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 2 arquiteturas, e a escolhida foi a arquitetura baseada em registradores e memória. Nesta arquitetura, as operações ocorrem entre um registrador e uma posição de memória ou valor imediato, utilizando um banco de registradores. Essa arquitetura foi a escolhida pois nessa arquitetura é possível fazer as operações desejadas com menos comandos, uma vez que há possibilidade de utilizar diversos registradores como acumuladores, e não apenas um.

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**	**HFI**	**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 4 bits para o OpCode, 2 bits para o endereço dos registradores, e 9 bits para o imediato respetivamente, elas são utilizadas da seguinte forma:

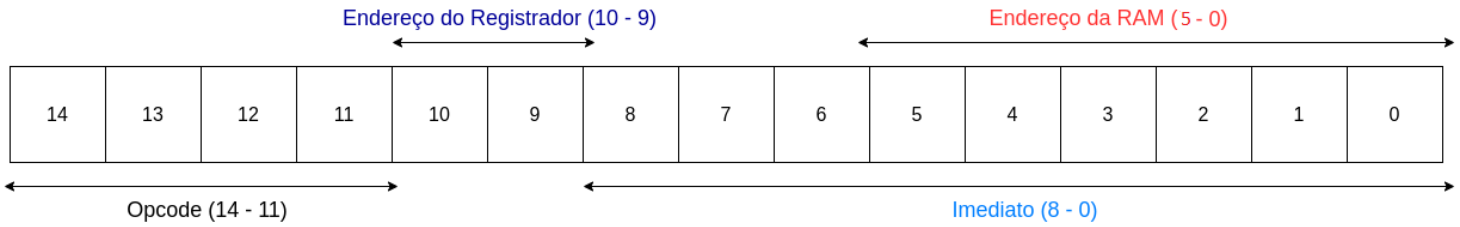

Imagem 1: Formato das instruções

Cada uma das instruções da ROM é representada por um comando ***tmp*** o qual representa as linhas de código do assembly, os comandos ***tmp*** são compostos pelos bits abaixo:

tmp(X) = OPCODE & END_REG & AM & x"DEST"

No qual **X** é a linha da instrução, ***OPCODE*** é uma instrução da tabela de instruções, **AM** é um bit que indica se a instrução irá armazenar dado na memória e

DEST é o destino da instrução em hexadecimal, podendo ser um endereço de memória, ou um valor de entrada, dependendo da instrução, e ***END_REG*** sendo os 2 bits reservados para o endereço dos registradores.

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.

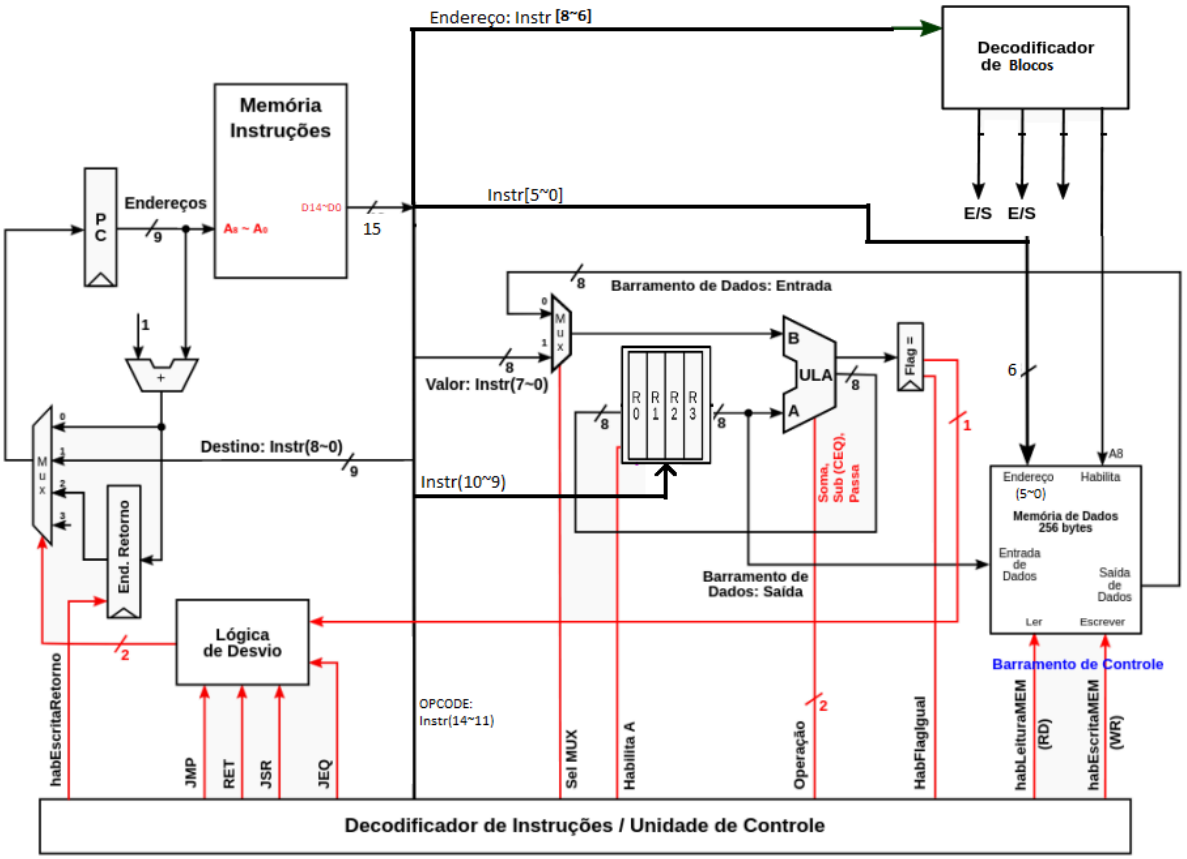

Imagem 2: Fluxo de Dados

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 (WR) na memória, 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.

HabFlagIgual: 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.

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

RET: Sua função é identificar um retorno 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 instrução da ROM ou o valor de saída da RAM será o valor na entrada B da 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, com pequenas alterações, como a adição das bases de tempo. Isto pode ser visto no diagrama abaixo:

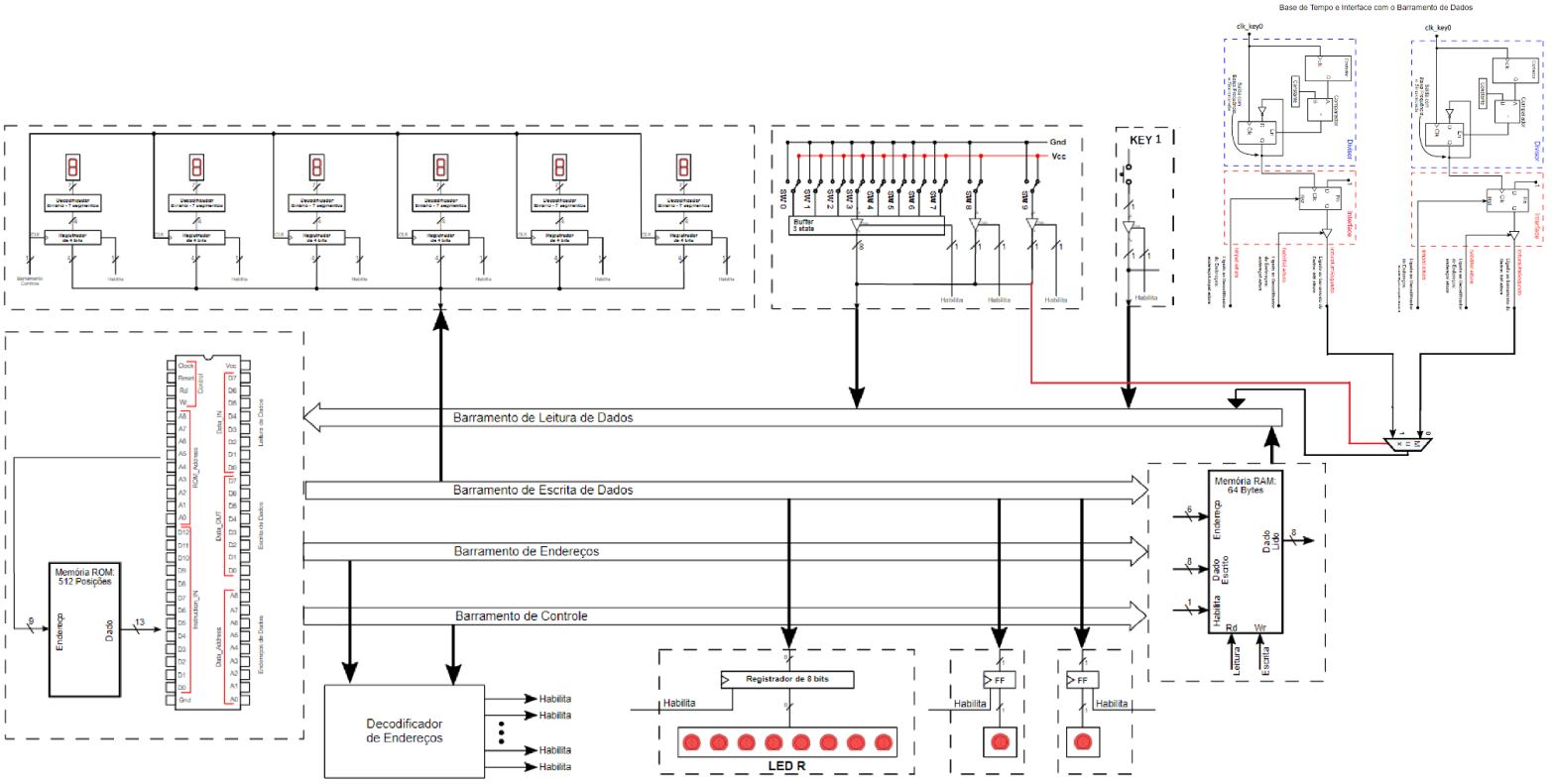

Imagem 3: Conexão com periféricos

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, porém nessa versão final ela é automática por conta da base de tempo inserida no projeto, não sendo mais utilizada.
- Key 1 - Sua função é entrar no modo de definição de horário, como acontece num relógio usual. Esse horário é definido por um valor de binário que vem das switches [0 a 7], se o relógio atingir 24 horas, o led de limite é acesso. Essa definição é feita em cada algarismo, ou seja, na dezena de minutos, unidade de segundo 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 horário de cada algarismo de tempo.
- SW(9) - Controla o seletor entre as duas bases de tempo, podendo contar normalmente de segundo em segundo ou mais rapidamente.
- LEDR(5 downto 0) - Esses LEDS devem indicar em qual algarismo está sendo definido o horário, ele acende de forma cumulativa, sendo que na definição da unidade de segundo apenas o LEDR0 acende, ao passo em que na definição da dezena de segundo, o LEDR0 e o LEDR1 acendem, e assim por diante até a definição da dezena de hora.

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";

       constant R0 : std_logic_vector(1 downto 0) := "00";
       constant R1 : std_logic_vector(1 downto 0) := "01";
       constant R2 : std_logic_vector(1 downto 0) := "10";
       constant R3 : std_logic_vector(1 downto 0) := "11";

  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[14] = 6
-- MEM[15] = 4
-- MEM[16] = 2

-- 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 & R2 & '0' & x"00"; -- Carregando 0 no R2
        tmp(1) := STA & R2 & '1' & x"20"; -- DISPLAY 0 MEM[288]
        tmp(2) := STA & R2 & '1' & x"21"; -- DISPLAY 1 MEM[289]
        tmp(3) := STA & R2 & '1' & x"22"; -- DISPLAY 2 MEM[290]
        tmp(4) := STA & R2 & '1' & x"23"; -- DISPLAY 3 MEM[291]
        tmp(5) := STA & R2 & '1' & x"24"; -- DISPLAY 4 MEM[292]
        tmp(6) := STA & R2 & '1' & x"25"; -- DISPLAY 5 MEM[293]



-- ========================================================================
-- Apagando os LEDS
-- ========================================================================
        tmp(7) := STA & R2 & '1' & x"00"; -- Apagando LEDS 0 downto 7 MEM[256]
        tmp(8) := STA & R2 & '1' & x"01"; -- Apagando LED8 MEM[257]
        tmp(9) := STA & R2 & '1' & x"02"; -- Apagando LED9 MEM[258]



-- ========================================================================
-- Inicializando variaveis dos displays
-- ========================================================================
        tmp(10) := STA & R2 & '0' & x"00"; -- Armazenando 0 no MEM[0] (UNIDADE)
        tmp(11) := STA & R2 & '0' & x"01"; -- Armazenando 0 no MEM[1] (DEZENA)
        tmp(12) := STA & R2 & '0' & x"02"; -- Armazenando 0 no MEM[2] (CENTENA)
        tmp(13) := STA & R2 & '0' & x"03"; -- Armazenando 0 no MEM[3] (UNIDADE DE MILHAR)
        tmp(14) := STA & R2 & '0' & x"04"; -- Armazenando 0 no MEM[4] (DEZENA DE MILHAR)
        tmp(15) := STA & R2 & '0' & x"05"; -- Armazenando 0 no MEM[5] (CENTENA DE MILHAR)



-- ========================================================================
```

```
-- Zerando flag de inibir contagem
-- ======================================================================
        tmp(16) := STA & R2 & '0' & x"06"; -- Armazenando 0 no MEM[6] (ZERA FLAG DE
INIBIR)

-- ======================================================================
-- Armazenando Variaveis
-- ======================================================================
        tmp(17) := LDI & R0 & '0' & x"00"; -- Carregando 0 no R0
        tmp(18) := STA & R0 & '0' & x"0A"; -- Armazenando incremento no MEM[10]

        tmp(19) := LDI & R0 & '0' & x"01"; -- Carregando 1 no R0
        tmp(20) := STA & R0 & '0' & x"0B"; -- Armazenando incremento no MEM[11]

        tmp(21) := LDI & R0 & '0' & x"0A"; -- Carregando 10 no R0
        tmp(22) := STA & R0 & '0' & x"0D"; -- Valor de comparacao armazenado no MEM[13]

        tmp(23) := LDI & R0 & '0' & x"09"; -- Carregando 9 no R0
        tmp(24) := STA & R0 & '0' & x"0C"; -- Valor de comparacao no MEM[12]
        tmp(25) := STA & R0 & '0' & x"14"; -- Armazenando limite da unidade no MEM[20]
        tmp(26) := STA & R0 & '0' & x"16"; -- Armazenando limite da dezena no MEM[21]
        tmp(27) := STA & R0 & '0' & x"15"; -- Armazenando limite da centena no MEM[22]
        tmp(28) := STA & R0 & '0' & x"17"; -- Armazenando limite da unidade de milhar no
MEM[23]
        tmp(29) := STA & R0 & '0' & x"18"; -- Armazenando limite da dezena de milhar no
MEM[24]
        tmp(30) := STA & R0 & '0' & x"19"; -- Armazenando limite da centena de milhar no
MEM[25]


-- ======================================================================
-- Armazenando limite de contagem
-- ======================================================================

        tmp(31) := LDI & R0 & '0' & x"06"; -- Carregando 6 no R0
        tmp(32) := STA & R0 & '0' & x"0E"; -- Armazenando 6 no MEM[14]
        tmp(33) := LDI & R0 & '0' & x"04"; -- Carregando 4 no R0
        tmp(34) := STA & R0 & '0' & x"0F"; -- Armazenando 4 no MEM[15]
        tmp(35) := LDI & R0 & '0' & x"02"; -- Carregando 2 no R0
        tmp(36) := STA & R0 & '0' & x"10"; -- Armazenando 2 no MEM[16]
        tmp(37) := NOP & R0 & '0' & x"00"; -- Pausa para sequencia de temps


-- ======================================================================
-- LOOP DOS BOTOES (CHECA KEY0, KEY1 E O FPGA RESET)
-- ======================================================================

        -- ======================================================================
        -- Checando flag de inibir contagem
        -- ======================================================================
                tmp(38) := NOP & R0 & '0' & x"00"; -- Pausa para sequencia de temps
                tmp(39) := LDA & R0 & '0' & x"06"; -- Olha pro valor da flag de inibir
                tmp(40) := CEQ & R0 & '0' & x"0B"; -- Compara o valor da flag com 1
                tmp(41) := JEQ & R0 & '0' & x"2E"; -- Pula quando a flag esta zerada
ignorando a KEY0

        -- ======================================================================
        --OBSERVANDO A KEY0
        -- ======================================================================

                tmp(42) := LDA & R0 & '1' & x"60"; -- Observa o valor da KEY0
                tmp(43) := CEQ & R0 & '0' & x"0A"; -- Verifica se a KEY0 foi ativada
                tmp(44) := JEQ & R0 & '0' & x"2E"; -- Caso a KEY0 for igual a 1, pula
para o incremento
                tmp(45) := JSR & R0 & '0' & x"3A"; -- Caso a KEY0 for igual a 1, pula
para o incremento
```

```
        -- ========================================================================
        --OBSERVANDO A KEY1
        -- ========================================================================

                tmp(46) := LDA & R0 & '1' & x"61"; -- Observa o valor da KEY1
                tmp(47) := CEQ & R0 & '0' & x"0A"; -- Verifica se a KEY1 foi ativada

                tmp(48) := JEQ & R0 & '0' & x"33"; -- Caso a KEY1 for igual a 1, pula
para a definicao do limite
                tmp(49) := JSR & R0 & '0' & x"BD"; -- Caso a KEY1 for igual a 1, pula
para o limite
                tmp(50) := NOP & R0 & '0' & x"00"; -- Pausa para sequencia de temps

        -- ========================================================================
        --OBSERVANDO A FPGA_RESET
        -- ========================================================================

                tmp(52) := LDA & R0 & '1' & x"64"; -- Observa o valor do FPGA RESET
                tmp(53) := CEQ & R0 & '0' & x"0B"; -- Compara o valor do FPGA RESET com 1

                tmp(54) := JEQ & R0 & '0' & x"38"; -- Caso a FPGA RESET for igual a 1,
pula para a reset
                tmp(55) := JSR & R0 & '0' & x"82"; -- Caso a FPGA RESET for igual a 1,
pula para o resetar a placa
                tmp(56) := JSR & R0 & '0' & x"8D"; -- Pula para atualizar os DISPLAYS
                tmp(57) := JMP & R0 & '0' & x"26"; -- Volta para o loop principal

-- ========================================================================
-- LOOP DE INCREMENTO (KEY0 SUB-ROTINA)
-- ========================================================================
                tmp(58) := STA & R0 & '1' & x"FF"; -- Zera KEY0

        -- ========================================================================
        -- Incremento da unidade
        -- ========================================================================
                tmp(59) := LDA & R0 & '0' & x"00"; -- Carrega MEM[0]
                tmp(60) := SOMA & R0 & '0' & x"0B"; -- Soma 1 na unidade
                tmp(61) := STA & R0 & '0' & x"00"; -- Armazena valor da soma no MEM[0]
                tmp(62) := CEQ & R0 & '0' & x"0D"; -- Compara o valor de MEM[0] com 10
                tmp(63) := JEQ & R0 & '0' & x"41"; -- Pula pra checar a dezena
                tmp(64) := RET & R0 & '0' & x"00"; -- Volta para o loop principal


        -- ========================================================================
        -- Incremento da dezena
        -- ========================================================================
                tmp(65) := LDA & R0 & '0' & x"0A"; -- Carrega MEM[10] = 0
                tmp(66) := STA & R0 & '0' & x"00"; -- Zera a unidade

                tmp(67) := LDA & R0 & '0' & x"01"; -- Carrega MEM[1]
                tmp(68) := SOMA & R0 & '0' & x"0B"; -- Soma 1 na dezena
                tmp(69) := STA & R0 & '0' & x"01"; -- Armazena valor da soma no MEM[1]
                tmp(70) := CEQ & R0 & '0' & x"0E"; -- Compara o valor de MEM[1] com 6
                tmp(71) := JEQ & R0 & '0' & x"49"; -- Pula pra checar a centena
                tmp(72) := RET & R0 & '0' & x"00"; -- Volta para o loop principal


        -- ========================================================================
        -- Incremento da centena
        -- ========================================================================
                tmp(73) := LDA & R0 & '0' & x"0A"; -- Carrega MEM[10] = 0
                tmp(74) := STA & R0 & '0' & x"01"; -- Zera a dezena

                tmp(75) := LDA & R0 & '0' & x"02"; -- Carrega MEM[2]
                tmp(76) := SOMA & R0 & '0' & x"0B"; -- Soma 1 na centena
                tmp(77) := STA & R0 & '0' & x"02"; -- Armazena valor da soma no MEM[2]
                tmp(78) := CEQ & R0 & '0' & x"0D"; -- Compara o valor de MEM[2] com 10
```

```
            tmp(79) := JEQ & R0 & '0' & x"51"; -- Pula pra checar a unidade de milhar
            tmp(80) := RET & R0 & '0' & x"00"; -- Volta para o loop principal


        -- ================================================================================
        -- Incremento da unidade de milhar
        -- ================================================================================
            tmp(81) := LDA & R0 & '0' & x"0A"; -- Carrega MEM[10] = 0
            tmp(82) := STA & R0 & '0' & x"02"; -- Zera a centena

            tmp(83) := LDA & R0 & '0' & x"03"; -- Carrega MEM[3]
            tmp(84) := SOMA & R0 & '0' & x"0B"; -- Soma 1 na unidade de milhar
            tmp(85) := STA & R0 & '0' & x"03"; -- Armazena valor da soma no MEM[3]
            tmp(86) := CEQ & R0 & '0' & x"0E"; -- Compara o valor de MEM[3] com 6
            tmp(87) := JEQ & R0 & '0' & x"59"; -- Pula pra checar dezena de milhar
            tmp(88) := RET & R0 & '0' & x"00"; -- Volta para o loop principal


        -- ================================================================================
        -- Incremento da dezena de milhar e centena de milhar
        -- ================================================================================
            tmp(89) := LDA & R0 & '0' & x"0A"; -- Carrega MEM[10] = 0
            tmp(90) := STA & R0 & '0' & x"03"; -- Zera a unidade de milhar

            tmp(91) := LDA & R0 & '0' & x"04"; -- Carrega MEM[4]
            tmp(92) := SOMA & R0 & '0' & x"0B"; -- Soma 1 na dezena de milhar
            tmp(93) := STA & R0 & '0' & x"04"; -- Armazena valor da soma no MEM[4]
            tmp(94) := CEQ & R0 & '0' & x"0F"; -- Compara o valor de MEM[4] com 4
            tmp(95) := JEQ & R0 & '0' & x"6A"; -- Pula pra checar a centena de milhar

            tmp(96) := LDA & R0 & '0' & x"04"; -- Carrega MEM[4]
            tmp(97) := CEQ & R0 & '0' & x"0D"; -- Compara o valor de MEM[4] com 10
            tmp(98) := JEQ & R0 & '0' & x"64"; -- Pula pra checar a centena de milhar
            tmp(99) := RET & R0 & '0' & x"00"; -- Volta para o loop principal

            tmp(100) := LDA & R0 & '0' & x"0A"; -- Carrega 0 no R0
            tmp(101) := STA & R0 & '0' & x"04"; -- Armazena o valor no MEM[4]
            tmp(102) := LDA & R0 & '0' & x"05"; --       Carrega MEM[5]
            tmp(103) := SOMA & R0 & '0' & x"0B"; -- Soma 1 na centena de milhar
            tmp(104) := STA & R0 & '0' & x"05"; -- Armazena o valor da soma no MEM[5]
            tmp(105) := RET & R0 & '0' & x"00"; -- Volta para o loop principal

            tmp(106) := LDA & R0 & '0' & x"05"; -- Carrega MEM[5]
            tmp(107) := CEQ & R0 & '0' & x"10"; -- Compara o valor de MEM[5] com 2
            tmp(108) := JEQ & R0 & '0' & x"6E"; -- Pula pra ativar flags
            tmp(109) := RET & R0 & '0' & x"00"; -- Volta para o loop principal

        -- ================================================================================
        -- Zerando displays quando 24 horas
        -- ================================================================================
            tmp(110) := JMP & R0 & '0' & x"82"; -- Zerar horario quando bater 24
horas
            tmp(111) := RET & R0 & '0' & x"00"; -- Retorna para loop principal


-- ================================================================================
-- REINICIALIZAR PLACA
-- ================================================================================
        tmp(130) := LDA & R1 & '0' & x"0A"; -- Carrega MEM[10] = 0
        tmp(131) := STA & R1 & '0' & x"00"; -- Armazena 0 no MEM[0]
        tmp(132) := STA & R1 & '0' & x"01"; -- Armazena 0 no MEM[1]
        tmp(133) := STA & R1 & '0' & x"02"; -- Armazena 0 no MEM[2]
        tmp(134) := STA & R1 & '0' & x"03"; -- Armazena 0 no MEM[3]
        tmp(135) := STA & R1 & '0' & x"04"; -- Armazena 0 no MEM[4]
        tmp(136) := STA & R1 & '0' & x"05"; -- Armazena 0 no MEM[5]

        tmp(137) := STA & R1 & '0' & x"06"; -- Zerando Flag de inibir
```

```
        tmp(138) := STA & R1 & '1' & x"01"; -- Zera o LED8 de Overflow
        tmp(139) := STA & R1 & '1' & x"02"; -- Zera o LED9 de limite de contagem
        tmp(140) := RET & R1 & '0' & x"00"; -- Retorna da sub-rotina do RESET_FPGA


-- ========================================================================
-- ATUALIZA DISPLAYS COM OS VALORES CORRESPONDENTES
-- ========================================================================

        -- ================================================================
        -- atualiza a unidade
        -- ================================================================
                tmp(141) := LDA & R2 & '0' & x"00"; -- Carrega MEM[0]
                tmp(142) := STA & R2 & '1' & x"20"; -- Salva o valor de MEM[0] no HEX0 =
MEM[288]

        -- ================================================================
        -- atualiza a dezena
        -- ================================================================
                tmp(143) := LDA & R2 & '0' & x"01"; -- Carrega MEM[1]
                tmp(144) := STA & R2 & '1' & x"21"; -- Salva o valor de MEM[1] no HEX1 =
MEM[289]

        -- ================================================================
        -- atualiza a centena
        -- ================================================================
                tmp(145) := LDA & R2 & '0' & x"02"; -- Carrega MEM[2]
                tmp(146) := STA & R2 & '1' & x"22"; -- Salva o valor de MEM[2] no HEX2 =
MEM[290]

        -- ================================================================
        -- atualiza a unidade de milhar
        -- ================================================================
                tmp(147) := LDA & R2 & '0' & x"03"; -- Carrega MEM[3]
                tmp(148) := STA & R2 & '1' & x"23"; -- Salva o valor de MEM[3] no HEX3 =
MEM[291]

        -- ================================================================
        -- atualiza a dezena de milhar
        -- ================================================================
                tmp(149) := LDA & R2 & '0' & x"04"; -- Carrega MEM[4]
                tmp(150) := STA & R2 & '1' & x"24"; -- Salva o valor de MEM[4] no HEX4 =
MEM[292]

        -- ================================================================
        -- atualiza a centena de milhar
        -- ================================================================
                tmp(151) := LDA & R2 & '0' & x"05"; -- Carrega MEM[5]
                tmp(152) := STA & R2 & '1' & x"25"; -- Salva o valor de MEM[5] no HEX5 =
MEM[293]

                tmp(153) := RET & R0 & '0' & x"00"; -- Volta para o LOOP PRINCIPAL


-- ========================================================================
-- DEINICAO DE LIMITE
-- ========================================================================

        -- ================================================================
        -- Definindo limite da unidade
        -- ================================================================
                tmp(189) := STA & R0 & '1' & x"FE"; -- Zera a KEY1
                tmp(190) := LDI & R3 & '0' & x"01"; -- Carrega 1 no imediato de R3
                tmp(191) := STA & R3 & '1' & x"00"; -- Ativa o LED0 atualizando algarismo
da unidade de segundo
```

```
                tmp(192) := LDA & R0 & '1' & x"61"; -- Verifica a KEY1
                tmp(193) := CEQ & R0 & '0' & x"0A"; -- Compara o valor da KEY1 com 0
                tmp(194) := LDA & R1 & '1' & x"40"; -- Pega o valor das SW(7 downto 0)
                tmp(195) := JEQ & R0 & '0' & x"BE"; -- Espera o click da KEY1 para pular
para unidade de segundo
                tmp(196) := STA & R1 & '0' & x"00"; -- Armazena novo horario para unidade
de segundo


        -- ========================================================================
        -- Definindo limite da dezena
        -- ========================================================================
                tmp(197) := STA & R0 & '1' & x"FE"; -- Zera a KEY1
                tmp(198) := LDI & R3 & '0' & x"03"; -- Carrega 3 no imediato de R3
                tmp(199) := STA & R3 & '1' & x"00"; -- Ativa o LED 1 e 0 atualizando
algarismo dezena de segundo

                tmp(200) := LDA & R0 & '1' & x"61"; -- Verifica a KEY1
                tmp(201) := CEQ & R0 & '0' & x"0A"; -- Compara o valor da KEY1 com 0
                tmp(202) := LDA & R1 & '1' & x"40"; -- Pega o valor das SW(7 downto 0)
                tmp(203) := JEQ & R0 & '0' & x"C6"; -- Espera o click da KEY1 para pular
para dezena de segundo
                tmp(204) := STA & R1 & '0' & x"01"; -- Armazena novo horario para dezena
de segundo


        -- ========================================================================
        -- Definindo limite da centena
        -- ========================================================================
                tmp(205) := STA & R0 & '1' & x"FE"; -- Zera a KEY1
                tmp(206) := LDI & R3 & '0' & x"07"; -- Carrega 7 no imediato de R3
                tmp(207) := STA & R3 & '1' & x"00"; -- Ativa o LED 2, 1 e 0 atualizando
unidade de minuto

                tmp(208) := LDA & R0 & '1' & x"61"; -- Verifica a KEY1
                tmp(209) := CEQ & R0 & '0' & x"0A"; -- Compara o valor da KEY1 com 0
                tmp(210) := LDA & R1 & '1' & x"40"; -- Pega o valor das SW(7 downto 0)
                tmp(211) := JEQ & R0 & '0' & x"CE"; -- Espera o click da KEY1 para pular
para a unidade de minuto
                tmp(212) := STA & R1 & '0' & x"02"; -- Armazena novo horario para centena


        -- ========================================================================
        -- Definindo limite da unidade de milhar
        -- ========================================================================
                tmp(213) := STA & R0 & '1' & x"FE"; -- Zera a KEY1
                tmp(214) := LDI & R3 & '0' & x"0F"; -- Carrega 15 no imediato de R3
                tmp(215) := STA & R3 & '1' & x"00"; -- Ativa o LED 3, 2, 1 e 0
atualizando a dezena de minuto

                tmp(216) := LDA & R0 & '1' & x"61"; -- Verifica a KEY1
                tmp(217) := CEQ & R0 & '0' & x"0A"; -- Compara o valor da KEY1 com 0
                tmp(218) := LDA & R1 & '1' & x"40"; -- Pega o valor das SW(7 downto 0)
                tmp(219) := JEQ & R0 & '0' & x"D6"; -- Espera o click da KEY1 para pular
para a dezena de minuto
                tmp(220) := STA & R1 & '0' & x"03"; -- Armazena novo horario para dezena
de minuto


        -- ========================================================================
        -- Definindo limite da dezena de milhar
        -- ========================================================================
                tmp(221) := STA & R0 & '1' & x"FE"; -- Zera a KEY1
                tmp(222) := LDI & R3 & '0' & x"1F"; -- Carrega 31 no imediato de R3
                tmp(223) := STA & R3 & '1' & x"00"; -- Ativa o LED 4, 3, 2, 1 e 0
atualizando a unidade de hora
```

```
                tmp(224) := LDA & R0 & '1' & x"61"; -- Verifica a KEY1
                tmp(225) := CEQ & R0 & '0' & x"0A"; -- Compara o valor da KEY1 com 0
                tmp(226) := LDA & R1 & '1' & x"40"; -- Pega o valor das SW(7 downto 0)
                tmp(227) := JEQ & R0 & '0' & x"DE"; -- Espera o click da KEY1 para pular
para a unidade de hora
                tmp(228) := STA & R1 & '0' & x"04"; -- Armazena novo horario para unidade
de hora


        -- ========================================================================
        -- Definindo limite da centena de milhar
        -- ========================================================================
                tmp(229) := STA & R0 & '1' & x"FE"; -- Zera a KEY1
                tmp(230) := LDI & R3 & '0' & x"3F"; -- Carrega 63 no imediato de R3
                tmp(231) := STA & R3 & '1' & x"00"; -- Ativa o LED 5, 4, 3, 2, 1 e 0
atualizando a dezena de hora

                tmp(232) := LDA & R0 & '1' & x"61"; -- Verifica a KEY1
                tmp(233) := CEQ & R0 & '0' & x"0A"; -- Compara o valor da KEY1 com 0
                tmp(234) := LDA & R1 & '1' & x"40"; -- Pega o valor das SW(7 downto 0)
                tmp(235) := JEQ & R0 & '0' & x"E6"; -- Pula para Zerar KEY1
                tmp(236) := STA & R1 & '0' & x"05"; -- Armazena novo horario para dezena
de hora

                tmp(237) := STA & R0 & '1' & x"FE"; -- Zerando KEY1
                tmp(238) := LDA & R0 & '0' & x"0A"; -- Carrega 0 no imediato de R0
                tmp(239) := STA & R0 & '1' & x"00"; -- Desativa todos os LEDS
                tmp(240) := RET & R0 & '0' & x"00"; -- Voltando ao loop principal

         return tmp;
    end initMemory;

    signal memROM : blocoMemoria := initMemory;

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