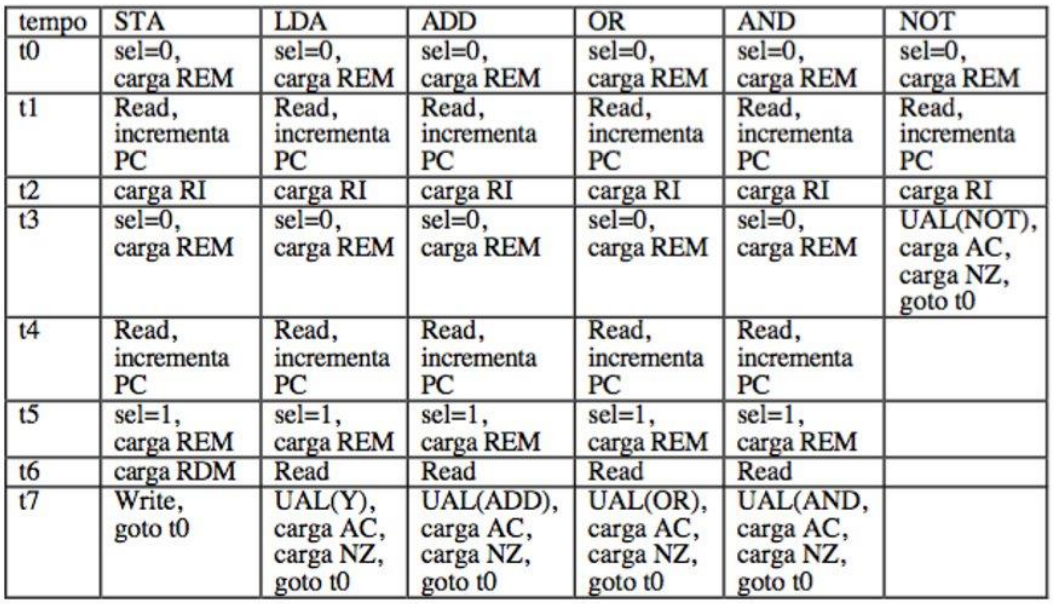
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMATICA
Aula 31/1/23 - Sistemas Digitais
Prof. Fernanda Kastensmit

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.

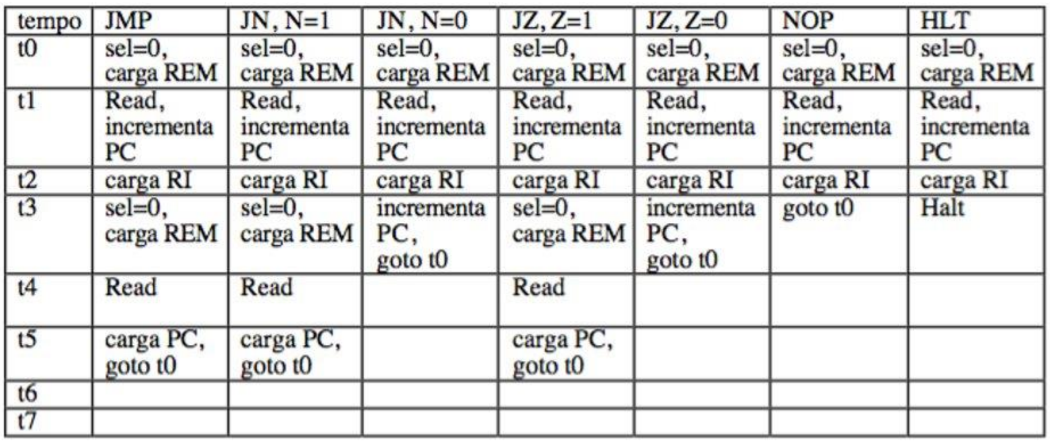
AULA 3 (31/1/2022)

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

Passo 1: 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 e as instruções novas de Desvio

Tempo	SHR	SHL	ROR	ROL	SUB	JV, V=1	JNV, V=0
T0	sel = 0, load REM	Sel = 0, load REM	Sel = 0, load REM	Sel = 0, load REM	Sel = 0, load REM	Sel = 0, load REM	Sel = 0, load REM
T1	Read, Inc PC	Read, Inc PC	Read, Inc PC	Read, Inc PC	Read, Inc PC	Read, Inc PC	Read, Inc PC
T2	load RI	load RI	load RI	load RI	Load RI	Load RI	Load RI
T3	ULA(SHR), load AC, load flags, Goto t0	ULA(SHL), load AC, load flags, Goto t0	ULA(ROR), load AC, load flags, Goto t0	ULA(ROL), load AC, load flags, Goto t0	Sel = 0, Load REM	Sel = 0, Load REM	Sel = 0, Load REM
T4					Read, Inc PC	Read	Read
T5					Sel = 1, Load REM	Load PC, Goto t0	Load PC, Goto t0
T6					Read		
T7					ULA(SUB), load AC, Load NZBC		

Tempo	JNZ, Z=0	JP, Z=0	JC, C=1	JNC, C=0	JB, B=1	JNB, B=1	
T0	Sel = 0, load REM	Sel = 0, load REM	Sel = 0, load REM	Sel = 0, load REM	Sel = 0, load REM	Sel = 0, load REM	
T1	Read, Inc PC	Read, Inc PC	Read, Inc PC	Read, Inc PC	Read, Inc PC	Read, Inc PC	
T2	Load RI	Load RI	Load RI	Load RI	Load RI	Load RI	
T3	Sel = 0, Load REM	Sel = 0, Load REM	Sel = 0, Load REM	Sel = 0, Load REM	Sel = 0, Load REM	Sel = 0, Load REM	
T4	Read	Read	Read	Read	Read	Read	
T5	Load PC, Goto t0	Load PC, Goto t0	Load PC, Goto t0	Load PC, Goto t0	Load PC, Goto t0	Load PC, Goto t0	
T6							
T7							

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

entity control_ahmes is
   Port(
            CLOCK: in std_logic;
      RESET: in std_logic;

                 -- controle dos registradores
            inc_pc: out std_logic;
                 load_ac: out std_logic;
      load_pc: out std_logic;
                 load_REM: out std_logic;
                 load_RDM: out std_logic;
      load_RI: out std_logic;
                 load_N: out std_logic;
                 load_Z: out std_logic;
                 load_V: out std_logic;
                 load_C: out std_logic;
                 load_B: out std_logic;

                 -- seletores
                 sel_ULA: out std_logic_vector(3 downto 0);
                 sel_MUXREM: out std_logic;
                 sel_MUXRDM: out std_logic;

                 -- controle da memoria
                 --mem_read: out  std_logic; -- nao utilizado
                 mem_write: out std_logic;

                 -- flags de estado
                 reg_N: in STD_LOGIC;
      reg_Z: in STD_LOGIC;
      reg_V: in STD_LOGIC;
      reg_C: in STD_LOGIC;
      reg_B: in STD_LOGIC;

                 -- flags das operacoes
                 op_nop: in std_logic;
            op_sta: in std_logic;
            op_lda: in std_logic;
                 op_add: in std_logic;
                 op_or:  in std_logic;
```

```
                    op_and: in std_logic;
                    op_not: in std_logic;
                    op_sub: in std_logic;
                    op_jmp: in std_logic;
                    op_jn:  in std_logic;
                    op_jp:  in std_logic;
                    op_jv:  in std_logic;
                    op_jnv: in std_logic;
                    op_jz:  in std_logic;
                    op_jnz: in std_logic;
                    op_jc:  in std_logic;
                    op_jnc: in std_logic;
                    op_jb:  in std_logic;
                    op_jnb: in std_logic;
                    op_shr: in std_logic;
                    op_shl: in std_logic;
                    op_ror: in std_logic;
                    op_rol: in std_logic;
                    op_hlt: in std_logic);
            end control_ahmes;

    architecture Behavioral of control_ahmes is
       type state_type is (S0, S1, S2, S3, S4, S5, S6, S7, S8);
            signal next_state, current_state: state_type;

       begin
                    process(CLOCK, RESET) -- controle de estados
                       begin
                                if(RESET = '1') then
                                        current_state <= S0;
                                elsif(rising_edge(CLOCK)) then
                                        current_state <= next_state;
                                else
                                        current_state <= current_state;
                                end if;
                            end process;

                    process(
                            next_state,

                            -- flags gerais
                            reg_N,
                            reg_Z,
                            reg_V,
                            reg_C,
                            reg_B,

                            --flags das ops
                            op_nop,
                            op_sta,
                            op_lda,
                            op_add,
op_or,
op_and,
op_not,
op_sub,
op_jmp,
op_jn,
op_jp,
op_jv,
op_jnv,
op_jz,
op_jnz,
op_jc,
op_jnc,
op_jb,
op_jnb,
op_shr,
op_shl,
op_ror,
op_rol,
op_hlt)
        begin

        -- reseta sinais de carga
        inc_pc <= '0';
        load_ac <= '0';
        load_pc <= '0';
        load_N <= '0';
        load_Z <= '0';
        load_V <= '0';
        load_C <= '0';
        load_B <= '0';
        load_RDM <= '0';
        load_REM <= '0';
        sel_MUXREM <= '0';
        sel_MUXRDM <= '0';
        sel_ULA <= "0000";
        mem_write <= '0';

        case current_state is
                when S0 =>
                        load_RDM <= '1';
                        next_state <= S1;
                when S1 =>
                        load_REM <= '0';
                        inc_pc <= '0';
                        next_state <= S2;
                when S2 =>
                        load_RDM <= '1';
                        inc_pc <= '1';
                        next_state <= S3;
                when S3 =>
                        inc_pc <= '0';
```

```
                                        load_RDM <= '0';
                                        if(op_nop = '1') then  -- NOP
                                             next_state <= S0;
                                        elsif(op_not = '1') then  -- NOT
                                             sel_ULA <= "0011";
                                             load_AC <= '1';
                                             load_N <= '1';
                                             load_Z <= '1';
                                             load_V <= '1';
                                             load_C <= '1';
                                             load_B <= '1';
                                             next_state <= S0;
                                        elsif(op_jn = '1' and reg_N = '0') then -- JN quando
n=0
                                             inc_PC <= '1';
                                             next_state <= S0;
                                        elsif(op_jp = '1' and reg_N = '1') then -- JP quando
n=1
                                             inc_PC <= '1';
                                             next_state <= S0;
                                        elsif(op_jv = '1' and reg_V = '0') then -- JV quando
v=0
                                             inc_PC <= '1';
                                             next_state <= S0;
                                        elsif(op_jv = '1' and reg_V = '1') then -- JNV quando
v=1
                                             inc_PC <= '1';
                                             next_state <= S0;
                                        elsif(op_jz = '1' and reg_Z = '0') then -- JZ quando
z=0
                                             inc_PC <= '1';
                                             next_state <= S0;
                                        elsif(op_jz = '1' and reg_Z = '1') then -- JNZ quando
z=1
                                             inc_PC <= '1';
                                             next_state <= S0;
                                        elsif(op_jc = '1' and reg_C = '0') then -- JC quando
c=0
                                             inc_PC <= '1';
                                             next_state <= S0;
                                        elsif(op_jb = '1' and reg_C = '1') then -- JNC
quando c=1
                                             inc_PC <= '1';
                                             next_state <= S0;
                                        elsif(op_jb = '1' and reg_B = '0') then -- JB quando
b=0
                                             inc_PC <= '1';
                                             next_state <= S0;
                                        elsif(op_jb = '1' and reg_B = '1') then -- JNB
quando b=1
                                             inc_PC <= '1';
                                             next_state <= S0;
```

```
        elsif(op_shr = '1') then  -- SHR
            sel_ULA <= "0101";
            load_AC <= '1';
            load_N <= '1';
            load_Z <= '1';
            load_V <= '1';
            load_C <= '1';
            load_B <= '1';
            next_state <= S0;
        elsif(op_shr = '1') then  -- SHL
            sel_ULA <= "0101";
            load_AC <= '1';
            load_N <= '1';
            load_Z <= '1';
            load_V <= '1';
            load_C <= '1';
            load_B <= '1';
            next_state <= S0;
        elsif(op_shr = '1') then  -- ROR
            sel_ULA <= "0101";
            load_AC <= '1';
            load_N <= '1';
            load_Z <= '1';
            load_V <= '1';
            load_C <= '1';
            load_B <= '1';
            next_state <= S0;
        elsif(op_shr = '1') then  -- ROL
            sel_ULA <= "0101";
            load_AC <= '1';
            load_N <= '1';
            load_Z <= '1';
            load_V <= '1';
            load_C <= '1';
            load_B <= '1';
            next_state <= S0;
        elsif(op_hlt = '1') then -- HLT
            inc_PC <= '0';
            next_state <= S8;
        else  -- qualquer outra operacao
            sel_MUXREM <= '0';
            load_REM <= '1';
            next_state <= S4;
        end if;
    when S4 =>
        sel_MUXREM <= '0';
        inc_PC <= '0';
        load_AC <= '0';
        load_REM <= '0';
        load_AC <= '0';
        load_N <= '0';
        load_Z <= '0';
```

```
          load_V <= '0';
          load_C <= '0';
          load_B <= '0';
          if(op_sta = '1' or
                    op_lda = '1' or
                    op_add = '1' or
                    op_or = '1' or
                    op_and = '1' or
                    op_sub = '1' or
                    op_jmp = '1' or
                    op_jn = '1' or
                    op_jp = '1' or
                    op_jv = '1' or
                    op_jnv = '1' or
                    op_jz = '1' or
                    op_jnz = '1' or
                    op_jc = '1' or
                    op_jnc = '1' or
                    op_jb = '1' or
                    op_jnb = '1') then
                              inc_PC <= '1';
          end if;
          next_state <= S5;
when S5 =>
          inc_PC <= '0';
          if(op_sta = '1' or
                    op_lda = '1' or
                    op_add = '1' or
                    op_or = '1' or
                    op_and = '1' or
                    op_sub = '1' or
                    op_jmp = '1' or
                    op_jn = '1' or
                    op_jp = '1' or
                    op_jv = '1' or
                    op_jnv = '1' or
                    op_jz = '1' or
                    op_jnz = '1' or
                    op_jc = '1' or
                    op_jnc = '1' or
                    op_jb = '1' or
                    op_jnb = '1') then
                              sel_MUXREM <= '1';
                              load_REM <= '1';
                              next_state <= S6;
          else
                    load_PC <= '1';
                    next_state <= S0;
          end if;
when S6 =>
          inc_PC <= '0';
          sel_MUXREM <= '0';
```

```
                    load_REM <= '0';
                    load_PC <= '0';
                    next_state <= S7;

                    if(op_sta = '1') then  -- STA
                        load_RDM <= '1';
                    end if;
                when S7 =>
                    if(op_sta = '1') then
                        mem_write <= '1';
                    elsif(op_lda = '1') then
                        sel_ULA <= "0100";  -- NOP(ULA Y)
                    elsif(op_add = '1') then
                        sel_ULA <= "0000";
                    elsif(op_or = '1') then
                        sel_ULA <= "0001";
                    elsif(op_and = '1') then
                        sel_ULA <= "0010";
                    elsif(op_not = '1') then
                        sel_ULA <= "0011";
                    elsif(op_sub = '1') then
                        sel_ULA <= "0100";
                    end if;
                when S8 =>
                    next_state <= S8;
                when others =>
                    next_state <= S0;
            end case;
            end process;
 end Behavioral;
```
*****ENTREGAR DIA 03/2/2023**** Editando esse DOC e submetendo no MS-TEAMS**