PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS - PUC CAMPINAS
CEATEC - CENTRO DE CIÊNCIAS EXATAS, AMBIENTAIS E DE TECNOLOGIA

Projeto 2 - CPU

Bruno Guilherme Spirlandeli Marini - RA 17037607
Caio Lima e Sousa Della Torre Sanches - RA 17225285
Gabriela Nelsina Vicente - RA 17757089
Jefferson Meneses da Silva - RA 17230400
Marcos Aurélio Tavares de Sousa Filho - RA 17042284

Campinas-SP
2018

Índice

Descrição e topologia da CPU

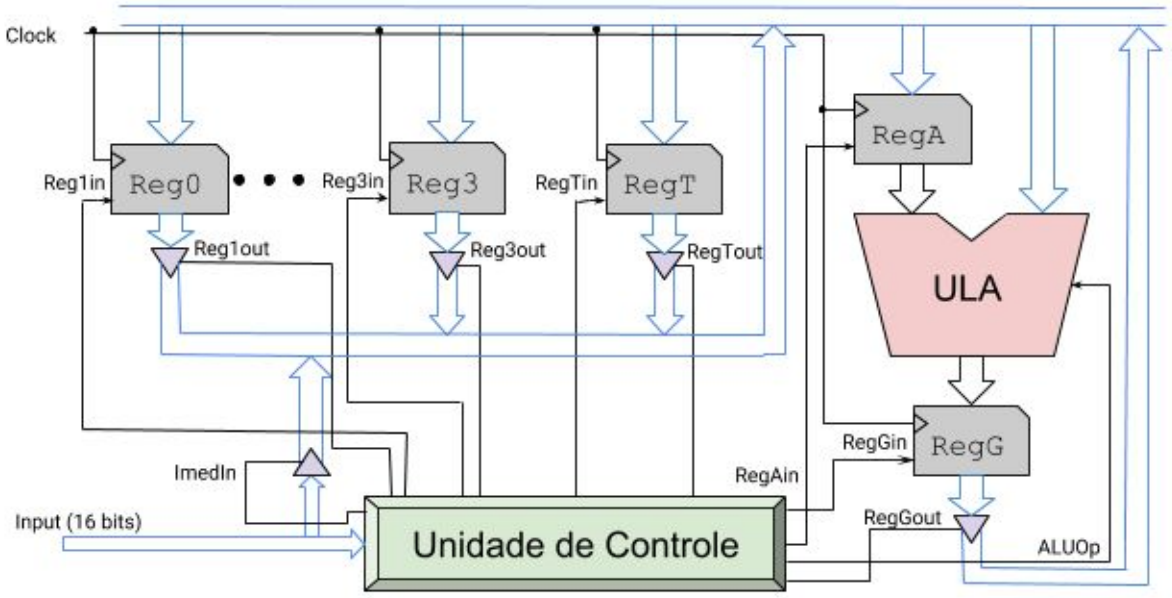
Como segundo projeto da disciplina de Arquitetura de Computadores (Prática), foi proposto o desenvolvimento de uma CPU capaz de executar as instruções contidas na tabela seguinte:

Instrução	Significado	Descrição
`MOV Ri,Rj`	`Ri <- Rj`	`Move`
`MOVI Ri,Imed`	`Ri <- Imed`	`Move Immediate`
`XCHG Ri,Rj`	`Ri <- Rj e Rj <- Ri`	`Exchange`
`ADD Ri,Rj,Rk`	`Ri <- Rj + Rk`	`Add`
`ADDI Ri,Rj,Imed`	`Ri <- Rj + Imed`	`Add Immediate`
`SUB Ri,Rj,Rk`	`Ri <- Rj - Rk`	`Subtract`
`SUBI Ri,Rj,Imed`	`Ri <- Rj - Imed`	`Subtract Immediate`
`AND Ri,Rj,Rk`	`Ri <- Rj & Rk`	`And`
`ANDI Ri,Rj,Imed`	`Ri <- Rj & Imed`	`And Immediate`
`OR Ri,Rj,Rk`	`Ri <- Rj | Rk`	`Or`
`ORI Ri,Rj,Imed`	`Ri <- Rj | Imed`	`Or Immediate`
`SHL Ri,Rj,Imed`	`Ri <- Rj << Imed`	`Shift Left`
`SHR Ri,Rj,Imed`	`Ri <- Rj >> Imed`	`Shift Right`

Para isso tomamos como base o projeto apresentado no livro "Fundamentals of Digital Logic with VHDL Design", Seção 7.14[1] que demonstra a criação de uma CPU multi ciclo simples. Com ela -e com seu ISA(Instruction Set Architecture)- foi possível criar, desenvolver e simular uma CPU experimental.

Inicialmente a CPU aguarda a entrada de uma instrução, que ao ser recebida será computada pela unidade de controle. Caso exista a necessidade de envolver um imediato, o sinal `ImedIn` se encarregará de fluir o dado para o bus interno. Neste bus, os dados estarão circulando a todo momento, paralelamente (bus com 16 bits), por isso é necessário que os sinais da unidade de controle sejam sincronizados a fim de que nenhuma informação seja perdida e consequentemente não danifique o canal. Segue abaixo, a topologia projetada para este trabalho.

Topologia da CPU



Detalhes de projeto

Registradores

Utilizamos então 4 registradores principais (sendo numerados e endereçáveis por 00b (0d) até 11b (3d)), complementando com 3 registradores auxiliares, sendo eles, A e G, utilizados respectivamente com o intuito de armazenar um valor temporário para a ULA e para armazenar o valor obtido da ULA e o último sendo utilizado como temporário para a instrução `XCHG`, não havendo a possibilidade de referenciá-lo na instrução. Por fim, cada registrador possui capacidade máxima de 16 bits.

Formato das instruções

Quanto aos opcodes, adotamos os 4 MSBs (bits 15-12) de cada instrução para referenciar ao tipo de execução necessária, uma vez que tínhamos 13 instruções básicas era preciso haver, no mínimo, 4 bits. A seguir, os bits 11-10 representam o registrador de destino. Para os instruções que necessitam de registradores de origem foram utilizados os bits 9-8, e caso ainda seja requerido algum registrador temporário, os bits 7-6 serão responsáveis por este papel. Os últimos bits (5-0), serão preenchidos com ‘0’, uma vez que estes não terão finalidades para os casos de instruções citados acima. Porém caso haja a necessidade de utilizar imediatos (e portanto, registradores temporários [bits 7-6] não serão envolvidos), os bits 7-0, serão encarregados com a tarefa de receber um

número de 8 bits sinalizado. Também é importante notar que a instrução XCHG utiliza de um registrador temporário T(citado no tópico de registradores) que não é mencionado ao escrever a instrução e não é alcançável por nenhuma outra instrução, já que este não pode ser endereçável. Para estipulação dos opcodes, foi planejado deixar instruções lógico-aritméticas no fim, para facilitar a diferenciação dos tipos R e I. A Tabela 1 trata e relaciona cada opcode com sua respectiva funcionalidade.

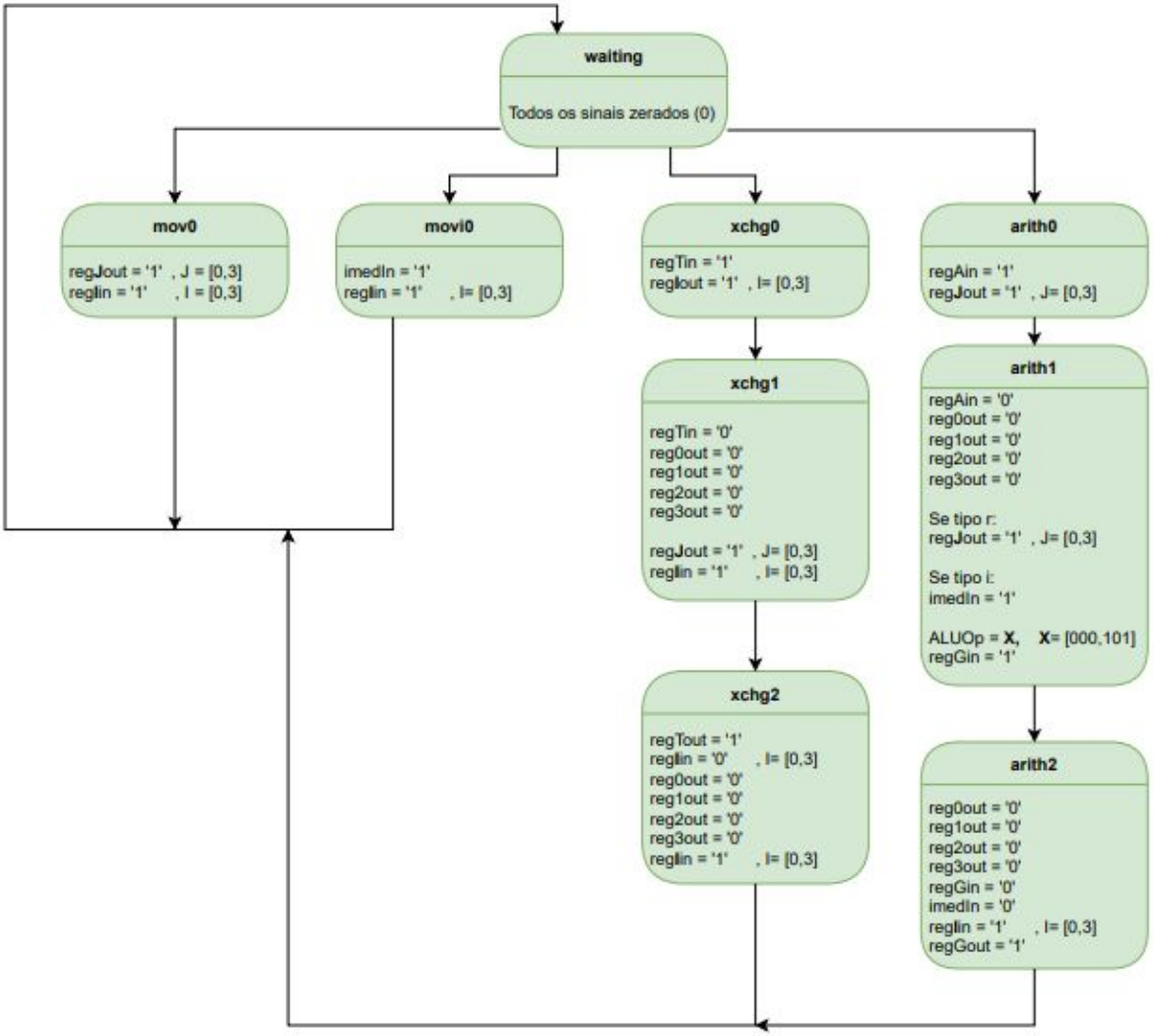
Tabela 1

Instrução	Opcode	Tipos
MOV	0000	Tipo R/I*
MOVI	0001	Tipo I
XCHG	0010	Tipo R/I*
SHL	0011	Tipo R
SHR	0100	
ADDI	0101	Tipo I
SUBI	0110	
ANDI	0111	
ORI	1000	
ADD	1001	Tipo R
SUB	1010	
AND	1011	
OR	1100	

* Estas instruções não cabem diretamente em um tipo específico por conta de ocuparem, além dos bits de opcode, apenas os bits de registrador de destino e registrador de origem, não havendo necessidade de utilizar os 8 bits restantes(como no caso de uma instrução imediata) e do campo do registrador temporário, uma vez que estas instruções não requerem uma especificação para o mesmo.

Unidade de controle

O diagrama de estados abaixo representa a FSM(*Finite State Machine*) que a unidade de controle segue:



A Tabela 2 engloba todos os sinais utilizados pela UC.

Tabela 2

Sinais (bits)	Definição
• `clock (1)`	Utilizado para coordenar as ações e armazenar valores em registradores
• `reg0in (1), reg0out (1);` • `reg1in (1), reg1out (1);` • `reg2in (1), reg2out (1);` • `reg3in (1), reg3out (1);` • `regAin (1);` • `regGin (1), regGout (1);` • `regTin (1), regTout (1);`	Sinais com o sufixo `in`, indicam que armazenará o valor do bus em um registrador específico (*e.g.* `reg0in` armazenará em 0). O mesmo funcionará com o sufixo `out`, indicando que liberará o buffer tri-state de um registrador, para escritura no bus.
• `ImedIn (1);`	Utilizado para habilitar o buffer tri-state que controla a entrada de dados imediatos
• `ALUOp (3);`	Sinal para informar a ULA qual operação deve ser feita, seguindo a seguinte relação: • `000 - Add;` • `001 - Sub;` • `010 - AND;` • `011 - OR;` • `100 - SLL;` • `101 - SRL`

Resultados

Testes Realizados

Foi realizado os seguintes testes junto ao simulador

Sequência	Instrução	Descrição
1	0001000000000110	MOVI REG0 <- IMED(6)
2	0001010000000101	MOVI REG1 <- IMED(5)
3	0010000100000000	XCHG REG0 <-> REG1
4	0101000000001000	REG0 <- REG0 ADDI IMED(8)
5	1000110000000000	REG3 <- REG0 ORI IMED(0)
6	1001000001000000	REG0 <- REG0 ADD REG1
7	1010001100000000	REG0 <- REG3 SUB REG0
8	0110000000000011	REG0 <- REG0 SUBI IMED(3)
9	0000000100000000	MOV REG0 <- REG1
10	0011010100000100	REG1 <- REG1 SHL IMED(4)
11	0100010100000010	REG1 <- REG1 SHR IMED(2)
12	0111111010101010	REG3 <- REG2 ANDI IMED(170)
13	1011111001000000	REG3 <- REG2 AND REG1
14	1100001000000000	REG0 <- REG2 OR REG0

As seguintes imagens representam os testes feitos na ordem dada na tabela. Os valores dos registradores estão numerados como sinais ao lado esquerdo. Na segunda coluna, os valores estão apresentados se referem ao 0 ps então não devem ser levados em consideração a não ser se forem analisados precisamente nesse ponto.

Imagem 1

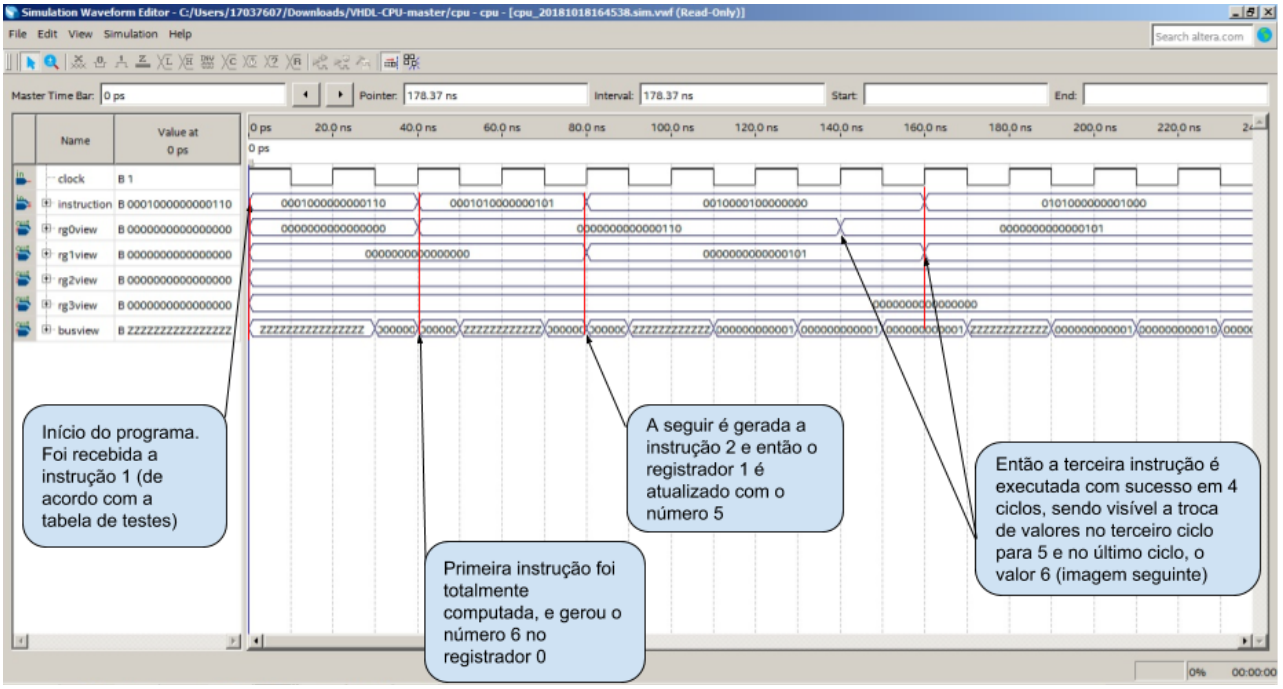


Imagem 2

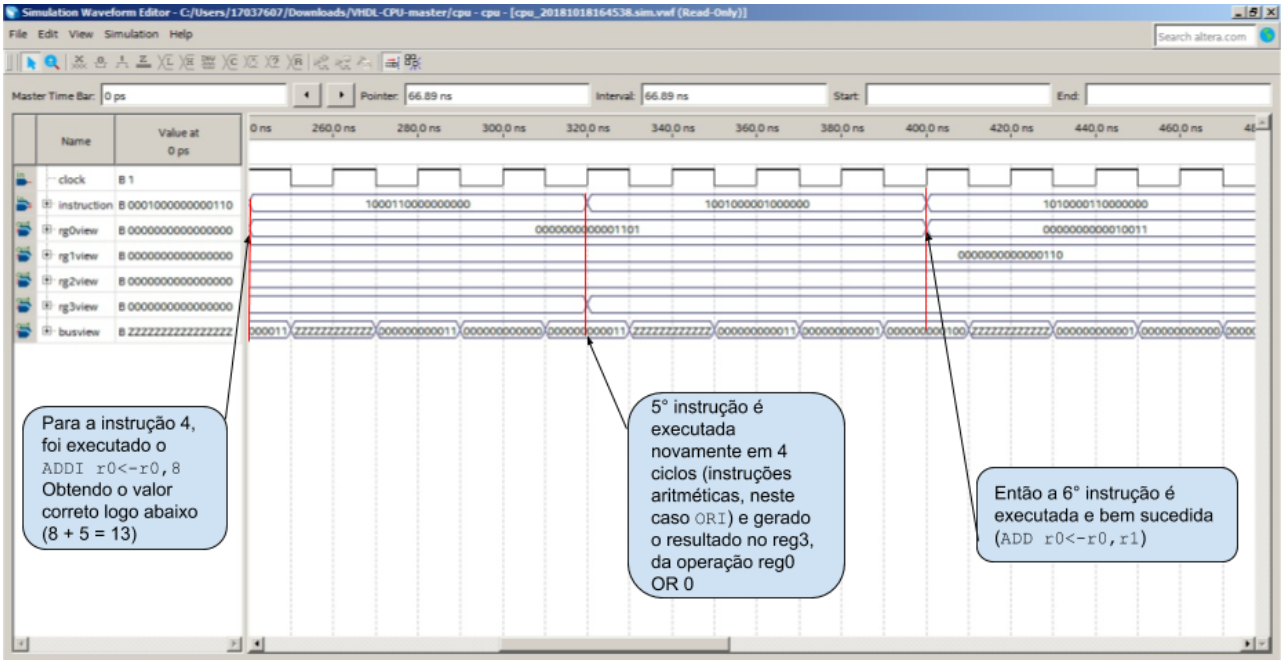


Imagem 3

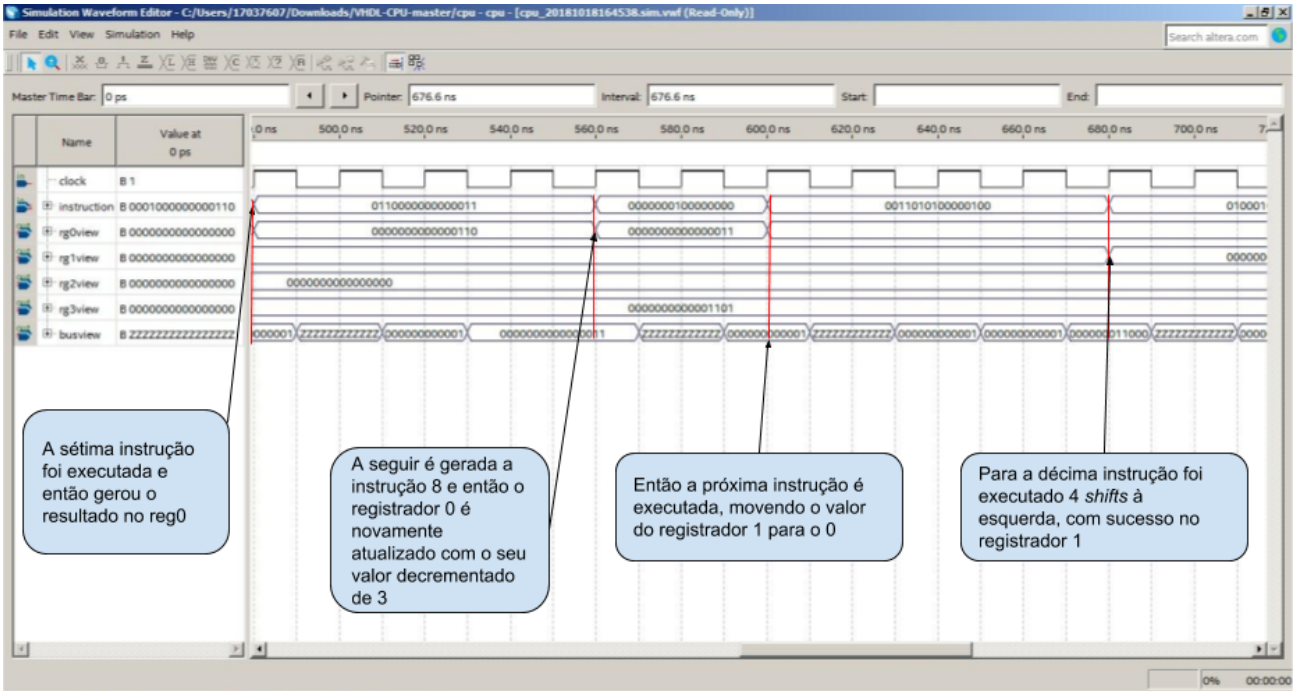


Imagem 4

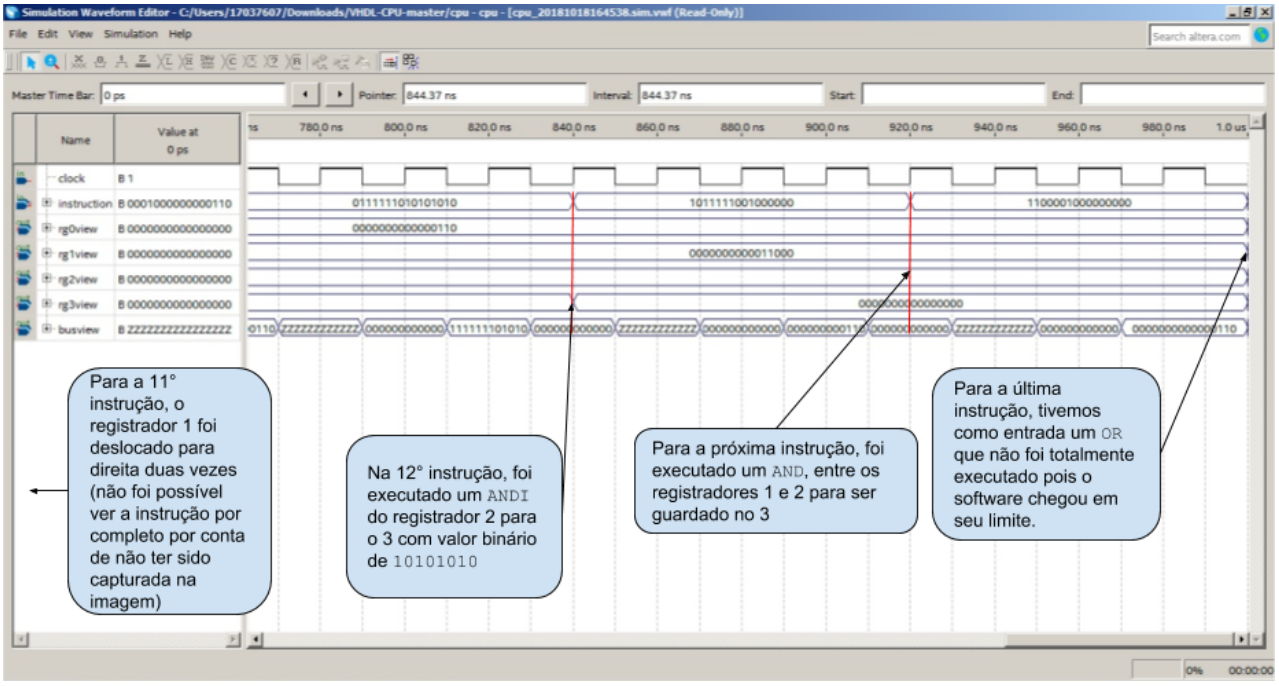
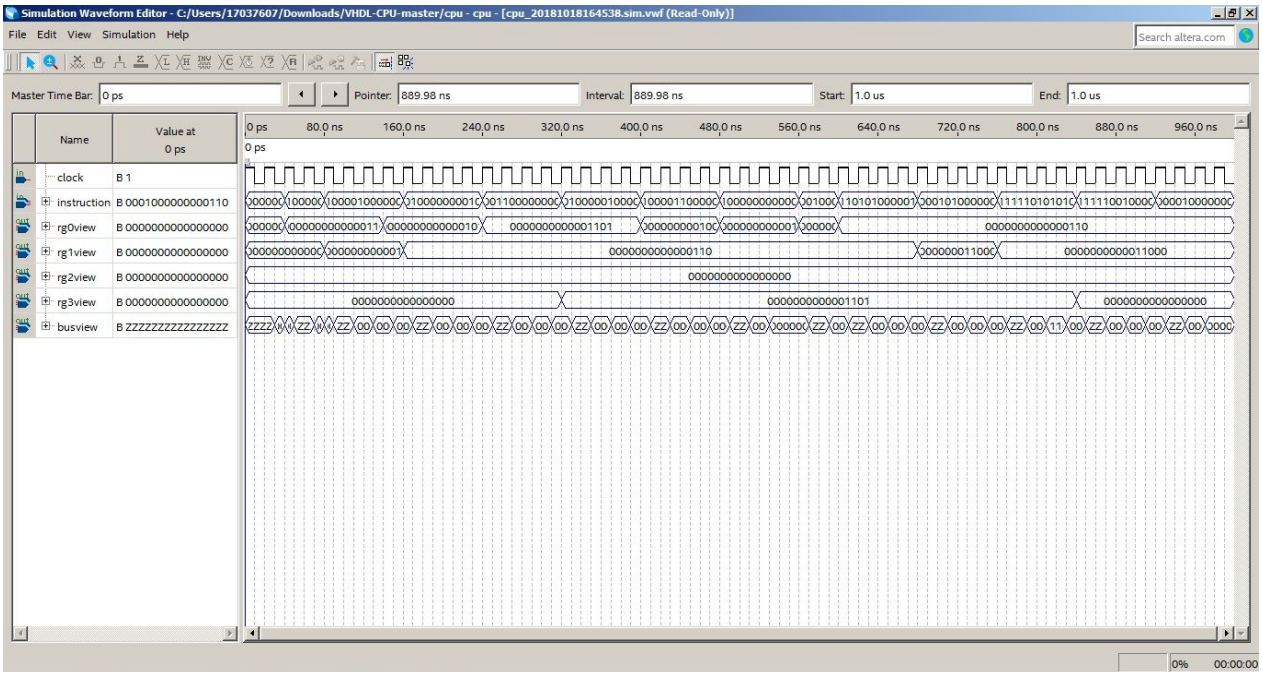


Imagem 5 (contendo todos os testes)



Resultados e discussão

Foram obtidos resultados totalmente satisfatórios, salvo na última instrução (OR) que acabou não gerando um resultado aparente por conta da limitação presente no software. Como já havíamos executado várias instruções aritméticas e tipos R (semelhante a instrução do “ou”), achamos que não haveria a necessidade de seguir com um teste novo apenas para inclusão da instrução OR.

Bibliografia

- Brown S., Vranesic S. “Fundamentals of Digital Logic with VHDL Design”, Capítulo 7 – Seção 7.14: Design Example;
- Slides disponibilizados no Ambiente Virtual de Aprendizagem (AVA) da disciplina de Arquitetura de Computadores;

Anexos

Arquivo `cpu.vhd` (principal)

```
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;
	 --reg0in, reg0out: in std_logic;
	 --reg1in, reg1out: in std_logic;
	 --reg2in, reg2out: in std_logic;
	 --reg3in, reg3out: in std_logic;
	 --regAin: in std_logic;
	 --regGin, regGout: in std_logic;
	 --regTin, regTout: in std_logic;
	 --imedIn: in std_logic;
	 busview: out std_logic_vector(0 to 15);
	 rg0view: out std_logic_vector(0 to 15);
	 rg1view: out std_logic_vector(0 to 15);
	 rg2view: out std_logic_vector(0 to 15);
	 rg3view: out std_logic_vector(0 to 15));
end cpu;

architecture behavior of cpu is
	signal mbus: std_logic_vector(0 to 15);

	--signal clock: std_logic;
	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 15);

	signal r0insid: std_logic_vector(0 to 15);
	signal r1insid: std_logic_vector(0 to 15);
	signal r2insid: std_logic_vector(0 to 15);
	signal r3insid: std_logic_vector(0 to 15);

	signal regAtoULA: std_logic_vector(0 to 15);
	signal ULAtoRegG: std_logic_vector(0 to 15);
	signal ALUOp: std_logic_vector(0 to 2);
	signal master_reset: std_logic;

	component register_16
		port (input: in std_logic_vector(0 to 15);
			clock: in std_logic;
			enable: in std_logic;
			reset: in std_logic;
```

```
                output: out std_logic_vector(0 to 15));
        end component;

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

        component triregister
                port (input: in std_logic_vector(0 to 15);
                        clock: in std_logic;
                        enablein: in std_logic;
                        enableout: in std_logic;
                        reset: in std_logic;
                        output: out std_logic_vector(0 to 15));
        end component;

        component ula
                port (regA: in std_logic_vector(0 to 15);
                        regB: in std_logic_vector(0 to 15);
                        op:    in std_logic_vector(0 to 2);
                        ula_out: out std_logic_vector(0 to 15));
        end component;

        component ctrlunit
                port (instruction: in std_logic_vector(0 to 15);
                        clock: in std_logic;
                        reg0in, reg0out: out std_logic;
                        reg1in, reg1out: out std_logic;
                        reg2in, reg2out: out std_logic;
                        reg3in, reg3out: out std_logic;
                        regAin: out std_logic;
                        regGin, regGout: out std_logic;
                        regTin, regTout: out std_logic;
                        imedIn: out std_logic;
                        ALUOp:  out std_logic_vector(0 to 2));
        end component;
begin
        busview <= mbus;
        rg0view <= r0insid;
        rg1view <= r1insid;
        rg2view <= r2insid;
        rg3view <= r3insid;

        process (instruction)
        begin
                if (instruction(8) = '0') then
                        immedi <= "00000000" & instruction(8 to 15);
                else
                        immedi <= "11111111" & instruction(8 to 15);
                end if;
        end process;
        --immedi <= "00000000" & instruction(8 to 15);
        imed: tristate port map (immedi, imedIn, mbus);

        reg0: register_16 port map (mbus, clock, reg0in, master_reset, r0insid);
        tri0: tristate port map (r0insid, reg0out, mbus);
        reg1: register_16 port map (mbus, clock, reg1in, master_reset, r1insid);
        tri1: tristate port map (r1insid, reg1out, mbus);
```

```
	reg2: register_16 port map (mbus, clock, reg2in, master_reset, r2insid);
	tri2: tristate port map (r2insid, reg2out, mbus);
	reg3: register_16 port map (mbus, clock, reg3in, master_reset, r3insid);
	tri3: tristate port map (r3insid, reg3out, mbus);

--	reg0: triregister port map (mbus, clock, reg0in, reg0out, master_reset, mbus);
--	reg1: triregister port map (mbus, clock, reg1in, reg1out, master_reset, mbus);
--	reg2: triregister port map (mbus, clock, reg2in, reg2out, master_reset, mbus);
--	reg3: triregister port map (mbus, clock, reg3in, reg3out, master_reset, mbus);

	regA: register_16 port map (mbus, clock, regAin, master_reset, regAtoULA);
	regG: triregister port map (ULAtoRegG, clock, regGin, regGout, master_reset,
mbus);
	regT: triregister port map (mbus, clock, regTin, regTout, master_reset, mbus);
	alu: ula port map (regAtoULA, mbus, ALUOp, ULAtoRegG);

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

Arquivo `ctrlunit.vhd` (Componente da unidade de controle)

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

entity ctrlunit is
	port (instruction: in std_logic_vector(0 to 15);
		clock: in std_logic;
		reg0in, reg0out: out std_logic;
		reg1in, reg1out: out std_logic;
		reg2in, reg2out: out std_logic;
		reg3in, reg3out: out std_logic;
		regAin: out std_logic;
		regGin, regGout: out std_logic;
		regTin, regTout: out std_logic;
		imedIn: out std_logic;
		ALUOp:  out std_logic_vector(0 to 2));
end ctrlunit;

architecture ctrl of ctrlunit is
	type states is (waiting, mov0, movi0, xchg0, xchg1, xchg2, arith0, arith1,
arith2);
	signal state: states := waiting;
	signal opcode: std_logic_vector(0 to 3);
	signal r1, r2, r3: std_logic_vector(0 to 1);
begin
	opcode <= instruction(0 to 3);
	r1 <= instruction(4 to 5);
	r2 <= instruction(6 to 7);
	r3 <= instruction(8 to 9);
	process(clock)
	begin
		if (clock'event and clock = '0') then
			case state is
				when waiting =>
					reg0in <= '0';
					reg0out <= '0';
					reg1in <= '0';
					reg1out <= '0';
					reg2in <= '0';
					reg2out <= '0';
					reg3in <= '0';
					reg3out <= '0';
```

```
                regAin <= '0';
                regGin <= '0';
                regGout <= '0';
                regTin <= '0';
                regTout <= '0';
                imedIn <= '0';
                case opcode is
                        when "0000" => state <= mov0;
                        when "0001" => state <= movi0;
                        when "0010" => state <= xchg0;
                        when "0011" => state <= arith0; -- SLL
                        when "0100" => state <= arith0; -- SRL
                        when "0101" => state <= arith0; -- ADDI
                        when "0110" => state <= arith0; -- SUBI
                        when "0111" => state <= arith0; -- ANDI
                        when "1000" => state <= arith0; -- ORI
                        when "1001" => state <= arith0; -- ADD
                        when "1010" => state <= arith0; -- SUB
                        when "1011" => state <= arith0; -- AND
                        when "1100" => state <= arith0; -- OR
                        when others => state <= waiting;
                end case;
        when mov0 => --============== MOV Ri, Rj =============
                case r2 is
                        when "00" => reg0out <= '1';
                        when "01" => reg1out <= '1';
                        when "10" => reg2out <= '1';
                        when "11" => reg3out <= '1';
                        when others => state <= waiting;
                end case;
                case r1 is
                        when "00" => reg0in <= '1';
                        when "01" => reg1in <= '1';
                        when "10" => reg2in <= '1';
                        when "11" => reg3in <= '1';
                        when others => state <= waiting;
                end case;
                state <= waiting;
        when movi0 => --============== MOV Ri, Imed =============
                imedIn <= '1';
                case r1 is
                        when "00" => reg0in <= '1';
                        when "01" => reg1in <= '1';
                        when "10" => reg2in <= '1';
                        when "11" => reg3in <= '1';
                        when others => state <= waiting;
                end case;
                state <= waiting;
        when xchg0 => --============== XCHG Ri, Rj =============
                regTin <= '1';
                case r1 is
                        when "00" => reg0out <= '1';
                        when "01" => reg1out <= '1';
                        when "10" => reg2out <= '1';
                        when "11" => reg3out <= '1';
                        when others => state <= waiting;
                end case;
                state <= xchg1;
        when xchg1 =>
                regTin <= '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 => state <= waiting;
```

```
                end case;
                case r2 is
                        when "00" => reg0out <= '1';
                        when "01" => reg1out <= '1';
                        when "10" => reg2out <= '1';
                        when "11" => reg3out <= '1';
                        when others => state <= waiting;
                end case;
                state <= xchg2;
        when xchg2 =>
                regTout <= '1';
                case r1 is
                        when "00" => reg0in <= '0';
                        when "01" => reg1in <= '0';
                        when "10" => reg2in <= '0';
                        when "11" => reg3in <= '0';
                        when others => state <= waiting;
                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 => state <= waiting;
                end case;
                state <= waiting;
        when arith0 => --------------- ARITMETICAS =============
                regAin <= '1';
                case r2 is
                        when "00" => reg0out <= '1';
                        when "01" => reg1out <= '1';
                        when "10" => reg2out <= '1';
                        when "11" => reg3out <= '1';
                        when others => state <= waiting;
                end case;
                state <= arith1;
        when arith1 =>
                regAin <= '0';
                reg0out <= '0';
                reg1out <= '0';
                reg2out <= '0';
                reg3out <= '0';
                if (opcode > "1000") then
                        case r3 is
                                when "00" => reg0out <= '1';
                                when "01" => reg1out <= '1';
                                when "10" => reg2out <= '1';
                                when "11" => reg3out <= '1';
                                when others => state <= waiting;
                        end case;
                else
                        imedIn <= '1';
                end if;
                case opcode is
                        when "0011" => ALUOp <= "100";
                        when "0100" => ALUOp <= "101";
                        when "0101" => ALUOp <= "000";
                        when "1001" => ALUOp <= "000";
                        when "0110" => ALUOp <= "001";
                        when "1010" => ALUOp <= "001";
                        when "0111" => ALUOp <= "010";
                        when "1011" => ALUOp <= "010";
                        when "1000" => ALUOp <= "011";
                        when "1100" => ALUOp <= "011";
                        when others => ALUOp <= "UUU";
                end case;
```

```
                                        regGin <= '1';
                                        state <= arith2;
                                when arith2 =>
                                        reg0out <= '0';
                                        reg1out <= '0';
                                        reg2out <= '0';
                                        reg3out <= '0';
                                        regGin <= '0';
                                        imedIn <= '0';
                                        case r1 is
                                                when "00" => reg0in <= '1';
                                                when "01" => reg1in <= '1';
                                                when "10" => reg2in <= '1';
                                                when "11" => reg3in <= '1';
                                                when others => state <= waiting;
                                        end case;
                                        regGout <= '1';
                                        state <= waiting;
                                when others =>
                                        state <= waiting; --WORK IN PROGRESS
                        end case;
                end if;
        end process;
end ctrl;
```

Arquivo `register_16.vhd` (Componente do registrador de 16 bits)

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

entity register_16 is
        port (input: in std_logic_vector(0 to 15);
                        clock: in std_logic;
                        enable: in std_logic;
                        reset: in std_logic;
                        output: out std_logic_vector(0 to 15));
end register_16;

architecture reg of register_16 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;
```

Arquivo `tristate.vhd` (Componente do tri-state buffer)

```
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 <= "ZZZZZZZZZZZZZZZ";
		end if;
	end process;
end buf;
```

Arquivo `triregister.vhd` (Componente utilizado para integrar e facilitar a utilização em conjunto do tri-state buffer com o registrador de 16 bits)

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

entity triregister is
	port (input: in std_logic_vector(0 to 15);
			clock: in std_logic;
			enablein: in std_logic;
			enableout: in std_logic;
			reset: in std_logic;
			output: out std_logic_vector(0 to 15));
end triregister;

architecture think of triregister is
	component register_16
		port (input: in std_logic_vector(0 to 15);
			clock: in std_logic;
			enable: in std_logic;
			reset: in std_logic;
			output: out std_logic_vector(0 to 15));
	end component;
	component tristate
		port (input: in std_logic_vector(0 to 15);
			ctrl: in std_logic;
			output: out std_logic_vector(0 to 15));
	end component;
	signal med: std_logic_vector(0 to 15);
begin
	reg: register_16 port map (input, clock, enablein, reset, med);
	tri: tristate port map (med, enableout, output);
end think;
```

Arquivo `ula.vhd` (Componente de unidade lógica e aritmética [ULA])

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

entity ula is
	port (regA: in std_logic_vector(0 to 15);
			regB: in std_logic_vector(0 to 15);
			op:    in std_logic_vector(0 to 2);
			ula_out: out std_logic_vector(0 to 15));
end ula;

architecture alu of ula is
begin
	process(regA, regB, op)
	begin
		case op is
```

```
                    when "000" => ula_out <= regA + regB;
                    when "001" => ula_out <= regA - regB;
                    when "010" => ula_out <= regA and regB;
                    when "011" => ula_out <= regA or regB;
                    --when "100" => ula_out <= std_logic_vector(unsigned(regA) sll
to_integer(unsigned(regB)));
                    when "100" => ula_out <= to_stdlogicvector(to_bitvector(regA) sll
to_integer(unsigned(regB)));
                    when "101" => ula_out <= to_stdlogicvector(to_bitvector(regA) srl
to_integer(unsigned(regB)));
                    when others => ula_out <= "0000000000000000";
                end case;
        end process;
end alu;
```