Insper Instituto de Ensino e Pesquisa

Projeto 1: Relógio
Utilizando um Processador Personalizado

Design de Computadores

Natália Queiroz Menezes Carreras
Willian Kenzo Asanuma Lee

1. Características do processador:

O processador utilizado neste projeto possui uma arquitetura de registrador memória. Um processador com essa arquitetura realiza operações entre um registrador e uma posição da memória RAM ou imediato, salvando o valor no mesmo registrador utilizado nos argumentos, esse tipo de arquitetura também pode ser chamado de x86.

Além disso o processador contém um switch (SW9) que se levantado altera a base tempo do relógio implementado.

2. Instruções do processador:

a. Formato das instruções:

Os primeiros 4 bits (14 ~ 11) representam o OpCode, os bits 10 e 9 representam o registrador que será utilizado na operação, e os outros 8 representam o imediato. O destino do Jump é o imediato (8 ~ 0), o endereço da RAM é o imediato (5 ~ 0). Já o valor é o imediato (7 ~ 0).

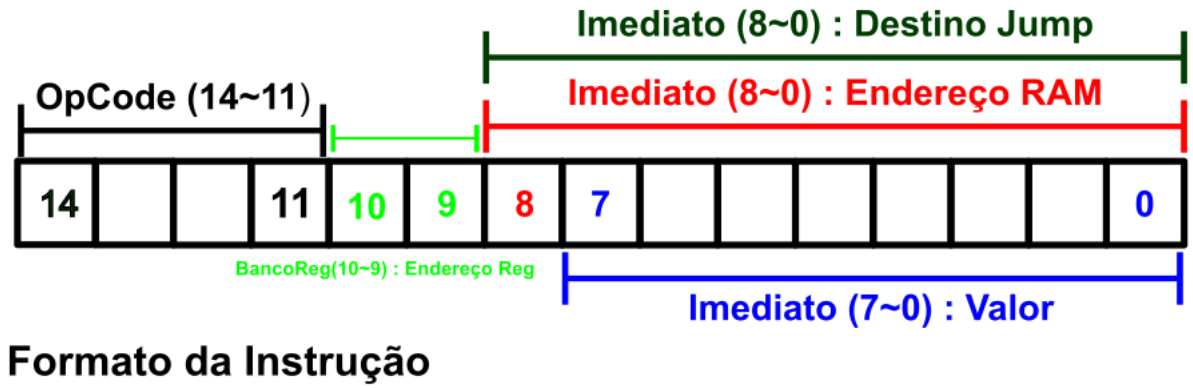

Figura 1: Formato das instruções

Um exemplo de instrução seria:

SOMA & R1& 0 $ 0x06

Com a seguinte execução:

R[1] = R[1] + Mem[0x06].

b. Instruções do processador: O processador possui um total de 11 instruções, sendo elas:

1. LDA: Carrega valor da memória para um registrador → Opcode = 0001
2. SOMA: Soma A e B e armazena em um registrador → Opcode = 0010
3. SUB: Subtrai B de A e armazena em um registrador → Opcode = 0011
4. LDI: Carrega valor imediato para um registrador → Opcode = 0100
5. STA: Salva valor do registrador para a memória → Opcode = 0101
6. JMP: Desvio de execução → Opcode = 0110
7. JEQ: Desvio condicional de execução, verifica o flag e, caso verdadeiro, faz o desvio. → Opcode = 0111
8. CEQ: Compara se o valor do acumulador é igual ao valor contido no endereço de memória. Caso sim ativa o flag IGUAL → Opcode = 1000
9. JSR: Chamada de sub-rotina. Desvia a execução para um trecho de código que, após executado retorna para a posição seguinte à chamada da sub-rotina → Opcode = 1001
10. RET: Retorno de sub-rotina. → Opcode = 1010
11. NOP: Sem operação → Opcode = 0000
12. ANDI: Faz uma operação de comparação entre bit a bit e salva o resultado no registrador→ Opcode = 1011
13. ADDI: Faz uma soma entre um registrador e o imediato → Opcode = 1100

c. Pontos de controle: o processador possui 11 pontos de controle sendo eles:

1. Habilita escrita retorno: será 1 quando precisar salvar o valor da posição seguinte quando for realizada uma chamada de sub-rotina.
2. JMP: será 1 quando for necessário realizar um desvio de execução.
3. RET: será 1 quando for para sair da sub-rotina e voltar para a rotina normal na posição indicada pelo endereço de retorno.
4. JSR: será 1 quando for realizado um desvio para uma sub-rotina.
5. JEQ: será 1 quando for necessário realizar (ou não) um desvio de execução condicional.
6. Sel Mux: será 1 quando for para selecionar o valor imediato e 0 quando for para selecionar a saída de dados da memória RAM.
7. Habilita A: será 1 quando for habilitar o acumulador (Registrador A).
8. Operação: será "01" quando a operação for de soma, será "00" quando a operação for de subtração e será "10". Para outras instruções, o valor pode ser qualquer um, então consideramos "00".
9. Habilita Flag: será 1 quando o valor de A for igual ao valor de B, se for diferente, ele retornará 0.
10. RD: será 1 quando a leitura da memória RAM estiver habilitada.
11. WR: será 1 quando a escrita na memória RAM estiver habilitada.

d. Utilização dos pontos de controle:

Inst.	Código binário	Hab. escrita retorno	JMP	RET	JSR	JEQ	Sel. Mux	Hab. A	Op.	Hab. Flag	RD	WR
NOP	0000	0	0	0	0	0	0	0	00	0	0	0
LDA	0001	0	0	0	0	0	0	1	10	0	1	0
SOM	0010	0	0	0	0	0	0	1	01	0	1	0
SUB	0011	0	0	0	0	0	0	1	00	0	1	0
LDI	0100	0	0	0	0	0	1	1	10	0	0	0
STA	0101	0	0	0	0	0	0	0	00	0	0	1
JMP	0110	0	1	0	0	0	0	0	00	0	0	0
JEQ	0111	0	0	0	0	1	0	0	00	0	0	0
CEQ	1000	0	0	0	0	0	0	0	00	1	1	0
JSR	1001	1	0	0	1	0	0	0	00	0	0	0
RET	1010	0	0	1	0	0	0	0	00	0	0	0
ANDI	1011	0	0	0	0	0	1	1	11	0	0	0
ADDI	1100	0	0	0	0	0	1	1	10	0	0	0

3. Fluxo de dados e conexões:

a. Fluxo de dados para o processador, com uma explicação resumida do seu funcionamento:

O processador faz a leitura das chaves (SW 9 a 0), botões (KEY 3 a 0), do Clock e o do botão de RESET, e retorna os valores que serão escritos nos leds (LED 9 a 0) e nos displays de sete segmentos. Ele é composto por:

1. Uma CPU 9 bits para endereços (512 endereços), 15 bits para instrução, 8 bits de entrada de dados (leitura) e 8 bits de saída de dados (escrita).
2. Uma memória ROM de 512 posições.
3. Uma memória RAM de 64 Bytes.
4. Um decodificador para sete segmentos.
5. Entrada de chaves e botões.
6. Interface de tempo

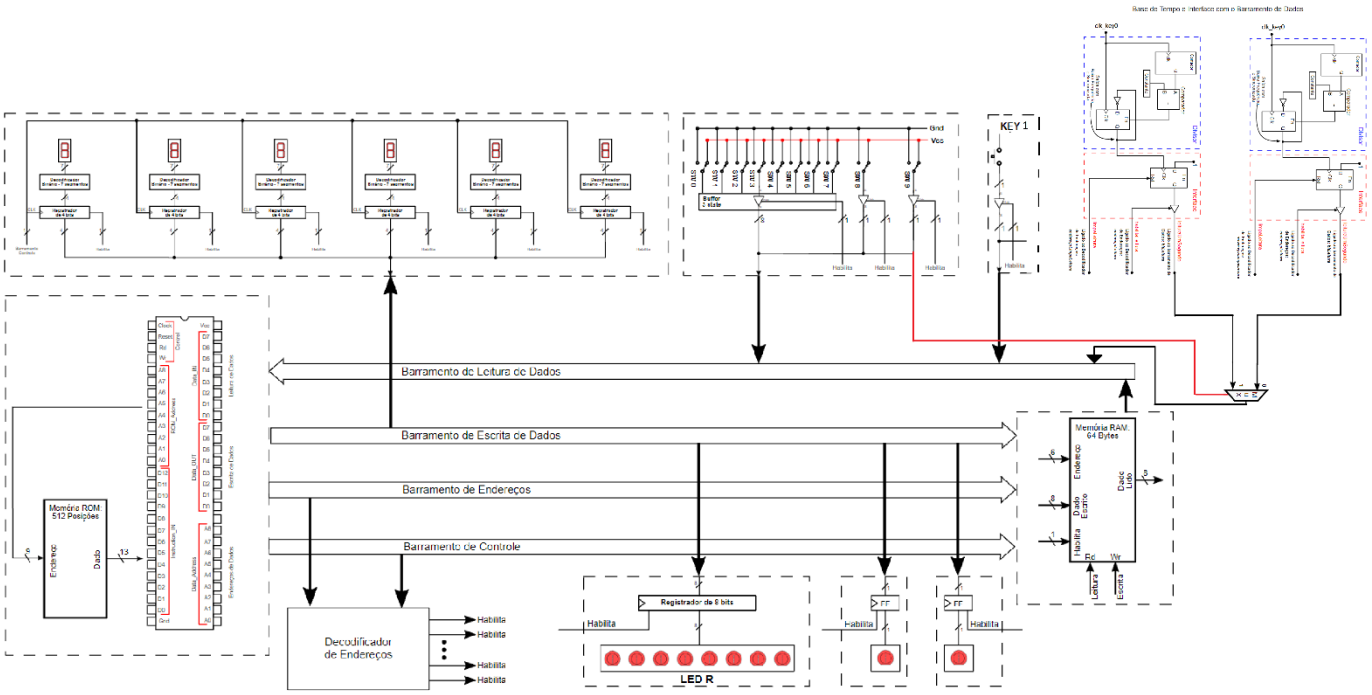

Figura 2: Esboço do computador

b. Rascunho do diagrama de conexão do processador com os periféricos:

1. Conexão do processador com a memória RAM:

A memória RAM ocupa o intervalo de endereços entre 0 e 63 e recebe os dados da CPU. O endereço que sai da CPU (Data_Address) possui 9 bits (A8 ~A0), dentre eles, os bits mais significativos (A8 ~A6) entram no decodificador 3x8 e se a saída for o Bloco 0, a RAM será habilitada. Já os 6 bits menos significativos (A5 ~A0) indicam o endereço da RAM.

A CPU também é responsável por enviar para a memória RAM os valores de habilitar a escrita e leitura. Quando estiver habilitada a leitura, a memória RAM enviará o dado da posição que foi endereçada, e quando estiver habilitada a escrita, a memória RAM receberá o valor que será armazenado em seu respectivo endereço.

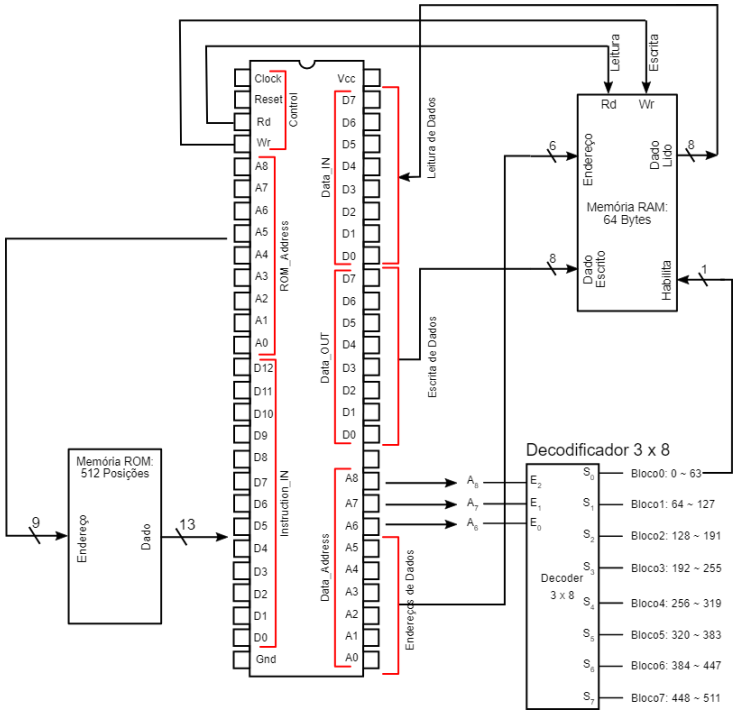

Figura 3: Rascunho da Memória RAM

2. Conexão do processador com os LEDs:

Os primeiros LEDs (LED 0 a LED 7) estão alocados no endereço 256, o LED 8 esta alocado no endereço 257 e o LED 9 está alocado no endereço 258. Assim como na memória RAM, foi usado um decodificador 3x8 que recebe de entrada os três bits menos significativos (A0 ~ A2). Caso o decodificador retorne o Endereço 0, a escrita esteja habilitada e o bloco 4 seja 1, os LEDS 0 a 7 serão ligados. No caso do Endereço ser 1, escrita habilitada e bloco 4 = 1, o LED 8 será ligado, e se o Endereço for 2, a escrita estiver habilitada e o bloco 4 for 1, o LED 9 será habilitado.

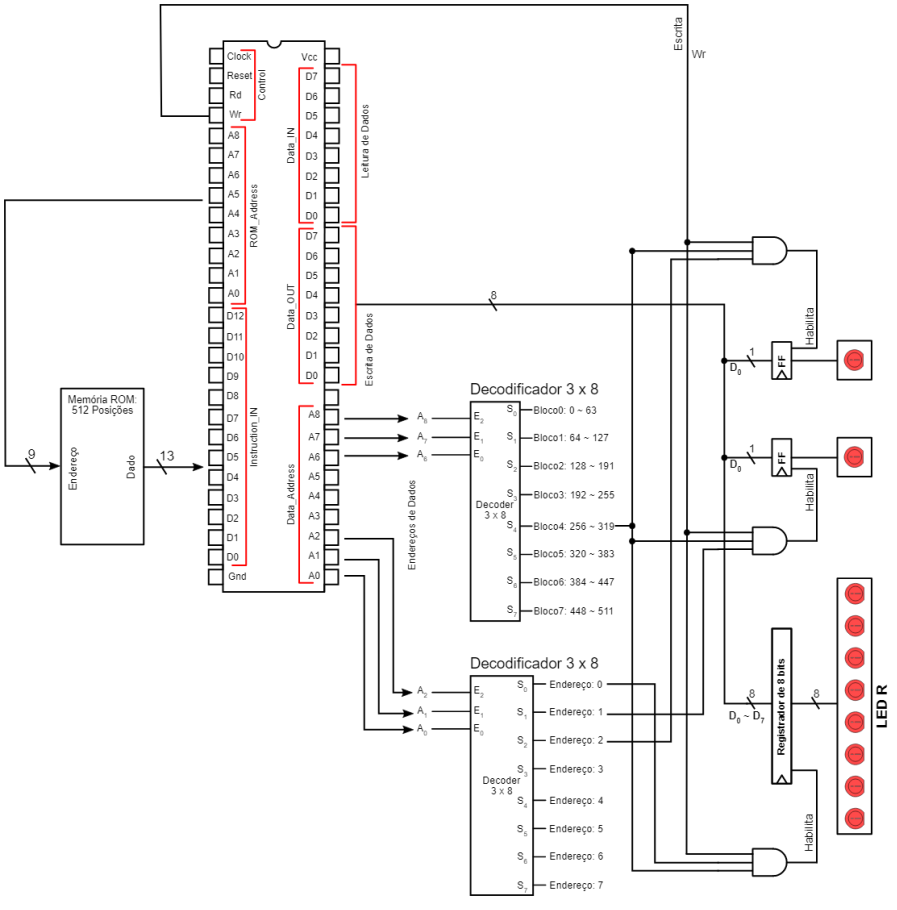

Figura 4: Rascunho dos LEDs

3. Conexão do processador com os displays de sete segmentos:

Os displays de sete segmentos foram alocados entre 288 e 293. Para ativar o Display HEX 0, será necessário que a escrita esteja habilitada e que o Endereço 0, o Bloco 4 e o A5 sejam 1, e o endereço de saída da CPU deve ser 288. O endereço do HEX 1 é 289, HEX 2 é 290, HEX 3 é 291, HEX 4 é 292 e HEX 5 é 293, e, assim como no HEX 0, os valores de habilita escrita, bloco 4 e A5 devem ser 1, e o Endereço do decoder será respectivo com o HEX, ou seja, HEX 1 precisa do Endereço 1, HEX 2 precisa do Endereço 2 e assim por diante.

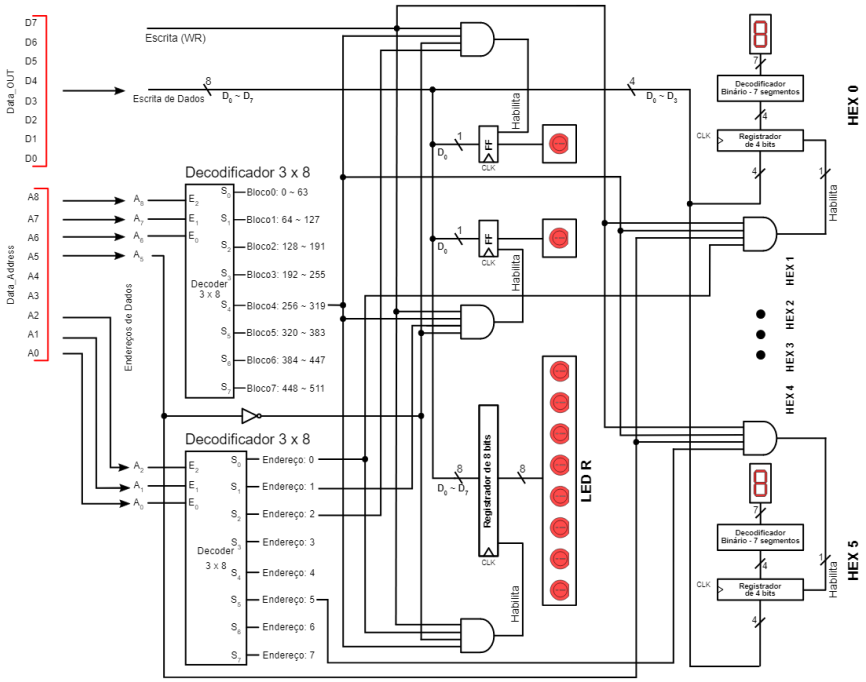

Figura 5: Racunho dos displays de sete segmentos

4. Conexão do processador com os Botões e Chaves:

As chaves foram alocadas entre os endereços 320 e 322. Para que sejam lidos como 1, é necessário que o valor de A5 seja 0, e que os valores de habilita leitura e Bloco 5 sejam 1. Além disso, o SW 9 precisa que o Endereço 2 seja 1, o SW8 precisa que o endereço 1 seja 1 e os SW 7 a 0 precisam que o endereço 0 seja 1.

Já os botões foram alocados entre os endereços 352 e 356. Para que sejam lidos como 1, é necessário que o valor de A5, Bloco 5 e leitura sejam 1. Além disso, para Key 0 o Endereço 0 deve ser 1, para Key 1 o Endereço 1 deve ser 1, para Key 2 o endereço 2 deve ser 1, para o Key 3 o endereço 3 deve ser 3 e para o botão de Reset o endereço 4 deve ser 1.

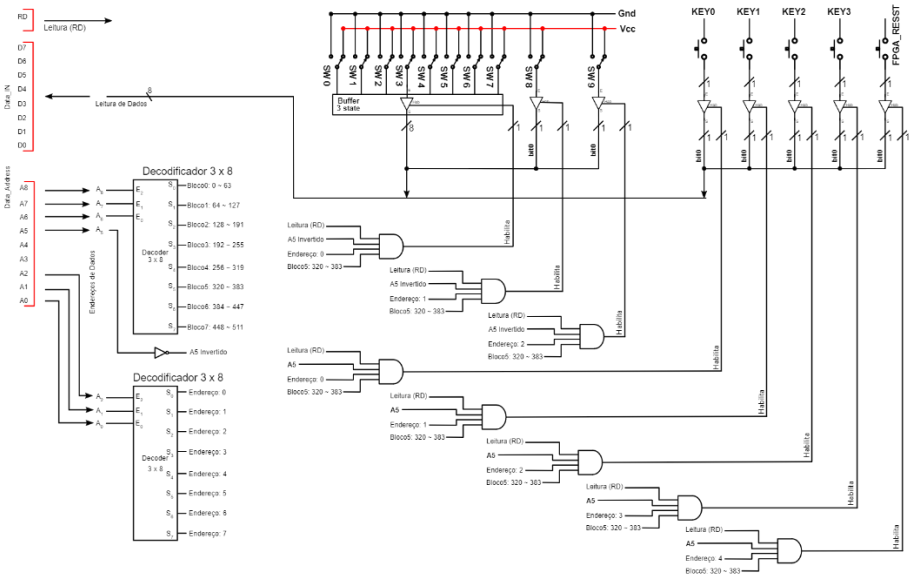

Figura 6: Racunho dos Botões e Chaves

5. Fluxo de dados dentro do processador:

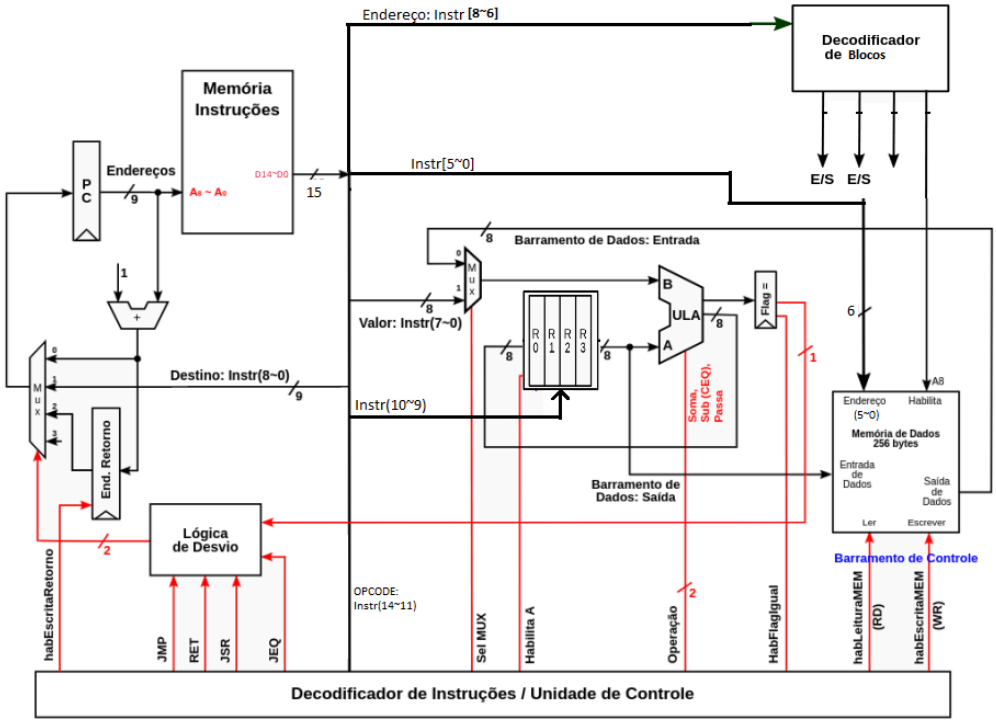

Figura 7: Fluxo de dados registrador memória

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	CLOCK	1 bit	Leitura	5
358 ~ 503				
504	Limpa leitura CLOCK	–	Escrita	7
507	Limpa leitura FPGA_RESET	–	Escrita	7
510	Limpa Leitura KEY1	–	Escrita	7
511	Limpa Leitura KEY0	–	Escrita	7

5. Fonte do programa em Assembly:

```
-- Segundos = REG[0] REG[1]

-- Uso geral = REG[2]

-- Clock_check = REG[3]
```

```
-- Armazenanmento de valor na RAM =  MEM[10] unidades
--                                   MEM[11] dezenas
--                                   MEM[12] centenas
--                                   MEM[13] milhares
--                                   MEM[14] dez milhares
--                                   MEM[15] cent milhares
--                                   MEM[16] flag
-- Armazenamento de limite na RAM = MEM[30] unidades
--                                  MEM[31] dezenas
--                                  MEM[32] centenas
--                                  MEM[33] milhares
--                                  MEM[34] dez milhares
--                                  MEM[35] cent milhares
--
--SP:

--LIMPA BOTOESBACK
tmp(0) := STA & R2 & '1' & x"FF";      -- STA %R2 .CLEARKEY0     #Limpa KEY 0
tmp(1) := STA & R2 & '1' & x"FE";      -- STA %R2 .CLEARKEY1     #Limpa KEY 1
tmp(2) := STA & R2 & '1' & x"FD";      -- STA %R2 .CLEARKEY2     #Limpa KEY 2
tmp(3) := STA & R2 & '1' & x"FC";      -- STA %R2 .CLEARKEY3     #Limpa KEY 3
tmp(4) := STA & R2 & '1' & x"FB";      -- STA %R2 .CLEARFPGA     #Limpa FPGA_RESET
tmp(5) := LDI & R2 & '0' & x"00";      -- LDI %R2 $0      #Carrega acumulador com valor 0

--ESCREVE 0 NOS DISPLAYS
tmp(6) := STA & R2 & '1' & x"20";      -- STA %R2 .HEX0    #Armazena o valor 0 no HEX0
tmp(7) := STA & R2 & '1' & x"21";      -- STA %R2 .HEX1    #Armazena o valor 0 no HEX1
tmp(8) := STA & R2 & '1' & x"22";      -- STA %R2 .HEX2    #Armazena o valor 0 no HEX2
tmp(9) := STA & R2 & '1' & x"23";      -- STA %R2 .HEX3    #Armazena o valor 0 no HEX3
tmp(10) := STA & R2 & '1' & x"24";     -- STA %R2 .HEX4    #Armazena o valor 0 no HEX4
tmp(11) := STA & R2 & '1' & x"25";     -- STA %R2 .HEX5    #Armazena o valor 0 no HEX5

--APAGANDO OS LEDS
```

```
tmp(12) := LDI & R2 & '0' & x"00";      -- LDI %R2 $0
tmp(13) := STA & R2 & '1' & x"00";      -- STA %R2 .LED07   #Armazena o valor 0 no LEDR7~0
tmp(14) := STA & R2 & '1' & x"01";      -- STA %R2 .LED8    #Armazena o valor 0 no LEDR8
tmp(15) := STA & R2 & '1' & x"02";      -- STA %R2 .LED9    #Armazena o valor 0 no LEDR9

--VARIAVEIS QUE ARMAZENAM O VALOR DO DISPLAY
tmp(16) := STA & R2 & '0' & x"0A";      -- STA %R2 @10      #Armazena o valor do acumulador em
MEM[10](unidades)
tmp(17) := STA & R2 & '0' & x"0B";      -- STA %R2 @11      #Armazena o valor do acumulador em
MEM[11](dezenas)
tmp(18) := STA & R2 & '0' & x"0C";      -- STA %R2 @12      #Armazena o valor do acumulador em
MEM[12](centenas)
tmp(19) := STA & R2 & '0' & x"0D";      -- STA %R2 @13      #Armazena o valor do acumulador em
MEM[13](milhares)
tmp(20) := STA & R2 & '0' & x"0E";      -- STA %R2 @14      #Armazena o valor do acumulador em
MEM[14](dez milhares)
tmp(21) := STA & R2 & '0' & x"0F";      -- STA %R2 @15      #Armazena o valor do acumulador em
MEM[15](cent milhares)

--FLAG
tmp(22) := STA & R2 & '0' & x"10";      -- STA %R2 @16      #Armazena o valor do acumulador em
MEM[16]=0 (flag)

--VARIAVEIS DE COMPARAÇÃO
tmp(23) := LDI & R0 & '0' & x"00";      -- LDI %R0 $0
tmp(24) := LDI & R2 & '0' & x"00";      -- LDI %R2 $0
tmp(25) := STA & R2 & '0' & x"00";      -- STA %R2 @0       #Armaena o valor do acumulador em
MEM[0]
tmp(26) := LDI & R2 & '0' & x"01";      -- LDI %R2 $1
tmp(27) := STA & R2 & '0' & x"01";      -- STA %R2 @1       #Armazena o valor do acumulador em
MEM[1]
tmp(28) := LDI & R2 & '0' & x"09";      -- LDI %R2 $9       #Carrega acumulador com valor 9
tmp(29) := STA & R2 & '0' & x"02";      -- STA %R2 @2       #Armazena o valor do acumulador em
MEM[2]
tmp(30) := LDI & R2 & '0' & x"0A";      -- LDI %R2 $10      #Carrega acumulador com valor 10
tmp(31) := STA & R2 & '0' & x"03";      -- STA %R2 @3       #Armazena o valor do acumulador em
MEM[3]
tmp(32) := LDI & R2 & '0' & x"04";      -- LDI %R2 $4       #Carrega acumulador com valor 10
tmp(33) := STA & R2 & '0' & x"04";      -- STA %R2 @4       #Armazena o valor do acumulador em
MEM[3]

--ARMAZENANDO LIMITES DE CONTAGEM
tmp(34) := LDI & R2 & '0' & x"0A";      -- LDI %R2 $10      #Carrega acumulador com valor 9
tmp(35) := STA & R2 & '0' & x"1E";      -- STA %R2 @30      #Armazena o limie de contagem em
MEM[30] (segundos)
```

```
tmp(36) := STA & R2 & '0' & x"20";      -- STA %R2 @32      #Armazena o limie de contagem em
MEM[32] (minutos)
tmp(37) := LDI & R2 & '0' & x"06";      -- LDI %R2 $6
tmp(38) := STA & R2 & '0' & x"1F";      -- STA %R2 @31      #Armazena o limie de contagem em
MEM[31] (segundos dezenas)
tmp(39) := STA & R2 & '0' & x"21";      -- STA %R2 @33      #Armazena o limie de contagem em
MEM[33] (minutos dezenas)
tmp(40) := LDI & R2 & '0' & x"0A";      -- LDI %R2 $10
tmp(41) := STA & R2 & '0' & x"22";      -- STA %R2 @34      #Armazena o limie de contagem em
MEM[34] (horas)
tmp(42) := LDI & R2 & '0' & x"02";      -- LDI %R2 $2
tmp(43) := STA & R2 & '0' & x"23";      -- STA %R2 @35      #Armazena o limie de contagem em
MEM[35] (horas dezenas)
--L:

--CHECA CLOCK
tmp(44) := LDA & R3 & '1' & x"65";      -- LDA %R3 .HABCLOCK      #Carrega acumulador com o valor
de KEY0
tmp(45) := ANDI & R3 & '0' & x"01";     -- ANDI %R3 $1      #Faz a operação AND com o valor 1
tmp(46) := CEQ & R3 & '0' & x"01";      -- CEQ %R3 @1       #OLha para se o valor do acumulador é
igual a 1 (Se key0 foi pressionado)
tmp(47) := JEQ & R3 & '0' & x"3B";      -- JEQ %R3 .INCREMENTA    #Se for igual pula para
fpga_reset
--B:

--CHECA KEY0
tmp(48) := LDA & R2 & '1' & x"60";      -- LDA %R2 .KEY0
tmp(49) := ANDI & R2 & '0' & x"01";     -- ANDI %R2 $1      #Faz a operação AND com o valor 1
tmp(50) := CEQ & R2 & '0' & x"01";      -- CEQ %R2 @1       #OLha para se o valor do acumulador é
igual a 1 (Se key0 foi pressionado)
tmp(51) := JEQ & R2 & '0' & x"81";      -- JEQ %R2 .AJUSTE_HORARIO

--CHECA FPGA_RESET
tmp(52) := LDA & R2 & '1' & x"64";      -- LDA %R2 .RST_FPGA      #Carrega acumulador com o valor
de FPGA_RESET
tmp(53) := ANDI & R2 & '0' & x"01";     -- ANDI %R2 $1      #Faz a operação AND com o valor 1
tmp(54) := CEQ & R2 & '0' & x"01";      -- CEQ %R2 @1       #Compara se o valor do acumulador é
igual a 0
tmp(55) := JEQ & R2 & '0' & x"00";      -- JEQ %R2 .SETUP   #Se  n foi pressionado pulta para
atualiza display

--ATUALIZA DISPLAY
tmp(56) := LDI & R2 & '0' & x"00";      -- LDI %R2 $0       #Carrega acumulador com valor 1
tmp(57) := JSR & R2 & '0' & x"75";      -- JSR %R2 .ATUALIZA_DISPLAY     #Chama a subrotina
atualiza display
tmp(58) := JMP & R2 & '0' & x"2C";      -- JMP %R2 .LOOP    #Volta para o loop principal
```

```
--loop INCREMENTO
--IEMENTA:
tmp(59) := STA & R3 & '1' & x"F8";      -- STA %R3 .CLEARCLOCK    #Limpa KEY 0

--INCREMENTA SEGUNDOS
tmp(60) := ADDI & R0 & '0' & x"01";     -- ADDI %R0 $1      #Incrementa o valor de REG[0] em 1
tmp(61) := CEQ & R0 & '0' & x"1E";      -- CEQ %R0 @30
tmp(62) := JEQ & R0 & '0' & x"40";      -- JEQ %R0 .INCREMENTA_DEZENAS   #Se REG[0] for igual a
REG[30] pula para incrementa dezenas segundos
tmp(63) := JMP & R0 & '0' & x"75";      -- JMP %R0 .ATUALIZA_DISPLAY     #Volta para o loop
principal
--IEMENTA_DEZENAS:
tmp(64) := LDI & R0 & '0' & x"00";      -- LDI %R0 $0       #Carrega acumulador com valor 0
tmp(65) := LDA & R1 & '0' & x"0B";      -- LDA %R1 @11      #Carrega o valor do acumulador em
MEM[30] (segundos)
tmp(66) := ADDI & R1 & '0' & x"01";     -- ADDI %R1 $1      #Soma o valor de REG[1] com o valor
de REG[0]
tmp(67) := STA & R1 & '0' & x"0B";      -- STA %R1 @11      #Armazena o valor do acumulador em
MEM[30] (segundos)
tmp(68) := CEQ & R1 & '0' & x"1F";      -- CEQ %R1 @31
tmp(69) := JEQ & R1 & '0' & x"47";      -- JEQ %R1 .INCREMENTA_MINUTOS   #Se REG[1] for igual a
REG[31] pula para incrementa minutos
tmp(70) := JMP & R1 & '0' & x"75";      -- JMP %R1 .ATUALIZA_DISPLAY
--IEMENTA_MINUTOS:
tmp(71) := LDI & R1 & '0' & x"00";      -- LDI %R1 $0       #Carrega acumulador com valor 0
tmp(72) := STA & R1 & '0' & x"0B";      -- STA %R1 @11      #Armazena o valor do acumulador em
MEM[30] (segundos)
tmp(73) := LDA & R1 & '0' & x"0C";      -- LDA %R1 @12
tmp(74) := ADDI & R1 & '0' & x"01";     -- ADDI %R1 $1      #Soma o valor de REG[1] com o valor
de REG[0]
tmp(75) := STA & R1 & '0' & x"0C";      -- STA %R1 @12      #Armazena o valor do acumulador em
MEM[30] (segundos)
tmp(76) := CEQ & R1 & '0' & x"20";      -- CEQ %R1 @32
tmp(77) := JEQ & R1 & '0' & x"4F";      -- JEQ %R1 .INCREMENTA_DEZENAS_MINUTOS  #Se REG[1] for
igual a REG[32] pula para incrementa dezenas minutos
tmp(78) := JMP & R1 & '0' & x"75";      -- JMP %R1 .ATUALIZA_DISPLAY
--IEMENTA_DEZENAS_MINUTOS:
tmp(79) := LDI & R1 & '0' & x"00";      -- LDI %R1 $0       #Carrega acumulador com valor 0
tmp(80) := STA & R1 & '0' & x"0C";      -- STA %R1 @12      #Armazena o valor do acumulador em
MEM[30] (segundos)
tmp(81) := LDA & R1 & '0' & x"0D";      -- LDA %R1 @13
tmp(82) := ADDI & R1 & '0' & x"01";     -- ADDI %R1 $1      #Soma o valor de REG[1] com o valor
de REG[0]
tmp(83) := STA & R1 & '0' & x"0D";      -- STA %R1 @13      #Armazena o valor do acumulador em
MEM[30] (segundos)
```

```
tmp(84) := CEQ & R1 & '0' & x"21";      -- CEQ %R1 @33
tmp(85) := JEQ & R1 & '0' & x"57";      -- JEQ %R1 .INCREMENTA_HORA      #Se REG[1] for igual a
REG[33] pula para incrementa horas
tmp(86) := JMP & R1 & '0' & x"75";      -- JMP %R1 .ATUALIZA_DISPLAY
--IEMENTA_HORA:
tmp(87) := LDI & R1 & '0' & x"00";      -- LDI %R1 $0       #Carrega acumulador com valor 0
tmp(88) := STA & R1 & '0' & x"0D";      -- STA %R1 @13      #Armazena o valor do acumulador em
MEM[30] (segundos)
tmp(89) := LDA & R1 & '0' & x"0E";      -- LDA %R1 @14
tmp(90) := ADDI & R1 & '0' & x"01";     -- ADDI %R1 $1      #Soma o valor de REG[1] com o valor
de REG[0]
tmp(91) := STA & R1 & '0' & x"0E";      -- STA %R1 @14      #Armazena o valor do acumulador em
MEM[30] (segundos)
tmp(92) := CEQ & R1 & '0' & x"04";      -- CEQ %R1 @4
tmp(93) := JEQ & R1 & '0' & x"68";      -- JEQ %R1 .CHECK_2       #Se REG[1] for igual a REG[4]
pula para incrementa horas
--SCHECK:
tmp(94) := LDA & R1 & '0' & x"0E";      -- LDA %R1 @14      #Armazena o valor do acumulador em
MEM[30] (segundos)
tmp(95) := CEQ & R1 & '0' & x"22";      -- CEQ %R1 @34
tmp(96) := JEQ & R1 & '0' & x"62";      -- JEQ %R1 .INCREMENTA_DEZENAS_HORAS   #Se REG[1] for
igual a REG[34] pula para incrementa dezenas horas
tmp(97) := JMP & R1 & '0' & x"75";      -- JMP %R1 .ATUALIZA_DISPLAY
--IEMENTA_DEZENAS_HORAS:
tmp(98) := LDI & R1 & '0' & x"00";      -- LDI %R1 $0       #Carrega acumulador com valor 0
tmp(99) := STA & R1 & '0' & x"0E";      -- STA %R1 @14      #Armazena o valor do acumulador em
MEM[30] (segundos)
tmp(100) := LDA & R1 & '0' & x"0F";     -- LDA %R1 @15
tmp(101) := ADDI & R1 & '0' & x"01";    -- ADDI %R1 $1      #Soma o valor de REG[1] com o valor
de REG[0]
tmp(102) := STA & R1 & '0' & x"0F";     -- STA %R1 @15      #Armazena o valor do acumulador em
MEM[30] (segundos)
tmp(103) := JMP & R1 & '0' & x"2C";     -- JMP %R1 .LOOP
--CK_2:
tmp(104) := LDA & R1 & '0' & x"0F";     -- LDA %R1 @15
tmp(105) := CEQ & R1 & '0' & x"23";     -- CEQ %R1 @35
tmp(106) := JEQ & R1 & '0' & x"6C";     -- JEQ %R1 .ZERADISPLAY   #Se REG[1] for igual a REG[35]
pula para incrementa horas
tmp(107) := JMP & R1 & '0' & x"5E";     -- JMP %R1 .SEM_CHECK
--ZDISPLAY:
tmp(108) := LDI & R2 & '0' & x"00";     -- LDI %R2 $0       #Carrega acumulador com valor 0
tmp(109) := LDI & R1 & '0' & x"00";     -- LDI %R1 $0       #Carrega acumulador com valor 0
tmp(110) := STA & R2 & '0' & x"0A";     -- STA %R2 @10      #Armazena o valor do acumulador em
MEM[10] (segundos)
tmp(111) := STA & R2 & '0' & x"0C";     -- STA %R2 @12      #Armazena o valor do acumulador em
MEM[11] (dezenas segundos)
```

```
tmp(112) := STA & R2 & '0' & x"0D";     -- STA %R2 @13      #Armazena o valor do acumulador em
MEM[12] (minutos)
tmp(113) := STA & R2 & '0' & x"0E";     -- STA %R2 @14      #Armazena o valor do acumulador em
MEM[13] (dezenas minutos)
tmp(114) := STA & R2 & '0' & x"0B";     -- STA %R2 @11      #Armazena o valor do acumulador em
MEM[14] (horas)
tmp(115) := STA & R2 & '0' & x"0F";     -- STA %R2 @15      #Armazena o valor do acumulador em
MEM[15] (dezenas horas)
tmp(116) := JMP & R2 & '0' & x"75";     -- JMP %R2 .ATUALIZA_DISPLAY
--ALIZA_DISPLAY:
tmp(117) := STA & R0 & '1' & x"20";     -- STA %R0 .HEX0    #Armazena o valor do REGISTRADOR em
HEX0
tmp(118) := LDA & R2 & '0' & x"0B";     -- LDA %R2 @11
tmp(119) := STA & R2 & '1' & x"21";     -- STA %R2 .HEX1    #Armazena o valor do REGISTRADOR em
HEX1
tmp(120) := LDA & R2 & '0' & x"0C";     -- LDA %R2 @12
tmp(121) := STA & R2 & '1' & x"22";     -- STA %R2 .HEX2    #Armazena o valor do REGISTRADOR em
HEX2
tmp(122) := LDA & R2 & '0' & x"0D";     -- LDA %R2 @13
tmp(123) := STA & R2 & '1' & x"23";     -- STA %R2 .HEX3    #Armazena o valor do REGISTRADOR em
HEX3
tmp(124) := LDA & R2 & '0' & x"0E";     -- LDA %R2 @14
tmp(125) := STA & R2 & '1' & x"24";     -- STA %R2 .HEX4    #Armazena o valor do REGISTRADOR em
HEX4
tmp(126) := LDA & R2 & '0' & x"0F";     -- LDA %R2 @15
tmp(127) := STA & R2 & '1' & x"25";     -- STA %R2 .HEX5    #Armazena o valor do REGISTRADOR em
HEX5
tmp(128) := JMP & R2 & '0' & x"2C";     -- JMP %R2 .LOOP
--ATE_HORARIO:
tmp(129) := STA & R2 & '1' & x"FF";     -- STA %R2 .CLEARKEY0
tmp(130) := LDI & R1 & '0' & x"01";     -- LDI %R1 $1
--AA_LED_SEGUNDO:
tmp(131) := STA & R1 & '1' & x"00";     -- STA %R1 .LED07
tmp(132) := LDA & R2 & '1' & x"60";     -- LDA %R2 .KEY0
tmp(133) := ANDI & R2 & '0' & x"01";    -- ANDI %R2 $1      #Faz a operação AND com o valor 1
tmp(134) := CEQ & R2 & '0' & x"00";     -- CEQ %R2 @0
tmp(135) := LDA & R3 & '1' & x"40";     -- LDA %R3 .SW0-7
tmp(136) := JEQ & R2 & '0' & x"83";     -- JEQ  %R2 .ATIVA_LED_SEGUNDO
tmp(137) := STA & R3 & '0' & x"14";     -- STA %R3 @20
tmp(138) := LDA & R0 & '0' & x"14";     -- LDA %R0 @20
tmp(139) := STA & R0 & '1' & x"20";     -- STA %R0 .HEX0
tmp(140) := STA & R2 & '1' & x"FF";     -- STA %R2 .CLEARKEY0
tmp(141) := LDI & R1 & '0' & x"03";     -- LDI %R1 $3
--AA_LED_DEZENA_SEGUNDO:
tmp(142) := STA & R1 & '1' & x"00";     -- STA %R1 .LED07
tmp(143) := LDA & R2 & '1' & x"60";     -- LDA %R2 .KEY0
```

```
tmp(144) := ANDI & R2 & '0' & x"01";    -- ANDI %R2 $1      #Faz a operação AND com o valor 1
tmp(145) := CEQ & R2 & '0' & x"00";     -- CEQ %R2 @0
tmp(146) := LDA & R3 & '1' & x"40";     -- LDA %R3 .SW0-7
tmp(147) := JEQ & R2 & '0' & x"8E";     -- JEQ  %R2 .ATIVA_LED_DEZENA_SEGUNDO
tmp(148) := STA & R3 & '0' & x"0B";     -- STA %R3 @11
tmp(149) := STA & R3 & '1' & x"21";     -- STA %R3 .HEX1
tmp(150) := STA & R2 & '1' & x"FF";     -- STA %R2 .CLEARKEY0
tmp(151) := LDI & R1 & '0' & x"07";     -- LDI %R1 $7
--AA_LED_MINUTO:
tmp(152) := STA & R1 & '1' & x"00";     -- STA %R1 .LED07
tmp(153) := LDA & R2 & '1' & x"60";     -- LDA %R2 .KEY0
tmp(154) := ANDI & R2 & '0' & x"01";    -- ANDI %R2 $1      #Faz a operação AND com o valor 1
tmp(155) := CEQ & R2 & '0' & x"00";     -- CEQ %R2 @0
tmp(156) := LDA & R3 & '1' & x"40";     -- LDA %R3 .SW0-7
tmp(157) := JEQ & R2 & '0' & x"98";     -- JEQ  %R2 .ATIVA_LED_MINUTO
tmp(158) := STA & R3 & '0' & x"0C";     -- STA %R3 @12
tmp(159) := STA & R3 & '1' & x"22";     -- STA %R3 .HEX2
tmp(160) := STA & R2 & '1' & x"FF";     -- STA %R2 .CLEARKEY0
tmp(161) := LDI & R1 & '0' & x"0F";     -- LDI %R1 $15
--AA_LED_DEZENA_MINUTO:
tmp(162) := STA & R1 & '1' & x"00";     -- STA %R1 .LED07
tmp(163) := LDA & R2 & '1' & x"60";     -- LDA %R2 .KEY0
tmp(164) := ANDI & R2 & '0' & x"01";    -- ANDI %R2 $1      #Faz a operação AND com o valor 1
tmp(165) := CEQ & R2 & '0' & x"00";     -- CEQ %R2 @0
tmp(166) := LDA & R3 & '1' & x"40";     -- LDA %R3 .SW0-7
tmp(167) := JEQ & R2 & '0' & x"A2";     -- JEQ  %R2 .ATIVA_LED_DEZENA_MINUTO
tmp(168) := STA & R3 & '0' & x"0D";     -- STA %R3 @13
tmp(169) := STA & R3 & '1' & x"23";     -- STA %R3 .HEX3
tmp(170) := STA & R2 & '1' & x"FF";     -- STA %R2 .CLEARKEY0
tmp(171) := LDI & R1 & '0' & x"1F";     -- LDI %R1 $31
--AA_LED_HORA:
tmp(172) := STA & R1 & '1' & x"00";     -- STA %R1 .LED07
tmp(173) := LDA & R2 & '1' & x"60";     -- LDA %R2 .KEY0
tmp(174) := ANDI & R2 & '0' & x"01";    -- ANDI %R2 $1      #Faz a operação AND com o valor 1
tmp(175) := CEQ & R2 & '0' & x"00";     -- CEQ %R2 @0
tmp(176) := LDA & R3 & '1' & x"40";     -- LDA %R3 .SW0-7
tmp(177) := JEQ & R2 & '0' & x"AC";     -- JEQ  %R2 .ATIVA_LED_HORA
tmp(178) := STA & R3 & '0' & x"0E";     -- STA %R3 @14
tmp(179) := STA & R3 & '1' & x"24";     -- STA %R3 .HEX4
tmp(180) := STA & R2 & '1' & x"FF";     -- STA %R2 .CLEARKEY0
tmp(181) := LDI & R1 & '0' & x"3F";     -- LDI %R1 $63
--AA_LED_DEZENA_HORA:
tmp(182) := STA & R1 & '1' & x"00";     -- STA %R1 .LED07
tmp(183) := LDA & R2 & '1' & x"60";     -- LDA %R2 .KEY0
tmp(184) := ANDI & R2 & '0' & x"01";    -- ANDI %R2 $1      #Faz a operação AND com o valor 1
tmp(185) := CEQ & R2 & '0' & x"00";     -- CEQ %R2 @0
```

```
tmp(186) := LDA & R3 & '1' & x"40";     -- LDA %R3 .SW0-7
tmp(187) := JEQ & R2 & '0' & x"B6";     -- JEQ  %R2 .ATIVA_LED_DEZENA_HORA
tmp(188) := STA & R3 & '0' & x"0F";     -- STA %R3 @15
tmp(189) := STA & R3 & '1' & x"25";     -- STA %R3 .HEX5
tmp(190) := LDA & R0 & '0' & x"14";     -- LDA %R0 @20
tmp(191) := STA & R2 & '1' & x"FF";     -- STA %R2 .CLEARKEY0
tmp(192) := LDI & R1 & '0' & x"00";     -- LDI %R1 $0
tmp(193) := STA & R1 & '1' & x"00";     -- STA $R1 .LED07
tmp(194) := JMP & R2 & '0' & x"2C";     -- JMP %R2 .LOOP
```

6. Código em assembler

```
# Segundos = REG[0] REG[1]
# Uso geral = REG[2]
# Clock_check = REG[3]
# Armazenanmento de valor na RAM =  MEM[10] unidades
#                                   MEM[11] dezenas
#                                   MEM[12] centenas
#                                   MEM[13] milhares
#                                   MEM[14] dez milhares
#                                   MEM[15] cent milhares
#                                   MEM[16] flag
# Armazenamento de limite na RAM = MEM[30] unidades
#                                  MEM[31] dezenas
#                                  MEM[32] centenas
#                                  MEM[33] milhares
#                                  MEM[34] dez milhares
#                                  MEM[35] cent milhares
#



SETUP:
#LIMPA BOTOESBACK
STA %R2 .CLEARKEY0 #Limpa KEY 0
STA %R2 .CLEARKEY1 #Limpa KEY 1
STA %R2 .CLEARKEY2 #Limpa KEY 2
STA %R2 .CLEARKEY3 #Limpa KEY 3
STA %R2 .CLEARFPGA #Limpa FPGA_RESET

LDI %R2 $0 #Carrega acumulador com valor 0

#ESCREVE 0 NOS DISPLAYS
STA %R2 .HEX0 #Armazena o valor 0 no HEX0
STA %R2 .HEX1 #Armazena o valor 0 no HEX1
STA %R2 .HEX2 #Armazena o valor 0 no HEX2
STA %R2 .HEX3 #Armazena o valor 0 no HEX3
STA %R2 .HEX4 #Armazena o valor 0 no HEX4
STA %R2 .HEX5 #Armazena o valor 0 no HEX5

#APAGANDO OS LEDS
LDI %R2 $0
STA %R2 .LED07 #Armazena o valor 0 no LEDR7~0
STA %R2 .LED8 #Armazena o valor 0 no LEDR8
STA %R2 .LED9 #Armazena o valor 0 no LEDR9

#VARIAVEIS QUE ARMAZENAM O VALOR DO DISPLAY
STA %R2 @10 #Armazena o valor do acumulador em MEM[10](unidades)
STA %R2 @11 #Armazena o valor do acumulador em MEM[11](dezenas)
STA %R2 @12 #Armazena o valor do acumulador em MEM[12](centenas)
STA %R2 @13 #Armazena o valor do acumulador em MEM[13](milhares)
STA %R2 @14 #Armazena o valor do acumulador em MEM[14](dez milhares)
STA %R2 @15 #Armazena o valor do acumulador em MEM[15](cent milhares)
```

```
#FLAG
STA %R2 @16 #Armazena o valor do acumulador em MEM[16]=0 (flag)

#VARIAVEIS DE COMPARAÇÃO
LDI %R0 $0
LDI %R2 $0
STA %R2 @0 #Armaena o valor do acumulador em MEM[0]

LDI %R2 $1
STA %R2 @1 #Armazena o valor do acumulador em MEM[1]

LDI %R2 $9 #Carrega acumulador com valor 9
STA %R2 @2 #Armazena o valor do acumulador em MEM[2]

LDI %R2 $10 #Carrega acumulador com valor 10
STA %R2 @3 #Armazena o valor do acumulador em MEM[3]

LDI %R2 $4 #Carrega acumulador com valor 10
STA %R2 @4 #Armazena o valor do acumulador em MEM[3]



#ARMAZENANDO LIMITES DE CONTAGEM
LDI %R2 $10 #Carrega acumulador com valor 9
STA %R2 @30  #Armazena o limie de contagem em MEM[30] (segundos)
STA %R2 @32 #Armazena o limie de contagem em MEM[32] (minutos)
LDI %R2 $6
STA %R2 @31 #Armazena o limie de contagem em MEM[31] (segundos dezenas)
STA %R2 @33 #Armazena o limie de contagem em MEM[33] (minutos dezenas)
LDI %R2 $10
STA %R2 @34 #Armazena o limie de contagem em MEM[34] (horas)
LDI %R2 $2
STA %R2 @35 #Armazena o limie de contagem em MEM[35] (horas dezenas)

LOOP:
#CHECA CLOCK
LDA %R3 .HABCLOCK #Carrega acumulador com o valor de KEY0
ANDI %R3 $1 #Faz a operação AND com o valor 1
CEQ %R3 @1 #OLha para se o valor do acumulador é igual a 1 (Se key0 foi pressionado)
JEQ %R3 .INCREMENTA #Se for igual pula para fpga_reset
BACK:

#CHECA KEY0
LDA %R2 .KEY0
ANDI %R2 $1 #Faz a operação AND com o valor 1
CEQ %R2 @1 #OLha para se o valor do acumulador é igual a 1 (Se key0 foi pressionado)
JEQ %R2 .AJUSTE_HORARIO

#CHECA FPGA_RESET
LDA %R2 .RST_FPGA #Carrega acumulador com o valor de FPGA_RESET
ANDI %R2 $1 #Faz a operação AND com o valor 1
CEQ %R2 @1 #Compara se o valor do acumulador é igual a 0
JEQ %R2 .SETUP #Se  n foi pressionado pulta para atualiza display

#ATUALIZA DISPLAY
LDI %R2 $0 #Carrega acumulador com valor 1
JSR %R2 .ATUALIZA_DISPLAY #Chama a subrotina atualiza display
JMP %R2 .LOOP #Volta para o loop principal



#loop INCREMENTO
INCREMENTA:
STA %R3 .CLEARCLOCK #Limpa KEY 0

#INCREMENTA SEGUNDOS
ADDI %R0 $1 #Incrementa o valor de REG[0] em 1
CEQ %R0 @30
JEQ %R0 .INCREMENTA_DEZENAS #Se REG[0] for igual a REG[30] pula para incrementa dezenas segundos
```

```
JMP %R0 .ATUALIZA_DISPLAY #Volta para o loop principal

INCREMENTA_DEZENAS:
LDI %R0 $0 #Carrega acumulador com valor 0

LDA %R1 @11 #Carrega o valor do acumulador em MEM[30] (segundos)
ADDI %R1 $1 #Soma o valor de REG[1] com o valor de REG[0]
STA %R1 @11 #Armazena o valor do acumulador em MEM[30] (segundos)
CEQ %R1 @31
JEQ %R1 .INCREMENTA_MINUTOS #Se REG[1] for igual a REG[31] pula para incrementa minutos
JMP %R1 .ATUALIZA_DISPLAY

INCREMENTA_MINUTOS:
LDI %R1 $0 #Carrega acumulador com valor 0
STA %R1 @11 #Armazena o valor do acumulador em MEM[30] (segundos)

LDA %R1 @12
ADDI %R1 $1 #Soma o valor de REG[1] com o valor de REG[0]
STA %R1 @12 #Armazena o valor do acumulador em MEM[30] (segundos)
CEQ %R1 @32
JEQ %R1 .INCREMENTA_DEZENAS_MINUTOS #Se REG[1] for igual a REG[32] pula para incrementa dezenas
minutos
JMP %R1 .ATUALIZA_DISPLAY

INCREMENTA_DEZENAS_MINUTOS:
LDI %R1 $0 #Carrega acumulador com valor 0
STA %R1 @12 #Armazena o valor do acumulador em MEM[30] (segundos)

LDA %R1 @13
ADDI %R1 $1 #Soma o valor de REG[1] com o valor de REG[0]
STA %R1 @13 #Armazena o valor do acumulador em MEM[30] (segundos)
CEQ %R1 @33
JEQ %R1 .INCREMENTA_HORA #Se REG[1] for igual a REG[33] pula para incrementa horas
JMP %R1 .ATUALIZA_DISPLAY



INCREMENTA_HORA:
LDI %R1 $0 #Carrega acumulador com valor 0
STA %R1 @13 #Armazena o valor do acumulador em MEM[30] (segundos)

LDA %R1 @14
ADDI %R1 $1 #Soma o valor de REG[1] com o valor de REG[0]
STA %R1 @14 #Armazena o valor do acumulador em MEM[30] (segundos)
CEQ %R1 @4
JEQ %R1 .CHECK_2 #Se REG[1] for igual a REG[4] pula para incrementa horas

SEM_CHECK:
LDA %R1 @14 #Armazena o valor do acumulador em MEM[30] (segundos)
CEQ %R1 @34
JEQ %R1 .INCREMENTA_DEZENAS_HORAS #Se REG[1] for igual a REG[34] pula para incrementa dezenas horas
JMP %R1 .ATUALIZA_DISPLAY

INCREMENTA_DEZENAS_HORAS:
LDI %R1 $0 #Carrega acumulador com valor 0
STA %R1 @14 #Armazena o valor do acumulador em MEM[30] (segundos)

LDA %R1 @15
ADDI %R1 $1 #Soma o valor de REG[1] com o valor de REG[0]
STA %R1 @15 #Armazena o valor do acumulador em MEM[30] (segundos)
JMP %R1 .LOOP

CHECK_2:
LDA %R1 @15
CEQ %R1 @35
JEQ %R1 .ZERADISPLAY #Se REG[1] for igual a REG[35] pula para incrementa horas
JMP %R1 .SEM_CHECK

ZERADISPLAY:
LDI %R2 $0 #Carrega acumulador com valor 0
```

```
LDI %R1 $0 #Carrega acumulador com valor 0
STA %R2 @10 #Armazena o valor do acumulador em MEM[10] (segundos)
STA %R2 @12 #Armazena o valor do acumulador em MEM[11] (dezenas segundos)
STA %R2 @13 #Armazena o valor do acumulador em MEM[12] (minutos)
STA %R2 @14 #Armazena o valor do acumulador em MEM[13] (dezenas minutos)
STA %R2 @11 #Armazena o valor do acumulador em MEM[14] (horas)
STA %R2 @15 #Armazena o valor do acumulador em MEM[15] (dezenas horas)
JMP %R2 .ATUALIZA_DISPLAY



ATUALIZA_DISPLAY:
STA %R0 .HEX0 #Armazena o valor do REGISTRADOR em HEX0

LDA %R2 @11
STA %R2 .HEX1 #Armazena o valor do REGISTRADOR em HEX1

LDA %R2 @12
STA %R2 .HEX2 #Armazena o valor do REGISTRADOR em HEX2

LDA %R2 @13
STA %R2 .HEX3 #Armazena o valor do REGISTRADOR em HEX3

LDA %R2 @14
STA %R2 .HEX4 #Armazena o valor do REGISTRADOR em HEX4

LDA %R2 @15
STA %R2 .HEX5 #Armazena o valor do REGISTRADOR em HEX5

JMP %R2 .LOOP

AJUSTE_HORARIO:
STA %R2 .CLEARKEY0
LDI %R1 $1
ATIVA_LED_SEGUNDO:
STA %R1 .LED07

LDA %R2 .KEY0
ANDI %R2 $1 #Faz a operação AND com o valor 1
CEQ %R2 @0
LDA %R3 .SW0-7
JEQ  %R2 .ATIVA_LED_SEGUNDO
STA %R3 @20
LDA %R0 @20
STA %R0 .HEX0

STA %R2 .CLEARKEY0
LDI %R1 $3
ATIVA_LED_DEZENA_SEGUNDO:
STA %R1 .LED07

LDA %R2 .KEY0
ANDI %R2 $1 #Faz a operação AND com o valor 1
CEQ %R2 @0
LDA %R3 .SW0-7
JEQ  %R2 .ATIVA_LED_DEZENA_SEGUNDO
STA %R3 @11
STA %R3 .HEX1

STA %R2 .CLEARKEY0
LDI %R1 $7
ATIVA_LED_MINUTO:
STA %R1 .LED07

LDA %R2 .KEY0
ANDI %R2 $1 #Faz a operação AND com o valor 1
CEQ %R2 @0
LDA %R3 .SW0-7
JEQ  %R2 .ATIVA_LED_MINUTO
STA %R3 @12
```

```
STA %R3 .HEX2

STA %R2 .CLEARKEY0
LDI %R1 $15
ATIVA_LED_DEZENA_MINUTO:
STA %R1 .LED07

LDA %R2 .KEY0
ANDI %R2 $1 #Faz a operação AND com o valor 1
CEQ %R2 @0
LDA %R3 .SW0-7
JEQ  %R2 .ATIVA_LED_DEZENA_MINUTO
STA %R3 @13
STA %R3 .HEX3

STA %R2 .CLEARKEY0
LDI %R1 $31
ATIVA_LED_HORA:
STA %R1 .LED07

LDA %R2 .KEY0
ANDI %R2 $1 #Faz a operação AND com o valor 1
CEQ %R2 @0
LDA %R3 .SW0-7
JEQ  %R2 .ATIVA_LED_HORA
STA %R3 @14
STA %R3 .HEX4

STA %R2 .CLEARKEY0
LDI %R1 $63
ATIVA_LED_DEZENA_HORA:
STA %R1 .LED07

LDA %R2 .KEY0
ANDI %R2 $1 #Faz a operação AND com o valor 1
CEQ %R2 @0
LDA %R3 .SW0-7
JEQ  %R2 .ATIVA_LED_DEZENA_HORA
STA %R3 @15
STA %R3 .HEX5

LDA %R0 @20
STA %R2 .CLEARKEY0
LDI %R1 $0
STA $R1 .LED07

JMP %R2 .LOOP
```