Insper

Relatório de atividade da disciplina Design de Computadores

Projeto 1: Relógio Utilizando um Processador Personalizado

Lais Nascimento da Silva

William Augusto Reis da Silva

João Guilherme Cintra de Freitas Almeida

Professor Paulo Carlos Santos

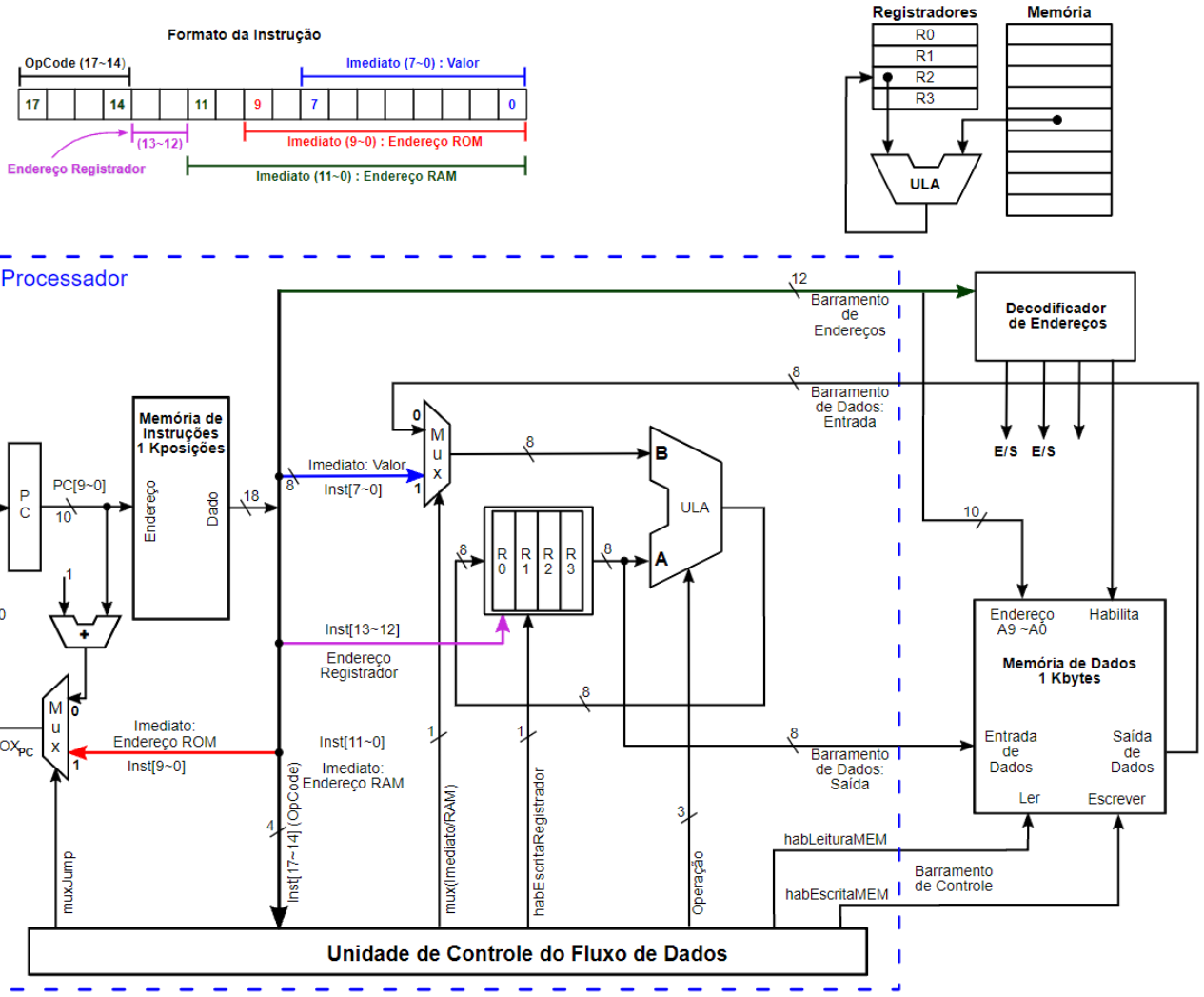
São Paulo
Maio/2020

1. Arquitetura do Processador

Para fazer a escolha da arquitetura do processador para o projeto, o grupo levou em consideração que até o início do projeto, a arquitetura da CPU estava baseada em Acumulador. Após o início do projeto, foi mais conveniente alterar a arquitetura do processador para Registrador-Memoria, devido ao fato de a alteração parecer simples.

Para se ter uma visão melhor, a arquitetura baseada em registrador e memória, o processador tem uma quantidade de registradores maior que 1 e as operações são entre um deles e uma posição de memória.

A imagem, que também se encontra na Aula 6, ilustrando essa arquitetura, se encontra abaixo:



Nossa implementação tem diferença no formato da instrução, que será explorado um pouco mais abaixo neste relatório, porém a ideia é semelhante, tendo em vista que se terá um banco de registradores (na imagem: R0, R1, R2, R3), onde se chega o endereço do registrador, seta com a linha roxa, a ser utilizado e sua saída entra na ULA e no Barramento de Dados. Para ilustrar novamente a pequena diferença, na imagem acima o *Endereço Registrador* são os bits 13 e 12 da instrução, entretando no nosso são os bits 10 e 9.

2. Total de Instruções e sua sintaxe

As instruções com sua sintaxe, mnemônico descritos foram feitos perto da Aula 5 e o grupo chegou ao final com as instruções descritas como na Tabela 1. A Tabela 2 refere-se à lógica de desvio, com o JMP, RET, JSR, JEQ, Flag= e SelMUX.

Tabela 1 - Instruções e Pontos de Controle

Instrução	Mnemônico	Código Binário	Hab Escrita Retorno	JMP	RET	JSR	JEQ	Sel MUX	Hab_A	Operação	habFlag=	RD	WR
Sem operação	NOP	0000	0	0	0	0	0	X	0	XX	0	0	0
Carrega valor da memória para A	LDA	0001	0	0	0	0	0	0	1	10	0	1	0
Soma A e B e armazena em A	SOMA	0010	0	0	0	0	0	0	1	01	0	1	0
Subtrai B de A e armazena em A	SUB	0011	0	0	0	0	0	0	1	00	0	1	0
Carrega valor imediato para A	LDI	0100	0	0	0	0	0	1	1	10	0	0	0
Salva valor de A para a memória	STA	0101	0	0	0	0	0	0	0	XX	0	0	1
Desvio de execução	JMP	0110	0	1	0	0	0	X	0	XX	0	0	0
Desvio condicional de execução	JEQ	0111	0	0	0	0	1	X	0	XX	0	0	0
Comparação	CEQ	1000	0	0	0	0	0	0	1	00	1	1	0
Chamada de Sub Rotina	JSR	1001	1	0	0	1	0	0	0	XX	0	0	0
Retorno de Sub Rotina	RET	1010	0	0	1	0	0	0	0	XX	0	0	0

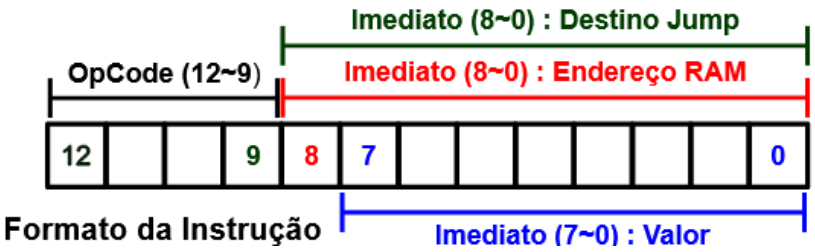
Tabela 2 - Lógica de Desvio (Próxima Instrução)

JMP	RET	JSR	JEQ	Flag de Igual	Seleção do Mux	Ação Resultante
0	0	0	0	X	00	Prox. Instrução
1	0	0	0	X	01	Desvio de JMP
0	0	0	1	0	00	Prox. Inst. JEQ
0	0	0	1	1	01	Desvio de JEQ
0	0	1	0	X	01	Desvio de JSR
0	1	0	0	X	10	Desvio de RET

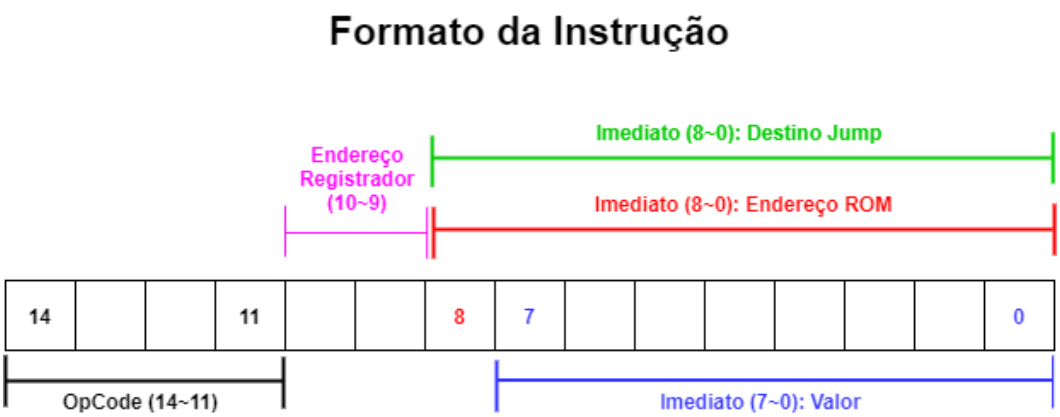
Vale ressaltar que onde se encontra X ou XX, é porque o valor naquela posição é irrelevante para a proposta da instrução, podendo ter qualquer combinação possível. No código do projeto, foi usado o valor 0 onde se tem o X.

3. Formato das Instruções

O formato das instruções já se alterou conforme foi se tendo o desenvolvimento do projeto, tendo em vista as modificações que iríamos fazendo conforme necessário. Por exemplo, a última era como na imagem abaixo:

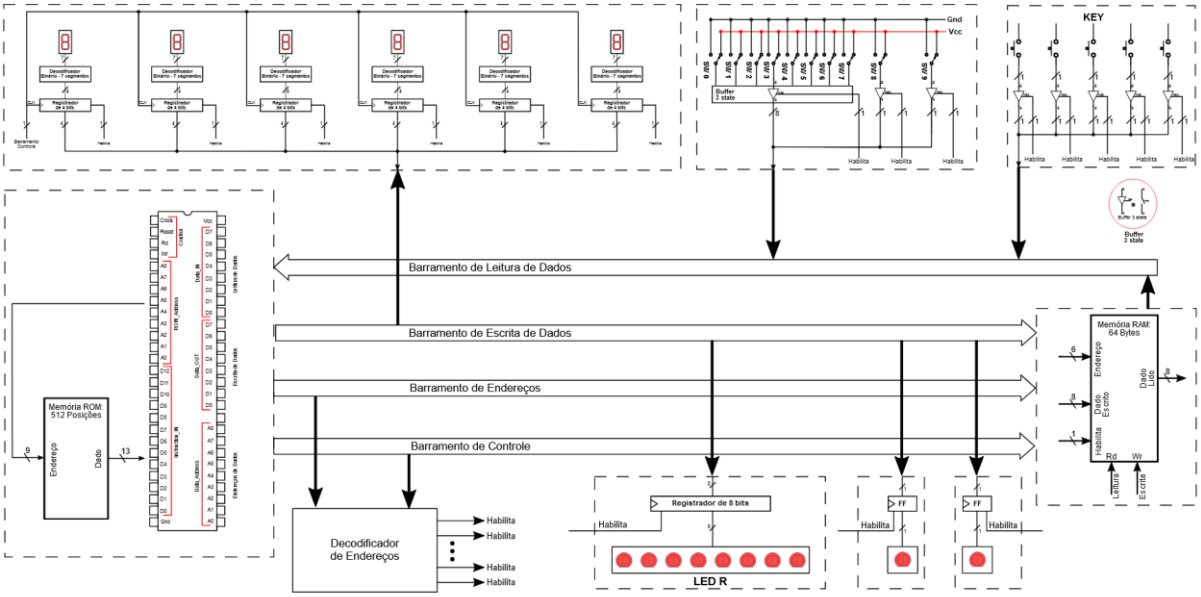


Como é possível observar, os bits 12~9 (12 downto 9 no código) eram reservados ao OpCode. Esse exemplo pode explicar como funciona os outros também. Porém, ao mudar a arquitetura para Registrador-Memória, foi necessário incluir dois bits a mais, por conta da seleção do registrador utilizado, tendo em vista que essa arquitetura possui um banco de 1 ou mais registradores e o grupo escolheu 4. Assim, para fazer a seleção, são necessários dois bits, ficando como na imagem abaixo:



4. Fluxo de dados para o processador

No processador entram instruções que são processadas. A partir disso, saem os controles de cada componente do projeto, além disso podem ir dados para serem escritos na RAM em um determinado endereço, que também sai do processador para a RAM. Ademais, podem sair dados da RAM que entram no processador. A imagem abaixo representa o que foi dito muito bem:



5. Listagem dos Pontos de Controle e Utilização

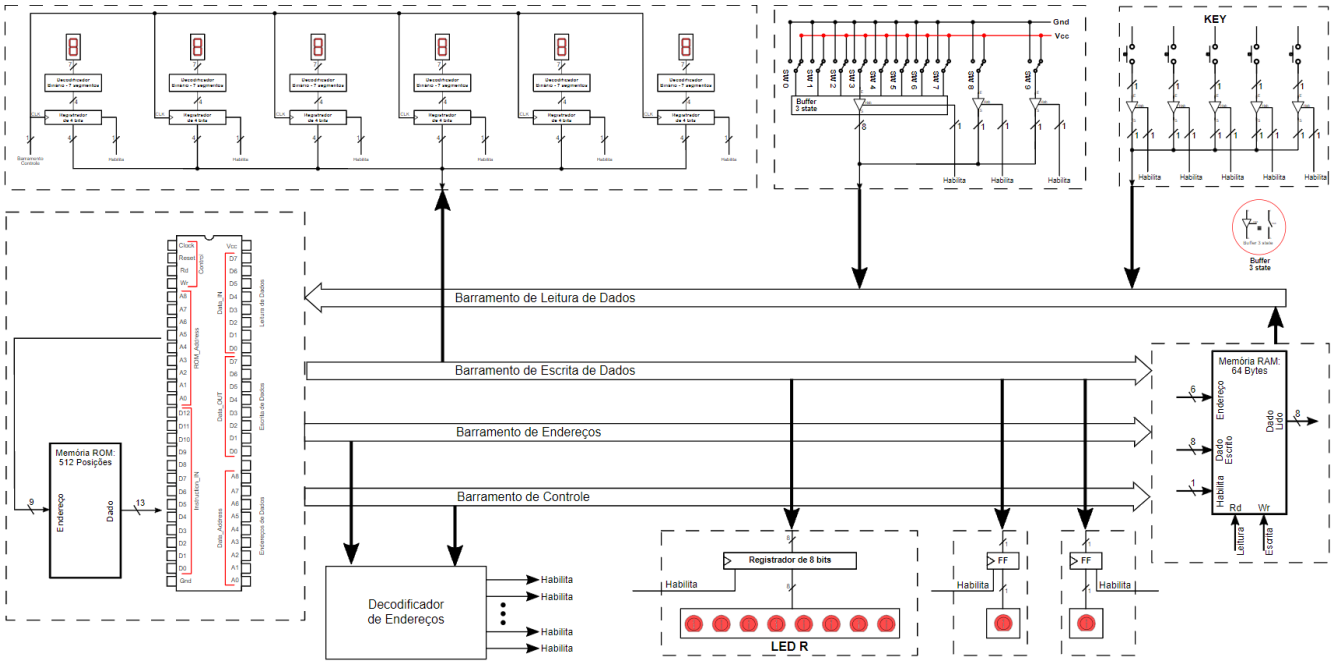
Os pontos de controle já foram citados no segundo tópico, na Tabela 1, entretando sem a explicação do que cada um pode realizar. Abaixo, na Tabela 3, isso é feito.

Tabela 3 – Pontos de Controle e Utilização

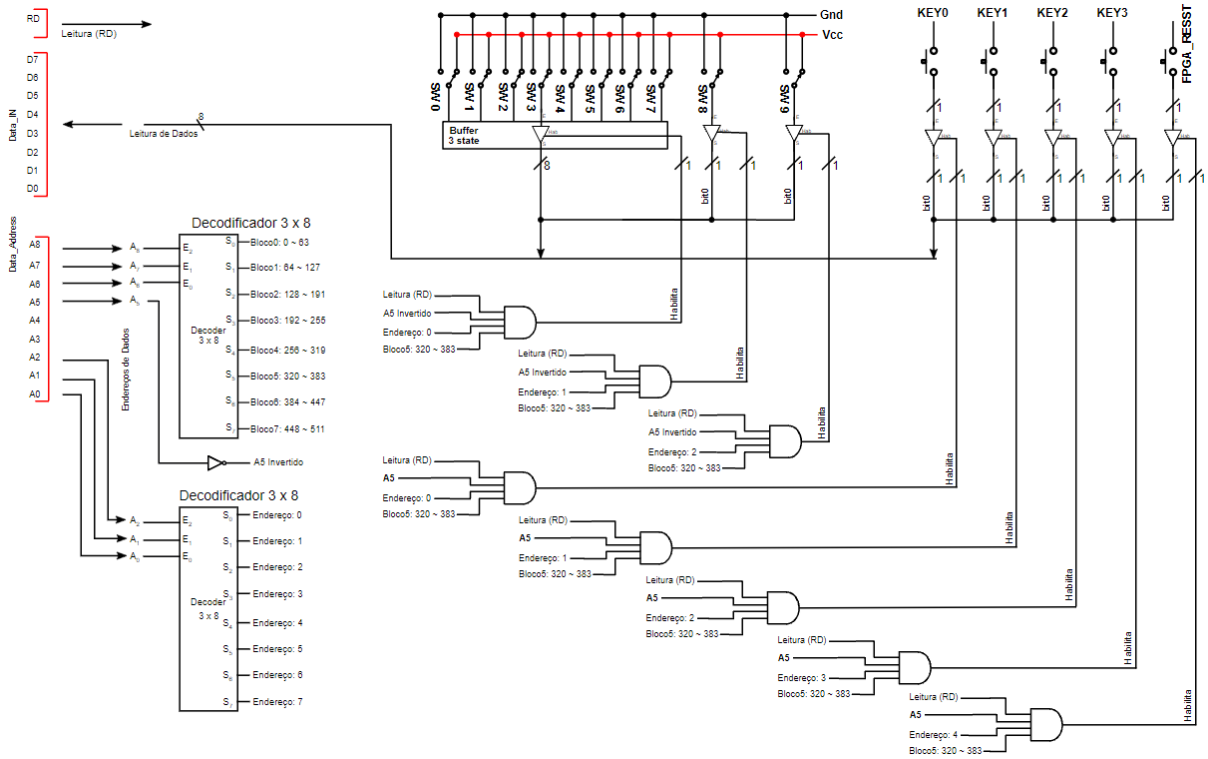
Pontos de Controle	Utilização
Habilita Escrita Retorno	Permite ou não a escrita no registrador Endereço Retorno.
JMP	JMP é o desvio, sem condições, apenas selecionando a "linha".
JSR	Responsável pelo desvio para a subrotina. É parecido com o JMP, mas específico para a subrotina, tendo o RET para voltar após a finalização.
RET	O RET é utilizado após a finalização de uma subrotina chamada pelo JSR. Ele guarda a linha que tem que voltar.
JEQ	Cumpre o papel de ativar o desvio condicional de execução, diferentemente do JMP, que é sem condição.
Seletor do Mux (SelMUX)	Seleciona se a saída é da instrução (7~0) ou do Barramento de Dados Entrada.
Operação	Determina se a ULA irá somar, subtrair ou passar.
Habilita_A	Antes, permitia com que o registrador A pudesse ser escrito, por isso o nome com A. Agora com a mudança de arquitetura, permite no banco de registradores.
Habilita Flag = (habFlag=)	Permite ou não a escrita no registrador Flag=, caso a operação da ULA seja CEQ.
RD – Read (habLeituraMEM)	Permite ou não com que a memória de dados possa fazer a leitura.
WR – Write (habEscritaMEM)	Permite ou não com que a memória de dados possa fazer a escrita.

6. Diagrama de Conexão do Processador com os Periféricos

O diagrama que melhor representa o que foi feito é o que está abaixo:



Cada periférico tinha um endereço específico na memória, que vai ser descrito no tópico 7 e esse endereço era ativado de acordo com uma porta AND. A imagem abaixo representa melhor essa ideia:



7. Mapa de Memória

O mapa utilizado pelo grupo durante todo o processo de criação e desenvolvimento do projeto sempre foi utilizando como base o que era descrito nas aulas e pelo professor, portando ficando como abaixo:

Tabela 4 – 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

Foi dado o destaque de ter sido feito com base no que foi feito durante as aulas pois ele é semelhante ao que se encontra nas anotações da Aula 9, no material do professor da disciplina.

8. Fonte do programa Assembly

O código Assembly utilizado no nosso projeto, contendo toda a lógica, se encontra no arquivo da memória ROM do projeto enviado, porém está descrito abaixo:

```
      tmp(0)  := LDI  & "01" & '0' & x"00"; -- manda 0 para acumulador

      -- Manda zero para os displays
      tmp(1)  := STA  & "01" & '1' & x"20";
      tmp(2)  := STA  & "01" & '1' & x"21";
      tmp(3)  := STA  & "01" & '1' & x"22";
      tmp(4)  := STA  & "01" & '1' & x"23";
      tmp(5)  := STA  & "01" & '1' & x"24";
      tmp(6)  := STA  & "01" & '1' & x"25";


      -- Manda zero para os LEDS 0 a 7
      tmp(7)  := STA  & "01" & '1' & x"00";
      -- Os LED 8 e 9 estão em baixo


      tmp(8)  := JMP  & "01" & '0' & x"80";


      tmp(9)  := LDI  & "01" & '0' & x"00";
      tmp(10) := STA  & "01" & '0' & x"00"; -- guarda 0 no mem[0]
      tmp(11) := STA  & "01" & '0' & x"02"; -- 0 no mem[2] (contador)
      tmp(12) := LDI  & "01" & '0' & x"01"; -- manda 1 para o acumulador
      tmp(13) := STA  & "01" & '0' & x"01"; -- armazena 1 no mem[1] (constante
1)
      tmp(14) := NOP  & "01" & '0' & x"00"; --
      tmp(15) := LDA  & "01" & '1' & x"60"; -- 352=x160 (KEY0) VERIFICA SE
APERTOU OU NÃO
      tmp(16) := CEQ  & "01" & '0' & x"00"; -- compara key0 com mem[0](que
esta guardando 0)
      tmp(17) := JEQ  & "01" & '0' & x"13"; -- se for igual, ou seja, key0 não
foi apertado, pulo para linha 19
      tmp(18) := JSR  & "01" & '0' & x"33"; -- se apertou o key0, vai para
subrotina, pula para linha 51
      tmp(19) := NOP  & "01" & '0' & x"00"; --
      tmp(20) := LDA  & "01" & '1' & x"61"; -- 353=x161 (KEY1) VERIFICA SE
APERTOU OU NÃO
      tmp(21) := CEQ  & "01" & '0' & x"00"; -- compara key1 com mem[0](que
esta guardando 0)
      tmp(22) := JEQ  & "01" & '0' & x"18"; -- se for igual, ou seja, key0 não
foi apertado, pulo para linha 24
```

```
      tmp(23) := JSR  & "01" & '0' & x"93";-- AINDA PENSAR EM QUAL SUBROTINA
(VOLTAR AQUIIIIIIIIIII) ----------------------------------------------------
------------------------
      tmp(24) := NOP  & "01" & '0' & x"00"; --
      tmp(25) := LDA  & "01" & '1' & x"64"; -- 356=x164 (FPGA_RESET) VERIFICA
SE APERTOU OU NÃO
      tmp(26) := CEQ  & "01" & '0' & x"00"; -- compara FPGA_RESET com
mem[0](que esta guardando 0)
      tmp(27) := JEQ  & "01" & '0' & x"1d"; -- se for igual, ou seja, key0 não
foi apertado, pulo para linha 29
      tmp(28) := JSR  & "01" & '0' & x"00";-- AINDA PENSAR EM QUAL
SUBROTINA  (VOLTAR AQUIIIIIIIIIII)
      tmp(29) := NOP  & "01" & '0' & x"00"; --
      tmp(30) := JMP  & "01" & '0' & x"0e"; -- Pula para 14 para ficar no laço
de verificar se o key0 foi apertado ou não
      tmp(31) := NOP  & "01" & '0' & x"00";
      tmp(32) := NOP  & "01" & '0' & x"00";
      tmp(33) := NOP  & "01" & '0' & x"00";
      tmp(34) := NOP  & "01" & '0' & x"00";
      tmp(35) := NOP  & "01" & '0' & x"00";
      tmp(36) := NOP  & "01" & '0' & x"00";
      tmp(37) := NOP  & "01" & '0' & x"00";
      tmp(38) := NOP  & "01" & '0' & x"00";
      tmp(39) := NOP  & "01" & '0' & x"00";
      tmp(40) := NOP  & "01" & '0' & x"00";
      tmp(41) := NOP  & "01" & '0' & x"00";
      tmp(42) := NOP  & "01" & '0' & x"00";
      tmp(43) := NOP  & "01" & '0' & x"00";
      tmp(44) := NOP  & "01" & '0' & x"00";
      tmp(45) := NOP  & "01" & '0' & x"00";
      tmp(46) := NOP  & "01" & '0' & x"00";
      tmp(47) := NOP  & "01" & '0' & x"00";
      tmp(48) := NOP  & "01" & '0' & x"00";
      tmp(49) := NOP  & "01" & '0' & x"00";
      tmp(50) := NOP  & "01" & '0' & x"00";

--    -- Apertou o key0 soma 1 no acumulador e muda os displays
      tmp(51) := STA  & "01" & '1' & x"ff"; -- 511=x1ff, limpa a leitura no
botão
      tmp(52) := LDI & "11" & '0' & x"01"; -- passa 1
      tmp(53) := CEQ & "11" & '0' & x"03"; -- ve se tem 1 na mem[3] flag
      tmp(54) := JEQ & "01" & '0' & x"0e"; -- se flag=1 não pode contar mais
volta para linha 14
      tmp(55) := LDA  & "01" & '0' & x"02"; -- bota o valor do mem[2] no
acumulador
      tmp(56) := SOMA & "01" & '0' & x"01"; -- Soma a constate 1 que esta no
MEM[1] com o valor que foi para o acumulador
```

```
      tmp(57) := STA  & "01" & '0' & x"02"; -- guarda o valor da soma em
mem[2] (contador)
      tmp(58) := STA  & "01" & '1' & x"02"; -- 258=x102 armazena o valor do
bit0 do acumulador no LDR9
      tmp(59) := JMP  & "01" & '0' & x"3d"; -- Vai para subrotina de colocar
os valores no display(na linha40)
      tmp(60) := RET  & "01" & '0' & x"00"; -- Retorna da subrotina (ou seja,
vai para linha 19)
      --tmp(58) := NOP  & "01" & '0' & x"00";
      -- SUBROTINA DE COLOCAR VALORES NO DISPLAY

      tmp(61) := LDA  & "10" & '0' & x"34"; -- Passa o valor atual de HEX0
para R2
      tmp(62) := SOMA & "10" & '0' & x"01"; -- soma 1
      tmp(63) := STA  & "10" & '0' & X"34"; -- Salva esse valor no mem[52]
      tmp(64) := CEQ  & "10" & '0' & x"2e"; -- Compara com o limite de HEX0
(que ta guardado em mem[46]) com R2
      tmp(65) := JEQ  & "00" & '0' & x"45"; -- Se for igual vai pular para a
linha que mexe na dezena
      -- se não pular é pq não chegou no limite!
      tmp(66) := LDA  & "10" & '0' & x"34"; -- Pega o valor que salvou no
mem[52]
      tmp(67) := STA  & "10" & '1' & x"20";
      tmp(68) := JMP  & "10" & '0' & x"3c"; -- Somou volta para o ret e fica
verificando verificando de novo o KEY0

      ---- SÓ FAZ ESSA PARTE DE BAIXO DE HEX0 PASSOU DO LIMITE
      tmp(69) := LDA  & "10" & '0' & x"00"; -- Salva 0 na R2
      tmp(70) := STA  & "10" & '0' & x"34"; -- Passa r2 para memoria do HEX0
(d52 = 0x34)
      tmp(71) := STA  & "10" & '1' & x"20"; -- manda 0 para HEX0
      tmp(72) := LDA  & "10" & '0' & x"35"; -- valor que ta em HEX1 para R2
      tmp(73) := SOMA & "10" & '0' & x"01";
      tmp(74) := STA  & "10" & '0' & X"35";
      tmp(75) := CEQ  & "10" & '0' & x"2f";
      tmp(76) := JEQ  & "10" & '0' & x"50"; -- Vai para a linha 80
      -- se não pular é pq não chegou no limite!
      tmp(77) := LDA  & "10" & '0' & x"35";
      tmp(78) := STA  & "10" & '1' & x"21";
      tmp(79) := JMP  & "01" & '0' & x"3c";

      ---- SÓ FAZ ESSA PARTE DE BAIXO DE HEX1 PASSOU DO LIMITE
      tmp(80) := LDA  & "10" & '0' & x"00";
      tmp(81) := STA  & "10" & '0' & x"35";
      tmp(82) := STA  & "10" & '1' & x"21"; -- bota 0 no HEX1
      tmp(83) := LDA  & "10" & '0' & x"36";
      tmp(84) := SOMA & "10" & '0' & x"01";
      tmp(85) := STA  & "10" & '0' & X"36";
```

```
    tmp(86) := CEQ  & "10" & '0' & x"30";
    tmp(87) := JEQ  & "10" & '0' & x"5b"; -- 91
     -- se não pular é pq não chegou no limite!
    tmp(88) := LDA  & "10" & '0' & x"36";
    tmp(89) := STA  & "10" & '1' & x"22";
    tmp(90) := JMP  & "01" & '0' & x"3c";

    ---- SÓ FAZ ESSA PARTE DE BAIXO DE HEX2 PASSOU DO LIMITE
    tmp(91) := LDA  & "10" & '0' & x"00";
    tmp(92) := STA  & "10" & '0' & x"36";
    tmp(93) := STA  & "10" & '1' & x"22"; -- bota 0 HEX2
    tmp(94) := LDA  & "10" & '0' & x"37";
    tmp(95) := SOMA & "10" & '0' & x"01";
    tmp(96) := STA  & "10" & '0' & X"37";
    tmp(97) := CEQ  & "10" & '0' & x"31";
    tmp(98) := JEQ  & "10" & '0' & x"66"; -- 102
    -- se não pular é pq não chegou no limite!
    tmp(99) := LDA  & "10" & '0' & x"37";
    tmp(100) := STA  & "10" & '1' & x"23";
    tmp(101) := JMP  & "01" & '0' & x"3c";


    ---- SÓ FAZ ESSA PARTE DE BAIXO DE HEX3 PASSOU DO LIMITE
    tmp(102) := LDA  & "10" & '0' & x"00";
    tmp(103) := STA  & "10" & '0' & x"37";
    tmp(104) := STA  & "10" & '1' & x"23";
    tmp(105) := LDA  & "10" & '0' & x"38";
    tmp(106) := SOMA & "10" & '0' & x"01";
    tmp(107) := STA  & "10" & '0' & X"38";
    tmp(108) := CEQ  & "10" & '0' & x"32";
    tmp(109) := JEQ  & "10" & '0' & x"71"; -- 113
    -- se não pular é pq não chegou no limite!
    tmp(110) := LDA  & "10" & '0' & x"38";
    tmp(111) := STA  & "10" & '1' & x"24";
    tmp(112) := JMP  & "01" & '0' & x"3c";

    ---- SÓ FAZ ESSA PARTE DE BAIXO DE HEX4 PASSOU DO LIMITE
    tmp(113) := LDA  & "10" & '0' & x"00";
    tmp(114) := STA  & "10" & '0' & x"38";
    tmp(115) := STA  & "10" & '1' & x"24";
    tmp(116) := LDA  & "10" & '0' & x"39";
    tmp(117) := SOMA & "10" & '0' & x"01";
    tmp(118) := STA  & "10" & '0' & X"39";
    tmp(119) := CEQ  & "10" & '0' & x"33";
    tmp(120) := JEQ  & "10" & '0' & x"7c"; --124
    -- se Não pular é pq não chegou no limite!
    tmp(121) := LDA  & "10" & '0' & x"39";
    tmp(122) := STA  & "10" & '1' & x"25";
```

```
      tmp(123) := JMP  & "01" & '0' & x"3c";


      ---- SÓ FAZ ESSA PARTE DE BAIXO DE HEX5 PASSOU DO LIMITE
      ---- OU SEJA, LIMITE MÁXIMO
      --- VAMOS ACENDER O LED 8
      tmp(124) := LDI  & "10" & '0' & x"01"; -- para 1 o R2
      tmp(125) := STA  & "10" & '1' & x"01"; -- ACENDE O LED 8
      tmp(126) := LDI & "11" & '0' & x"01";
      tmp(127) := STA & "11" & '0' & x"03"; -- bota 1 na flag (mem3) então
para a contagem

      tmp(128) := LDI  & "11" & '0' & x"0a"; -- 9 PARA O REG 3
      tmp(129) := STA  & "11" & '0' & x"2e"; -- 46
      tmp(130) := STA  & "11" & '0' & x"2f"; -- 47
      tmp(131) := STA  & "11" & '0' & x"30"; -- 48
      tmp(132) := STA  & "11" & '0' & x"31"; -- 49
      tmp(133) := STA  & "11" & '0' & x"32"; -- 50
      tmp(134) := STA  & "11" & '0' & x"33"; -- 51
      tmp(135) := LDI  & "11" & '0' & x"00";
      tmp(136) := STA  & "11" & '1' & x"01";
      tmp(137) := STA  & "11" & '1' & x"02";
      tmp(138) := STA  & "11" & '0' & x"34"; -- 52
      tmp(139) := STA  & "11" & '0' & x"35"; -- 53
      tmp(140) := STA  & "11" & '0' & x"36"; -- 54
      tmp(141) := STA  & "11" & '0' & x"37"; -- 55
      tmp(142) := STA  & "11" & '0' & x"38"; -- 56
      tmp(143) := STA  & "11" & '0' & x"39"; -- 57
      tmp(144) := STA & "11" & '0' & x"03"; --flag na MEM[3]
      tmp(145) := JMP  & "00" & '0' & x"09";
      tmp(146) := NOP  & "01" & '0' & x"00";


           -- SUBROTINA DE CONFIGURAR LIMITES!!
        --configunrando as unidade
      tmp(147) := STA  & "01" & '1' & x"fe";
      tmp(148) := LDA  & "00" & '1' & x"40"; -- le as chaves
      tmp(149) := STA  & "00" & '0' & x"2e"; -- 46
        -- loop KEY1
      tmp(150) := NOP  & "01" & '0' & x"00"; --
      tmp(151) := LDA  & "01" & '1' & x"61"; -- 353=x161 (KEY1) VERIFICA SE
APERTOU OU NÃO
      tmp(152) := CEQ  & "01" & '0' & x"00"; -- compara key1 com mem[0](que
esta guardando 0)
      tmp(153) := JEQ  & "01" & '0' & x"96"; -- se for igual, ou seja, key0
não foi apertado, pulo para linha 24
      tmp(154) := JMP  & "01" & '0' & x"9c";-- AINDA PENSAR EM QUAL SUBROTINA
(VOLTAR AQUIIIIIIIIIII)
```

```
      tmp(155) := NOP  & "01" & '0' & x"00"; --
        -- configunrando as dezenas
      tmp(156) := STA  & "01" & '1' & x"fe";
      tmp(157) := LDA  & "00" & '1' & x"40"; -- le as chaves
      tmp(158) := STA  & "00" & '0' & x"2f"; -- 47
      -- loop KEY1
      tmp(159) := NOP  & "01" & '0' & x"00"; --
      tmp(160) := LDA  & "01" & '1' & x"61"; -- 353=x161 (KEY1) VERIFICA SE
APERTOU OU NÃO
      tmp(161) := CEQ  & "01" & '0' & x"00"; -- compara key1 com mem[0](que
esta guardando 0)
      tmp(162) := JEQ  & "01" & '0' & x"9f"; -- se for igual, ou seja, key0
não foi apertado, pulo para linha 24
      tmp(163) := JMP  & "01" & '0' & x"a5";-- AINDA PENSAR EM QUAL SUBROTINA
(VOLTAR AQUIIIIIIIIIII)
      tmp(164) := NOP  & "01" & '0' & x"00"; --
        -- configunrando as centenas
      tmp(165) := STA  & "01" & '1' & x"fe";
     tmp(166) := LDA  & "00" & '1' & x"40"; -- le as chaves
      tmp(167) := STA  & "00" & '0' & x"30"; -- 48
        -- loop KEY1
      tmp(168) := NOP  & "01" & '0' & x"00"; --
      tmp(169) := LDA  & "01" & '1' & x"61"; -- 353=x161 (KEY1) VERIFICA SE
APERTOU OU NÃO
      tmp(170) := CEQ  & "01" & '0' & x"00"; -- compara key1 com mem[0](que
esta guardando 0)
      tmp(171) := JEQ  & "01" & '0' & x"a8"; -- se for igual, ou seja, key0
não foi apertado, pulo para linha 24
      tmp(172) := JMP  & "01" & '0' & x"ae";-- AINDA PENSAR EM QUAL SUBROTINA
(VOLTAR AQUIIIIIIIIIII)
      tmp(173) := NOP  & "01" & '0' & x"00"; --
        -- configunrando as unidade de milhares
      tmp(174) := STA  & "01" & '1' & x"fe";
     tmp(175) := LDA  & "00" & '1' & x"40"; -- le as chaves
      tmp(176) := STA  & "00" & '0' & x"31"; -- 49
     -- loop KEY1
      tmp(177) := NOP  & "01" & '0' & x"00"; --
      tmp(178) := LDA  & "01" & '1' & x"61"; -- 353=x161 (KEY1) VERIFICA SE
APERTOU OU NÃO
      tmp(179) := CEQ  & "01" & '0' & x"00"; -- compara key1 com mem[0](que
esta guardando 0)
      tmp(180) := JEQ  & "01" & '0' & x"b1"; -- se for igual, ou seja, key0
não foi apertado, pulo para linha 24
      tmp(181) := JMP  & "01" & '0' & x"b7";-- AINDA PENSAR EM QUAL SUBROTINA
      tmp(182) := NOP  & "01" & '0' & x"00"; --
        -- configunrando as dezenas de milhares
      tmp(183) := STA  & "01" & '1' & x"fe";
     tmp(184) := LDA  & "00" & '1' & x"40"; -- le as chaves
```

```
      tmp(185) := STA  & "00" & '0' & x"32"; -- 50
        -- loop KEY1
      tmp(186) := NOP  & "01" & '0' & x"00"; --
      tmp(187) := LDA  & "01" & '1' & x"61"; -- 353=x161 (KEY1) VERIFICA SE
APERTOU OU NÃO
      tmp(188) := CEQ  & "01" & '0' & x"00"; -- compara key1 com mem[0](que
esta guardando 0)
      tmp(189) := JEQ  & "01" & '0' & x"ba"; -- se for igual, ou seja, key0
não foi apertado, pulo para linha 24
      tmp(190) := JMP  & "01" & '0' & x"c0";-- AINDA PENSAR EM QUAL SUBROTINA
      tmp(191) := NOP  & "01" & '0' & x"00"; --
    -- configunrando as centenas de milhares
      tmp(192) := STA  & "01" & '1' & x"fe";
     tmp(193) := LDA  & "00" & '1' & x"40"; -- le as chaves
      tmp(194) := STA  & "00" & '0' & x"33"; -- 51
      tmp(195) := RET  & "01" & '0' & x"00";
      tmp(196) := NOP  & "01" & '0' & x"00";
```