UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMATICA
Aula 31/1/23 - Sistemas Digitais
Prof. Fernanda Kastensmit

TRABALHO FINAL AHMES
Pontuação: 10 pontos (vale 1/4 da nota do semestre)

Nome: Vinícius Boff Alves Cartão: 00335551

Objetivo: projetar e descrever em VHDL o processador Ahmes, implementar 2 programas em sua memória e mostrar através de simulação lógica sem e com atraso o funcionamento.

PASSO 1: 3 pontos

Descrever o DATAPATH do processador AHMES em VHDL em uma entidade apenas chamada de datapath_ahmes.

Cole aqui o código completo em VHDL do datapath

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 29.01.2023 11:08:53
-- Design Name:
-- Module Name: Ahmes - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
```

```
--use UNISIM.VComponents.all;

entity datapath_ahmes is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;

           PC : out STD_LOGIC_VECTOR (7 downto 0);
           cargaPC: in STD_LOGIC;
           incPC: in STD_LOGIC;

           regREM : out STD_LOGIC_VECTOR (7 downto 0);
           cargaREM: in STD_LOGIC;
           selREM: in STD_LOGIC;

           RDM : out STD_LOGIC_VECTOR (7 downto 0);
           cargaRDM: in STD_LOGIC;
           selRDM: in STD_LOGIC;

           RI: out STD_LOGIC_VECTOR (7 downto 0);
           cargaRI: in STD_LOGIC;

           AC: out STD_LOGIC_VECTOR (7 downto 0);
           cargaAC: in STD_LOGIC;

           N: out STD_LOGIC;

           Z: out STD_LOGIC;
           cargaNZ: in STD_LOGIC;

           C: out STD_LOGIC;
           cargaC: in STD_LOGIC;

           V: out STD_LOGIC;
           cargaV: in STD_LOGIC;

           B: out STD_LOGIC;
            cargaB: in STD_LOGIC
           );
end datapath_ahmes;

architecture Behavioral of datapath_ahmes is
signal reg_PC: STD_LOGIC_VECTOR (7 downto 0);
signal reg_REM: STD_LOGIC_VECTOR (7 downto 0);
signal reg_RDM: STD_LOGIC_VECTOR (7 downto 0);
signal reg_RI: STD_LOGIC_VECTOR (7 downto 0);
signal reg_AC: STD_LOGIC_VECTOR (7 downto 0);
signal ULASaida: STD_LOGIC_VECTOR (7 downto 0);
signal selULA: STD_LOGIC_VECTOR (23 downto 0);
signal Nf: STD_LOGIC;
signal Zf: STD_LOGIC;
signal Cf: STD_LOGIC;
signal Vf: STD_LOGIC;
```

```
signal Bf: STD_LOGIC;
signal NUla: STD_LOGIC;
signal ZUla: STD_LOGIC;
signal CUla: STD_LOGIC;
signal VUla: STD_LOGIC;
signal BUla: STD_LOGIC;
signal Write: STD_LOGIC_VECTOR(0 DOWNTO 0);

COMPONENT blk_mem_gen_0
  PORT (
    clka : IN STD_LOGIC;
    wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
    addra : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    dina : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
END COMPONENT;

begin
your_instance_name : blk_mem_gen_0
  PORT MAP (
    clka => clk,
    wea => Write,
    addra => reg_REM,
    dina => reg_AC,
    douta => reg_RDM
  );

process(clk,rst) --PC
begin
    if rst = '1' then
        reg_PC <= ( others => '0');
    elsif (clk'event and clk='1') then
        if (cargaPC='1' and incPC='0') then
            reg_PC <= reg_RDM;
        elsif (cargaPC='0' and incPC='1') then
            reg_PC <= std_logic_vector(unsigned(reg_PC) + 1);
        else
            reg_PC <= reg_PC;
        end if;
    end if;
end process;

process(clk,rst) --REM
begin
    if rst = '1' then
        reg_REM <= ( others => '0');
    elsif (clk'event and clk='1') then
        if (cargaREM='0') then
            reg_REM <= reg_REM;
        else
            if (selREM = '0') then
```

```
            reg_REM <= reg_PC;
         else
            reg_REM <= reg_RDM;
         end if;
      end if;
   end if;
end process;

process(clk,rst) --RI
begin
   if rst = '1' then
      reg_RI <= ( others => '0');
   elsif (clk'event and clk='1') then
      if (cargaRI='0') then
         reg_RI <= reg_RI;
      else
         reg_RI <= reg_RDM;
      end if;
   end if;
end process;

process(clk,rst) --AC
begin
   if rst = '1' then
      reg_AC <= ( others => '0');
   elsif (clk'event and clk='1') then
      if (cargaAC='0') then
         reg_AC <= reg_AC;
      else
         reg_AC <= ULASaida;
      end if;
   end if;
end process;

process(clk,rst) --N
begin
   if rst = '1' then
      Nf <= '0';
   elsif (clk'event and clk='1') then
      if (cargaNZ='0') then
         Nf <= Nf;
      else
         Nf <= NUla;
      end if;
   end if;
end process;

process(clk,rst) --Z
begin
   if rst = '1' then
      Zf <= '0';
   elsif (clk'event and clk='1') then
```

```
        if (cargaNZ='0') then
            Zf <= Zf;
        else
            Zf <= ZUla;
        end if;
    end if;
end process;

process(clk,rst) --C
begin
    if rst = '1' then
        Cf <= '0';
    elsif (clk'event and clk='1') then
        if (cargaC='0') then
            Cf <= Cf;
        else
            Cf <= CUla;
        end if;
    end if;
end process;

process(clk,rst) --V
begin
    if rst = '1' then
        Vf <= '0';
    elsif (clk'event and clk='1') then
        if (cargaV='0') then
            Vf <= Vf;
        else
            Vf <= VUla;
        end if;
    end if;
end process;

process(clk,rst) --B
begin
    if rst = '1' then
        Bf <= '0';
    elsif (clk'event and clk='1') then
        if (cargaB='0') then
            Bf <= Bf;
        else
            Bf <= BUla;
        end if;
    end if;
end process;

process(reg_RDM,reg_AC,selULA,CUla,ULASaida) --ULA
variable temp : STD_LOGIC_VECTOR(8 DOWNTO 0);
begin
    ULASaida <= "00000000";
    ZUla <= '0';
```

```
    NUla <= '0';
    CUla <= '0';
    VUla <= '0';
    BUla <= '0';
        case selULA is
            when "00000000000000000000100" =>
            ULASaida <= reg_RDM; --LDA
            when "00000000000000000001000" =>
            temp := std_logic_vector(unsigned('0'&reg_AC) + unsigned('0'&reg_RDM)); --ADD
            ULASaida <= temp(7 DOWNTO 0);
            CUla <= temp(8);
            if (reg_AC(7)=reg_RDM(7)) then
                if (reg_AC(7) /= temp(7)) then
                    VUla <= '1';
                end if;
            end if;
            when "00000000000000000010000" =>
            ULASaida <= reg_AC or reg_RDM; --OR
            when "00000000000000000100000" =>
            ULASaida <= reg_AC and reg_RDM; --AND
            when "00000000000000001000000" =>
            ULASaida <= not reg_AC; --NOT
            when "00000000000000010000000" =>
            temp := std_logic_vector(unsigned('0'&reg_AC) + unsigned('0'&(not reg_RDM)) + 1); --
SUB
            ULASaida <= temp(7 DOWNTO 0);
            BUla <= not(temp(8));
            if (reg_AC(7)/=reg_RDM(7)) then
                if (reg_AC(7) /= temp(7)) then
                    VUla <= '1';
                end if;
            end if;
            when "00001000000000000000000" =>
            CUla <= reg_AC(0);
            ULASaida(6 downto 0) <= reg_AC(7 downto 1);
            ULASaida(7) <= '0'; --SHR
            when "00010000000000000000000" =>
            CUla <= reg_AC(7);
            ULASaida(7 downto 1) <= reg_AC(6 downto 0);
            ULASaida(0) <= '0'; --SHL
            when "00100000000000000000000" =>
            ULASaida(7) <= Cf;
            CUla <= reg_AC(0);
            ULASaida(6 downto 0) <= reg_AC(7 downto 1);--ROR
            when "01000000000000000000000" =>
            ULASaida(0) <= Cf;
            CUla <= reg_AC(7);
            ULASaida(7 downto 1) <= reg_AC(6 downto 0); --ROL
            when others =>
            ULASaida <= "00000000";
            ZUla <= '0';
            NUla <= '0';
```

```
            CUla <= '0';
            VUla <= '0';
            BUla <= '0';
        end case;
    if (ULASaida(7) = '1') then
        NUla <= '1';
    end if;
    if (ULASaida = "00000000") then
        ZUla <= '1';
    end if;
end process;

process(reg_RI) --Decod
begin
    case reg_RI(7 downto 4) is
        when "0000" =>
        selULA <= "00000000000000000000001"; --NOP
        when "0001" =>
        selULA <= "00000000000000000000010"; --STA
        when "0010" =>
        selULA <= "00000000000000000000100"; --LDA
        when "0011" =>
        selULA <= "00000000000000000001000"; --ADD
        when "0100" =>
        selULA <= "00000000000000000010000"; --OR
        when "0101" =>
        selULA <= "00000000000000000100000"; --AND
        when "0110" =>
        selULA <= "00000000000000001000000"; --NOT
        when "0111" =>
        selULA <= "00000000000000010000000"; --SUB
        when "1000" =>
        selULA <= "00000000000000100000000"; --JMP
        when "1001" =>
        case reg_RI(3 downto 2) is
            when "00" =>
            selULA <= "00000000000001000000000"; --JN
            when "01" =>
            selULA <= "00000000000010000000000"; --JP
            when "10" =>
            selULA <= "00000000000100000000000"; --JV
            when "11" =>
            selULA <= "00000000001000000000000"; --JNV
            when others =>
            selULA <= "00000000000000000000000";
            end case;
        when "1010" =>
        case reg_RI(2) is
            when '0' =>
            selULA <= "00000000010000000000000"; --JZ
            when '1' =>
            selULA <= "00000000100000000000000"; --JNZ
```

```
            when others =>
            selULA <= "00000000000000000000000";
            end case;
        when "1011" =>
        case reg_RI(3 downto 2) is
            when "00" =>
            selULA <= "00000000100000000000000"; --JC
            when "01" =>
            selULA <= "00000001000000000000000"; --JNC
            when "10" =>
            selULA <= "00000010000000000000000"; --JB
            when "11" =>
            selULA <= "00000100000000000000000"; --JNB
            when others =>
            selULA <= "00000000000000000000000";
            end case;
        when "1110" =>
        case reg_RI(1 downto 0) is
            when "00" =>
            selULA <= "00001000000000000000000"; --SHR
            when "01" =>
            selULA <= "00010000000000000000000"; --SHL
            when "10" =>
            selULA <= "00100000000000000000000"; --ROR
            when "11" =>
            selULA <= "01000000000000000000000"; --ROL
            when others =>
            selULA <= "00000000000000000000000";
            end case;
        when "1111" =>
        selULA <= "10000000000000000000000";
        when others =>
        selULA <= "00000000000000000000000";
        end case;
    end process;
 PC <= reg_PC;
 regREM <= reg_REM;
 RDM <= reg_RDM;
 RI <= reg_RI;
 AC <= reg_AC;
 N <= Nf;
 Z <= Zf;
 C <= Cf;
 V <= Vf;
 B <= Bf;

end Behavioral;
```

Qual componente FPGA escolheste para a síntese? xc7k70tfbv676-1

Quantos registradores tem o datapath do RAMSES? 9

Quantas operações diferentes tem a ULA? 10

A área do DATAPTH em # LUTs: 86 e #ffps: 37

PASSO 2: 3 pontos

Descrever a parte de controle do AHMES em VHDL como uma maquina de estados.

Dada as tabelas com as instruções do Neander por estado da máquina de estrados

tempo	STA	LDA	ADD	OR	AND	NOT
t0	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM
t1	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC
t2	carga RI	carga RI	carga RI	carga RI	carga RI	carga RI
t3	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	UAL(NOT), carga AC, carga NZ, goto t0
t4	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	
t5	sel=1, carga REM	sel=1, carga REM	sel=1, carga REM	sel=1, carga REM	sel=1, carga REM	
t6	carga RDM	Read	Read	Read	Read	
t7	Write, goto t0	UAL(Y), carga AC, carga NZ, goto t0	UAL(ADD), carga AC, carga NZ, goto t0	UAL(OR), carga AC, carga NZ, goto t0	UAL(AND, carga AC, carga NZ, goto t0	

tempo	JMP	JN, N=1	JN, N=0	JZ, Z=1	JZ, Z=0	NOP	HLT
t0	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM	sel=0, carga REM
t1	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC	Read, incrementa PC
t2	carga RI	carga RI	carga RI	carga RI	carga RI	carga RI	carga RI
t3	sel=0, carga REM	sel=0, carga REM	incrementa PC, goto t0	sel=0, carga REM	incrementa PC, goto t0	goto t0	Halt
t4	Read	Read		Read			
t5	carga PC, goto t0	carga PC, goto t0		carga PC, goto t0			
t6							
t7							

Código	Instrução	Significado
0000 xxxx	NOP	nenhuma operação
0001 xxxx	STA end	MEM(end) ← AC
0010 xxxx	LDA end	AC ← MEM(end)
0011 xxxx	ADD end	AC ← AC + MEM(end)
0100 xxxx	OR end	AC ← AC **OR** MEM(end) *("ou" bit-a-bit)*
0101 xxxx	AND end	AC ← AC **AND** MEM(end) *("e" bit-a-bit)*
0110 xxxx	NOT	AC ← **NOT** AC (complemento de 1)
0111 xxxx	SUB end	AC ← AC - MEM(end)
1000 xxxx	JMP end	PC ← end (desvio incondicional)
1001 00xx	JN end	IF N=1 THEN PC ← end
1001 01xx	JP end	IF N=0 THEN PC ← end
1001 10xx	JV end	IF V=1 THEN PC ← end
1001 11xx	JNV end	IF V=0 THEN PC ← end

Código	Instrução	Significado
1010 00xx	JZ end	IF Z=1 THEN PC ← end
1010 01xx	JNZ end	IF Z=0 THEN PC ← end
1011 00xx	JC end	IF C=1 THEN PC ← end
1011 01xx	JNC end	IF Z=0 THEN PC ← end
1011 10xx	JB end	IF B=1 THEN PC ← end
1011 11xx	JNB end	IF Z=0 THEN PC ← end
1110 xx00	SHR	C ← AC(0); AC(i-1) ← AC(i); AC(7) ← 0
1110 xx01	SHL	C ← AC(7); AC(i) ← AC(i-1); AC(0) ← 0
1110 xx10	ROR	C ← AC(0); AC(i-1) ← AC(i); AC(7) ← C
1110 xx11	ROL	C ← AC(7); AC(i) ← AC(i-1); AC(0) ← C
1111 xxxx	HLT	término da execução (halt)

Completar a tabela a seguir com as instruções AHMES que não tem no NEANDER

Tempo	**SHR**	**SHL**	**ROR**	**ROL**	**SUB**
T0	Sel=0, cargaREM	Sel=0, cargaREM	Sel=0, cargaREM	Sel=0, cargaREM	Sel=0, cargaREM
T1	Read, incPC	Read, incPC	Read, incPC	Read, incPC	Read, incPC
T2	cargaRI	cargaRI	cargaRI	cargaRI	cargaRI
T3	UAL(SHR) Carga AC Carga NZ Carga C goto t0	UAL(SHL) Carga AC Carga NZ Carga C goto t0	UAL(ROR) Carga AC Carga NZ Carga C goto t0	UAL(ROL) Carga AC Carga NZ Carga C goto t0	Sel=0 cargaREM
T4					Read, incPC
T5					Sel=1 cargaREM
T6					Read
T7					UAL(SUB) Carga AC Carga NZ Carga V Carga B goto t0

E as instruções novas de Desvio

Tempo	JV (V=1)	JV (V=0)	JNV (V=1)	JNV (V=0)	JC (C=1)	JC (C=0)
T0	Sel=0, cargaREM	Sel=0, cargaREM	Sel=0, cargaREM	Sel=0, cargaREM	Sel=0, cargaREM	Sel=0, cargaREM
T1	Read, incPC	Read, incPC	Read, incPC	Read, incPC	Read, incPC	Read, incPC
T2	cargaRI	cargaRI	cargaRI	cargaRI	cargaRI	cargaRI
T3	Sel=0, cargaREM	incPC, goto t0	incPC, goto t0	Sel=0, cargaREM	Sel=0, cargaREM	incPC, goto t0
T4	Read			Read	Read	
T5	cargaPC goto t0			cargaPC goto t0	cargaPC goto t0	
T6						
T7						

Tempo	JNC (C=1)	JNC (C=0)	JB (B=1)	JB (B=0)	JNB (B=1)	JNB (B=0)
T0	Sel=0, cargaREM	Sel=0, cargaREM	Sel=0, cargaREM	Sel=0, cargaREM	Sel=0, cargaREM	Sel=0, cargaREM
T1	Read, incPC	Read, incPC	Read, incPC	Read, incPC	Read, incPC	Read, incPC
T2	cargaRI	cargaRI	cargaRI	cargaRI	cargaRI	cargaRI
T3	incPC, goto t0	Sel=0, cargaREM	Sel=0, cargaREM	incPC, goto t0	incPC, goto t0	Sel=0, cargaREM
T4		Read	Read			Read
T5		cargaPC goto t0	cargaPC goto t0			cargaPC goto t0
T6						
T7						

Cole aqui o VHDL da parte de controle usando FSM com dois process.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

entity maquina_estados is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;

           reg_RI: in STD_LOGIC_VECTOR (7 downto 0);

           Nf: in STD_LOGIC;

           Zf: in STD_LOGIC;

           Cf: in STD_LOGIC;

           Vf: in STD_LOGIC;

           Bf: in STD_LOGIC;

           incPC_est: out STD_LOGIC;
           cargaPC_est: out STD_LOGIC;
           cargaREM_est: out STD_LOGIC;
           selREM_est: out STD_LOGIC;
           cargaRDM_est: out STD_LOGIC;
           cargaRI_est: out STD_LOGIC;
           cargaAC_est: out STD_LOGIC;
           Read_est: out STD_LOGIC;
           Write_est: out STD_LOGIC_VECTOR (0 downto 0);
           cargaNZ_est: out STD_LOGIC;
           cargaC_est: out STD_LOGIC;
           cargaB_est: out STD_LOGIC;
           cargaV_est: out STD_LOGIC
           );
End maquina_estados;

architecture Behavioral of maquina_estados is
 type tipoestado is (t0, t1, t2, t3, t4, t5, t6, t7);
 signal estado : tipoestado;
 signal next_state, current_state: tipoestado;

 signal incPC : STD_LOGIC;
 signal cargaPC : STD_LOGIC;
 signal cargaREM : STD_LOGIC;
 signal selREM : STD_LOGIC;
 signal cargaRDM: STD_LOGIC;
```

```
 signal cargaRI: STD_LOGIC;
 signal cargaAC: STD_LOGIC;
 signal Read: STD_LOGIC;
 signal Write: STD_LOGIC_VECTOR (0 downto 0);
 signal cargaNZ: STD_LOGIC;
signal cargaC: STD_LOGIC;
signal cargaB: STD_LOGIC;
signal cargaV: STD_LOGIC;

Begin

state_reg: process(clk, rst)
begin
          if (rst='1') then
       current_state <= t0;
          elsif (clk'event and clk='1') then
              current_state <= next_state;
          else
            current_state <= current_state;
          end if;

end process;

comb_logic: process(current_state, reg_RI, Nf, Zf, Vf, Bf, Cf)
   begin
   incPC <= '0';
   cargaPC <= '0';
   cargaREM <= '0';
   selREM <= '0';
   cargaRDM <= '0';
   cargaRI <= '0';
   cargaAC <= '0';
   Read <= '0';
   cargaNZ <= '0';
   cargaC <= '0';
   cargaV <= '0';
   cargaB <= '0';
   Write <= "0";
          case current_state is
             when t0 =>  if reg_RI(7 downto 4)="1111" then
                    next_state <= t0;
                       else
                    incPC <= '0';
                    cargaPC <= '0';
                    cargaREM <= '1';
                    selREM <= '0';
                    cargaRDM <= '0';
                    cargaRI <= '0';
                    cargaAC <= '0';
                    Read <= '0';
                    cargaNZ <= '0';
                    next_state <= t1;
```

```
            end if;
          when t1 =>  incPC <= '1';
            cargaREM <= '0';
            cargaRI <= '1';
            Read <= '1';
            next_state <= t2;

          when t2 =>  incPC <= '0';
            cargaRI <= '1';
            Read <= '0';
            next_state <= t3;

          when t3 =>
                    if reg_RI(7 downto 4)="0110" then
          cargaAC <= '1';
          cargaNZ <= '1';
          cargaRI <= '0';
          next_state <= t0;
        elsif reg_RI(7 downto 4)="0000" then
          cargaRI <= '0';
          next_state <= t0;
        elsif (reg_RI(7)='0') or (reg_RI(7 downto 4)="1000") or (reg_RI(7 downto 2)="100100" and Nf='1') or (reg_RI(7 downto 2)="100101" and Nf='0') or (reg_RI(7 downto 2)="100110" and Vf='1') or (reg_RI(7 downto 2)="100111" and Vf='0') or (reg_RI(7 downto 2)="101000" and Zf='1') or (reg_RI(7 downto 2)="101001" and Zf='0') or (reg_RI(7 downto 2)="101100" and Cf='1') or (reg_RI(7 downto 2)="101101" and Cf='0') or (reg_RI(7 downto 2)="101110" and Bf='1') or (reg_RI(7 downto 2)="101111" and Bf='0') then
            cargaREM <= '1';
            cargaRI <= '0';
            next_state <= t4;
        elsif (reg_RI(7 downto 2)="100100" and Nf='0') or (reg_RI(7 downto 2)="100101" and Nf='1') or (reg_RI(7 downto 2)="100110" and Vf='0') or (reg_RI(7 downto 2)="100111" and Vf='1') or (reg_RI(7 downto 2)="101000" and Zf='0') or (reg_RI(7 downto 2)="101001" and Zf='1') or (reg_RI(7 downto 2)="101100" and Cf='0') or (reg_RI(7 downto 2)="101101" and Cf='1') or (reg_RI(7 downto 2)="101110" and Bf='0') or (reg_RI(7 downto 2)="101111" and Bf='1') then
            incPC <= '1';
            cargaRI <= '0';
            next_state <= t0;
        elsif (reg_RI(7 downto 4)="1110") then
          cargaAC <= '1';
          cargaC <= '1';
          cargaNZ <= '1';
          cargaRI <= '0';
          next_state <= t0;
        elsif reg_RI(7 downto 4)="1111" then
          next_state <= t0;
          --HALT
        end if;
      when t4 =>
        cargaREM <= '0';
        Read <= '1';
        next_state <= t5;
```

```
        if reg_RI(7)='0' then
            incPC <= '1';
        end if;
    when t5 =>
        Read <= '0';
        incPC <= '0';
        if reg_RI(7)='0' then
            selREM <= '1';
            cargaREM <= '1';
            next_state <= t6;
        else
            cargaPC <= '1';
            next_state <= t0;
        end if;
    when t6 =>
        selREM <= '0';
        cargaREM <= '0';
        if reg_RI (7 downto 4)="0001" then
            cargaRDM <= '1';
        else
            Read <= '1';
        end if;
        next_state <= t7;
    when t7 =>
        Read <= '0';
        cargaRDM <= '0';
        if reg_RI (7 downto 4)="0001" then
            Write <= "1";
        else
            cargaAC <= '1';
            cargaNZ <= '1';
            if reg_RI (7 downto 4)="0011" then
                cargaV<='1';
                cargaC<='1';
            elsif reg_RI (7 downto 4)="0111" then
                cargaV<='1';
                cargaB<='1';
            end if;
        end if;
        next_state <= t0;
    when others =>
        next_state <= t0;
                        --HALT
    end case;

end process;

        incPC_est <= incPC;
        cargaPC_est <= cargaPC;
        cargaREM_est <= cargaREM;
        selREM_est <= selREM;
        cargaRDM_est <= cargaRDM;
```

```
        cargaRI_est <= cargaRI;
        cargaAC_est <= cargaAC;
        Read_est <= Read;
        Write_est <= Write;
        cargaNZ_est <= cargaNZ;
        cargaC_est <= cargaC;
        cargaB_est <= cargaB;
        cargaV_est <= cargaV;

end Behavioral;
```

PASSO 3: 1 ponto

Descrever o programa em Assembly do AHMES que realize a multiplicação de dois números inteiros positivos de 8 bits por Deslocamento e soma em binário e colocar no arquivo .COE na memória BRAM.

Inserir aqui o programa em Assembly com explicação

```
inicio:
LDA oito
STA i
LDA zero
STA P1
LDA var2
STA mult            ;mult = var2
continua:           ;while(i != 0)
LDA mult
SHR                 ;SHR mult
STA mult
LDA P1
JNC not_carry       ;if(mult(0) == 1) P1 = P1 + var1
ADD var1
not_carry:          ;ROR P1
ROR
STA P1
JNC not_carry_2             ;if(P1(0) == 1) mult = mult OR "10000000"
LDA mult
OR neg
STA mult
not_carry_2:
LDA i               ;i = i-1
SUB um
STA i
JNZ continua
HLT
ORG 128
var1: DB 2
var2: DB 5
P1: DB 0
mult: DB 0
i: DB 0
```

```
zero: DB 0
um: DB 1
oito: DB 8
neg: DB 128
Inserir aqui o .coe

memory_initialization_radix=10;
memory_initialization_vector=
32,135,16,132,32,133,16,130,32,129,16,131,32,131,224,16,131,32,130,180,23,48,128,226,16,130,1
80,34,32,131,64,136,16,131,32,132,112,134,16,132,164,12,240,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,2,5,0,0,0,0,1,8,128,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
```

PASSO 4: 3 pontos

Simular sem atraso o AHMES com um programa teste a ser feito pelo aluno e depois que testado e funcionando, simular com o programa do passo 3. Depois de tudo funcionando, simular também com atraso.

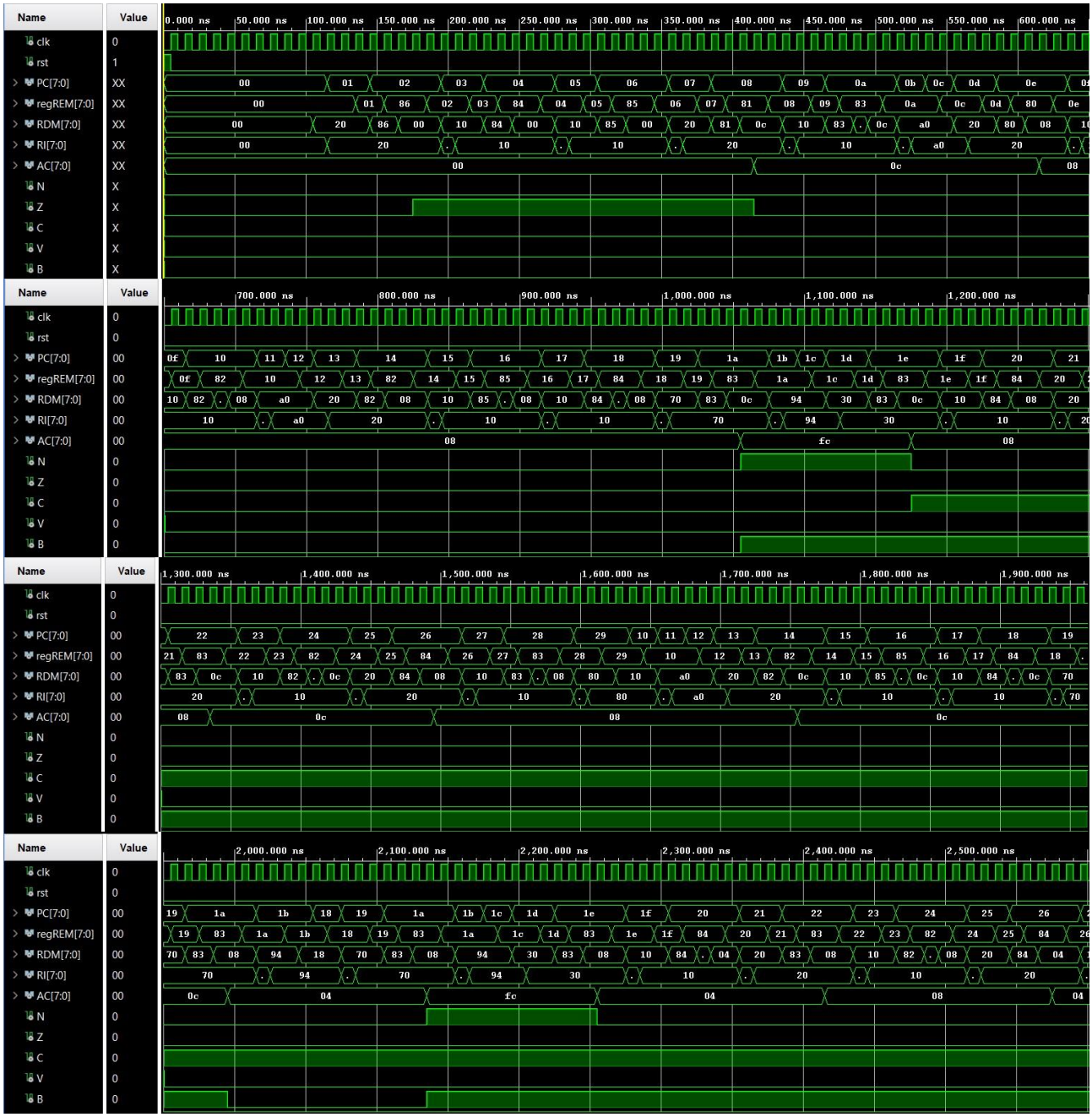
Lembrem-se que deve ser feito um testbench para a simulação.

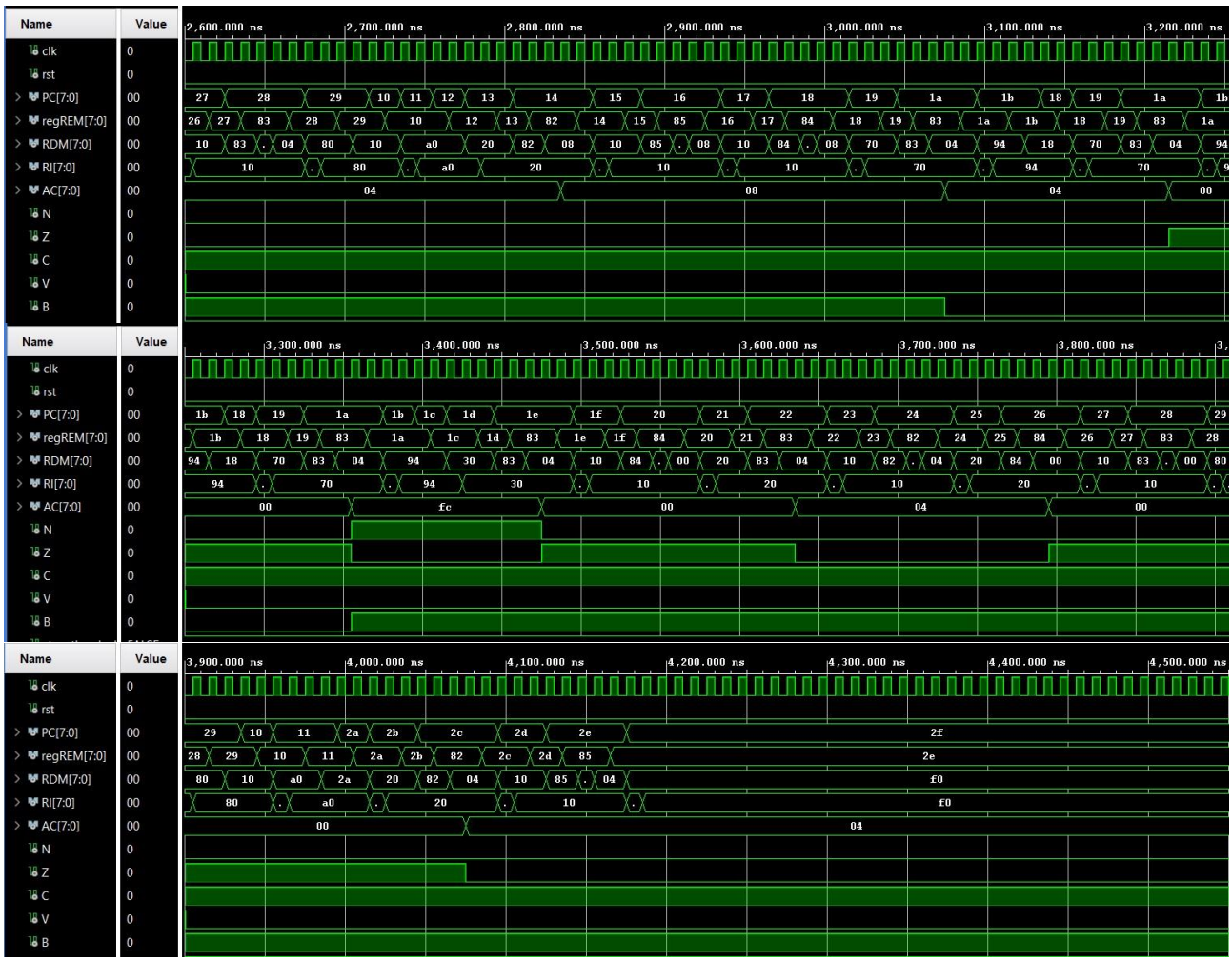
Colar aqui o programa teste e simulações (.JPG)

```
inicio:
LDA zero
STA var3
STA res
LDA num2
STA var2
JZ fim
LDA num1    ;int mdv(int i, int j){
STA var1    ;if (j == 0)
loop:       ;       return i
JZ fim      ;else
LDA var1    ;       return mdv(j, i%j);
STA res     ;Maximo divisor comum recursivo
STA var3
loop2:
SUB var2
JP loop2
ADD var2
STA var3
LDA var2
STA var1
LDA var3
STA var2
JMP loop
fim:
```

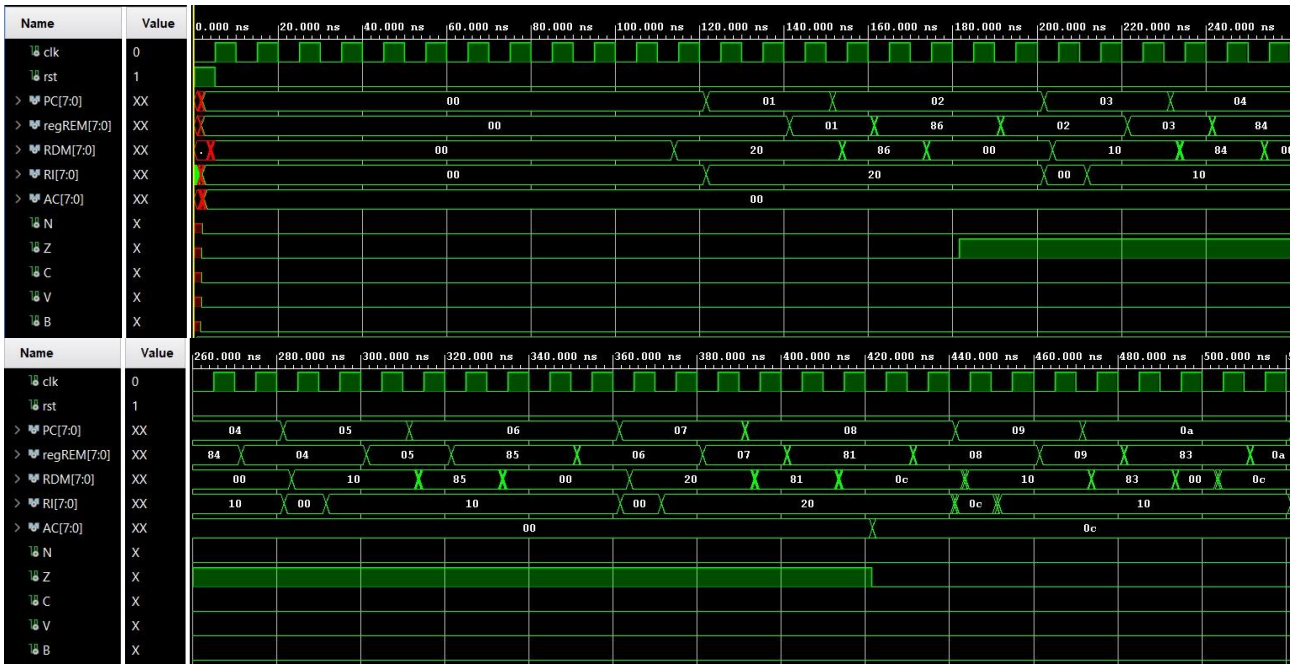
```
LDA var1
STA res
HLT
ORG 128
num1: DB 8
num2: DB 12
var1: DB 0
var2: DB 0
var3: DB 0
res: DB 0
zero: DB 0
```

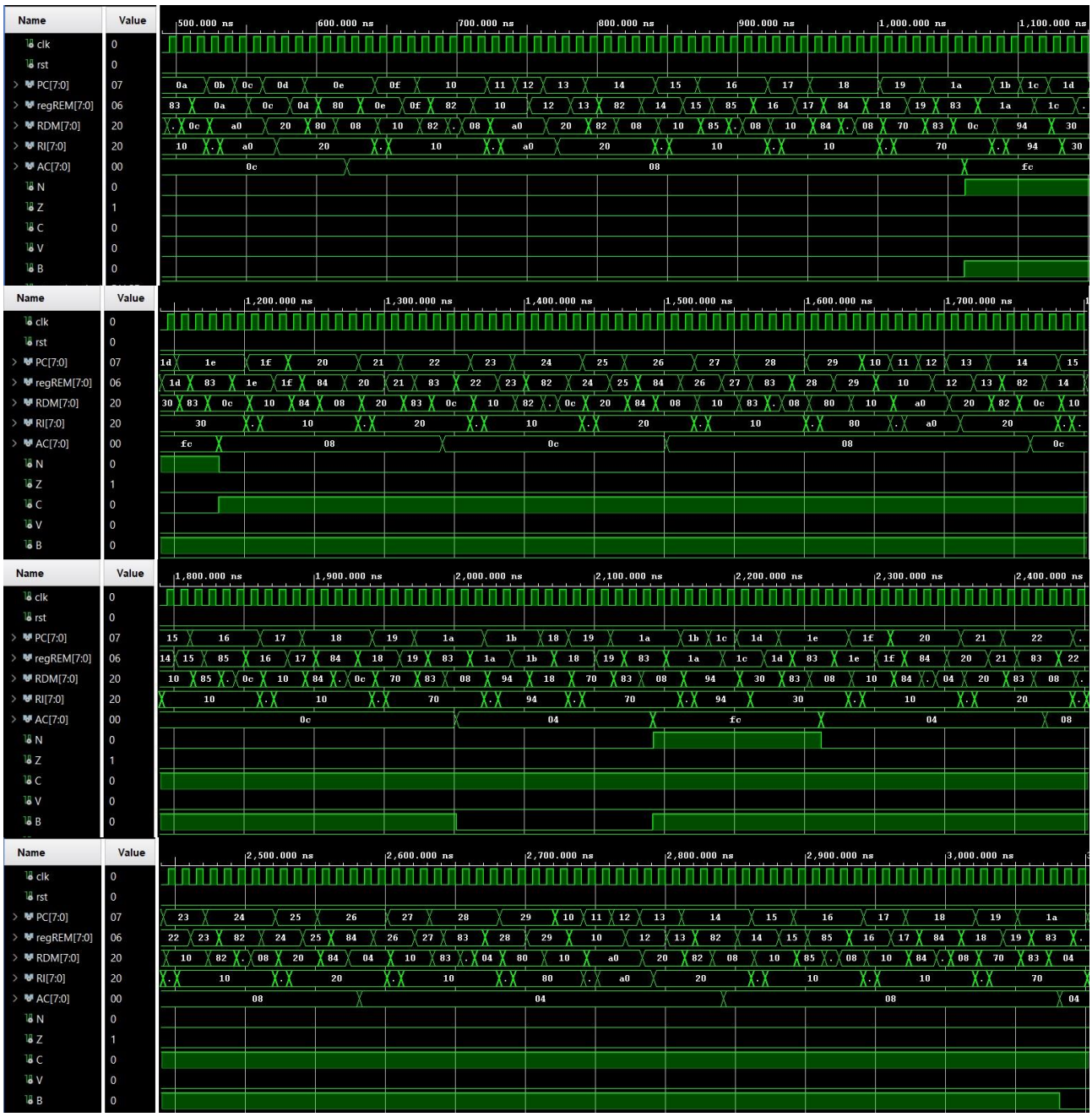
Sem atraso:

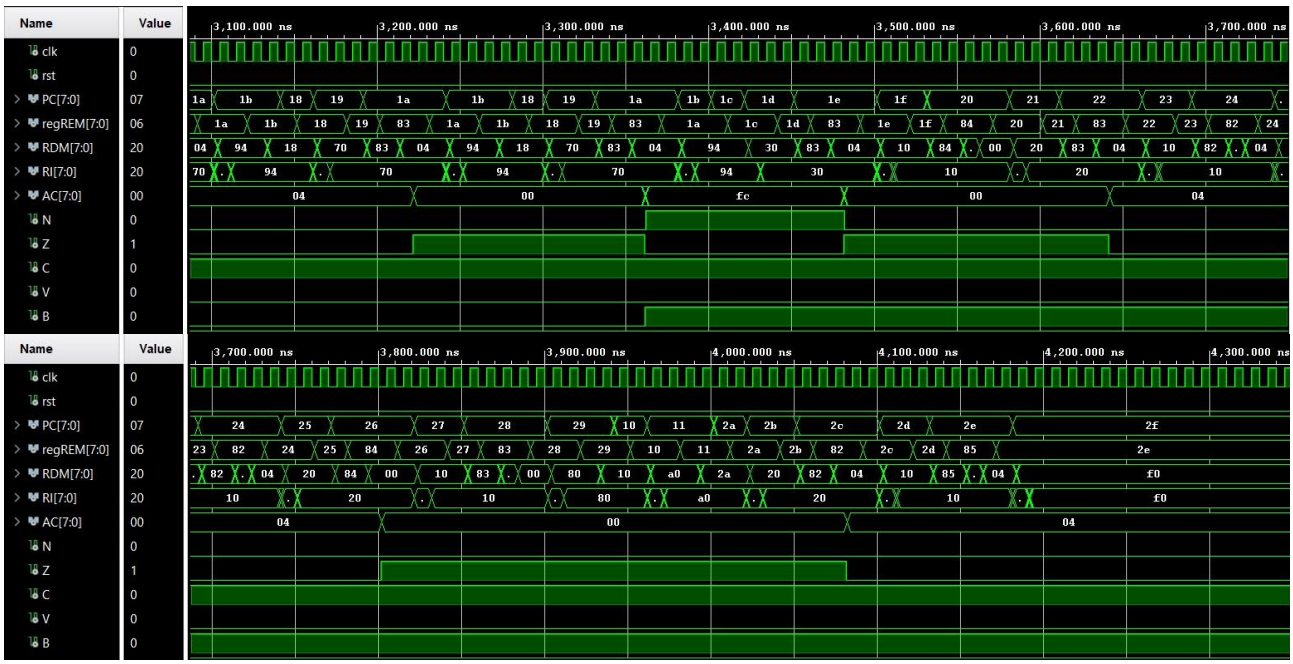



Name
Value
clk
rst
PC[7:0]
regREM[7:0]
RDM[7:0]
RI[7:0]
AC[7:0]
N
Z
C
V
B

Com atraso

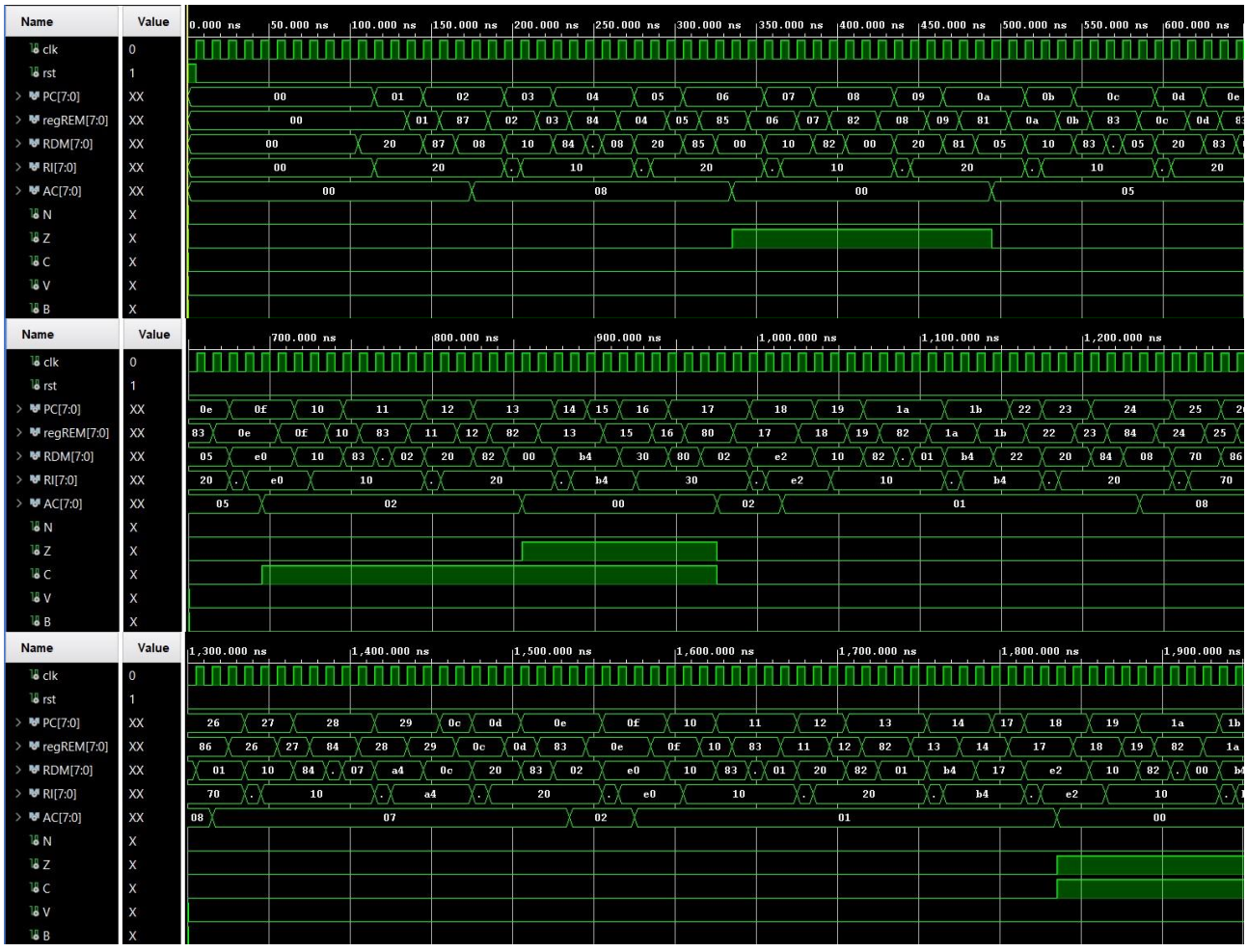

Name
Value
clk
rst
PC[7:0]
regREM[7:0]
RDM[7:0]
RI[7:0]
AC[7:0]
N
Z
C
V
B

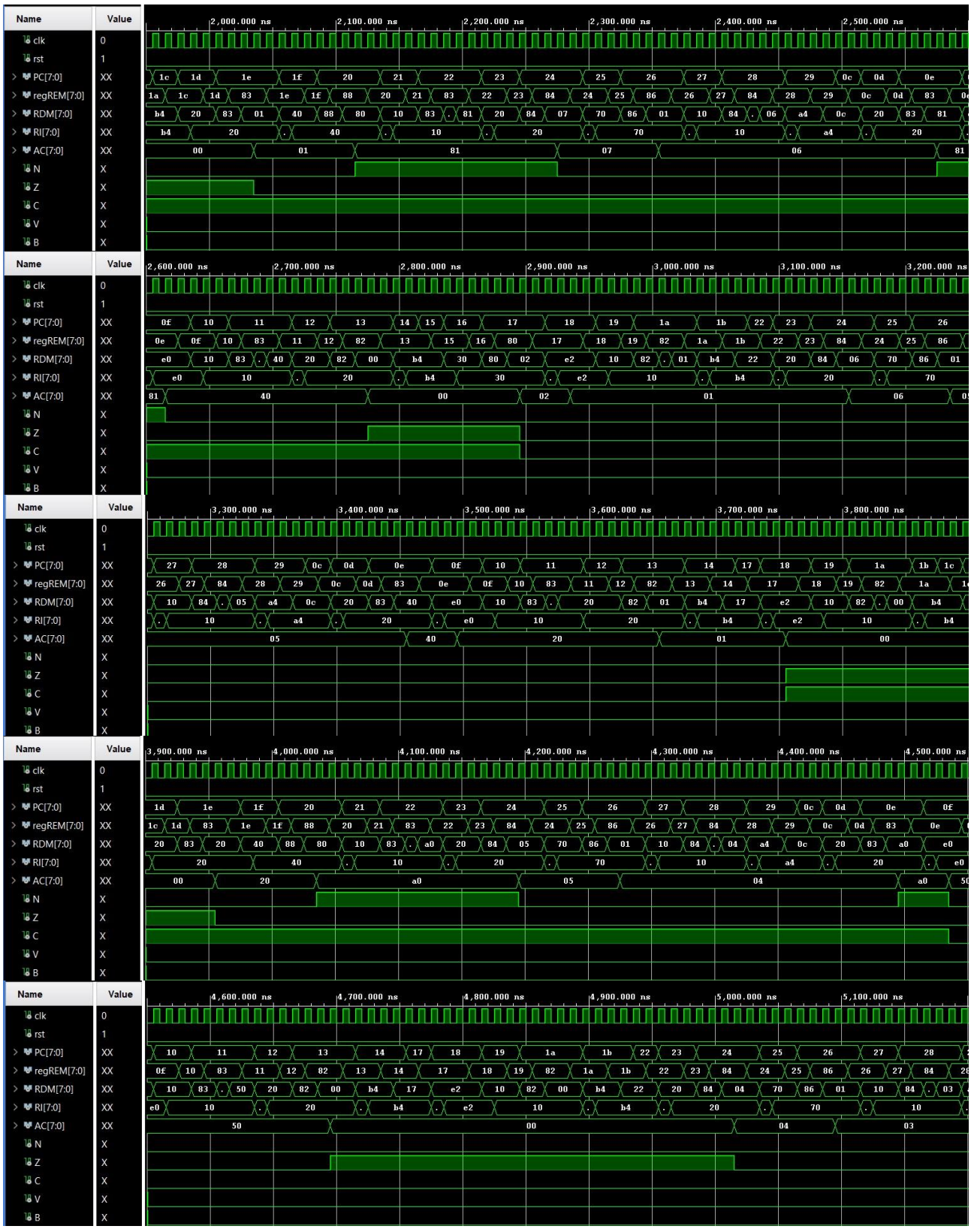

Name
Value
clk
rst
PC[7:0]
regREM[7:0]
RDM[7:0]
RI[7:0]
AC[7:0]
N
Z
C
V
B
500.000 ns
600.000 ns
700.000 ns
800.000 ns
900.000 ns
1,000.000 ns
1,100.000 ns
1,200.000 ns
1,300.000 ns
1,400.000 ns
1,500.000 ns
1,600.000 ns
1,700.000 ns
1,800.000 ns
1,900.000 ns
2,000.000 ns
2,100.000 ns
2,200.000 ns
2,300.000 ns
2,400.000 ns
2,500.000 ns
2,600.000 ns
2,700.000 ns
2,800.000 ns
2,900.000 ns
3,000.000 ns

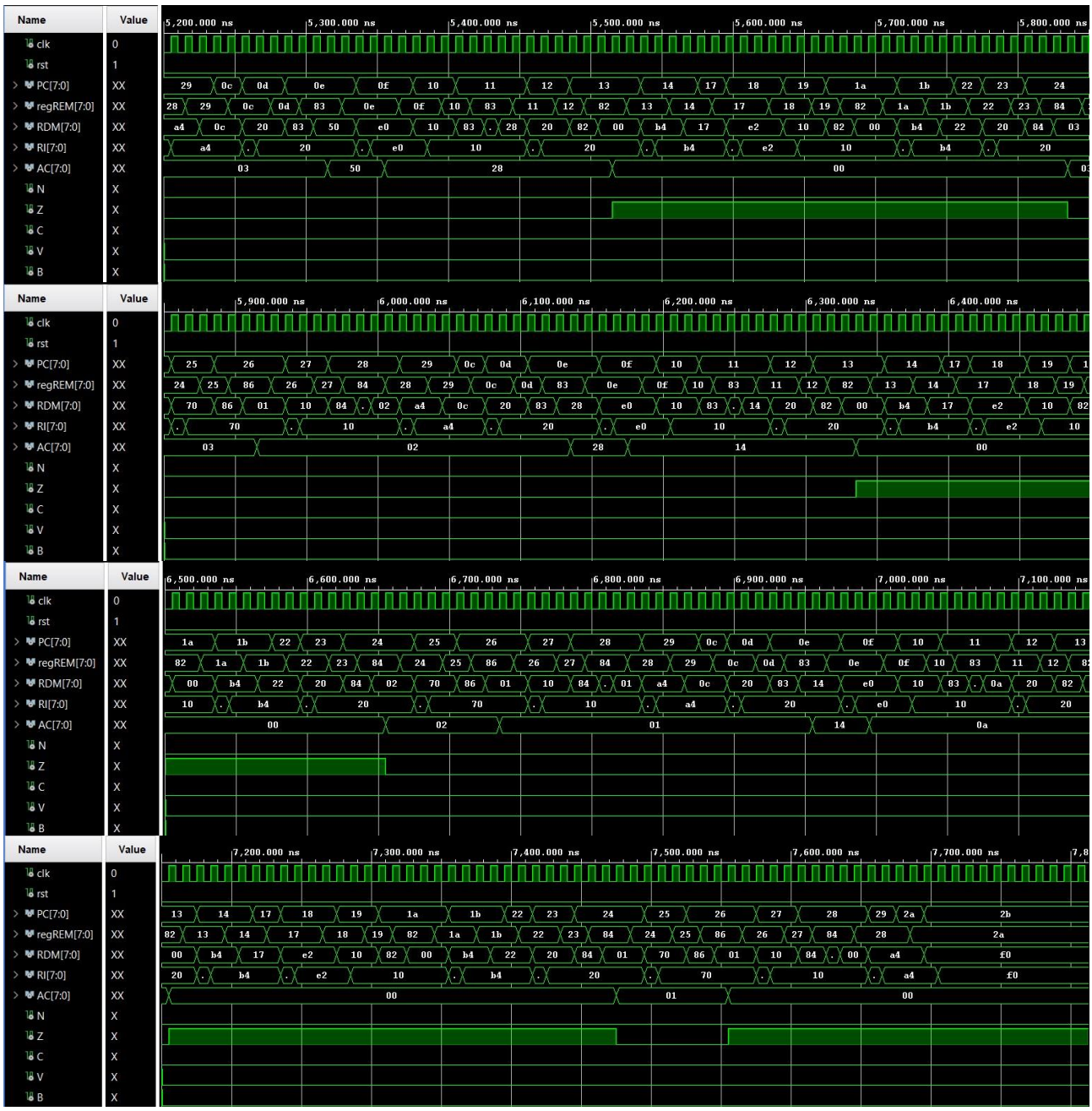


Colar aqui as simulações do programa do passo 3:

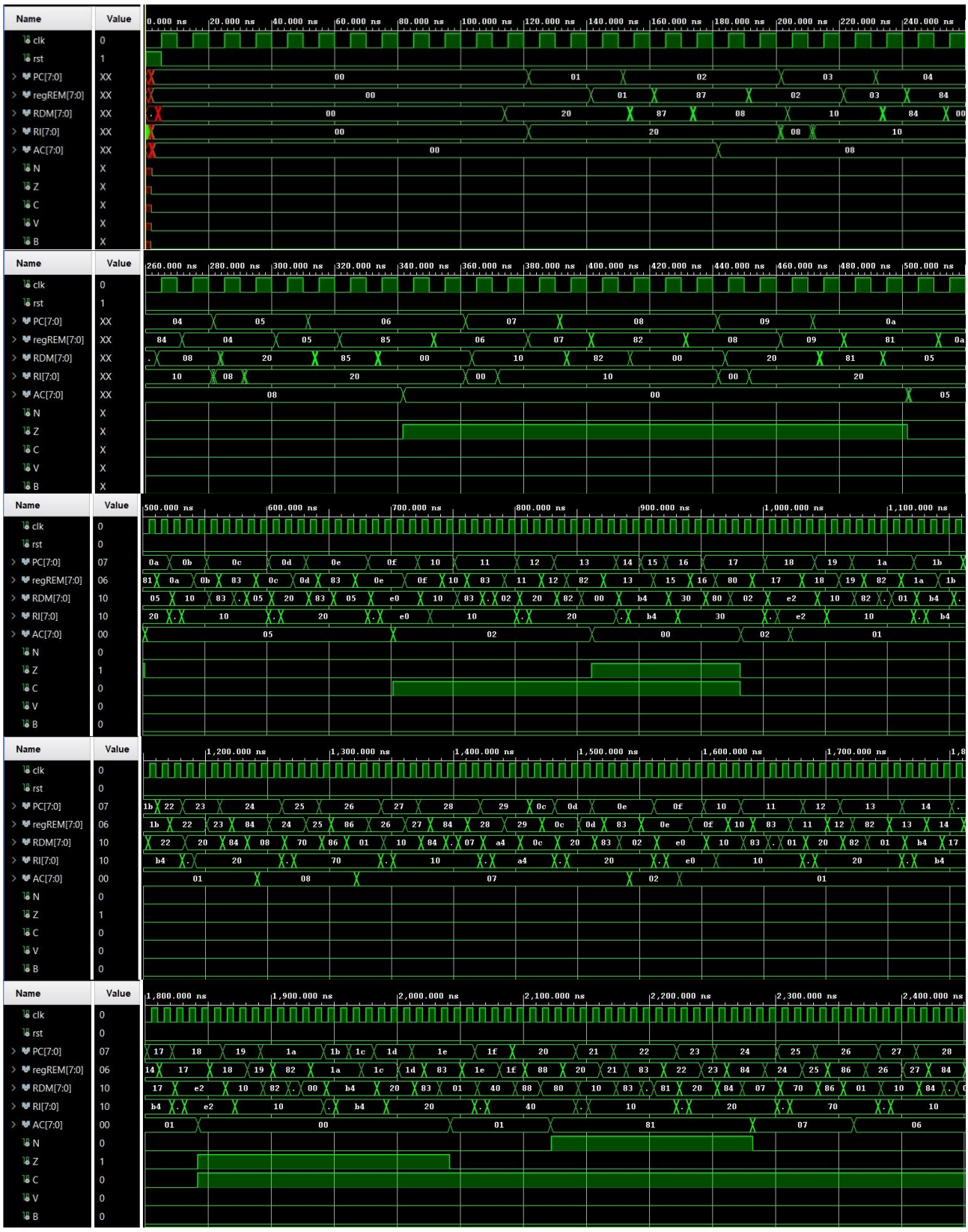
Sem atraso:

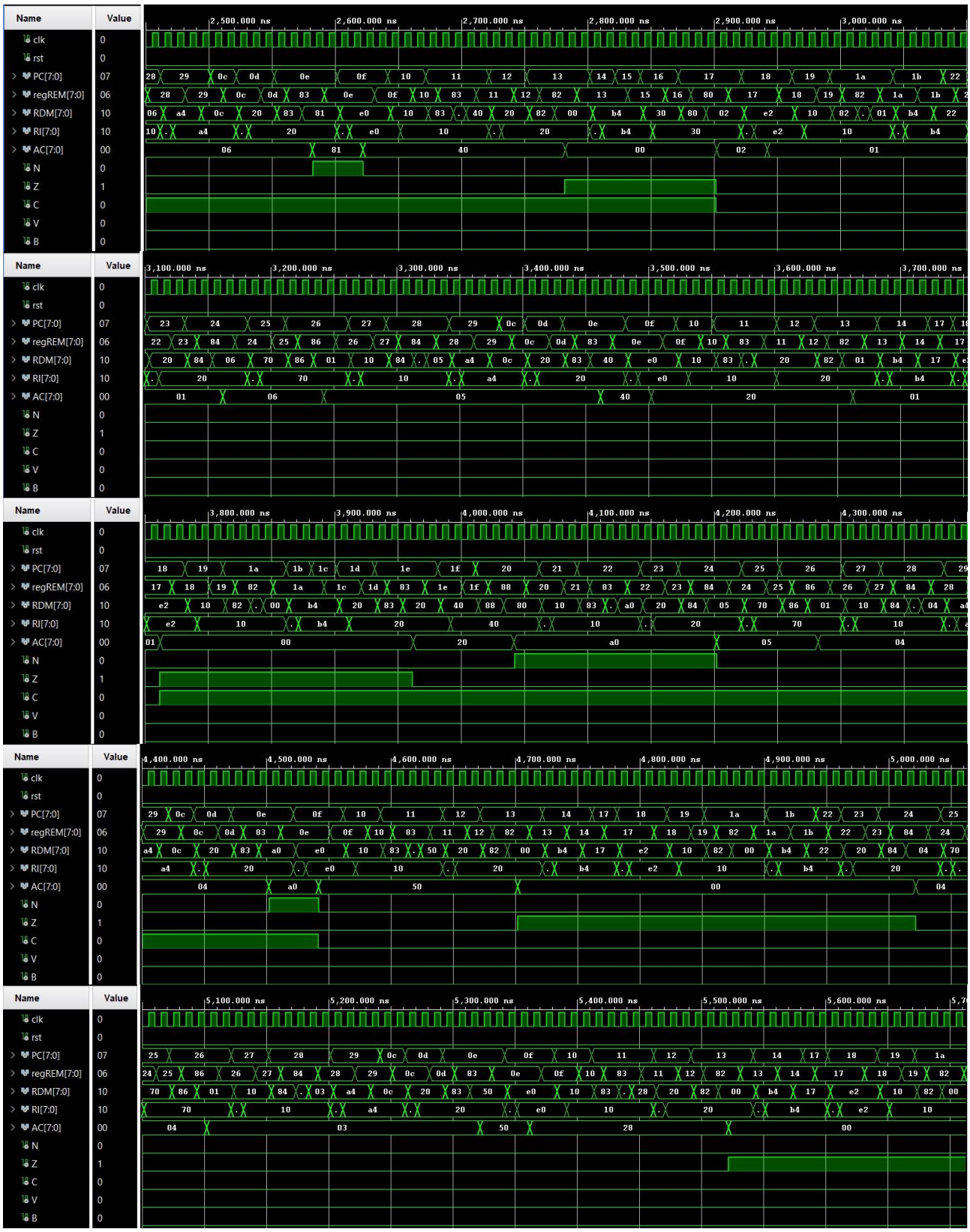


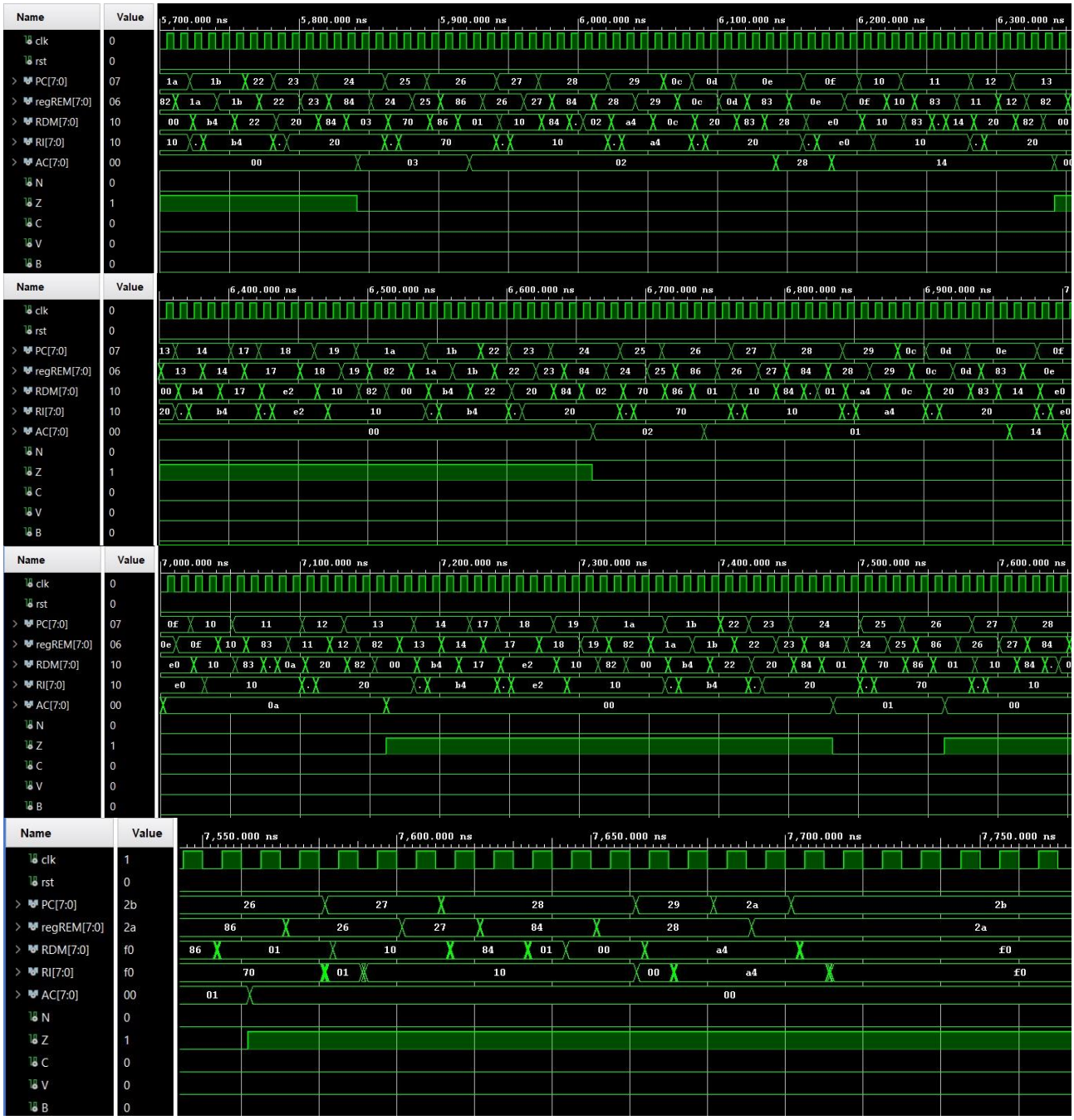

Name
Value
clk
rst
PC[7:0]
regREM[7:0]
RDM[7:0]
RI[7:0]
AC[7:0]
N
Z
C
V
B
2,000.000 ns
2,100.000 ns
2,200.000 ns
2,300.000 ns
2,400.000 ns
2,500.000 ns
2,600.000 ns
2,700.000 ns
2,800.000 ns
2,900.000 ns
3,000.000 ns
3,100.000 ns
3,200.000 ns
3,300.000 ns
3,400.000 ns
3,500.000 ns
3,600.000 ns
3,700.000 ns
3,800.000 ns
3,900.000 ns
4,000.000 ns
4,100.000 ns
4,200.000 ns
4,300.000 ns
4,400.000 ns
4,500.000 ns
4,600.000 ns
4,700.000 ns
4,800.000 ns
4,900.000 ns
5,000.000 ns
5,100.000 ns


Name
Value
clk
rst
PC[7:0]
regREM[7:0]
RDM[7:0]
RI[7:0]
AC[7:0]
N
Z
C
V
B
5,200.000 ns
5,300.000 ns
5,400.000 ns
5,500.000 ns
5,600.000 ns
5,700.000 ns
5,800.000 ns
5,900.000 ns
6,000.000 ns
6,100.000 ns
6,200.000 ns
6,300.000 ns
6,400.000 ns
6,500.000 ns
6,600.000 ns
6,700.000 ns
6,800.000 ns
6,900.000 ns
7,000.000 ns
7,100.000 ns
7,200.000 ns
7,300.000 ns
7,400.000 ns
7,500.000 ns
7,600.000 ns
7,700.000 ns

Com atraso:


Name
Value
clk
rst
PC[7:0]
regREM[7:0]
RDM[7:0]
RI[7:0]
AC[7:0]
N
Z
C
V
B
0.000 ns
20.000 ns
40.000 ns
60.000 ns
80.000 ns
100.000 ns
120.000 ns
140.000 ns
160.000 ns
180.000 ns
200.000 ns
220.000 ns
240.000 ns
260.000 ns
280.000 ns
300.000 ns
320.000 ns
340.000 ns
360.000 ns
380.000 ns
400.000 ns
420.000 ns
440.000 ns
460.000 ns
480.000 ns
500.000 ns
500.000 ns
600.000 ns
700.000 ns
800.000 ns
900.000 ns
1,000.000 ns
1,100.000 ns
1,200.000 ns
1,300.000 ns
1,400.000 ns
1,500.000 ns
1,600.000 ns
1,700.000 ns
1,800.000 ns
1,900.000 ns
2,000.000 ns
2,100.000 ns
2,200.000 ns
2,300.000 ns
2,400.000 ns


Name
Value
clk
rst
PC[7:0]
regREM[7:0]
RDM[7:0]
RI[7:0]
AC[7:0]
N
Z
C
V
B
2,500.000 ns
2,600.000 ns
2,700.000 ns
2,800.000 ns
2,900.000 ns
3,000.000 ns
3,100.000 ns
3,200.000 ns
3,300.000 ns
3,400.000 ns
3,500.000 ns
3,600.000 ns
3,700.000 ns
3,800.000 ns
3,900.000 ns
4,000.000 ns
4,100.000 ns
4,200.000 ns
4,300.000 ns
4,400.000 ns
4,500.000 ns
4,600.000 ns
4,700.000 ns
4,800.000 ns
4,900.000 ns
5,000.000 ns
5,100.000 ns
5,200.000 ns
5,300.000 ns
5,400.000 ns
5,500.000 ns
5,600.000 ns



Quantos ciclos de relógio foram necessários para a execução do programa de multiplicação no AHMES? 770

Codigo completo do Ahmes

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Ahmes is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;

           PC : out STD_LOGIC_VECTOR (7 downto 0);

           regREM : out STD_LOGIC_VECTOR (7 downto 0);

           RDM : out STD_LOGIC_VECTOR (7 downto 0);

           RI: out STD_LOGIC_VECTOR (7 downto 0);

           AC: out STD_LOGIC_VECTOR (7 downto 0);

           N: out STD_LOGIC;

           Z: out STD_LOGIC;

           C: out STD_LOGIC;

           V: out STD_LOGIC;

           B: out STD_LOGIC
           );
end Ahmes;

architecture Behavioral of Ahmes is
signal reg_PC: STD_LOGIC_VECTOR (7 downto 0);
signal reg_REM: STD_LOGIC_VECTOR (7 downto 0);
signal reg_RDM: STD_LOGIC_VECTOR (7 downto 0);
signal reg_RI: STD_LOGIC_VECTOR (7 downto 0);
signal reg_AC: STD_LOGIC_VECTOR (7 downto 0);
signal ULASaida: STD_LOGIC_VECTOR (7 downto 0);
signal selULA: STD_LOGIC_VECTOR (23 downto 0);
signal Nf: STD_LOGIC;
signal Zf: STD_LOGIC;
signal Cf: STD_LOGIC;
signal Vf: STD_LOGIC;
signal Bf: STD_LOGIC;
signal NUla: STD_LOGIC;
signal ZUla: STD_LOGIC;
signal CUla: STD_LOGIC;
signal VUla: STD_LOGIC;
```

```
signal BUla: STD_LOGIC;
signal Write: STD_LOGIC_VECTOR(0 DOWNTO 0);
signal Read: STD_LOGIC;
type tipoestado is (t0, t1, t2, t3, t4, t5, t6, t7);
signal estado : tipoestado;
signal next_state, current_state: tipoestado;
signal incPC : STD_LOGIC;
signal cargaPC : STD_LOGIC;
signal cargaREM : STD_LOGIC;
signal selREM : STD_LOGIC;
signal cargaRDM: STD_LOGIC;
signal cargaRI: STD_LOGIC;
signal cargaAC: STD_LOGIC;
signal cargaNZ: STD_LOGIC;
signal cargaC: STD_LOGIC;
signal cargaB: STD_LOGIC;
signal cargaV: STD_LOGIC;

COMPONENT blk_mem_gen_0
  PORT (
    clka : IN STD_LOGIC;
    wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
    addra : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    dina : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
END COMPONENT;

begin
your_instance_name : blk_mem_gen_0
  PORT MAP (
    clka => clk,
    wea => Write,
    addra => reg_REM,
    dina => reg_AC,
    douta => reg_RDM
  );

process(clk,rst) --PC
begin
    if rst = '1' then
        reg_PC <= ( others => '0');
    elsif (clk'event and clk='1') then
        if (cargaPC='1' and incPC='0') then
            reg_PC <= reg_RDM;
        elsif (cargaPC='0' and incPC='1') then
            reg_PC <= std_logic_vector(unsigned(reg_PC) + 1);
        else
            reg_PC <= reg_PC;
        end if;
    end if;
end process;

process(clk,rst) --REM
begin
    if rst = '1' then
        reg_REM <= ( others => '0');
```

```
    elsif (clk'event and clk='1') then
        if (cargaREM='0') then
            reg_REM <= reg_REM;
        else
            if (selREM = '0') then
                reg_REM <= reg_PC;
            else
                reg_REM <= reg_RDM;
            end if;
        end if;
    end if;
end process;

process(clk,rst) --RI
begin
    if rst = '1' then
        reg_RI <= ( others => '0');
    elsif (clk'event and clk='1') then
        if (cargaRI='0') then
            reg_RI <= reg_RI;
        else
            reg_RI <= reg_RDM;
        end if;
    end if;
end process;

process(clk,rst) --AC
begin
    if rst = '1' then
        reg_AC <= ( others => '0');
    elsif (clk'event and clk='1') then
        if (cargaAC='0') then
            reg_AC <= reg_AC;
        else
            reg_AC <= ULASaida;
        end if;
    end if;
end process;

process(clk,rst) --N
begin
    if rst = '1' then
        Nf <= '0';
    elsif (clk'event and clk='1') then
        if (cargaNZ='0') then
            Nf <= Nf;
        else
            Nf <= NUla;
        end if;
    end if;
end process;

process(clk,rst) --Z
begin
    if rst = '1' then
        Zf <= '0';
    elsif (clk'event and clk='1') then
```

```
        if (cargaNZ='0') then
            Zf <= Zf;
        else
            Zf <= ZUla;
        end if;
    end if;
end process;

process(clk,rst) --C
begin
    if rst = '1' then
        Cf <= '0';
    elsif (clk'event and clk='1') then
        if (cargaC='0') then
            Cf <= Cf;
        else
            Cf <= CUla;
        end if;
    end if;
end process;

process(clk,rst) --V
begin
    if rst = '1' then
        Vf <= '0';
    elsif (clk'event and clk='1') then
        if (cargaV='0') then
            Vf <= Vf;
        else
            Vf <= VUla;
        end if;
    end if;
end process;

process(clk,rst) --B
begin
    if rst = '1' then
        Bf <= '0';
    elsif (clk'event and clk='1') then
        if (cargaB='0') then
            Bf <= Bf;
        else
            Bf <= BUla;
        end if;
    end if;
end process;

process(reg_RDM,reg_AC,selULA,CUla,ULASaida) --ULA
variable temp : STD_LOGIC_VECTOR(8 DOWNTO 0);
begin
    ULASaida <= "00000000";
    ZUla <= '0';
    NUla <= '0';
    CUla <= '0';
    VUla <= '0';
    BUla <= '0';
            case selULA is
```

```
            when "00000000000000000000100" =>
            ULASaida <= reg_RDM; --LDA
            when "00000000000000000001000" =>
                temp := std_logic_vector(unsigned('0'&reg_AC) + unsigned('0'&reg_RDM)); --ADD
                ULASaida <= temp(7 DOWNTO 0);
                CUla <= temp(8);
        if (reg_AC(7)=reg_RDM(7)) then
           if (reg_AC(7) /= temp(7)) then
              VUla <= '1';
           end if;
        end if;
                when "00000000000000000010000" =>
                ULASaida <= reg_AC or reg_RDM; --OR
                when "00000000000000000100000" =>
                ULASaida <= reg_AC and reg_RDM; --AND
            when "00000000000000001000000" =>
            ULASaida <= not reg_AC; --NOT
            when "00000000000000010000000" =>
            temp := std_logic_vector(unsigned('0'&reg_AC) + unsigned('0'&(not reg_RDM)) + 1); --SUB
            ULASaida <= temp(7 DOWNTO 0);
            BUla <= not(temp(8));
        if (reg_AC(7)/=reg_RDM(7)) then
           if (reg_AC(7) /= temp(7)) then
              VUla <= '1';
           end if;
        end if;
            when "00001000000000000000000" =>
            CUla <= reg_AC(0);
            ULASaida(6 downto 0) <= reg_AC(7 downto 1);
            ULASaida(7) <= '0'; --SHR
            when "00010000000000000000000" =>
            CUla <= reg_AC(7);
            ULASaida(7 downto 1) <= reg_AC(6 downto 0);
            ULASaida(0) <= '0'; --SHL
            when "00100000000000000000000" =>
            ULASaida(7) <= Cf;
            CUla <= reg_AC(0);
        ULASaida(6 downto 0) <= reg_AC(7 downto 1);--ROR
            when "01000000000000000000000" =>
            ULASaida(0) <= Cf;
            CUla <= reg_AC(7);
        ULASaida(7 downto 1) <= reg_AC(6 downto 0); --ROL
            when others =>
            ULASaida <= "00000000";
        ZUla <= '0';
        NUla <= '0';
        CUla <= '0';
        VUla <= '0';
        BUla <= '0';
                end case;
    if (ULASaida(7) = '1') then
       NUla <= '1';
    end if;
    if (ULASaida = "00000000") then
       ZUla <= '1';
    end if;
end process;
```

```
process(reg_RI) --Decod
begin
   case reg_RI(7 downto 4) is
      when "0000" =>
      selULA <= "00000000000000000000001"; --NOP
      when "0001" =>
      selULA <= "00000000000000000000010"; --STA
      when "0010" =>
      selULA <= "00000000000000000000100"; --LDA
      when "0011" =>
      selULA <= "00000000000000000001000"; --ADD
      when "0100" =>
      selULA <= "00000000000000000010000"; --OR
      when "0101" =>
      selULA <= "00000000000000000100000"; --AND
      when "0110" =>
      selULA <= "00000000000000001000000"; --NOT
      when "0111" =>
      selULA <= "00000000000000010000000"; --SUB
      when "1000" =>
      selULA <= "00000000000000100000000"; --JMP
      when "1001" =>
      case reg_RI(3 downto 2) is
         when "00" =>
         selULA <= "00000000000001000000000"; --JN
         when "01" =>
         selULA <= "00000000000010000000000"; --JP
         when "10" =>
         selULA <= "00000000000100000000000"; --JV
         when "11" =>
         selULA <= "00000000001000000000000"; --JNV
         when others =>
         selULA <= "00000000000000000000000";
         end case;
      when "1010" =>
      case reg_RI(2) is
         when '0' =>
         selULA <= "00000000010000000000000"; --JZ
         when '1' =>
         selULA <= "00000000100000000000000"; --JNZ
         when others =>
         selULA <= "00000000000000000000000";
         end case;
      when "1011" =>
      case reg_RI(3 downto 2) is
         when "00" =>
         selULA <= "00000001000000000000000"; --JC
         when "01" =>
         selULA <= "00000010000000000000000"; --JNC
         when "10" =>
         selULA <= "00000100000000000000000"; --JB
         when "11" =>
         selULA <= "00001000000000000000000"; --JNB
         when others =>
         selULA <= "00000000000000000000000";
         end case;
```

```
        when "1110" =>
        case reg_RI(1 downto 0) is
           when "00" =>
           selULA <= "00001000000000000000000000"; --SHR
           when "01" =>
           selULA <= "00010000000000000000000000"; --SHL
           when "10" =>
           selULA <= "00100000000000000000000000"; --ROR
           when "11" =>
           selULA <= "01000000000000000000000000"; --ROL
           when others =>
           selULA <= "00000000000000000000000000";
           end case;
        when "1111" =>
        selULA <= "10000000000000000000000000";
        when others =>
        selULA <= "00000000000000000000000000";
        end case;
    end process;

state_reg: process(clk, rst)
begin
           if (rst='1') then
        current_state <= t0;
           elsif (clk'event and clk='1') then
              current_state <= next_state;
           else
             current_state <= current_state;
           end if;

end process;

comb_logic: process(current_state, reg_RI, Nf, Zf, Vf, Bf, Cf)
    begin
    incPC <= '0';
    cargaPC <= '0';
    cargaREM <= '0';
    selREM <= '0';
    cargaRDM <= '0';
    cargaRI <= '0';
    cargaAC <= '0';
    Read <= '0';
    cargaNZ <= '0';
    cargaC <= '0';
    cargaV <= '0';
    cargaB <= '0';
    Write <= "0";
           case current_state is
              when t0 =>  if reg_RI(7 downto 4)="1111" then
                    next_state <= t0;
                        else
                    incPC <= '0';
                    cargaPC <= '0';
                    cargaREM <= '1';
                    selREM <= '0';
                    cargaRDM <= '0';
                    cargaRI <= '0';
```

```
                cargaAC <= '0';
                Read <= '0';
                cargaNZ <= '0';
                next_state <= t1;
            end if;
          when t1 =>  incPC <= '1';
            cargaREM <= '0';
            cargaRI <= '1';
            Read <= '1';
            next_state <= t2;

          when t2 =>  incPC <= '0';
            cargaRI <= '1';
            Read <= '0';
            next_state <= t3;

          when t3 =>
                        if reg_RI(7 downto 4)="0110" then
            cargaAC <= '1';
            cargaNZ <= '1';
            cargaRI <= '0';
            next_state <= t0;
        elsif reg_RI(7 downto 4)="0000" then
            cargaRI <= '0';
            next_state <= t0;
                        elsif (reg_RI(7)='0') or (reg_RI(7 downto 4)="1000") or (reg_RI(7 downto 2)="100100" and Nf='1') or (reg_RI(7 downto 2)="100101" and Nf='0') or (reg_RI(7 downto 2)="100110" and Vf='1') or (reg_RI(7 downto 2)="100111" and Vf='0') or (reg_RI(7 downto 2)="101000" and Zf='1') or (reg_RI(7 downto 2)="101001" and Zf='0') or (reg_RI(7 downto 2)="101100" and Cf='1') or (reg_RI(7 downto 2)="101101" and Cf='0') or (reg_RI(7 downto 2)="101110" and Bf='1') or (reg_RI(7 downto 2)="101111" and Bf='0') then
            cargaREM <= '1';
            cargaRI <= '0';
            next_state <= t4;
        elsif (reg_RI(7 downto 2)="100100" and Nf='0') or (reg_RI(7 downto 2)="100101" and Nf='1') or (reg_RI(7 downto 2)="100110" and Vf='0') or (reg_RI(7 downto 2)="100111" and Vf='1') or (reg_RI(7 downto 2)="101000" and Zf='0') or (reg_RI(7 downto 2)="101001" and Zf='1') or (reg_RI(7 downto 2)="101100" and Cf='0') or (reg_RI(7 downto 2)="101101" and Cf='1') or (reg_RI(7 downto 2)="101110" and Bf='0') or (reg_RI(7 downto 2)="101111" and Bf='1') then
                            incPC <= '1';
                            cargaRI <= '0';
            next_state <= t0;
                        elsif (reg_RI(7 downto 4)="1110") then
            cargaAC <= '1';
            cargaC <= '1';
            cargaNZ <= '1';
            cargaRI <= '0';
            next_state <= t0;
        elsif reg_RI(7 downto 4)="1111" then
            next_state <= t0;
            --HALT
                        end if;
      when t4 =>
        cargaREM <= '0';
        Read <= '1';
        next_state <= t5;
        if reg_RI(7)='0' then
```

```
            incPC <= '1';
        end if;
    when t5 =>
        Read <= '0';
        incPC <= '0';
        if reg_RI(7)='0' then
            selREM <= '1';
            cargaREM <= '1';
            next_state <= t6;
        else
            cargaPC <= '1';
            next_state <= t0;
        end if;
    when t6 =>
        selREM <= '0';
        cargaREM <= '0';
        if reg_RI (7 downto 4)="0001" then
            cargaRDM <= '1';
        else
            Read <= '1';
        end if;
        next_state <= t7;
    when t7 =>
        Read <= '0';
        cargaRDM <= '0';
        if reg_RI (7 downto 4)="0001" then
            Write <= "1";
        else
            cargaAC <= '1';
            cargaNZ <= '1';
            if reg_RI (7 downto 4)="0011" then
                cargaV<='1';
                cargaC<='1';
            elsif reg_RI (7 downto 4)="0111" then
                cargaV<='1';
                cargaB<='1';
            end if;
        end if;
        next_state <= t0;
    when others =>
        reg_RI<="11110000";
        next_state <= t0;
                            --HALT
    end case;

end process;

PC <= reg_PC;
regREM <= reg_REM;
RDM <= reg_RDM;
RI <= reg_RI;
AC <= reg_AC;
N <= Nf;
Z <= Zf;
C <= Cf;
V <= Vf;
B <= Bf;
```

```
end Behavioral;
```