Insper

Relatório de atividade da disciplina Design de Computadores

P1 - Relógio Utilizando um Processador Personalizado

André Tavernaro, Antonio Fuziy e Marcelo Cesário Miguel

Professor Paulo Santos

Professor auxiliar Marco Alves

São Paulo
Outubro/2021

Sumário

1 Introdução

Esta atividade, realizada na disciplina Design de Computadores do Insper, no curso de Engenharia de Computação, tem por objetivo a implementação de um processador, que será utilizado em um relógio com as seguintes características: indicar horas, minutos e segundos por meio de um display de sete segmentos; sistema para acertar o horário; seleção da base de tempo. O Hardware utilizado para o projeto é uma FPGA que foi programada por meio do Software Quartus.

2 Arquitetura do processador

Para a realização do projeto, o grupo poderia escolher entre 3 arquiteturas do processador, entre elas, o acumulador, o registrador memória e o registrador registrador. A primeira consiste em armazenar o resultado sempre no acumulador, utilizando sempre um registrador apenas para armazenar os valores, já o registrador-memória, as operações ocorrem entre um registrador e uma posição de memória ou valor imediato, este utiliza um banco de registradores ao invés de um do acumulador, ao passo em que no registrador-registrador as operações só ocorrem entre registradores e os valores são carregados na memória ou imediato. A arquitetura escolhida pelo grupo foi a arquitetura de acumulador, pelo fato de que essa arquitetura já estava sendo utilizada pelo grupo nas atividades anteriores do curso, facilitando a implementação do projeto.

3 Instruções

Para o projeto, foi utilizado um total de onze instruções, sendo mostradas na tabela abaixo com o código binário indicado.

Instrução	Mnemônico	Código Binário	HEM	HLM	HFl	OP	HabA	Sel Mux	JEQ	JSR	RET	JMP	HER
Sem Operação	NOP	0000	0	0	0	11	0	0	0	0	0	0	0
Carrega valor da memória para A	LDA	0001	0	1	0	10	1	0	0	0	0	0	0
Soma A e B e armazena em A	SOMA	0010	0	1	0	01	1	0	0	0	0	0	0
Subtrai B de A e armazena em A	SUB	0011	0	1	0	00	1	0	0	0	0	0	0
Carrega valor imediato para A	LDI	0100	0	0	0	10	1	1	0	0	0	0	0
Salva valor de A para a memória	STA	0101	1	0	0	10	0	0	0	0	0	0	0
Desvio de execução	JMP	0110	0	0	0	11	0	0	0	0	0	1	0
Desvio condicional de execução	JEQ	0111	0	0	0	11	0	0	1	0	0	0	0
Comparação	CEQ	1000	0	1	1	11	0	0	0	0	0	0	0
Chamada de Sub Rotina	JSR	1001	0	0	0	11	0	0	0	1	0	0	1
Retorno de Subrotina	RET	1010	0	0	0	11	0	0	0	0	1	0	0

Tabela 1: Instruções

4 Formato das instruções

As instruções definidas na Tabela 1 tem um formato com 3 bits para o endereço dos registradores, 4 bits para o OpCode e 9 bits para o imediato, elas são utilizadas da seguinte forma:

tmp(X) = INS & AM & x"DI"

No qual **X** é a linha da instrução, **INS** é uma instrução da tabela de instruções, **AM** é um bit que indica se a instrução irá armazenar dado na memória e **DI** é o destino da instrução em hexadecimal, podendo ser um endereço de memória, ou um valor de entrada, dependendo da instrução.

5 Fluxo de dados para o processador

O fluxo de dados do processador é composto por três componentes externos, sendo eles a memória ROM de 512 posições, uma memória RAM de 64 Bytes e um decodificador 3 x 8. Esse fluxo de dados pode ser melhor analisado no diagrama abaixo.

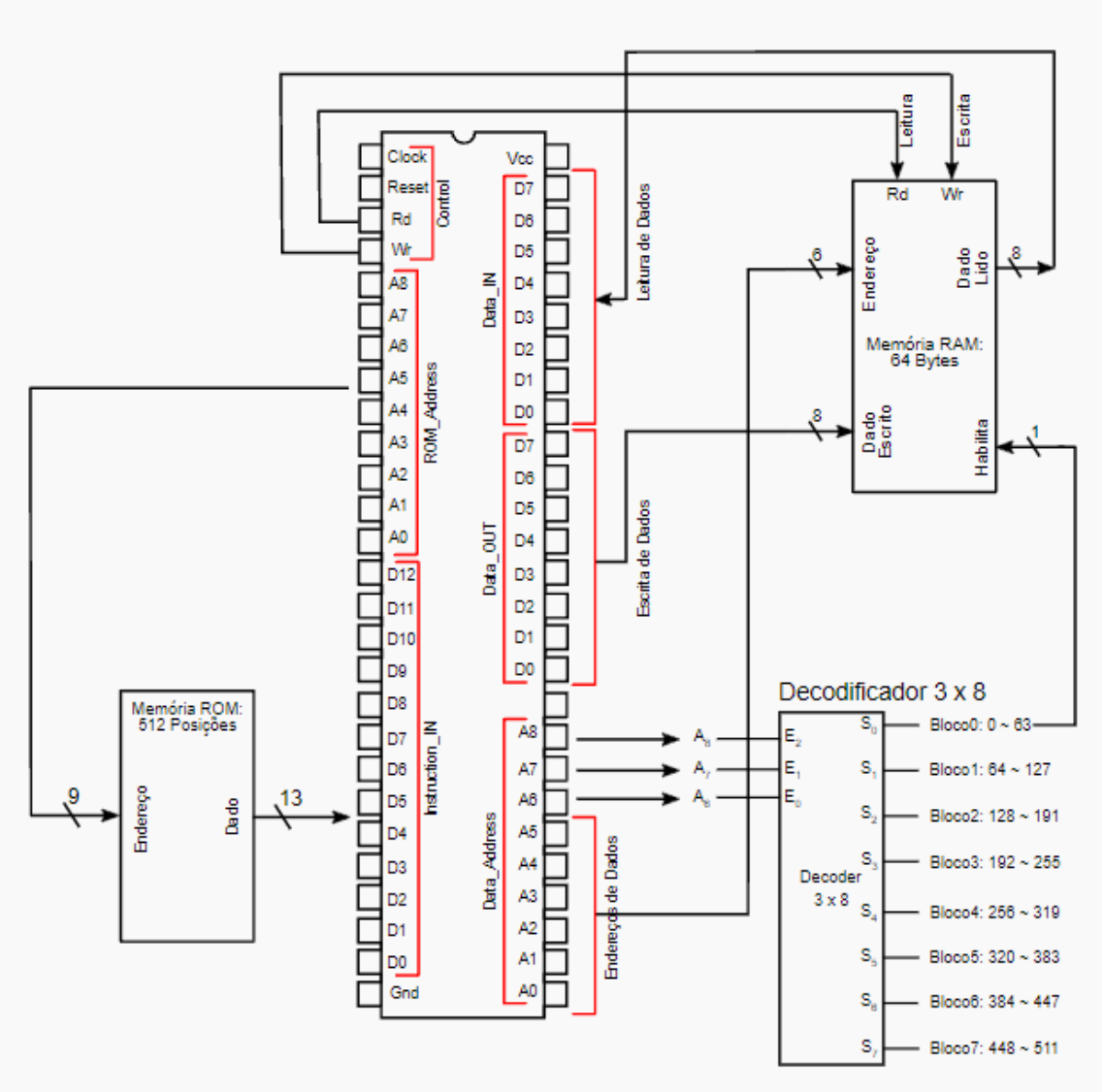

Imagem 1: Fluxo de Dados

Explicando o fluxo de dados do diagrama acima, tem-se uma ROM que recebe do processador a próxima instrução da saídas A0-A8 que envia o endereço do próximo estado. Já na memória RAM, é recebida o endereço de Dados, o Dado Escrito e as entradas RD e WR, que indica se o dado é para ser lido (RD) ou escrito na memória (WR), além disso, a RAM também recebe, por meio do decodificador, o bloco de memória indicado pela instrução.

6 Pontos de Controle

Os pontos de controle são instruções que serão decodificadas para o processador, para que este faça as operações desejadas na ordem correta, entre elas estão:

Habilita Escrita Retorno: Sua função é habilitar o registrador responsável por guardar o endereço de retorno.

HabFlag: Sua função é identificar se a saída da ULA é 0

HabLeitura: Sua função é controlar a leitura da memória RAM, podendo ativá-la ou desativá-la.

HabEscrita: Sua função é controlar a escrita da memória RAM, podendo ativá-la ou desativá-la.

Jump: Sua função é decidir efetuar um jump ou seguir para a próxima instrução sem fazer nenhum salto.

RET: Sua função é identificar um return de uma sub-rotina.

JSR: Sua função é identificar que um jump de subrotina deve ser feito. Além disso, o valor para onde se deve voltar da subrotina é armazenado.

JEQ: Sua função é identificar que um jump deve ser feito caso a saída da ULA seja 0

SelMux: Sua função é decidir se a ROM ou a RAM será o valor na entrada ULA

Operação: Sua função é identificar as operações desejadas da ULA

7 Diagrama de conexão

A conexão com os periféricos continua sendo o modelo apresentado pelo professor, sem alterações. Isto pode ser visto no diagrama abaixo:

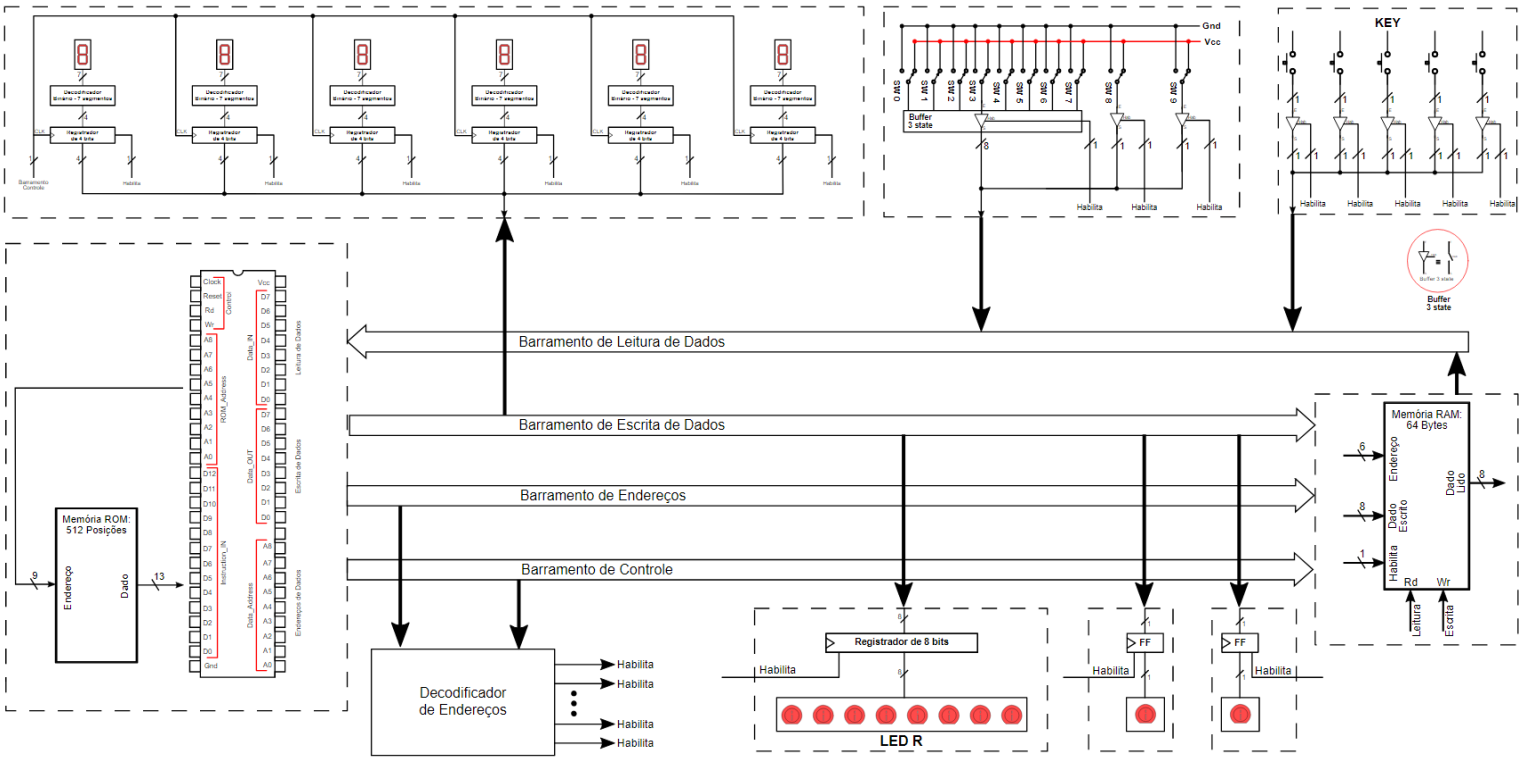

Imagem 2: Conexão com periféricos

8 Mapa de Memória

Assim como o diagrama de conexões com periféricos, o mapa de memória também continua sendo o modelo apresentado pelo professor. Isto pode ser visto na tabela abaixo.

Mapa de Memória				
Endereço em Decimal	**Periférico**	**Largura dos Dados**	**Tipo de Acesso**	**Bloco (Página) de Memória**
0 ~ 63	RAM	8 bits	Leitura/Escrita	0
64 ~ 127	Reservado	--	--	1
128 ~ 191	Reservado	--	--	2
192 ~ 255	Reservado	--	--	3
256	LEDR0 ~ LEDR7	8 bits	Escrita	4
257	LEDR8	1 bit	Escrita	4
258	LEDR9	1 bit	Escrita	4
259 ~ 287	Reservado	--	--	4
288	HEX0	4 bits	Escrita	4
289	HEX1	4 bits	Escrita	4
290	HEX2	4 bits	Escrita	4
291	HEX3	4 bits	Escrita	4
292	HEX4	4 bits	Escrita	4
293	HEX5	4 bits	Escrita	4
294 ~ 319	Reservado	--	--	4
320	SW0 ~ SW7	8 bits	Leitura	5
321	SW8	1 bit	Leitura	5
322	SW9	1 bit	Leitura	5
323 ~ 351	Reservado	--	--	5
352	KEY0	1 bit	Leitura	5
353	KEY1	1 bit	Leitura	5
354	KEY2	1 bit	Leitura	5
355	KEY3	1 bit	Leitura	5
356	FPGA_RESET	1 bit	Leitura	5
357 ~ 383	Reservado	--	--	5
384 ~ 447	Reservado	--	--	6
448 ~ 510	Reservado	--	--	7
510	Limpa Leitura KEY1	--	Escrita	7
511	Limpa Leitura KEY0	--	Escrita	7

Tabela 2: Mapa de memória

9 Funcionamento da placa

Esta seção tem como objetivo demonstrar o funcionamento de alguns inputs da placa nessa entrega intermediária.

- Key 0 - Sua função é incrementar o valor que está no display
- Key 1 - Sua função é entrar no modo de definição de limite da contagem. Esse limite é definido por um valor de binário que vem das switches [0 a 7], se o limite for atingido, o led de limite é acesso. Essa definição é feita em cada algarismo, ou seja, na dezena, centena e etc…
- FPGA RESET - Sua função é zerar o display da placa e apagar os led de overflow e limite de contagem.

- SW(7 downto 0) - Têm por finalidade definir em binário o valor do limite da cada algarismo
- LEDR(5 downto 0) - Esses LEDS devem indicar em qual algarismo está sendo definido o limite de contagem, ele acende de forma cumulativa, sendo que na definição da unidade apenas o LEDR0 acende, ao passo em que na definição da dezena, o LEDR0 e o LEDR1 acendem, e assim por diante até a definição da centena de milhar.

10 Fonte do Programa em Assembly

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

entity ROM is
  generic (
          dataWidth: natural := 8;
          addrWidth: natural := 4
          );
  port (
          Endereco : in std_logic_vector (addrWidth-1 DOWNTO 0);
          Dado : out std_logic_vector (dataWidth-1 DOWNTO 0)
          );
end entity;

architecture assincrona of ROM is

   constant NOP : std_logic_vector(3 downto 0) := "0000";
   constant LDA : std_logic_vector(3 downto 0) := "0001";
   constant SOMA: std_logic_vector(3 downto 0) := "0010";
   constant SUB : std_logic_vector(3 downto 0) := "0011";
   constant LDI : std_logic_vector(3 downto 0) := "0100";
   constant STA : std_logic_vector(3 downto 0) := "0101";
   constant JMP : std_logic_vector(3 downto 0) := "0110";
   constant JEQ : std_logic_vector(3 downto 0) := "0111";
   constant CEQ : std_logic_vector(3 downto 0) := "1000";
   constant JSR : std_logic_vector(3 downto 0) := "1001";
   constant RET : std_logic_vector(3 downto 0) := "1010";

  type blocoMemoria is array(0 TO 2**addrWidth - 1) of std_logic_vector(dataWidth-1 DOWNTO 0);

  function initMemory
          return blocoMemoria is variable tmp : blocoMemoria := (others => (others => '0'));
  begin

-- MEM[0] = UNIDADE
-- MEM[1] = DEZENA
-- MEM[2] = CENTENA
```

```
-- MEM[3] = UNIDADE DE MILHAR
-- MEM[4] = DEZENA DE MILHAR
-- MEM[5] = CENTENA DE MILHAR

-- MEM[6] = FLAG INIBE CONTAGEM

-- MEM[10] = 0
-- MEM[11] = 1
-- MEM[12] = 9
-- MEM[13] = 10

-- MEM[20] = limite da unidade
-- MEM[21] = limite da dezena
-- MEM[22] = limite da centena
-- MEM[23] = limite da unidade de milhar
-- MEM[24] = limite da dezena de milhar
-- MEM[25] = limite da centena de milhar

-- MEM[256] LEDS (7 downto 0) indicacao dos algarismos no modo de definicao de limite
-- MEM[257] LED8 Overflow
-- MEM[258] LED9 limite de contagem

-- MEM[288] DISPLAY 0
-- MEM[289] DISPLAY 1
-- MEM[290] DISPLAY 2
-- MEM[291] DISPLAY 3
-- MEM[292] DISPLAY 4
-- MEM[293] DISPLAY 5


-- ================================================================
-- Escrevendo 0 nos displays
-- ================================================================
tmp(0) := LDI & '0' & x"00"; -- Carregando 0 no acumulador
tmp(1) := STA & '1' & x"20"; -- DISPLAY 0 MEM[288]
tmp(2) := STA & '1' & x"21"; -- DISPLAY 1 MEM[289]
tmp(3) := STA & '1' & x"22"; -- DISPLAY 2 MEM[290]
tmp(4) := STA & '1' & x"23"; -- DISPLAY 3 MEM[291]
tmp(5) := STA & '1' & x"24"; -- DISPLAY 4 MEM[292]
tmp(6) := STA & '1' & x"25"; -- DISPLAY 5 MEM[293]



-- ================================================================
-- Apagando os LEDS
-- ================================================================
tmp(7) := STA & '1' & x"00"; -- Apagando LEDS 0 downto 7 MEM[256]
tmp(8) := STA & '1' & x"01"; -- Apagando LED8 MEM[257]
tmp(9) := STA & '1' & x"02"; -- Apagando LED9 MEM[258]
```

```
-- ================================================================
-- Inicializando variaveis dos displays
-- ================================================================
tmp(10) := STA & '0' & x"00"; -- Armazenando 0 no MEM[0] (UNIDADE)
tmp(11) := STA & '0' & x"01"; -- Armazenando 0 no MEM[1] (DEZENA)
tmp(12) := STA & '0' & x"02"; -- Armazenando 0 no MEM[2] (CENTENA)
tmp(13) := STA & '0' & x"03"; -- Armazenando 0 no MEM[3] (UNIDADE DE MILHAR)
tmp(14) := STA & '0' & x"04"; -- Armazenando 0 no MEM[4] (DEZENA DE MILHAR)
tmp(15) := STA & '0' & x"05"; -- Armazenando 0 no MEM[5] (CENTENA DE MILHAR)



-- ================================================================
-- Zerando flag de inibir contagem
-- ================================================================
tmp(16) := STA & '0' & x"06"; -- Armazenando 0 no MEM[6] (ZERA FLAG DE INIBIR)

-- ================================================================
-- Armazenando Variaveis
-- ================================================================
tmp(17) := LDI & '0' & x"00"; -- Carregando 0 no acumulador
tmp(18) := STA & '0' & x"0A"; -- Armazenando incremento no MEM[10]

tmp(19) := LDI & '0' & x"01"; -- Carregando 1 no acumulador
tmp(20) := STA & '0' & x"0B"; -- Armazenando incremento no MEM[11]

tmp(21) := LDI & '0' & x"09"; -- Carregando 9 no acumulador
tmp(22) := STA & '0' & x"0C"; -- Valor de comparacao no MEM[12]



-- ================================================================
-- Armazenando limite de contagem
-- ================================================================
tmp(24) := STA & '0' & x"14"; -- Armazenando limite da unidade no MEM[20]

tmp(25) := STA & '0' & x"15"; -- Armazenando limite da dezena no MEM[21]

tmp(26) := STA & '0' & x"16"; -- Armazenando limite da centena no MEM[22]

tmp(27) := STA & '0' & x"17"; -- Armazenando limite da unidade de milhar no MEM[23]

tmp(28) := STA & '0' & x"18"; -- Armazenando limite da dezena de milhar no MEM[24]

tmp(29) := STA & '0' & x"19"; -- Armazenando limite da centena de milhar no MEM[25]

--Carregando 10 no MEM[13]
tmp(30) := LDI & '0' & x"0A"; -- Carregando 10 no acumulador
tmp(31) := STA & '0' & x"0D"; -- Valor de comparacao armazenado no MEM[13]
```

```
-- ================================================================
-- LOOP DOS BOTOES (CHECA KEY0, KEY1 E O FPGA RESET)
-- ================================================================
-- Checando flag de inibir contagem
-- ================================================================
tmp(32) := NOP & '0' & x"00"; -- Retorno ao loop principal
tmp(33) := LDA & '0' & x"06"; -- Olha pro valor da flag de inibir
tmp(34) := CEQ & '0' & x"0B"; -- Compara o valor da flag com 1
tmp(35) := JEQ & '0' & x"28"; -- Pula quando a flag esta zerada ignorando a KEY0

-- ================================================================ --OBSERVANDO A KEY0
-- ================================================================
tmp(36) := LDA & '1' & x"60"; -- Observa o valor da KEY0
tmp(37) := CEQ & '0' & x"0A"; -- Verifica se a KEY0 foi ativada
tmp(38) := JEQ & '0' & x"28"; -- Caso a KEY0 for igual a 1, pula para o incremento
tmp(39) := JSR & '0' & x"36"; -- Caso a KEY0 for igual a 1, pula para o incremento

-- ================================================================
--OBSERVANDO A KEY1
-- ================================================================   tmp(40) := LDA & '1' &
x"61"; -- Observa o valor da KEY1
tmp(41) := CEQ & '0' & x"0A"; -- Verifica se a KEY1 foi ativada
tmp(42) := JEQ & '0' & x"2D"; -- Caso a KEY1 for igual a 1, pula para a definicao do limite
tmp(43) := JSR & '0' & x"BD"; -- Caso a KEY1 for igual a 1, pula para o limite

tmp(44) := NOP & '0' & x"00"; -- Pausa entre sub-rotinas
tmp(45) := JSR & '0' & x"A1"; -- Pula pra limite de contagem

-- ================================================================
--OBSERVANDO A FPGA_RESET
-- ================================================================   tmp(46) := LDA & '1' &
x"64"; -- Observa o valor do FPGA RESET
tmp(47) := CEQ & '0' & x"0B"; -- Compara o valor do FPGA RESET com 1    tmp(48) := JEQ & '0' & x"32"; --
Caso a FPGA RESET for igual a 1, pula para a reset
tmp(49) := JSR & '0' & x"78"; -- Caso a FPGA RESET for igual a 1, pula para o resetar a placa
tmp(50) := JSR & '0' & x"8D"; -- Pula para atualizar os DISPLAYS
tmp(51) := JMP & '0' & x"20"; -- Volta para o loop principal

-- ================================================================
-- LOOP DE INCREMENTO (KEY0 SUB-ROTINA)
-- ================================================================
tmp(54) := STA & '1' & x"FF"; -- Zerar KEY0
-- ================================================================
-- Incremento da unidade
-- ================================================================
tmp(55) := LDA & '0' & x"00"; -- Carrega MEM[0]
tmp(56) := SOMA & '0' & x"0B"; -- Soma 1 na unidade
tmp(57) := STA & '0' & x"00"; -- Armazena valor da soma no MEM[0]
```

```
tmp(58) := CEQ & '0' & x"0D"; -- Compara o valor de MEM[0] com 10
tmp(59) := JEQ & '0' & x"3D"; -- Pula pra checar a dezena
tmp(60) := RET & '0' & x"00"; -- Volta para o loop principal

-- ================================================================
-- Incremento da dezena
-- ================================================================
tmp(61) := LDA & '0' & x"0A"; -- Carrega MEM[10] = 0
tmp(62) := STA & '0' & x"00"; -- Zera a unidade
tmp(63) := LDA & '0' & x"01"; -- Carrega MEM[1]
tmp(64) := SOMA & '0' & x"0B"; -- Soma 1 na dezena
tmp(65) := STA & '0' & x"01"; -- Armazena valor da soma no MEM[1]
tmp(66) := CEQ & '0' & x"0D"; -- Compara o valor de MEM[1] com 10
tmp(67) := JEQ & '0' & x"45"; -- Pula pra checar a centena
tmp(68) := RET & '0' & x"00"; -- Volta para o loop principal

-- ================================================================
-- Incremento da centena
-- ================================================================
tmp(69) := LDA & '0' & x"0A"; -- Carrega MEM[10] = 0
tmp(70) := STA & '0' & x"00"; -- Zera a dezena
tmp(71) := LDA & '0' & x"02"; -- Carrega MEM[2]
tmp(72) := SOMA & '0' & x"0B"; -- Soma 1 na centena
tmp(73) := STA & '0' & x"02"; -- Armazena valor da soma no MEM[2]
tmp(74) := CEQ & '0' & x"0D"; -- Compara o valor de MEM[2] com 10
tmp(75) := JEQ & '0' & x"4D"; -- Pula pra checar a unidade de milhar
tmp(76) := RET & '0' & x"00"; -- Volta para o loop principal

-- ================================================================
-- Incremento da unidade de milhar
-- ================================================================
tmp(77) := LDA & '0' & x"0A"; -- Carrega MEM[10] = 0
tmp(78) := STA & '0' & x"00"; -- Zera a centena
tmp(79) := LDA & '0' & x"03"; -- Carrega MEM[3]
tmp(80) := SOMA & '0' & x"0B"; -- Soma 1 na unidade de milhar
tmp(81) := STA & '0' & x"03"; -- Armazena valor da soma no MEM[3]
tmp(82) := CEQ & '0' & x"0D"; -- Compara o valor de MEM[3] com 10
tmp(83) := JEQ & '0' & x"55"; -- Pula pra checar dezena de milhar
tmp(84) := RET & '0' & x"00"; -- Volta para o loop principal

-- ================================================================
-- Incremento da dezena de milhar
-- ================================================================
tmp(85) := LDA & '0' & x"0A"; -- Carrega MEM[10] = 0
tmp(86) := STA & '0' & x"00"; -- Zera a unidade de milhar          tmp(87) := LDA & '0' & x"04"; -- Carrega
MEM[4]
tmp(88) := SOMA & '0' & x"0B"; -- Soma 1 na dezena de milhar
tmp(89) := STA & '0' & x"04"; -- Armazena valor da soma no MEM[4]
tmp(90) := CEQ & '0' & x"0D"; -- Compara o valor de MEM[4] com 10
tmp(91) := JEQ & '0' & x"5D"; -- Pula pra checar a centena de milhar
```

```
tmp(92) := RET & '0' & x"00"; -- Volta para o loop principal

-- ============================================================
-- Incremento da centena de milhar
-- ============================================================
tmp(93) := LDA & '0' & x"0A"; -- Carrega MEM[10] = 0
tmp(94) := STA & '0' & x"00"; -- Zera a dezena de milhar
tmp(95) := LDA & '0' & x"05"; -- Carrega MEM[5]
tmp(96) := SOMA & '0' & x"0B"; -- Soma 1 na centena de milhar
tmp(97) := STA & '0' & x"05"; -- Armazena valor da soma no MEM[5]
tmp(98) := CEQ & '0' & x"0D"; -- Compara o valor de MEM[5] com 10
tmp(99) := JEQ & '0' & x"65"; -- Pula pra flag de inibir contagem
tmp(100) := RET & '0' & x"00"; -- Volta para o loop principal

-- ============================================================
-- Ativacao dos LEDS
-- ============================================================
tmp(101) := LDA & '0' & x"0B"; -- Carrega MEM[11] = 1
tmp(102) := STA & '0' & x"06"; -- Ativa Flag de inibir contagem
tmp(103) := STA & '1' & x"02"; -- Ativa LED9 de limite da contage
tmp(104) := RET & '0' & x"00"; -- Retorna para loop principal

-- ============================================================
-- REINICIALIZAR PLACA
-- ============================================================
tmp(120) := LDA & '0' & x"0A"; -- Carrega MEM[10] = 0
tmp(121) := STA & '0' & x"00"; -- Armazena 0 no MEM[0]
tmp(122) := STA & '0' & x"01"; -- Armazena 0 no MEM[1]
tmp(123) := STA & '0' & x"02"; -- Armazena 0 no MEM[2]
tmp(124) := STA & '0' & x"03"; -- Armazena 0 no MEM[3]
tmp(125) := STA & '0' & x"04"; -- Armazena 0 no MEM[4]
tmp(126) := STA & '0' & x"05"; -- Armazena 0 no MEM[5]
tmp(127) := STA & '0' & x"06"; -- Zerando Flag de inibir
tmp(128) := STA & '1' & x"01"; -- Zera o LED8 de Overflow
tmp(129) := STA & '1' & x"02"; -- Zera o LED9 de limite de contagem
tmp(130) := RET & '0' & x"00"; -- Retorna da sub-rotina do RESET_FPGA

-- ============================================================
-- ATUALIZA DISPLAYS COM OS VALORES CORRESPONDENTES
-- ============================================================
-- atualiza a unidade
-- ============================================================
tmp(141) := LDA & '0' & x"00"; -- Carrega MEM[0]
tmp(142) := STA & '1' & x"20"; -- Salva o valor de MEM[0] no HEX0     = MEM[288]

-- ============================================================
-- atualiza a dezena
-- ============================================================
tmp(143) := LDA & '0' & x"01"; -- Carrega MEM[1]
tmp(144) := STA & '1' & x"21"; -- Salva o valor de MEM[1] no HEX1 = MEM[289]
```

```
-- ================================================================
-- atualiza a centena
-- ================================================================
tmp(145) := LDA & '0' & x"02"; -- Carrega MEM[2]
tmp(146) := STA & '1' & x"22"; -- Salva o valor de MEM[2] no HEX2 = MEM[290]

-- ================================================================
-- atualiza a unidade de milhar
-- ================================================================
tmp(147) := LDA & '0' & x"03"; -- Carrega MEM[3]
tmp(148) := STA & '1' & x"23"; -- Salva o valor de MEM[3] no HEX3 = MEM[291]

-- ================================================================
-- atualiza a dezena de milhar
-- ================================================================
tmp(149) := LDA & '0' & x"04"; -- Carrega MEM[4]
tmp(150) := STA & '1' & x"24"; -- Salva o valor de MEM[4] no HEX4 = MEM[292]

-- ================================================================
-- atualiza a centena de milhar
-- ================================================================
tmp(151) := LDA & '0' & x"05"; -- Carrega MEM[5]
tmp(152) := STA & '1' & x"25"; -- Salva o valor de MEM[5] no HEX5 = MEM[293]
tmp(153) := RET & '0' & x"00"; -- Volta para o LOOP PRINCIPAL

-- ================================================================
-- CHECAGEM DO LIMITE DA CONTAGEM
-- ================================================================

--checando unidade
-- ================================================================
tmp(161) := LDA & '0' & x"14"; -- Carrega limite unidade = MEM[20]
tmp(162) := CEQ & '0' & x"00"; -- Compara o valor de MEM[12] com a unidade
tmp(163) := JEQ & '0' & x"A5"; -- Pula pra checar a unidade
tmp(164) := RET & '0' & x"00"; -- Retorna da sub-rotina da checagem

-- ================================================================
--checando centena
-- ================================================================
tmp(165) := LDA & '0' & x"15"; -- Carrega limite dezena = MEM[21]
tmp(166) := CEQ & '0' & x"01"; -- Compara o valor de MEM[12] com a dezena
tmp(167) := JEQ & '0' & x"A9"; -- Pula pra checar a dezena
tmp(168) := RET & '0' & x"00"; -- Retorna da sub-rotina da checagem

-- ================================================================
--checando centena
-- ================================================================
tmp(169) := LDA & '0' & x"16"; -- Carrega limite centena = MEM[22]
tmp(170) := CEQ & '0' & x"02"; -- Compara o valor de MEM[12] com a centena
```

```
tmp(171) := JEQ & '0' & x"AD"; -- Pula pra checar a centena
tmp(172) := RET & '0' & x"00"; -- Retorna da sub-rotina da checagem

-- ============================================================
--checando unidade de milhar
-- ============================================================
tmp(173) := LDA & '0' & x"17"; -- Carrega limite unidade de milhar = MEM[23]
tmp(174) := CEQ & '0' & x"03"; -- Compara o valor de MEM[12] com a unidade de milhar
tmp(175) := JEQ & '0' & x"B1"; -- Pula pra checar a unidade de milhar
tmp(176) := RET & '0' & x"00"; -- Retorna da sub-rotina da checagem

--============================================================
--checando dezena de milhar
--============================================================
tmp(177) := LDA & '0' & x"18"; -- Carrega limite dezena de milhar = MEM[24]
tmp(178) := CEQ & '0' & x"04"; -- Compara o valor de MEM[12] com a dezena de milhar
tmp(179) := JEQ & '0' & x"B5"; -- Pula pra checar a dezena de milhar
tmp(180) := RET & '0' & x"00"; -- Retorna da sub-rotina da checagem

--============================================================
--checando centena de milhar
--============================================================
tmp(181) := LDA & '0' & x"19"; -- Carrega limite centena de milhar = MEM[25]
tmp(182) := CEQ & '0' & x"05"; -- Compara o valor de MEM[12] com a centena de milhar
tmp(183) := JEQ & '0' & x"B9"; -- Pula pra checar a centena de milhar
tmp(184) := RET & '0' & x"00"; -- Retorna da sub-rotina da checagem

--============================================================
--checando flags
--============================================================
tmp(185) := LDA & '0' & x"0B"; -- Carrega limite MEM[11] = 1
tmp(186) := STA & '0' & x"06"; -- Inibe a contagem
tmp(187) := STA & '1' & x"01"; -- Ativa o LED9
tmp(188) := RET & '0' & x"00"; -- Retorna da sub-rotina da checagem

--============================================================
-- DEINICAO DE LIMITE
--============================================================      -- Definindo limite da
unidade
--============================================================
tmp(189) := STA & '1' & x"FE"; -- Zera a KEY1
tmp(190) := LDI & '0' & x"01"; -- Carrega 1 no imediato
tmp(191) := STA & '1' & x"00"; -- Ativa o LED0 atualizando algarismo da unidade

tmp(192) := LDA & '1' & x"61"; -- Verifica a KEY1
tmp(193) := CEQ & '0' & x"0A"; -- Compara o valor da KEY1 com 0
tmp(194) := LDA & '1' & x"40"; -- Pega o valor das SW(7 downto 0)

tmp(195) := JEQ & '0' & x"BE"; -- Espera o click da KEY1 para pular para dezena
```

```
tmp(196) := STA & '0' & x"14"; -- Armazena novo limite para unidade

--=============================================================
-- Definindo limite da dezena
--=============================================================
tmp(197) := STA & '1' & x"FE"; -- Zera a KEY1
tmp(198) := LDI & '0' & x"03"; -- Carrega 3 no imediato
tmp(199) := STA & '1' & x"00"; -- Ativa o LED 1 e 0 atualizando algarismo da dezena

tmp(200) := LDA & '1' & x"61"; -- Verifica a KEY1
tmp(201) := CEQ & '0' & x"0A"; -- Compara o valor da KEY1 com 0
tmp(202) := LDA & '1' & x"40"; -- Pega o valor das SW(7 downto 0)
tmp(203) := JEQ & '0' & x"C6"; -- Espera o click da KEY1 para pular para centena

tmp(204) := STA & '0' & x"15"; -- Armazena novo limite para dezena
--=============================================================
-- Definindo limite da centena
--=============================================================
tmp(205) := STA & '1' & x"FE"; -- Zera a KEY1
tmp(206) := LDI & '0' & x"07"; -- Carrega 7 no imediato
tmp(207) := STA & '1' & x"00"; -- Ativa o LED 2, 1 e 0 atualizando algarismo da centena

tmp(208) := LDA & '1' & x"61"; -- Verifica a KEY1
tmp(209) := CEQ & '0' & x"0A"; -- Compara o valor da KEY1 com 0
tmp(210) := LDA & '1' & x"40"; -- Pega o valor das SW(7 downto 0)
tmp(211) := JEQ & '0' & x"CE"; -- Espera o click da KEY1 para pular para a unidade de milhar
tmp(212) := STA & '0' & x"16"; -- Armazena novo limite para centena

--=============================================================   -- Definindo limite da
unidade de milhar
--=============================================================
tmp(213) := STA & '1' & x"FE"; -- Zera a KEY1
tmp(214) := LDI & '0' & x"0F"; -- Carrega 15 no imediato
tmp(215) := STA & '1' & x"00"; -- Ativa o LED 3, 2, 1 e 0 atualizando algarismo da unidade de milhar

tmp(216) := LDA & '1' & x"61"; -- Verifica a KEY1
tmp(217) := CEQ & '0' & x"0A"; -- Compara o valor da KEY1 com 0
tmp(218) := LDA & '1' & x"40"; -- Pega o valor das SW(7 downto 0)
tmp(219) := JEQ & '0' & x"D6"; -- Espera o click da KEY1 para pular para a dezena de milhar
tmp(220) := STA & '0' & x"17"; -- Armazena novo limite para unidade de milhar

--=============================================================
-- Definindo limite da dezena de milhar
--=============================================================   tmp(221) := STA & '1' &
x"FE"; -- Zera a KEY1
tmp(222) := LDI & '0' & x"1F"; -- Carrega 31 no imediato
tmp(223) := STA & '1' & x"00"; -- Ativa o LED 4, 3, 2, 1 e 0 atualizando algarismo da dezena de milhar

tmp(224) := LDA & '1' & x"61"; -- Verifica a KEY1
tmp(225) := CEQ & '0' & x"0A"; -- Compara o valor da KEY1 com 0
```

```
tmp(226) := LDA & '1' & x"40"; -- Pega o valor das SW(7 downto 0)
tmp(227) := JEQ & '0' & x"DE"; -- Espera o click da KEY1 para pular para a centena de milhar
tmp(228) := STA & '0' & x"18"; -- Armazena novo limite para dezena de milhar

--============================================================-- Definindo limite da
centena de milhar
--============================================================
tmp(229) := STA & '1' & x"FE"; -- Zera a KEY1
tmp(230) := LDI & '0' & x"3F"; -- Carrega 63 no imediato
tmp(231) := STA & '1' & x"00"; -- Ativa o LED 5, 4, 3, 2, 1 e 0 atualizando algarismo da centena de milhar

tmp(232) := LDA & '1' & x"61"; -- Verifica a KEY1
tmp(233) := CEQ & '0' & x"0A"; -- Compara o valor da KEY1 com 0
tmp(234) := LDA & '1' & x"40"; -- Pega o valor das SW(7 downto 0)
tmp(235) := JEQ & '0' & x"E6"; -- Pula para Zerar KEY1
tmp(236) := STA & '0' & x"19"; -- Armazena novo limite para centena de milhar

tmp(237) := STA & '1' & x"FE"; -- Zerando KEY1
tmp(238) := LDA & '0' & x"0A"; -- Carrega 0 no imediato
tmp(239) := STA & '1' & x"00"; -- Desativa todos os LEDS
tmp(240) := RET & '0' & x"00"; -- Voltando ao loop principal

        return tmp;
        end initMemory;

        signal memROM : blocoMemoria := initMemory;

begin
        Dado <= memROM (to_integer(unsigned(Endereco)));
end architecture;
```