

PROJETO 1 - ENTREGA FINAL

Entregue: 08/05/2023

PROJETO DE UM RELÓGIO

Gabriel Onishi | gabrielhso@al.insper.edu.br

Sarah Pimenta | sarahcp@al.insper.edu.br

DESCRIÇÃO DO PROJETO

Esse relatório é parte da entrega final do Projeto 1 da disciplina de Design de Computadores, parte da graduação em Engenharia de Computação pelo Insper. Trata-se da descrição de um computador com as funcionalidades básicas de um relógio, como mostrar hora atual, configurar hora e configurar um alarme.

Esse relatório é composto por uma descrição do processador utilizado, explorando sua arquitetura e fluxo de dados, passando para um detalhamento sobre as instruções disponíveis e o código principal do programa. Por fim, há um breve manual de instruções do relógio.

PROCESSADOR

Arquitetura do Processador e Formato de Instrução

O contador projetado tem topologia baseada em registradores-memória. Nessa arquitetura, há mais do que um registrador, podendo ocorrer operações entre ele e uma posição de memória ou valor imediato. No caso, o computador projetado possui 4 registradores.

Toda instrução é formada de um código de operação (*opcode*) unida ao registrador utilizado e um endereço. Os *opcodes* são nada mais que instruções, como carregamento de uma posição de memória ou pulo de uma linha de instrução para outra. Para cada operação, é necessário explicitar também qual o registrador envolvido. O endereço, por sua vez, especifica uma posição de memória ou um valor imediato. No caso do computador desenhado para esse projeto, a instrução é composta por 4 bits, a especificação do registrador é de 2 bits, enquanto o endereçamento é de 12 bits. O formato de instrução pode ser melhor entendido através da Figura 1 abaixo.

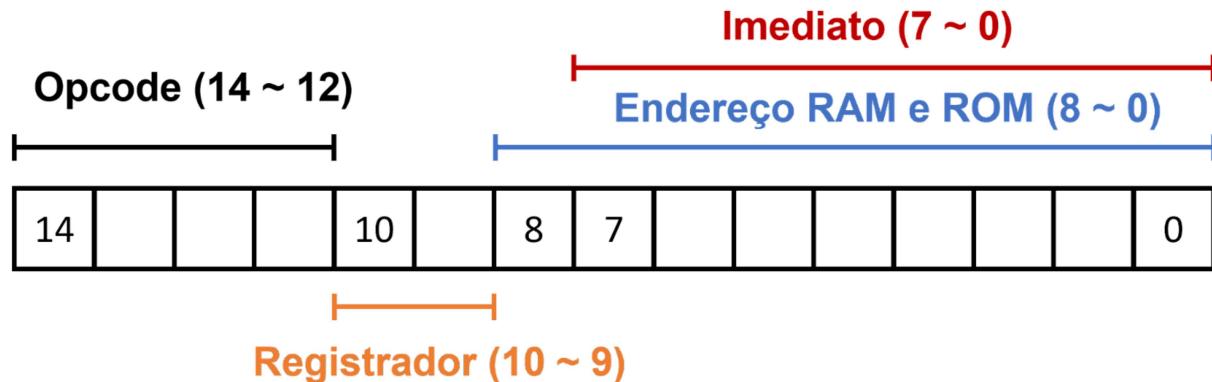


Figura 1 - Esquema de Formato de Instrução

Fluxo de Dados

A CPU do contador é composta de diversos componentes, detalhados individualmente abaixo:

- Unidade Lógica Aritmética (ULA)

A ULA trata os dados vindos do acumulador (conectado à sua entrada B) com os dados do valor imediato ou de uma posição de memória (conectados à entrada A e selecionados através do uso de um MUX). Esse componente é capaz de fazer operações lógicas, selecionadas a partir da instrução mandada, e repassa o novo resultado para o acumulador.

- Unidade de Controle

Esse componente faz o tratamento dos códigos de instrução enviados. Ele é composto por um decodificador, que faz a associação do *opcode* e a ativação dos respectivos pontos de controle.

- Lógica de Desvio

A lógica de desvio controla o MUX de próxima instrução, definindo qual o endereço de linha salvo no Program Counter.

- Program Counter (PC)

O PC nada mais é do que um registrador cujo único objetivo é armazenar a posição da próxima linha de instruções. Ele recebe essa informação da lógica de desvio da unidade de controle do fluxo de dados e a envia para a ROM.

O fluxo de dados pode ser melhor compreendido pela Figura 2, abaixo.

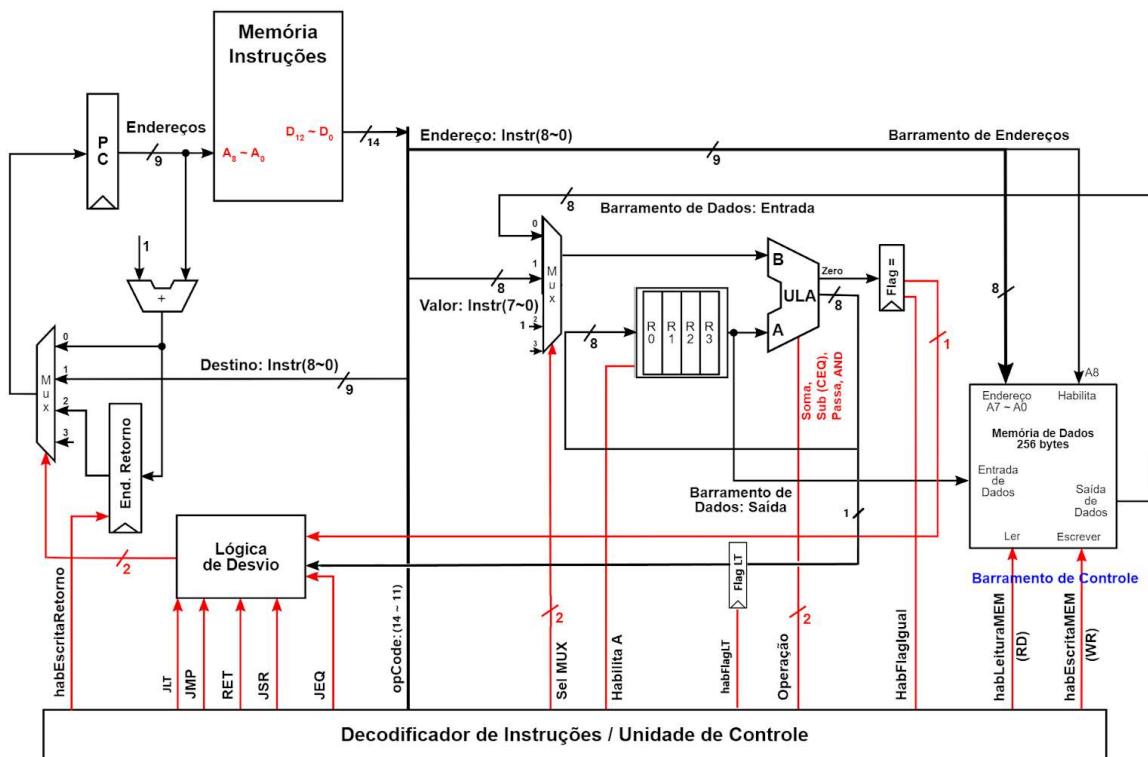


Figura 2 - Esquema ilustrativo da CPU

Pontos de Controle

Este processador apresenta 14 pontos de controle

| Bit | Nome | Descrição |
|-----|---------------|--|
| 0 | habEscritaMEM | Habilita a escrita da memória RAM |
| 1 | habLeituraMEM | Habilita a leitura da memória RAM |
| 2 | habFlagIgual | Habilita registrador de flag igual na saída da ULA |
| 3 | habFlagLT | Habilita registrador de flag less than na saída da ULS |
| 4 | Operação | Faz a seleção da operação (00: Subtração, 01: Soma ou INC, 10: Passa, 11: ANDI) |

| | | |
|----|---------------|---|
| 5 | Operação | Faz a seleção da operação (00: Subtração, 01: Soma ou INC, 10: Passa, 11: ANDI) |
| 6 | Habilita A | Habilita a escrita no banco de registradores |
| 7 | Sel MUX | Faz a seleção do MUX de entrada na ULA (00: Recebe dados da memória, 01: Recebe valor imediato, 10: Recebe valor 1, 11: Invalido) |
| 8 | SelMUX | Faz a seleção do MUX de entrada na ULA (00: Recebe dados da memória, 01: Recebe valor imediato, 10: Recebe valor 1, 11: Invalido) |
| 9 | JEQ | Ativa desvio de instrução quando flag igual está ativada |
| 10 | JSR | Ativa desvio de instrução para sub-rotina |
| 11 | RET | Ativa desvio de instrução de retorno de subrotina |
| 12 | JMP | Ativa desvio de instrução incondicional |
| 13 | JLT | Ativa desvio de instrução quando flag less than está ativada |
| 14 | habEscritaRet | Habilita registrador de endereço de retorno |

Conexão com Periféricos

A CPU se conecta com 4 segmentos de periféricos que podem ser ativados e desativados com as instruções do programa. O contador projetado utiliza de quatro periféricos, descritos individualmente abaixo.

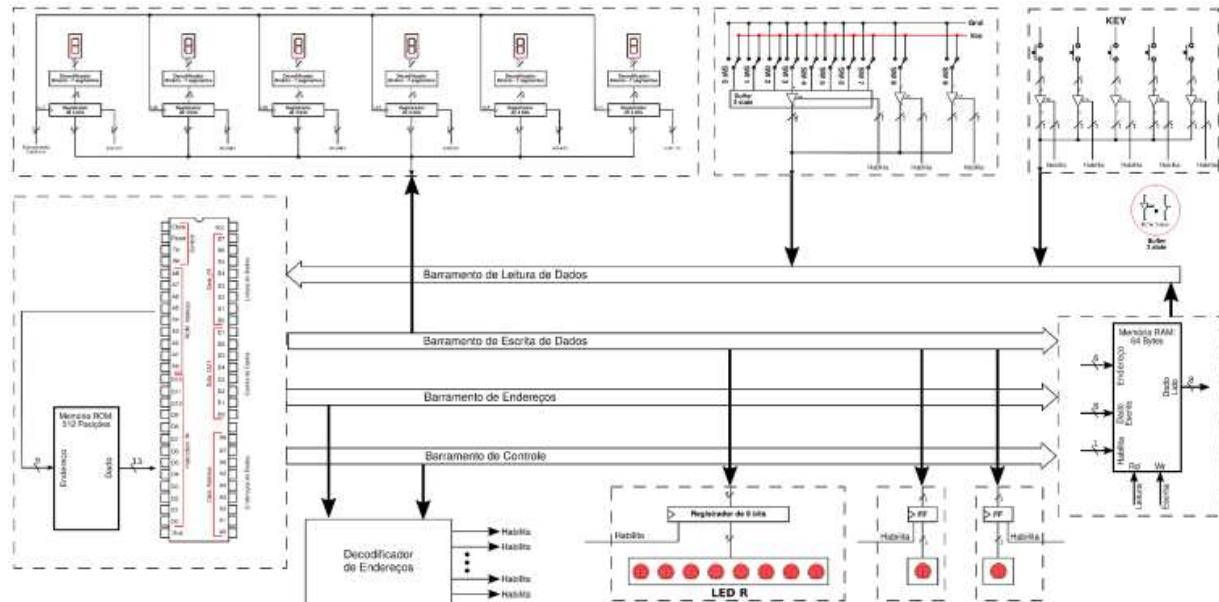


Figura 3 - Diagrama de Conexão com Periféricos

A FPGA utilizada dispõe de 10 LEDs cujo estado atual é mantido em registradores. O estado dos LEDs 0 até 7 são mantidos em um só registrador, enquanto os LEDs 8 e 9 possuem registradores individuais para sua ativação.

A placa também dispõe de 6 displays de sete segmentos. O estado de cada um deles é armazenado em um registrador de 4 bits, que depois passam por um decodificador que associa o número armazenado com os segmentos ativados no display.

Também dispõe de 10 chaves (SW0 ~ SW9). O estado dessas chaves podem ser lidos (não escritos) pelo computador, sendo utilizado buffers 3 state para garantir um tratamento dos dados.

Por fim, a placa também conta com 5 botões: KEY 0 ~ KEY 3 e um botão FPGA Reset. Esses botões também são apenas para leitura, com estados também administrados por buffers 3 state.

INSTRUÇÕES SUPORTADAS

Este processador tem um total de 15 instruções, a descrição de cada uma delas pode ser encontrada na tabela abaixo:

| Instrução | Código Binário | Mnemônico | Argumento | Exemplo |
|---------------------------------|----------------|-----------|------------------------|--------------|
| Sem Operação | 0000 | NOP | - | NOP |
| Carrega valor da memória para A | 0001 | LDA | Posição de Memória | LDA R0, @5 |
| Soma A e B e armazena em A | 0010 | SOMA | Posição de Memória | SOMA R0, @5 |
| Subtrai B de A e armazena em A | 0011 | SUB | Posição de Memória | SUB R0, @5 |
| Carrega valor imediato para A | 0100 | LDI | Valor Imediato | LDA R0, \$6 |
| Salva valor de A para a memória | 0101 | STA | Posição de Memória | STA R0, @5 |
| Desvio de execução | 0110 | JMP | Posição de Memória ROM | JMP @LABEL |
| Desvio condicional de execução | 0111 | JEQ | Posição de Memória ROM | JEQ @LABEL |
| Comparação | 1000 | CEQ | Posição de Memória | CEQ R0, @5 |
| Chamada de Sub Rotina | 1001 | JSR | Posição de Memória ROM | JSR @LABEL |
| Retorno de Subrotina | 1010 | RET | - | RET |
| Operação Lógica AND | 1011 | ANDI | Valor Imediato | ANDI R0, \$1 |
| Incremento | 1100 | INC | - | INC |
| Comparação Menor Que | 1101 | CLT | Posição de Memória | CLT R0, @5 |
| Desvio Condicional de menor que | 1110 | JLT | Posição de Memória ROM | JLT @LABEL |

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 |
| 0 ~ 5 | Valores de armazenamento do horário | 8 bits | Leitura/Escrita | 0 |
| 6 ~ 18 | Valores para comparação | 8 bits | Leitura | 0 |
| 19 | Reservado | 8 bits | Leitura | 0 |
| 20 ~ 25 | Valores de armazenamento do despertador | 8 bit | 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 (indica alarme) | 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 ~ 509 | Reservado | – | – | 7 |
| 508 | Limpa Leitura KEY3 | – | Escrita | 7 |
| 509 | Limpa Leitura KEY2 | – | Escrita | 7 |
| 510 | Limpa Leitura KEY1 | – | Escrita | 7 |
| 511 | Limpa Leitura KEY0 | – | Escrita | 7 |

PROGRAMA DA ROTINA PRINCIPAL

O contador roda o seguinte código em assembly:

```
# Setup

SETUP:
    # Zerando os displays de sete segmentos

    LDI R0, $0          #Carrega o acumulador com o valor 0
    STA R0, @288        #Armazena o valor do acumulador em HEX0
    STA R0, @289        #Armazena o valor do acumulador em HEX1
    STA R0, @290        #Armazena o valor do acumulador em HEX2
    STA R0, @291        #Armazena o valor do acumulador em HEX3
    STA R0, @292        #Armazena o valor do acumulador em HEX4
    STA R0, @293        #Armazena o valor do acumulador em HEX5

    #Limpar os botoes (rst do FF)

    STA R0, @511        #Limpando FF do KEY0
    STA R0, @510        #Limpando FF do KEY1
    STA R0, @509        #Limpando FF do KEY2
    STA R0, @508        #Limpando FF do KEY3

    #Inicializando as variaveis do tempo

    STA R0, @0          #Armazena o valor do acumulador em MEM[0] (segundos unidade)
    STA R0, @1          #Armazena o valor do acumulador em MEM[1] (segundos dezenas)
    STA R0, @2          #Armazena o valor do acumulador em MEM[2] (minutos unidade)
    STA R0, @3          #Armazena o valor do acumulador em MEM[3] (minutos dezenas)
    STA R0, @4          #Armazena o valor do acumulador em MEM[4] (hora unidade)
    STA R0, @5          #Armazena o valor do acumulador em MEM[5] (hora dezenas)

    #Apagando os LEDs

    STA R0, @257        #Armazena o valor do bit0 do acumulador no LDR8 (indica Overflow)
```

```
STA R0, @258      #Armazena o valor do bit0 do acumulador no LDR9 (indica Limite Atingido)
```

```
STA R0, @256      #Armazena o valor do acumulador no LDR0 ~ LDR7
```

```
#Inicializando as variaveis de comparação
```

```
LDI R0, $10      #Carrega o acumulador com o valor 10
```

```
STA R0, @6       #Armazena o valor do acumulador em MEM[6] (segundos unidade)
```

```
STA R0, @8       #Armazena o valor do acumulador em MEM[8] (minutos unidade)
```

```
LDI R0, $6       #Carrega o acumulador com o valor 6
```

```
STA R0, @7       #Armazena o valor do acumulador em MEM[7] (segundos dezenas)
```

```
STA R0, @9       #Armazena o valor do acumulador em MEM[9] (minutos dezenas)
```

```
LDI R0, $4       #Carrega o acumulador com o valor 6
```

```
STA R0, @10      #Armazena o valor do acumulador em MEM[10] (hora unidade)
```

```
LDI R0, $3       #Carrega o acumulador com o valor 3
```

```
STA R0, @11      #Armazena o valor do acumulador em MEM[11] (hora dezenas)
```

```
# Variáveis para comparação de setar hora
```

```
LDI R0, $9
```

```
STA R0, @15      #Valor para comparação setar segundo, minuto
```

```
LDI R0, $5
```

```
STA R0, @16      #Valor para comparação setar dez segundo, minuto
```

```
LDI R0, $3
```

```
STA R0, @17      #valor para comparação setar hora
```

```
LDI R0, $2
```

```
STA R0, @18      #Valor para comparação setar dez hora
```

```
#Salvando variaveis utilizadas em comparacoes
```

```
LDI R0, $0       #Carrega o acumulador com o valor 0 para fazer comparacao com o botao
```

```
STA R0, @13      #Armazena o valor do acumulador em MEM[13] (valor para comparacao do aperto do botao)
```

```
LDI R0, $1       #Carrega o acumulador com o valor 1 para fazer comparacao com o botao
```

```
STA R0, @14      #Armazena o valor do acumulador em MEM[14] (valor para  
comparacao do aperto do botao)
```

```
#flag alarme
```

```
LDI R0, $0      #Carrega acumulador com o valor 0 para flag
```

```
STA R0, @19      #Armazena na posicao 19 a flag do alarme
```

```
#Inicializando as variaveis do tempo
```

```
LDI R0, $9
```

```
STA R0, @20      #Armazena o valor do acumulador em MEM[20] (segundos  
unidade)
```

```
STA R0, @21      #Armazena o valor do acumulador em MEM[21] (segundos  
dezenas)
```

```
STA R0, @22      #Armazena o valor do acumulador em MEM[22] (minutos  
unidade)
```

```
STA R0, @23      #Armazena o valor do acumulador em MEM[23] (minutos  
dezenas)
```

```
STA R0, @24      #Armazena o valor do acumulador em MEM[24] (hora unidade)
```

```
STA R0, @25      #Armazena o valor do acumulador em MEM[25] (hora dezenas)
```

```
# -----
```

INICIO:

```
# Ler o botão de incremento de contagem (KEY0):
```

```
LDA R0, @352      # Lê KEY0
```

```
ANDI R0, $1      # Limpa KEY0
```

```
# Caso tenha sido pressionado, desviar para a sub-rotina de incremento de  
valor.
```

```
C EQ R0, @13      # Compara valor com 0 (MEM[13])
```

```
J EQ @CHECK_KEY1  # Caso não tenha sido apertado, checa KEY1
```

```
J SR @INCREMENTA # Se for apertado (KEY0=1), pula pra INCREMENTA
```

CHECK_KEY1:

```
# Ler o botão de configuração do limite de incremento (KEY1):  
LDA R0, @353          # Lê KEY1  
ANDI R0, $1            # Limpa KEY1  
  
# Caso não esteja pressionado, verifica FPGA_RESET  
CEQ R0, @13           # Compara valor com 0 (MEM[13])  
JEQ @CHECK_KEY2        # Caso não tenha sido apertado, checa KEY2  
  
# Caso esteja pressionado, vai para sub-rotina de checar limite  
JMP @SET_HORARIO      # Se for apertado (KEY1=1), pula pra  
SET_HORARIO  
  
  
CHECK_KEY2:  
  
# Ler o botão de configuração do limite de incremento (KEY2):  
LDA R0, @354          # Lê KEY2  
ANDI R0, $1            # Limpa KEY2  
  
# Caso não esteja pressionado, verifica FPGA_RESET  
CEQ R0, @13           # Compara valor com 0 (MEM[13])  
JEQ @CHECK_RESET       # Caso não tenha sido apertado, checa  
FPGA_RESET  
  
# Caso esteja pressionado, vai para sub-rotina de setar alarme  
JMP @ALAR_ALARME     # Se for apertado (KEY2=1), pula pra  
ALAR_ALARME  
  
  
CHECK_RESET:  
  
# Ler o botão de reiniciar contagem (FPGA_RESET):  
LDA R0, @356          # Lê FPG_RESET  
ANDI R0, $1            # Limpa FPG_RESET  
  
# Caso esteja pressionado, desviar para a sub-rotina de reiniciar contagem.  
CEQ R0, @13           # Compara valor com 0 (MEM[13])  
JEQ @SETUP             # Pula para fazer o setup novamente  
  
  
# Escrever os valores das variáveis nos respectivos displays (pode ser uma  
sub-rotina).  
JSR @DISPLAY
```

```
#Verifica se chegou no alarme
```

```
JSR @VERIFICA_ALARME
```

```
# Desviar para o **INÍCIO**.
```

```
JMP @INICIO
```

```
# -----
```

```
# SUBROTINAS
```

```
DISPLAY:
```

```
# Carrega valor de memória de cada unidade
```

```
# e os mostra no display
```

```
LDA R0, @0
```

```
STA R0, @288
```

```
LDA R0, @1
```

```
STA R0, @289
```

```
LDA R0, @2
```

```
STA R0, @290
```

```
LDA R0, @3
```

```
STA R0, @291
```

```
LDA R0, @4
```

```
STA R0, @292
```

```
LDA R0, @5
```

```
STA R0, @293
```

```
RET
```

```
INCREMENTA:
```

```
STA R0, @511 # Limpa FF de KEY0
```

```
# Verificando flag inibição de contagem
LDA R0, @19          #Carrega valor flag
CEQ R0, @14          #Compara com 1 (FALSE)
JEQ @INICIO

#INC_SEG_UNI:
LDA R0, @0          # Carrega valor das unidades
INC R0              # Incrementa valor
# Checa se ultrapassou 10
CEQ R0, @6
JEQ @INC_SEG_DEZ
# Caso contrário, carrega novo valor
STA R0, @0          # Carrega novo valor da unidade (MEM[0])
RET

INC_SEG_DEZ:
# Reseta valor da unidade
LDI R0, $0
STA R0, @0          #Carrega valor segundo unidade
LDA R0, @1          #Carrega valor segundo dezena
INC R0
CEQ R0, @7
JEQ @INC_MIN_UNI
STA R0, @1
RET

INC_MIN_UNI:
# Reseta valor da unidade
LDI R0, $0
STA R0, @1
LDA R0, @2
```

```
INC R0
```

```
CEQ R0, @8
```

```
JEQ @INC_MIN_DEZ
```

```
STA R0, @2
```

```
RET
```

INC_MIN_DEZ:

```
# Reseta valor da unidade
```

```
LDI R0, $0
```

```
STA R0, @2
```

```
LDA R0, @3
```

```
INC R0
```

```
CEQ R0, @9
```

```
JEQ @INC_HORA_UNI
```

```
STA R0, @3
```

```
RET
```

INC_HORA_UNI:

```
# Reseta valor da unidade
```

```
LDI R0, $0
```

```
STA R0, @3
```

```
LDA R0, @4
```

```
INC R0
```

```
CEQ R0, @10
```

```
JEQ @INC_HORA_DEZ
```

```
STA R0, @4
```

```
RET
```

INC_HORA_DEZ:

```
# Reseta valor da unidade
```

```
LDI R0, $0
```

```
STA R0, @4
LDA R0, @5
INC R0
CEQ R0, @11
JEQ @OVERFLOW
STA R0, @5
RET
```

OVERFLOW:

```
LDI R0, $0
STA R0, @0      #Armazena o valor do acumulador em MEM[0] (segundos unidade)
STA R0, @1      #Armazena o valor do acumulador em MEM[1] (segundos dezenas)
STA R0, @2      #Armazena o valor do acumulador em MEM[2] (minutos unidade)
STA R0, @3      #Armazena o valor do acumulador em MEM[3] (minutos dezenas)
STA R0, @4      #Armazena o valor do acumulador em MEM[4] (hora unidade)
STA R0, @5      #Armazena o valor do acumulador em MEM[5] (hora dezenas)
RET
```

SET_HORARIO:

#Setando limite (segundos unidade) :

```
STA R0, @510      #Limpa FF da KEY1
LDA R0, @320      #Armazena o valor lido nas chaves (ler SW0~SW7)
ANDI R0, $15      #Mascara para limpar valor
```

```
CLT R0, @15      #Faz R0-9
```

```
JLT @PULA_1      #Se o que usuário colocar for > 9, vira 9
```

```
LDI R0, $9
```

PULA_1:

```
STA R0, @0        #Armazena o valor do acumulador no segundo unidadeE
```

```
STA R0, @288      #Carrega valor no display
```

```
# Setando limite (dezenas):  
LDI R0, $2          #Carrega o acumulador com o valor 2  
STA R0, @256        #Armazena 2 no LDR0 até LDR7 (segundos dezenas)  
  
SET_SEG_DEZ:  
LDA R0, @353        #Armazena o valor lido no KEY1 (ler KEY1)  
ANDI R0, @1          #Utiliza mascara para limpar o valor lido do botao  
CEQ R0, @13          #Compara o valor lido de KEY1 com 0 (zero esta  
salvo na posicao 13)  
JEQ @SET_SEG_DEZ    #KEY1 nao foi apertado então faz o desvio (fica  
esperando)  
  
STA R0, @510         #Limpando FF do KEY1  
LDA R0, @320         #Armazena o valor lido nas chaves (ler SW0~SW7)  
ANDI R0, $15          #Mascara para limpar valor  
  
CLT R0, @16          #Faz 5 - R0  
JLT @PULA_2          #Se o que usuario colocar for > 5, vira 5  
LDI R0, $5  
  
PULA_2:  
STA R0, @1          #Armazena o valor do acumulador no espaco das  
dezenas do limite  
  
#Carrega valor no display  
STA R0, @289  
  
#Setando horario dos minutos:  
LDI R0, $4          #Carrega o acumulador com o valor 4  
STA R0, @256        #Armazena 1 no LDR0 até LDR7 (mostrando centenas)  
SET_MIN_UNI:  
LDA R0, @353        #Armazena o valor lido no KEY1 (ler KEY1)  
ANDI R0, @1          #Utiliza mascara para limpar o valor lido do botao
```

```
CEQ R0, @13          #Compara o valor lido de KEY1 com 0 (zero esta  
salvo na posicao 13)  
  
JEQ @SET_MIN_UNI    #KEY1 nao foi apertado então faz o desvio (fica  
esperando)  
  
STA R0, @510         #Limpando FF do KEY1  
LDA R0, @320         #Armazena o valor lido nas chaves (ler SW0~SW7)  
ANDI R0, $15         #Mascara para limpar valor  
  
  
CLT R0, @15          #Faz 9 - R0  
JLT @PULA_3          #Se o que usuário colocar for > 9, vira 9  
LDI R0, $9  
  
PULA_3:  
STA R0, @2            #Armazena o valor do acumulador no espaco das  
centenas do limite  
  
  
#Carrega valor no display  
STA R0, @290  
  
  
#Setando limite (dezenas de minuto):  
LDI R0, $8            #Carrega o acumulador com o valor 8  
STA R0, @256          #Armazena 1 no LDR0 até LDR7 (mostrando milhares)  
  
SET_MIN_DEZ:  
LDA R0, @353          #Armazena o valor lido no KEY1 (ler KEY1)  
ANDI R0, @1             #Utiliza mascara para limpar o valor lido do botao  
CEQ R0, @13            #Compara o valor lido de KEY1 com 0 (zero esta  
salvo na posicao 13)  
  
JEQ @SET_MIN_DEZ      #KEY1 nao foi apertado então faz o desvio (fica  
esperando)  
  
STA R0, @510           #Limpando FF do KEY11  
LDA R0, @320           #Armazena o valor lido nas chaves (ler SW0~SW7)  
ANDI R0, $15            #Mascara para limpar valor
```

```
CLT R0, @16          #Faz 5 - R0
JLT @PULA_4          #Se o que usuário colocar for > 5, vira 5
LDI R0, $5
PULA_4:
STA R0, @3           #Armazena o valor do acumulador no espaço das
milhares do limite

#Carrega valor no display
STA R0, @291

#Setando limite unidade de hora:
LDI R0, $16          #Carrega o acumulador com o valor 16
STA R0, @256          #Armazena 1 no LDR0 até LDR7 (mostrando dezenas de
milhares)
SET_HORA_UNI:
LDA R0, @353          #Armazena o valor lido no KEY1 (ler KEY1)
ANDI R0, @1             #Utiliza máscara para limpar o valor lido do botão
CEQ R0, @13            #Compara o valor lido de KEY1 com 0 (zero está
salvo na posição 13)
JEQ @SET_HORA_UNI      #KEY1 não foi apertado então faz o desvio (fica
esperando)

STA R0, @510          #Limpa FF do KEY1
LDA R0, @320          #Armazena o valor lido nas chaves (ler SW0~SW7)
ANDI R0, $15           #Máscara para limpar valor

CLT R0, @17          #Faz 3 - R0
JLT @PULA_5          #Se o que usuário colocar for > 3, vira 3
LDI R0, $3
PULA_5:
```

```
STA R0, @4          #Armazena o valor do acumulador no espaço das  
dezenas de milhares do limite
```

```
#Carrega valor no display
```

```
STA R0, @292
```

```
#Setando limite (centenas de milhares):
```

```
LDI R0, $32          #Carrega o acumulador com o valor 16  
STA R0, @256         #Armazena 1 no LDR0 até LDR7 (mostrando centenas de  
milhares)
```

```
SET_HORA_DEZ:
```

```
LDA R0, @353         #Armazena o valor lido no KEY1 (ler KEY1)  
ANDI R0, @1           #Utiliza máscara para limpar o valor lido do botão  
CEQ R0, @13           #Compara o valor lido de KEY1 com 0 (zero esta  
salvo na posição 13)  
JEQ @SET_HORA_DEZ    #KEY1 não foi apertado então faz o desvio (fica  
esperando)
```

```
STA R0, @510          #Limpa FF do KEY1  
LDA R0, @320          #Armazena o valor lido nas chaves (ler SW0~SW7)  
ANDI R0, $15          #Máscara para limpar valor
```

```
CLT R0, @18          #Faz 2 - R0
```

```
JLT @PULA_6          #Se o que usuário colocar for > 2, vira 2
```

```
LDI R0, $2
```

```
PULA_6:
```

```
STA R0, @5          #Armazena o valor do acumulador no espaço das  
centenas de milhares do limite
```

```
#Carrega valor no display
```

```
STA R0, @293
```

```
LDI R0, $1          #Carrega o acumulador com o valor 2
```

```
STA R0, @256          #Armazena 2 no LDR0 até LDR7 (mostrando dezenas)
```

```
JMP @CHECK_KEY2
```

```
#Subrotina que ajusta alarme
```

```
ALAR_ALARME:
```

```
#Setando limite (segundos unidade):
```

```
STA R0, @509          #Limpa FF da KEY2
```

```
LDA R0, @320          #Armazena o valor lido nas chaves (ler SW0~SW7)
```

```
ANDI R0, $15          #Mascara para limpar valor
```

```
CLT R0, @15           #Faz R0- 9
```

```
JLT @PULA_10          #Se o que usuário colocar for > 9, vira 9
```

```
LDI R0, $9
```

```
PULA_10:
```

```
STA R0, @20            #Armazena o valor do acumulador no segundo unidadeE
```

```
STA R0, @288            #Carrega valor no display
```

```
# Setando limite (segundos dezenas):
```

```
LDI R0, $2              #Carrega o acumulador com o valor 2
```

```
STA R0, @256            #Armazena 2 no LDR0 até LDR7 (segundos dezenas)
```

```
ALAR_SEG_DEZ:
```

```
LDA R0, @354            #Armazena o valor lido no KEY2 (ler KEY2)
```

```
ANDI R0, $1              #Utiliza mascara para limpar o valor lido do botao
```

```
CEQ R0, @13              #Compara o valor lido de KEY1 com 0 (zero esta  
salvo na posicao 13)
```

```
JEQ @ALAR_SEG_DEZ       #KEY1 nao foi apertado então faz o desvio (fica  
esperando)
```

```
STA R0, @509            #Limpando FF do KEY1
```

```
LDA R0, @320            #Armazena o valor lido nas chaves (ler SW0~SW7)
```

```
ANDI R0, $15          #Mascara para limpar valor

CLT R0, @16          #Faz 5 - R0
JLT @PULA_20         #Se o que usuário colocar for > 5, vira 5
LDI R0, $5
PULA_20:
STA R0, @21          #Armazena o valor do acumulador no espaço das
dezenas do limite

#Carrega valor no display
STA R0, @289

#Setando horário dos minutos:
LDI R0, $4          #Carrega o acumulador com o valor 4
STA R0, @256         #Armazena 1 no LDR0 até LDR7 (mostrando centenas)

ALAR_MIN_UNI:
LDA R0, @354         #Armazena o valor lido no KEY2 (ler KEY2)
ANDI R0, @1           #Utiliza máscara para limpar o valor lido do botão
CEQ R0, @13           #Compara o valor lido de KEY1 com 0 (zero está
salvo na posição 13)
JEQ @ALAR_MIN_UNI   #KEY1 não foi apertado então faz o desvio (fica
esperando)

STA R0, @509          #Limpa FF do KEY1
LDA R0, @320          #Armazena o valor lido nas chaves (ler SW0~SW7)
ANDI R0, $15          #Máscara para limpar valor

CLT R0, @15          #Faz 9 - R0
JLT @PULA_30         #Se o que usuário colocar for > 9, vira 9
LDI R0, $9
PULA_30:
```

```
STA R0, @22          #Armazena o valor do acumulador no espaço das  
centenas do limite
```

```
#Carrega valor no display
```

```
STA R0, @290
```

```
#Setando limite (dezenas de minuto):
```

```
LDI R0, $8          #Carrega o acumulador com o valor 8
```

```
STA R0, @256        #Armazena 1 no LDR0 até LDR7 (mostrando milhares)
```

```
ALAR_MIN_DEZ:
```

```
LDA R0, @354        #Armazena o valor lido no KEY2 (ler KEY2)
```

```
ANDI R0, @1          #Utiliza máscara para limpar o valor lido do botão
```

```
CEQ R0, @13          #Compara o valor lido de KEY1 com 0 (zero está  
salvo na posição 13)
```

```
JEQ @ALAR_MIN_DEZ  #KEY1 não foi apertado então faz o desvio (fica  
esperando)
```

```
STA R0, @509        #Limpando FF do KEY2
```

```
LDA R0, @320        #Armazena o valor lido nas chaves (ler SW0~SW7)
```

```
ANDI R0, $15         #Máscara para limpar valor
```

```
CLT R0, @16          #Faz 5 - R0
```

```
JLT @PULA_40        #Se o que usuário colocar for > 5, vira 5
```

```
LDI R0, $5
```

```
PULA_40:
```

```
STA R0, @23          #Armazena o valor do acumulador no espaço das  
milhares do limite
```

```
#Carrega valor no display
```

```
STA R0, @291
```

```
#Setando limite unidade de hora:
```

```
LDI R0, $16          #Carrega o acumulador com o valor 16
STA R0, @256         #Armazena 1 no LDR0 até LDR7 (mostrando dezenas de
                     milhares)

ALAR_HORA_UNI:
LDA R0, @354         #Armazena o valor lido no KEY2 (ler KEY2)
ANDI R0, @1           #Utiliza mascara para limpar o valor lido do botao
CEQ R0, @13           #Compara o valor lido de KEY1 com 0 (zero esta
                     salvo na posicao 13)
JEQ @ALAR_HORA_UNI  #KEY1 nao foi apertado então faz o desvio (fica
                     esperando)

STA R0, @509          #Limpando FF do KEY2
LDA R0, @320          #Armazena o valor lido nas chaves (ler SW0~SW7)
ANDI R0, $15           #Mascara para limpar valor

CLT R0, @17           #Faz 3 - R0
JLT @PULA_50          #Se o que usuario colocar for > 3, vira 3
LDI R0, $3
PULA_50:
STA R0, @24           #Armazena o valor do acumulador no espaco das
                     dezenas de milhares do limite

#Carrega valor no display
STA R0, @292

#Setando limite (centenas de milhares):
LDI R0, $32           #Carrega o acumulador com o valor 16
STA R0, @256           #Armazena 1 no LDR0 até LDR7 (mostrando centenas de
                     milhares)

ALAR_HORA_DEZ:
LDA R0, @354           #Armazena o valor lido no KEY2 (ler KEY2)
ANDI R0, @1             #Utiliza mascara para limpar o valor lido do botao
```

```
CEQ R0, @13          #Compara o valor lido de KEY1 com 0 (zero esta  
salvo na posicao 13)  
  
JEQ @ALAR_HORA_DEZ    #KEY1 nao foi apertado então faz o desvio (fica  
esperando)  
  
STA R0, @509          #Limpando FF do KEY2  
LDA R0, @320          #Armazena o valor lido nas chaves (ler SW0~SW7)  
ANDI R0, $15           #Mascara para limpar valor  
  
  
CLT R0, @18          #Faz 2 - R0  
JLT @PULA_60          #Se o que usuário colocar for > 2, vira 2  
LDI R0, $2  
  
PULA_60:  
STA R0, @25          #Armazena o valor do acumulador no espaco das  
centenas de milhares do limite  
  
  
#Carrega valor no display  
STA R0, @293  
  
  
LDI R0, $1           #Carrega o acumulador com o valor 2  
STA R0, @256          #Armazena 2 no LDR0 até LDR7 (mostrando dezenas)  
  
  
JMP @CHECK_RESET  
  
  
VERIFICA_ALARME:  
# Comparação do segundo unidade  
LDA R0, @0  
CEQ R0, @20  
JEQ @VERIFICA_SEG_DEZ  
RET  
  
  
VERIFICA_SEG_DEZ:
```

```
LDA R0, @1
```

```
CEQ R0, @21
```

```
JEQ @VERIFICA_MIN
```

```
RET
```

VERIFICA_MIN:

```
LDA R0, @2
```

```
CEQ R0, @22
```

```
JEQ @VERIFICA_MIN_DEZ
```

```
RET
```

VERIFICA_MIN_DEZ:

```
LDA R0, @3
```

```
CEQ R0, @23
```

```
JEQ @VERIFICA_HORA
```

```
RET
```

VERIFICA_HORA:

```
LDA R0, @4
```

```
CEQ R0, @24
```

```
JEQ @VERIFICA_HORA_DEZ
```

```
RET
```

VERIFICA_HORA_DEZ:

```
LDA R0, @5
```

```
CEQ R0, @25
```

```
JEQ @ALARME_ATINGIDO
```

```
RET
```

ALARME_ATINGIDO:

```
# Liga LED9 (de limite atingido)
```

```
LDI R0, $1
STA R0, @258
# Ativa flag de inibição de contagem
STA R0, @19
JMP @INICIO
```

UTILIZAÇÃO DO RELÓGIO

A verificação do funcionamento do projeto depende da correta utilização da placa. Essa seção do relatório se preocupa em instruir o usuário para tanto.

Ao ligar a FPGA, o usuário observará que todos os displays de sete segmentos estão ligados com o valor zero e o relógio já começará a contar. Além disso, o LED 0 também é ligado, enquanto os demais permanecem desativados. O motivo desse LED estar ligado inicialmente está descrito na próxima subseção.

Definindo horário

Para fazer a customização do horário é necessário utilizar as chaves SW0 ~ SW4. A ativação delas fornece uma palavra que, em binário, delimitam o limite de uma casa decimal.

Inicialmente a placa tem o LED 0 ligado porque a placa já está preparada para receber o limite das unidades (casa decimal 0). Ao manipular as chaves SW0 ~ SW4 e apertar o botão 1 (KEY1), a primeira casa decimal do limite customizado é modificada. Esse valor pode ser confirmado ao se observar o display de sete segmentos. O mesmo protocolo deve ser seguido para as casas decimais subsequentes.

Atenção: a partir da primeira vez que o botão KEY1 é apertado, o relógio não funcionará até que todas as casas decimais do limite sejam configuradas.

Observação: O restante das chaves podem ser ativados, porém somente as chaves SW0~SW4 são lidas (validas).

Definindo alarme

A definição do alarme também utiliza as chaves SW0~SW4. A ativação delas fornece uma palavra que, em binário, delimitam o limite de uma casa decimal.

Inicialmente a placa tem o LED 0 ligado porque a placa já está preparada para receber o limite das unidades (casa decimal 0). Ao manipular as chaves SW0 ~ SW4 e apertar o botão 2 (KEY2), a primeira casa decimal do limite customizado é modificada. Esse valor pode ser

confirmado ao se observar o display de sete segmentos. O mesmo protocolo deve ser seguido para as casas decimais subsequentes.

É importante notar que se o usuário tentar aplicar um valor maior do que o suportado por uma casa decimal (valor maior que 9), o programa travará esse valor em 9.

Atenção: a partir da primeira vez que o botão KEY2 é apertado, o relógio não funcionará até que todas as casas decimais do limite sejam configuradas.

Observação: O restante das chaves podem ser ativados, porém somente as chaves SW0~SW4 são lidas (validas).

Mudar Base de Tempo

O relógio pode funcionar em duas bases de tempo diferentes, a seleção desse tempo é feita pela chave 9 (SW9). Com SW9 desativado, o relógio funciona com passagem de tempo de 1 segundo. Ao acionar a chave o relógio funcionará com passagem de tempo mais rápida para facilitar a visualização da passagem de horas.

Reinício do Relógio

Para reiniciar o relógio (00:00:00), pode-se tanto fazer o carregamento do programa na placa novamente (Hard-reset) quanto apertar o botão FPGA Reset (Program-reset). A segunda alternativa é mais rápida e faz essencialmente a mesma coisa que a primeira.