;
		entradaAux0, saidaAux0: out std_logic;
		entradaAux2, saidaAux2: out std_logic;
		imediato: out std_logic;
		ALUOp:  out std_logic_vector(0 to 1));
end UnidadeControle;

architecture controle of UnidadeControle is
	type estadosUC is (esperandoOperacao, mov, movi, xchg, xchg1, xchg2,
op0, op1, op2);
	signal estado: estadosUC := esperandoOperacao;
	signal opcode: std_logic_vector(0 to 3);
	signal r1, r2, r3: std_logic_vector(0 to 1);

begin
	opcode <= instrucao(0 to 3);
	r1 <= instrucao(4 to 5);
	r2 <= instrucao(6 to 7);
```

```
r3 <= instrucao(8 to 9);
process(clock)
begin
if (clock'event and clock = '0') then
        case estado is
                when esperandoOperacao =>
                entradaReg0 <= '0';
                saidaReg0 <= '0';
                entradaReg1 <= '0';
                saidaReg1 <= '0';
                entradaReg2 <= '0';
                saidaReg2 <= '0';
                entradaReg3 <= '0';
                saidaReg3 <= '0';
                aux1 <= '0';
                entradaAux2 <= '0';
                saidaAux2 <= '0';
                entradaAux0 <= '0';
                saidaAux0 <= '0';
                imediato <= '0';
                case opcode is
                when "0000" => estado <= mov;
                when "0001" => estado <= movi;
                when "0010" => estado <= xchg;
                when "0011" => estado <= op0;
                when "0100" => estado <= op0;
                when "0101" => estado <= op0;
                when "0110" => estado <= op0;
                when "0111" => estado <= op0;
                when "1000" => estado <= op0;
```

```
            when "1001" => estado <= op0;
            when "1010" => estado <= op0;
            when others => estado <= esperandoOperacao;
            end case;
        when mov =>
            case r2 is
            when "00" => saidaReg0 <= '1';
            when "01" => saidaReg1 <= '1';
            when "10" => saidaReg2 <= '1';
            when "11" => saidaReg3 <= '1';
            when others => estado <= esperandoOperacao;
            end case;
            case r1 is
            when "00" => entradaReg0 <= '1';
            when "01" => entradaReg1 <= '1';
            when "10" => entradaReg2 <= '1';
            when "11" => entradaReg3 <= '1';

            when others => estado <= esperandoOperacao;

            end case;
            estado <= esperandoOperacao;
        when movi =>
            imediato <= '1';
            case r1 is
            when "00" => entradaReg0 <= '1';
            when "01" => entradaReg1 <= '1';
            when "10" => entradaReg2 <= '1';
            when "11" => entradaReg3 <= '1';
            when others => estado <= esperandoOperacao;
            end case;
```

```
            estado <= esperandoOperacao;
        when xchg => -
            entradaAux0 <= '1';
            case r1 is
            when "00" => saidaReg0 <= '1';
            when "01" => saidaReg1 <= '1';
            when "10" => saidaReg2 <= '1';
            when "11" => saidaReg3 <= '1';
            when others => estado <= esperandoOperacao;

            end case;
            estado <= xchg1;
        when xchg1 =>
            entradaAux0 <= '0';
            saidaReg0 <= '0';
            saidaReg1 <= '0';
            saidaReg2 <= '0';
            saidaReg3 <= '0';
            case r1 is
            when "00" => entradaReg0 <= '1';
            when "01" => entradaReg1 <= '1';
            when "10" => entradaReg2 <= '1';
            when "11" => entradaReg3 <= '1';

            when others => estado <= esperandoOperacao;

            end case;
            case r2 is
            when "00" => saidaReg0 <= '1';
            when "01" => saidaReg1 <= '1';
            when "10" => saidaReg2 <= '1';
            when "11" => saidaReg3 <= '1';
```

```
            when others => estado <= esperandoOperacao;
            end case;
            estado <= xchg2;
        when xchg2 =>
            saidaAux0 <= '1';
            case r1 is
            when "00" => entradaReg0 <= '0';
            when "01" => entradaReg1 <= '0';
            when "10" => entradaReg2 <= '0';
            when "11" => entradaReg3 <= '0';
            when others => estado <= esperandoOperacao;
            end case;
            saidaReg0 <= '0';
            saidaReg1 <= '0';
            saidaReg2 <= '0';
            saidaReg3 <= '0';
            case r2 is
            when "00" => entradaReg0 <= '1';
            when "01" => entradaReg1 <= '1';
            when "10" => entradaReg2 <= '1';
            when "11" => entradaReg3 <= '1';
            when others => estado <= esperandoOperacao;
            end case;
            estado <= esperandoOperacao;
        when op0 =>
            aux1 <= '1';
            case r2 is
            when "00" => saidaReg0 <= '1';
            when "01" => saidaReg1 <= '1';
            when "10" => saidaReg2 <= '1';
```

```
            when "11" => saidaReg3 <= '1';
            when others => estado <= esperandoOperacao;
            end case;
            estado <= op1;
        when op1 =>
            aux1 <= '0';
            saidaReg0 <= '0';
            saidaReg1 <= '0';
            saidaReg2 <= '0';
            saidaReg3 <= '0';
            if (opcode > "0110") then
                case r3 is
                when "00" => saidaReg0 <= '1';
                when "01" => saidaReg1 <= '1';
                when "10" => saidaReg2 <= '1';
                when "11" => saidaReg3 <= '1';
                when others => estado <= esperandoOperacao;
                end case;
            else
                imediato <= '1';
            end if;
            case opcode is
            when "0011" => ALUOp <= "00";
            when "0111" => ALUOp <= "00";
            when "0100" => ALUOp <= "01";
            when "1000" => ALUOp <= "01";
            when "0101" => ALUOp <= "10";
            when "1001" => ALUOp <= "10";
            when "0110" => ALUOp <= "11";
            when "1010" => ALUOp <= "11";
```

```
                when others => ALUOp <= "ZZ";
                end case;
                entradaAux2 <= '1';
                estado <= op2;
            when op2 =>
                saidaReg0 <= '0';
                saidaReg1 <= '0';
                saidaReg2 <= '0';
                saidaReg3 <= '0';
                entradaAux2 <= '0';
                imediato <= '0';
                case r1 is
                when "00" => entradaReg0 <= '1';
                when "01" => entradaReg1 <= '1';
                when "10" => entradaReg2 <= '1';
                when "11" => entradaReg3 <= '1';
                when others => estado <= esperandoOperacao;
                end case;
                saidaAux2 <= '1';
                estado <= esperandoOperacao;
            when others =>
                estado <= esperandoOperacao;
            end case;
        end if;
end process;
end controle;
```

triStateRegistrador.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity triStateRegistrador is
	port (entradaTR: in std_logic_vector(0 to 7);
			clock: in std_logic;
			enablein: in std_logic;
			enableout: in std_logic;
			reset: in std_logic;
			saidaTR: out std_logic_vector(0 to 7));
end triStateRegistrador;
architecture aux of triStateRegistrador is
	component registrador_8
		port (entradaReg: in std_logic_vector(0 to 7);
			clock: in std_logic;
			enableReg: in std_logic;
			resetReg: in std_logic;
			saidaReg: out std_logic_vector(0 to 7));
	end component;
	component triStateBuffer
		port (entradaTriState: in std_logic_vector(0 to 7);
			controleTriState: in std_logic;
			saidaTriState: out std_logic_vector(0 to 7));
	end component;
	signal med: std_logic_vector(0 to 7);
begin
	reg: registrador_8 port map (entradaTR, clock, enablein, reset, med);
	tri: triStateBuffer port map (med, enableout, saidaTR);
end aux;
```

triStateBuffer.vhd

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

entity triStateBuffer is
	port (entradaTriState: in std_logic_vector(0 to 7);
			controleTriState: in std_logic;
			saidaTriState: out std_logic_vector(0 to 7));
end triStateBuffer;


architecture TSB of triStateBuffer is
begin
	process(controleTriState, entradaTriState)
	begin
		if (controleTriState = '1') then
			saidaTriState <= entradaTriState;
		else
			saidaTriState <= "ZZZZZZZZ";
		end if;
	end process;
end TSB;
```

registrador_8.vhd

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

entity registrador_8 is
	port (entradaReg: in std_logic_vector(0 to 7);
			clock: in std_logic;
			enableReg: in std_logic;
			resetReg: in std_logic;
			saidaReg: out std_logic_vector(0 to 7));
end registrador_8;

architecture registrador8B of registrador_8 is
begin
	process(clock, resetReg)
	begin
		if (clock'event and clock = '1' and enableReg = '1') then
			saidaReg <= entradaReg;
		end if;
		if (resetReg = '1') then
			saidaReg <= "00000000";
		end if;
	end process;
end registrador8B;
```

cpu.vhd

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

entity cpu is
        port(instruction: in std_logic_vector(0 to 15);
         clock: in std_logic;
         busview: out std_logic_vector(0 to 7);
         reg0view: out std_logic_vector(0 to 7);
         reg1view: out std_logic_vector(0 to 7);
         reg2view: out std_logic_vector(0 to 7);
         rg3view: out std_logic_vector(0 to 7));
end cpu;

architecture behavior of cpu is
        signal mbus: std_logic_vector(0 to 7);
        signal reg0in, reg0out: std_logic;
        signal reg1in, reg1out: std_logic;
        signal reg2in, reg2out: std_logic;
        signal reg3in, reg3out: std_logic;
        signal regAin: std_logic;
        signal regGin, regGout: std_logic;
        signal regTin, regTout: std_logic;
        signal imedIn: std_logic;
        signal immedi: std_logic_vector(0 to 7);
        signal r0insid: std_logic_vector(0 to 7);
        signal r1insid: std_logic_vector(0 to 7);
        signal r2insid: std_logic_vector(0 to 7);
        signal r3insid: std_logic_vector(0 to 7);
```

```
signal regAtoULA: std_logic_vector(0 to 7);
signal ULAtoRegG: std_logic_vector(0 to 7);
signal ALUOp: std_logic_vector(0 to 1);
signal master_reset: std_logic;
component registrador_8
        port (entradaReg: in std_logic_vector(0 to 7);
                clock: in std_logic;
                enableReg: in std_logic;
                resetReg: in std_logic;
                saidaReg: out std_logic_vector(0 to 7));
end component;
component triStateBuffer
        port (entradaTriState: in std_logic_vector(0 to 7);
                controleTriState: in std_logic;
                saidaTriState: out std_logic_vector(0 to 7));
end component;
component triStateRegistrador
        port (entradaTR: in std_logic_vector(0 to 7);
                clock: in std_logic;
                enablein: in std_logic;
                enableout: in std_logic;
                reset: in std_logic;
                saidaTR
                : out std_logic_vector(0 to 7));
end component;
component ULA
        port (registrador1: in std_logic_vector(0 to 7);
                registrador2: in std_logic_vector(0 to 7);
                operacao:     in std_logic_vector(0 to 1);
                saidaULA: out std_logic_vector(0 to 7));
```

```
        end component;
        component UnidadeControle
                port (instrucao: in std_logic_vector(0 to 15);
                        clock: in std_logic;
                        entradaReg0, saidaReg0: out std_logic;
                        entradaReg1, saidaReg1: out std_logic;
                        entradaReg2, saidaReg2: out std_logic;
                        entradaReg3, saidaReg3: out std_logic;
                        aux1: out std_logic;
                        entradaAux0, saidaAux0: out std_logic;
                        entradaAux2, saidaAux2: out std_logic;
                        imediato: out std_logic;
                        ALUOp:  out std_logic_vector(0 to 1));
        end component;
begin
        busview <= mbus;
        reg0view <= r0insid;
        reg1view <= r1insid;
        reg2view <= r2insid;
        rg3view <= r3insid;
        process (instruction)
        begin
                if (instruction(8) = '0') then
                        immedi <=  instruction(8 to 15);
                else
                        immedi <=  instruction(8 to 15);
                end if;
        end process;
        imed: triStateBuffer port map (immedi, imedIn, mbus);
        reg0: registrador_8 port map (mbus, clock, reg0in, master_reset, r0insid);
```

tri0: triStateBuffer port map (r0insid, reg0out, mbus);

reg1: registrador_8 port map (mbus, clock, reg1in, master_reset, r1insid);

tri1: triStateBuffer port map (r1insid, reg1out, mbus);

reg2: registrador_8 port map (mbus, clock, reg2in, master_reset, r2insid);

tri2: triStateBuffer port map (r2insid, reg2out, mbus);

reg3: registrador_8 port map (mbus, clock, reg3in, master_reset, r3insid);

tri3: triStateBuffer port map (r3insid, reg3out, mbus);

regA: registrador_8 port map (mbus, clock, regAin, master_reset, regAtoULA);

regG: triStateRegistrador port map (ULAtoRegG, clock, regGin, regGout, master_reset, mbus);

regT: triStateRegistrador port map (mbus, clock, regTin, regTout, master_reset, mbus);

alu: ULA port map (regAtoULA, mbus, ALUOp, ULAtoRegG);

unit: UnidadeControle port map (instruction, clock, reg0in, reg0out, reg1in, reg1out, reg2in, reg2out, reg3in, reg3out, regAin, regGin, regGout, regTin, regTout, imedIn, ALUOp);

end behavior;