Rafael Eli Katri

Felipe Catapano Emrich Melo

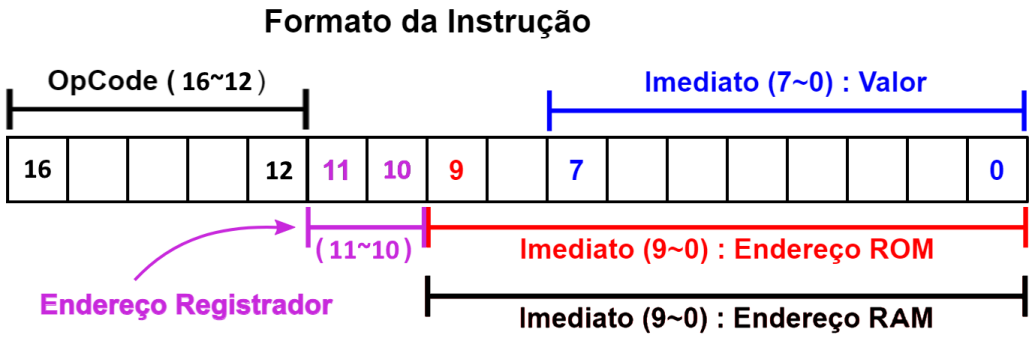
Projeto 1- Relógio (Entrega Final)

Arquitetura do processador: Registrador-memória

Essa arquitetura foi escolhida pois apresenta mais de um registrador para armazenar dados. Isso é relevante uma vez que acessar dados de registradores é significativamente mais rápido do que acesso direto à memória (o que ocorre mais vezes em uma arquitetura baseada em acumulador).

Instruções:

- Número de instruções: 15
- Largura: 17 bits
- Formato:



- Foi escolhido um opcode de largura 5 para possibilitar 32 instruções diferentes.
- Endereçamento de registradores permite até 4 registradores.
- Imediato de endereços apresentam largura 10 para endereçar até 1024 endereços.
- imediato de valor apresenta largura de 8 bits para concordar com o processamento da ULA.

● Descrição:

- NOP : No operation. Não usa argumentos.
- LDA: Carrega valor da memória para determinado registrador. Usa 2 bits de endereçamento para registradores e 10 bits para endereçamento da memória.
- LDI: Carrega valor do imediato para determinado registrador. Usa 2 bits de endereçamento para registradores e 8 bits para o imediato.
- STA: Carrega valor de determinado registrador para a memória. Usa 2 bits de endereçamento para registradores e 10 bits para endereçamento da memória.
- SOMA: Soma valor de determinado registrador com a memória e salva o resultado no mesmo registrador. Usa 2 bits de endereçamento para registradores e 10 bits para endereçamento da memória.
- SUB: Subtrai o valor da memória do valor de determinado registrador e salva o resultado no mesmo registrador (R= R - M). Usa 2 bits de endereçamento para registradores e 10 bits para endereçamento da memória.
- OP_AND: Realiza operação AND entre o valor de determinado registrador e o valor da memória. Usa 2 bits de endereçamento para registradores e 10 bits para endereçamento da memória.

- JMP: Salta para o endereço da memória ROM indicado no imediato. Usa 10 bits para o destino do salto.
- JEQ: Salta para o endereço da memória indicado no imediato caso a flag de igual esteja ativa. Usa 10 bits para o destino do salto.
- CEQ: Ativa a flag de igual caso o valor de determinado registrador for igual ao valor da memória. Usa 2 bits de endereçamento para registradores e 10 bits para endereçamento da memória.
- JSR: Salta para um endereço da memória ROM em que se encontra uma subrotina, indicado no imediato. Usa 2 bits de endereçamento para registradores e 10 bits para o destino do salto.
- RET: Retorna de uma subrotina. Não usa argumentos.
- SOMAI: Soma valor de determinado registrador com imediato e salva o resultado no mesmo registrador. Usa 2 bits de endereçamento para registradores e 8 bits para o imediato.
- CLT: Ativa a flag de “menor que” caso o valor de determinado registrador seja estritamente menor que o valor da memória. Usa 2 bits de endereçamento para registradores e 10 bits para endereçamento da memória.
- JLT: Salta para o endereço da memória indicado no imediato caso a flag de “menor que” esteja ativa. Usa 10 bits para o destino do salto.

Pontos de controle:

- Largura: 14 bits

- Descrição:

- Write - bit 0: Sinal de escrita para a memória/IO
- Read - bit 1: Sinal de leitura para memória/IO
- Habilita Igual - bit 2: Habilita escrita para flag de igual
- Operação ULA - bit 3 a 5: Indica operação da ULA

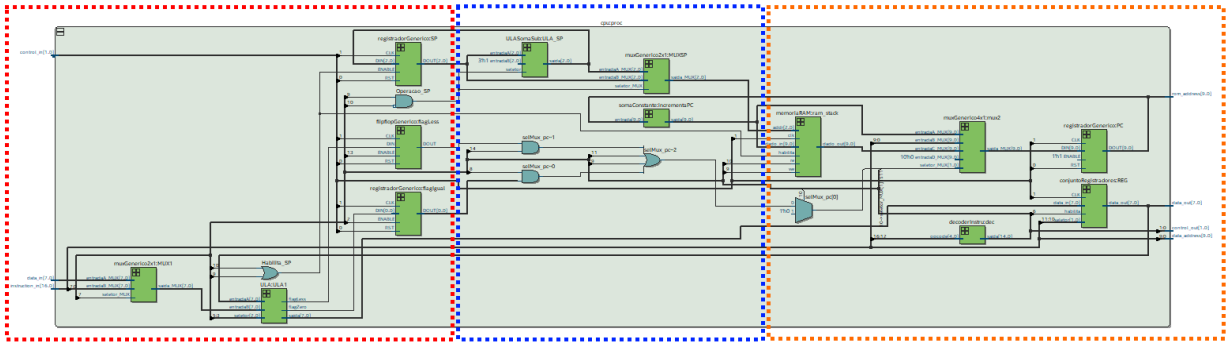
```
saida <= soma when (seletor = "000") else
    subtracao when (seletor = "001") else
    entradaA when  (seletor = "010") else
    entradaB when  (seletor = "011") else
    op_xor when    (seletor = "100") else
    op_not when    (seletor = "101") else
    op_and when    (seletor = "110") else
    op_or when     (seletor = "111") else
    entradaA;      -- outra opcao: saida = entradaA
```

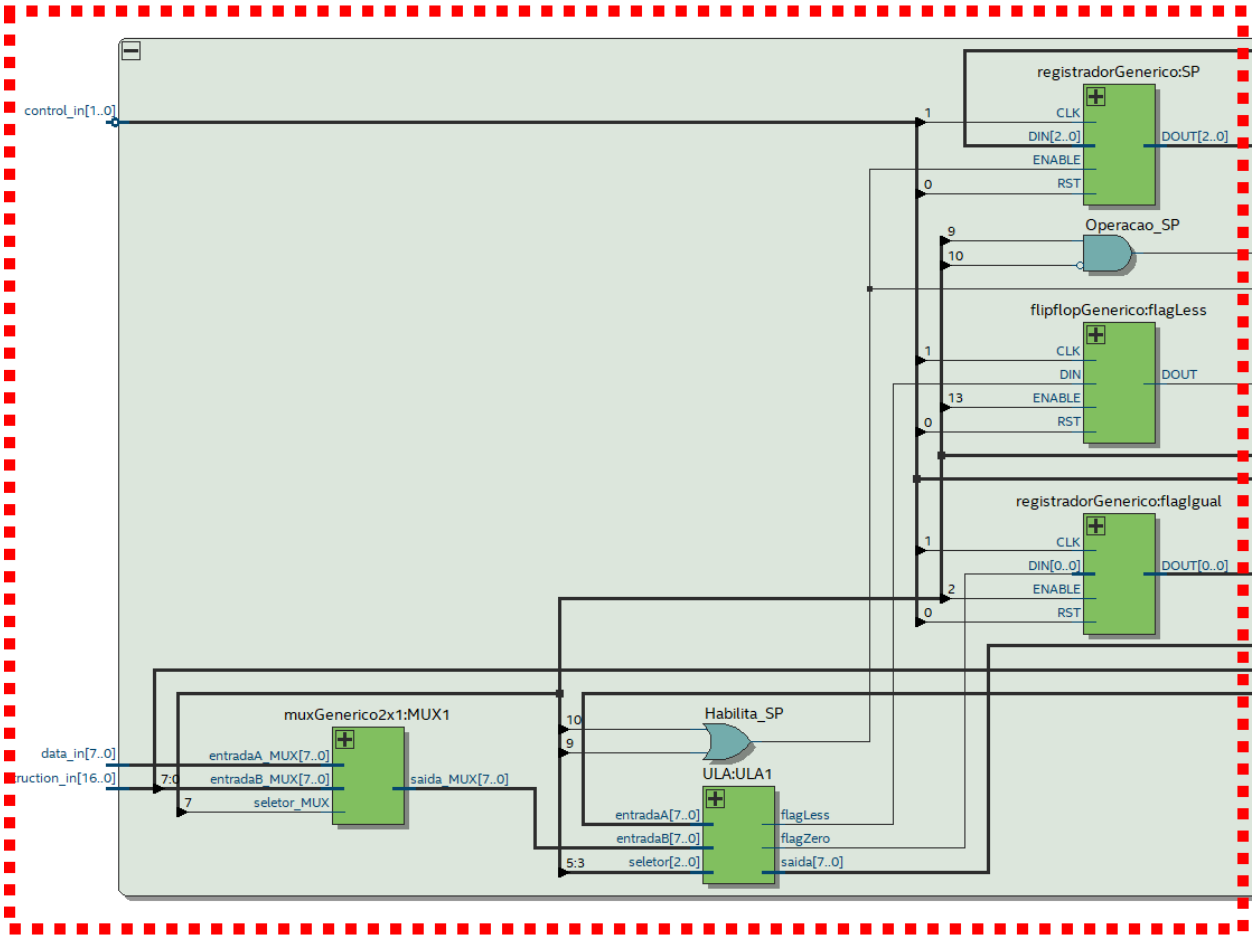
- Habilita registradores - bit 6: Habilita escrita no bloco de registradores. Esse sinal é uma das entradas do bloco de registradores, que também considera o imediato de 2 bits para habilitar apenas o registrador desejado.
- Seletor Imediato/Dados - bit 7: Seletor do MUX que indica se valores do imediato ou da memória serão usados como a entrada B da ULA.
- JEQ - bit 8: Realiza salto condicional se a flag de igual está ativa.
- JSR - bit 9: Realiza salto de subrotina.
- RET - bit 10: Realiza salto para o valor do registrador do endereço de retorno.
- JMP - bit 11: Realiza salto incondicional.
- Habilita Menor Que - bit 12: Habilita escrita para flag de Menor Que.

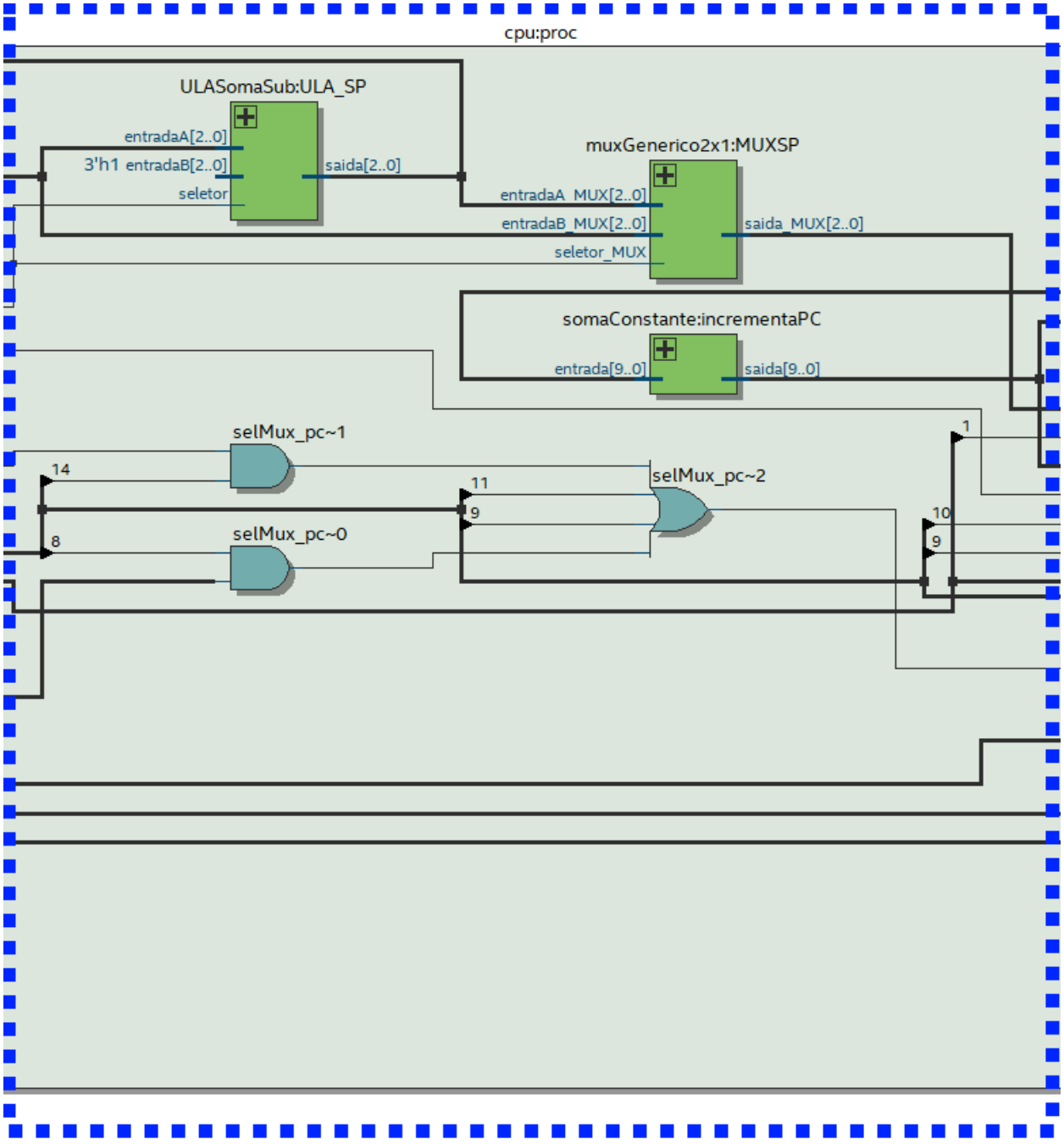
- JLT - bit 13: Realiza salto condicional se a flag de Menor Que está ativa.

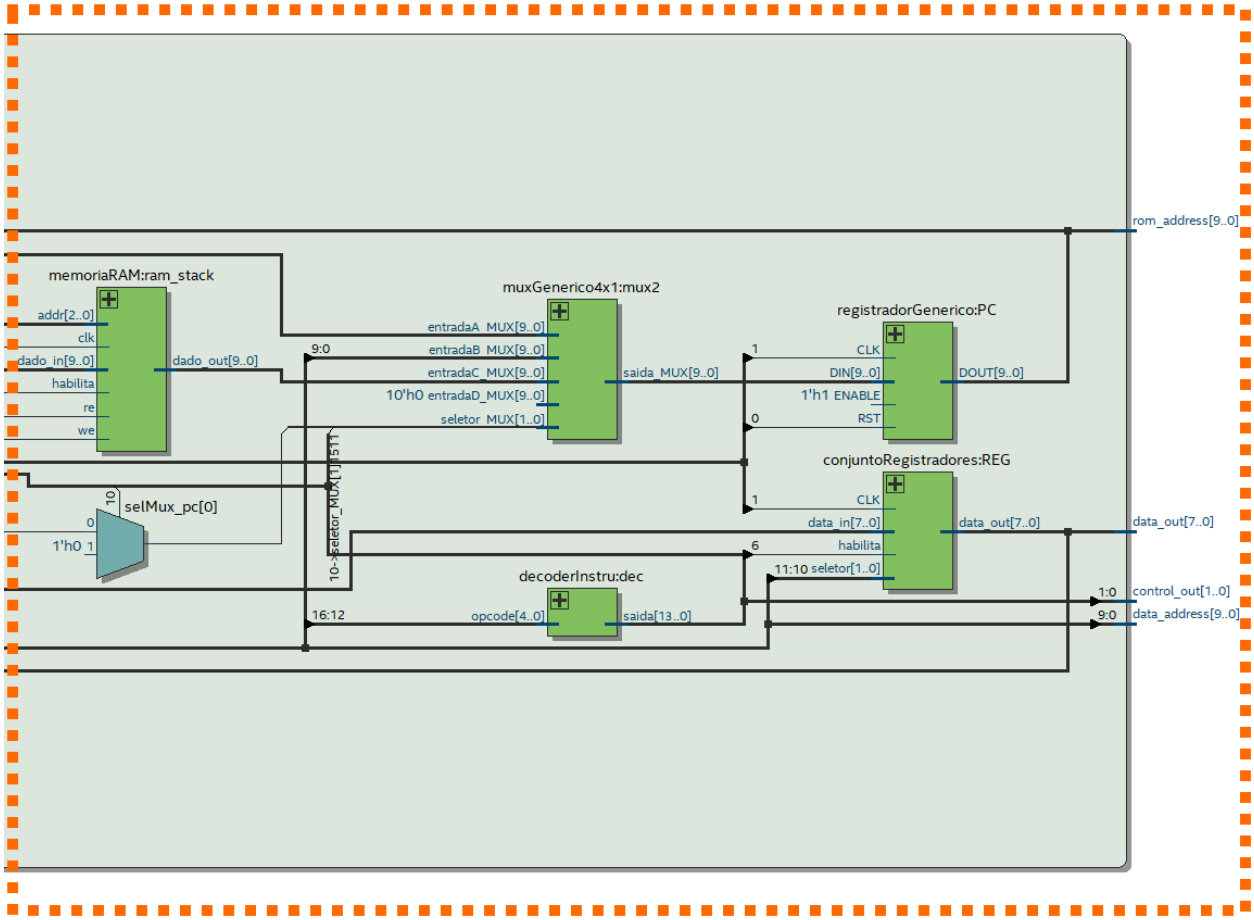
Fluxo de dados e funcionamento da CPU:

- RTL gerado pelo Quartus (instância que utiliza cpu.vhd):

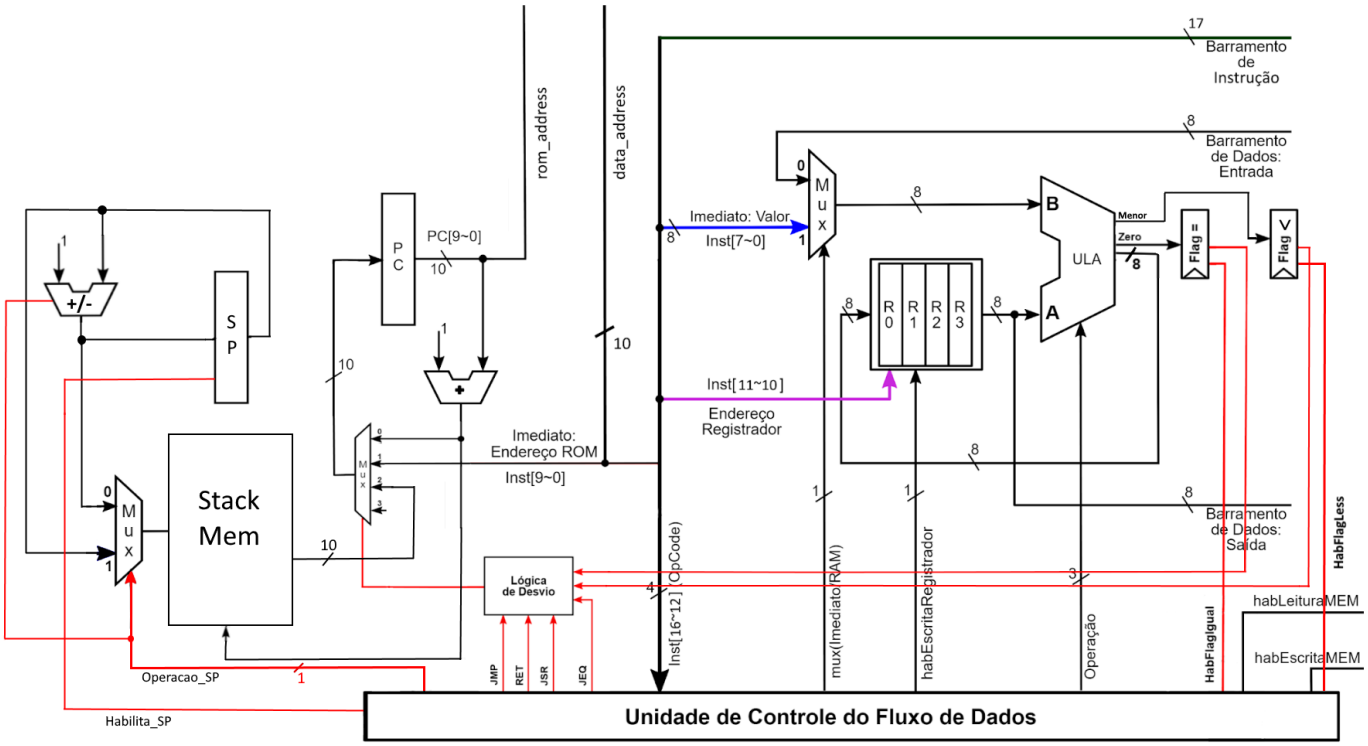





cpu:proc
ULASomaSub:ULA_SP
entradaA[2..0]
3'h1 entradaB[2..0]
seletor
saida[2..0]
muxGenerico2x1:MUXSP
entradaA_MUX[2..0]
entradaB_MUX[2..0]
seletor_MUX
saida_MUX[2..0]
somaConstante:incrementaPC
entrada[9..0]
saida[9..0]
selMux_pc~1
selMux_pc~2
selMux_pc~0
14
8
11
9
1
10
9


rom_address[9..0]
memoriaRAM:ram_stack
addr[2..0]
clk
dado_in[9..0]
habilita
re
we
dado_out[9..0]
9:0
muxGenerico4x1:mux2
entradaA_MUX[9..0]
entradaB_MUX[9..0]
entradaC_MUX[9..0]
10'h0 entradaD_MUX[9..0]
seletor_MUX[1..0]
saida_MUX[9..0]
registradorGenerico:PC
1
CLK
DIN[9..0]
1'h1 ENABLE
RST
DOUT[9..0]
0
conjuntoRegistradores:REG
1
CLK
data_in[7..0]
6
habilita
11:10 seletor[1..0]
data_out[7..0]
data_out[7..0]
10
selMux_pc[0]
0
1'h0 1
10->seletor_MUX[1] 511
decoderInstru:dec
16:12
opcode[4..0]
saida[13..0]
1:0
control_out[1..0]
9:0
data_address[9..0]

- Rascunho no formato baseado em exemplos da disciplina:



- Observações sobre funcionamento:

- Vide Pontos de controle para mais detalhes
- Entradas: clock, reset, dados_entrada, instrução.
- Saídas: rom_address, data_saída, data_address, read, write.
- Clock e reset não foram desenhados no rascunho por simplificação.
- Uma memória RAM armazena endereços de retorno de subrotinas usando uma lógica de Stack.

- Uma instrução JSR seria análoga ao “PUSH”, escrevendo o endereço da próxima instrução do PC e incrementando o Stack Pointer.
- Já a instrução RET é análoga ao “POP”, fazendo uma leitura da memória de endereço de retorno e decrementando o Stack Pointer.
- Isso permite subrotinas aninhadas.
- O Stack Pointer foi implementado como um registrador de 3 bits de largura. Assim permitindo até 8 subrotinas aninhadas.
- Lógica de salto controla um MUX com 3 entradas relevantes (0-3): Próxima instrução, valor imediato e valor da memória de endereço de retorno
- Mux Imediato/Ram indica qual será a entrada B da ULA
- Bloco de Registradores possui duas entradas relevantes, o endereço do registrador e o habilita bloco de registradores. O habilita indica que uma escrita será feita no bloco e o endereço indica qual registrador será habilitado.
- ULA apresenta 3 saídas: saída de resultados, saída da flag igual e saída da flag menor que. A saída de resultados tem largura de 8 bits e apresenta o resultado de A operação B. A saída da flag igual tem largura de 1 bit e é ativada caso o valor de A seja igual a B.
- A saída da flag menor que tem largura de 1 bit e é ativada caso o valor de A seja menor que B. Para isso foram criados sinais estendidos idênticos as entradas, mas com 9 bits de largura, tendo o bit mais significativo igual a zero. Isso permite analisar os dados como complemento de dois e portanto, quando a operação de A-B resultar em um valor que tem bit mais significativo ‘1’, ou seja negativo, pode-se afirmar que A é menor que B.

Mapa de Memória:

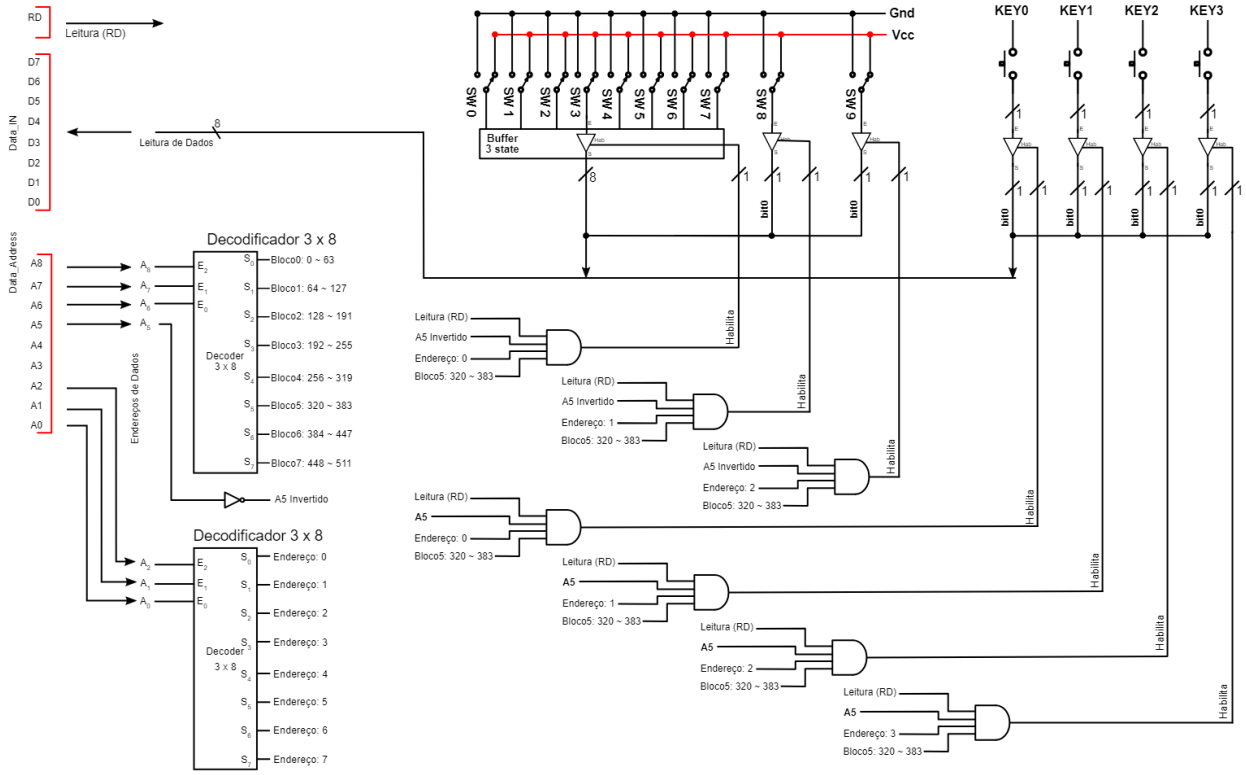
Os endereços de memória utilizados pelo relógio, pelo qual referenciamos os periféricos, é muito semelhante ao mapeamento utilizado pelo computador da disciplina, com as únicas diferenças sendo o fato de que o botão FPGA_RESET não é mapeado e a inclusão dos endereços de leitura de passagem de tempo, limpa leitura de tempo e mudança da base de tempo.

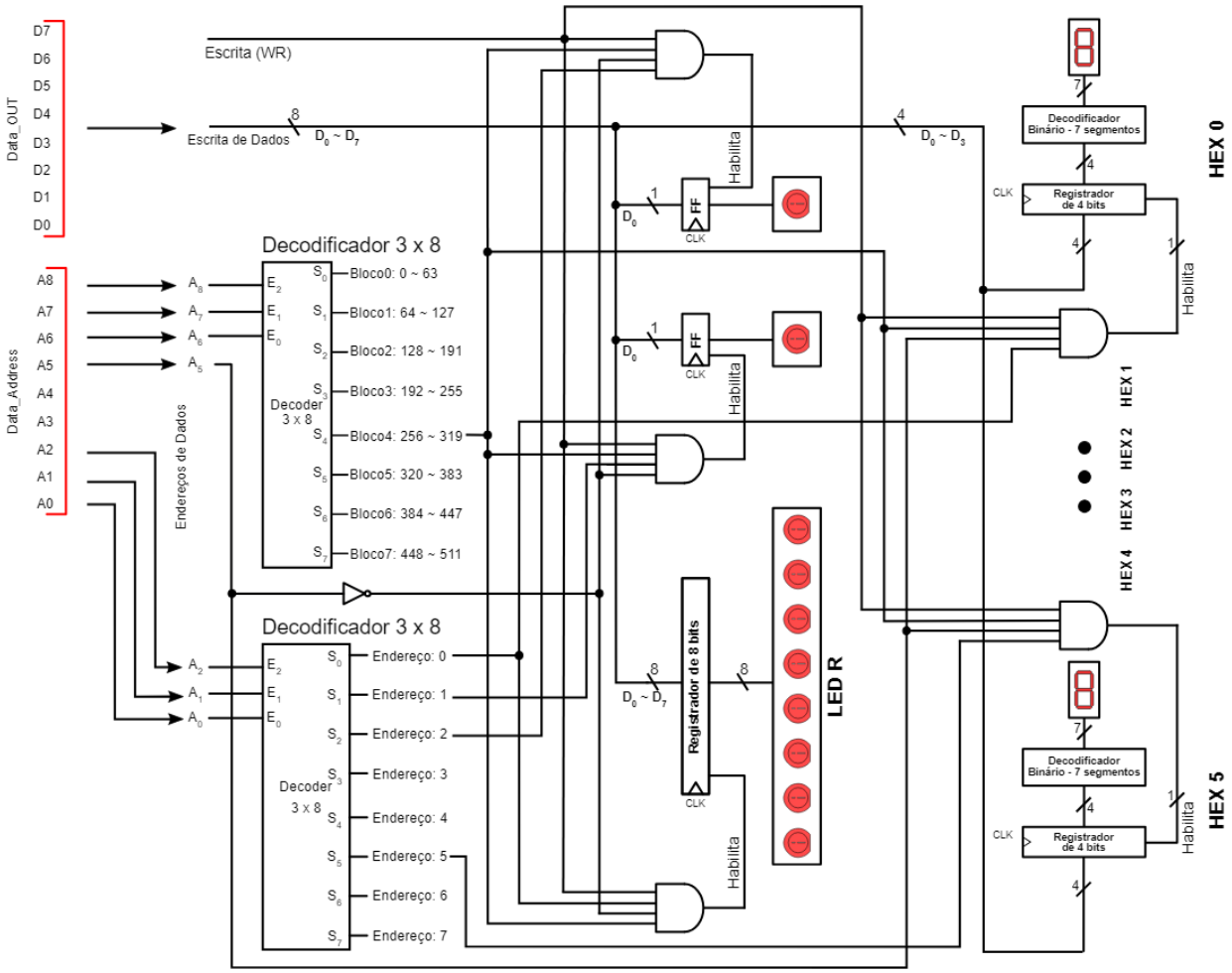
O botão FPGA_RESET não é endereçado porque ele está ligado diretamente, associado a uma porta NOT, ao sinal de RESET da CPU, o qual é passado para todos os registradores, incluindo PC, assim reiniciando o programa.

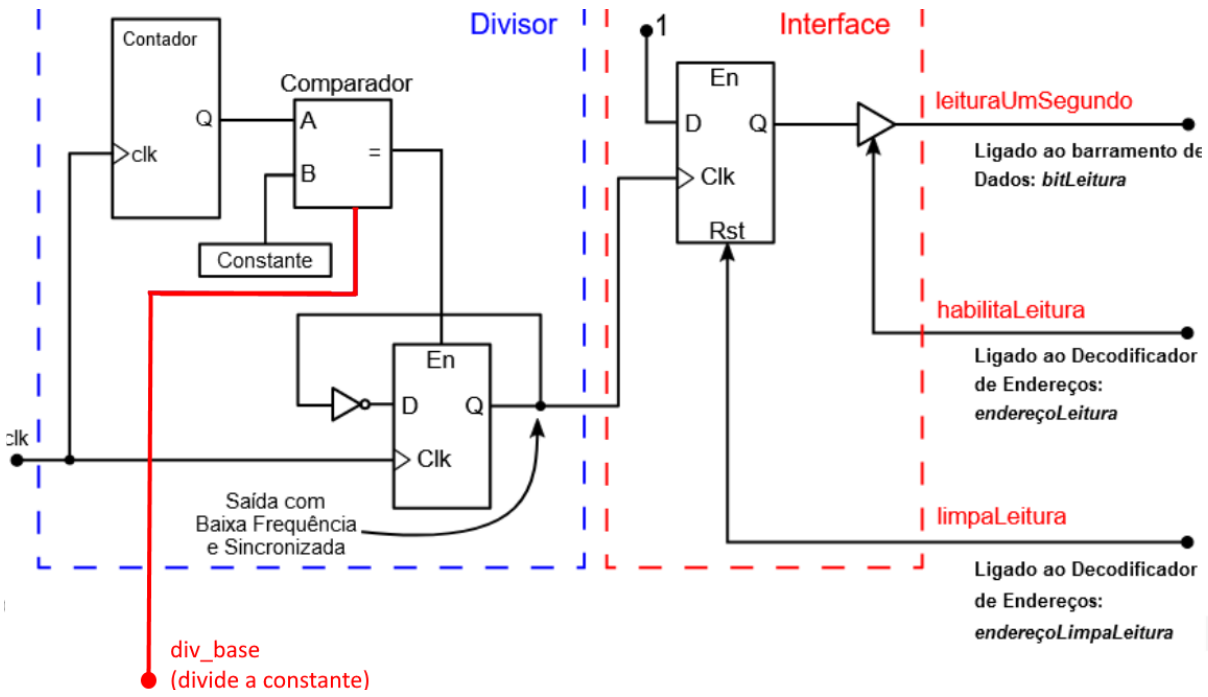
O primeiro bloco (bloco zero) de endereços é utilizado para a memória RAM e são usados 8 bits de largura para seus endereços. Já os periféricos de outputs: os LEDs (e seus respectivos registradores), displays de 7 segmentos (e seus respectivos conversores) e a mudança da base de tempo, e os de inputs: botões, switches e passagem de tempo (e seus respectivos registradores) usam os blocos 4 e 5 respectivamente. Os endereços 511, 510 e 509, do bloco 7, são usados para a limpeza da leitura dos inputs dos botões KEY0, KEY1 e leitura de passagem de tempo, dado que ao pressionar um botão ou certo tempo se passar seu estado se mantém até a próxima limpeza. Os blocos 1, 2, 3 e 6 estão inteiramente reservados para expansão de periféricos.

Endereço em Decimal	Uso	Largura	Bloco de Memória
0 ~ 63	RAM	8 bits	0
64 ~ 255	(não usado)	-	1, 2 e 3
256	LEDs 0 a 7	8 bits	4
257 ~ 258	LEDs 8 e 9	1 bit	
259	Divisor de tempo	8 bits	4
260 ~ 287	(não usado)	-	4
288 ~ 293	Displays de 7 segmentos	4 bits	4
294 ~ 319	(não usado)	-	4
320	Chaves 0 a 7	8 bits	5
321 ~ 322	Chaves 8 e 9	1 bit	5
323 ~ 351	(não usado)	-	5
352 ~ 355	Botões “KEY”	1 bit	5
356	Botão RESET	1 bit	5
357 ~ 508	(não usado)	-	5, 6 e 7
509	Limpeza Tempo	-	7
510 ~ 511	Limpeza leitura botões	-	7

Rascunho do diagrama de conexão do processador com os periféricos:




Data_OUT
D7
D6
D5
D4
D3
D2
D1
D0
Escrita (WR)
Escrita de Dados
8
D0 ~ D7
4
D0 ~ D3
Habilita
1
D0
FF
CLK
Decodificador 3 x 8
Data_Address
A8
A7
A6
A5
A4
A3
A2
A1
A0
Endereços de Dados
E2
E1
E0
S0
S1
S2
S3
S4
S5
S6
S7
Decoder
3 x 8
Bloco0: 0 ~ 63
Bloco1: 64 ~ 127
Bloco2: 128 ~ 191
Bloco3: 192 ~ 255
Bloco4: 256 ~ 319
Bloco5: 320 ~ 383
Bloco6: 384 ~ 447
Bloco7: 448 ~ 511
Endereço: 0
Endereço: 1
Endereço: 2
Endereço: 3
Endereço: 4
Endereço: 5
Endereço: 6
Endereço: 7
Registrador de 8 bits
LED R
Decodificador Binário - 7 segmentos
Registrador de 4 bits
CLK
7
HEX 0
HEX 1
HEX 2
HEX 3
HEX 4
HEX 5



- Por padrão, o divisor conta até 25.000.000, gerando um sinal de frequência 1 Hz (1 sinal por segundo)
- O endereço de mudança de base de tempo controla um valor de 1 até 255 que divide o padrão de 25.000.000, sendo capaz de acelerar a base de tempo
- Endereço de leitura da passagem de tempo se encontra no bloco 5 junto com os outros inputs
- Endereço limpa leitura de passagem de tempo se encontra no bloco 7 junto com outros limpadores de leitura

Descrição de uso:

- IO:

- Display de 7 segmentos: Indica o horário no formato HH:MM:SS. Valores restritos ao formato 24hs.
- LED 0 até 5: Acendem para indicar qual casa do horário (unidade de segundo, dezena de segundo, etc) está tendo seu valor configurado.
- Key 0: Inicia a subrotina de mudança de horário ou avança o seu estado.
- Key 1: Inicia a subrotina de mudança de base de tempo ou avança o seu estado.
- Chaves 0-3: Usada para configurar a mudança de horário.
- Chaves 0-7: Usada para configurar a base de tempo.
- FPGA_RESET: faz reset do programa.

- Modo de uso:

- Assim que o programa é inicializado, o horário começa a contar a partir de 00:00:00 e sempre segue o formato 24 horas
- Apertar Key 0 para parar a contagem de tempo e utilizar as chaves para selecionar um novo horário atual. As chaves SW0 a SW3 são usadas para escrever o valor em binário (para cima é 1 e para baixo é 0). Os LEDs LEDR0 a LEDR5 são utilizados para indicar qual é a casa atual sendo alterada, e Key 0 avança a casa. Valores inválidos (segundo ou minuto acima de 59 ou hora acima de 23) são automaticamente limitados ao menor valor mais próximo
- Apertar Key 1 para parar a contagem de tempo e utilizar as chaves para selecionar um novo divisor para a base de tempo (para acelerar o relógio). Por padrão o divisor é 1 (uma contagem por segundo). As chaves SW0 a SW7 são usadas para definir de uma vez em

quantas vezes o relógio é acelerado, em binário, que pode assumir um valor em decimal de 1 a 255. Valores de 0 são tratados como 1 pelo Quartus para evitar divisão por zero. Aperte Key 1 novamente para confirmar

- Ao passar de 23:59:59, o horário volta para 00:00:00 sem interrupção

Extras:

- Chamada de sub-rotina aninhada (até 8 chamadas).
- Ajuste de horário que não seja através de aumento da frequência da base de tempo.
- Instrução *ADDI*, soma do acumulador com imediato.
- Instrução *JLT*, desvia se menor que.
- Instrução *CLT*, compara se menor que.

Fonte do programa do Contador, em formato ASM comentado:

```
LDI R0, $0 #setup inicializa em 0
STA R0, .LEDS
STA R0, .LED8
STA R0, .LED9
STA R0, .HEX0
STA R0, .HEX1
STA R0, .HEX2
STA R0, .HEX3
STA R0, .HEX4
STA R0, .HEX5
STA R0, @0 #unidade de segundo
STA R0, @1 #dezena de segundo
```

```
STA R0, @2 #unidade de minuto

STA R0, @3 #dezena de minuto

STA R0, @4 #unidade de hora

STA R0, @5 #dezena de hora

STA R0, @6 #flag inibe contagem

STA R0, @15 #cte 0 para saber se botao esta solto

LDI R1, $1

STA R1, .TIME_DIV #base de tempo por padrao 1 seg

LDI R2, $10

STA R2, @14 #cte 10 para verificar estouro

STA R2, .CK0 #limpa k0

STA R2, .CK1 #limpa k1

STA R2, .CTIME #limpa tempo

LDI R3, $6

STA R3, @16 #cte 6 para verificar 60

LDI R3, $2

STA R3, @17 #cte 2 para verificar 24

LDI R3, $4

STA R3, @18 #cte 6 para verificar 24

LDI R3, $3

STA R3, @19 #cte 3 para verificar menor ou igual a 2


MAIN:

LDI R0, $0 #carrega zero

CEQ R0, .TIME  #ve se passou segundo

JEQ .DEPOIS_DO_TIME #pula se estiver solto

JSR .INC #incrementa contagem

JSR .CHECA_60 #checa por carry de horario

DEPOIS_DO_TIME:

JSR .ATUALIZA_DISPLAY #atualiza info da memoria no display

LDI R1, $0 #carrega zero

CEQ R1, .K0 #ve se k0 esta pressionado

JEQ .DEPOIS_DO_K0
```

```
JSR .CONFIGURA_TEMPO #configura horario atual
DEPOIS_DO_K0:
LDI R2, $0 #carrega zero
CEQ R2, .K1 #ve se k0 esta pressionado
JEQ .DEPOIS_DO_K1
JSR .MUDA_BASE #muda base de tempo
DEPOIS_DO_K1:
JMP .MAIN #volta para o inicio do laco


ATUALIZA_DISPLAY:
LDA R0, @0 #carrega unidade de segundo
STA R0, .HEX0 #armazena em hex0
LDA R1, @1 #carrega dezena de segundo
STA R1, .HEX1 #armazena em hex1
LDA R2, @2 #carrega unidade de minuto
STA R2, .HEX2 #armazena em hex2
LDA R3, @3 #carrega dezena de minuto
STA R3, .HEX3 #armazena em hex3
LDA R0, @4 #carrega unidade de hora
STA R0, .HEX4 #armazena em hex4
LDA R1, @5 #carrega dezena de hora
STA R1, .HEX5 #armazena em hex5
RET


INC:
STA R0, .CTIME #limpa tempo
LDI R0, $0 #carrega zero
CEQ R0, @6 #compara com flag inibe contagem e retorna se for diferente de zero
JEQ .FAZ_CONTAGEM
RET
FAZ_CONTAGEM:
```

```
LDA R1, @0 #carrega unidade
SOMAI R1, $1 #soma com um
STA R1, @0 #armazena unidade
JSR .VERIFICA_ESTOURO #verifica estouro
RET


VERIFICA_ESTOURO:
LDA R1, @0 #carrega unidade
CEQ R1, @14 #compara com 10 para verificar estouro
JEQ .CARRY_UN_S
RET
CARRY_UN_S:
LDI R1, $0 #carrega zero
STA R1, @0 #zera unidade seg
LDA R1, @1 #carrega dezena seg
SOMAI R1, $1 #incrementa
STA R1, @1 #armazena dezena seg
CEQ R1, @14 #compara com 10 para verificar estouro
JEQ .CARRY_D_S
RET
CARRY_D_S:
LDI R2, $0 #carrega zero
STA R2, @1 #zera dezena seg
VERIFICA_ESTOURO_MINUTO:
LDA R2, @2 #carrega unidade min
SOMAI R2, $1 #incrementa
STA R2, @2 #armazena unidade min
CEQ R2, @14 #compara com 10 para verificar estouro
JEQ .CARRY_UN_M
RET
CARRY_UN_M:
LDI R3, $0 #carrega zero
```

```
STA R3, @2 #zera unidade min
LDA R3, @3 #carrega dezena min
SOMAI R3, $1 #incrementa
STA R3, @3 #armazena dezena min
CEQ R3, @14 #compara com 10 para verificar estouro
JEQ .CARRY_D_M
RET
CARRY_D_M:
LDI R0, $0 #carrega zero
STA R0, @3 #zera dezena min
VERIFICA_ESTOURO_HORA:
LDA R0, @4 #carrega unidade hora
SOMAI R0, $1 #incrementa
STA R0, @4 #armazena unidade hora
CEQ R0, @14 #compara com 10 para verificar estouro
JEQ .CARRY_UN_H
RET
CARRY_UN_H:
LDI R1, $0 #carrega zero
STA R1, @4 #zera unidade hora
LDA R1, @5 #carrega dezena hora
SOMAI R1, $1 #incrementa
STA R1, @5 #armazena dezena hora
CEQ R1, @14 #compara com 10 para verificar estouro
JEQ .CARRY_D_H
RET
CARRY_D_H:
LDI R2, $0 #carrega zero
STA R2, @5 #zera dezena hora
LDI R2, $1 #carrega um
STA R2, .LED9 #acende led de erro
RET
```

```
CONFIGURA_TEMPO:
STA R0, .CK0 #limpa K0
LDI R0, $1 #carrega um
STA R0, .LEDS #ativa led0 pra indicar unidade segundo
ESPERA_UN_S:
LDA R1, .K0 #carrega k0
CEQ R1, @15 #compara com zero para saber se esta pressionado
JEQ .ESPERA_UN_S
LDA R1, .SWS #carrega chaves
STA R1, @0 #armazena em unidade segundo
STA R1, .CK0 #limpa k0
LDI R1, $2 #carrega 2
STA R1, .LEDS #ativa led1
ESPERA_D_S:
LDA R2, .K0 #carrega k0
CEQ R2, @15 #compara com zero para saber se esta pressionado
JEQ .ESPERA_D_S
LDA R2, .SWS #carrega chaves
STA R2, @1 #armazena em dezena segundo
STA R2, .CK0 #limpa k0
LDI R2, $4 #carrega 4
STA R2, .LEDS #ativa led2
ESPERA_UN_M:
LDA R3, .K0 #carrega k0
CEQ R3, @15 #compara com zero para saber se esta pressionado
JEQ .ESPERA_UN_M
LDA R3, .SWS #carrega chaves
STA R3, @2 #armazena em unidade minuto
STA R3, .CK0 #limpa k0
LDI R3, $8 #carrega 8
STA R3, .LEDS #ativa led3
ESPERA_D_M:
```

```
LDA R0, .K0 #carrega k0
CEQ R0, @15 #compara com zero para saber se esta pressionado
JEQ .ESPERA_D_M
LDA R0, .SWS #carrega chaves
STA R0, @3 #armazena em dezena minuto
STA R0, .CK0 #limpa k0
LDI R0, $16 #carrega 16
STA R0, .LEDS #ativa led4
ESPERA_UN_H:
LDA R1, .K0 #carrega k0
CEQ R1, @15 #compara com zero para saber se esta pressionado
JEQ .ESPERA_UN_H
LDA R1, .SWS #carrega chaves
STA R1, @4 #armazena em unidade hora
STA R1, .CK0 #limpa k0
LDI R1, $32 #carrega 32
STA R1, .LEDS #ativa led5
ESPERA_D_H:
LDA R2, .K0 #carrega k0
CEQ R2, @15 #compara com zero para saber se esta pressionado
JEQ .ESPERA_D_H
LDA R2, .SWS #carrega chaves
STA R2, @5 #armazena em dezena hora
STA R2, .CK0 #limpa k0
LDI R2, $0 #carrega 0
STA R2, .LEDS #desativa leds
JSR .VERIFICA_CONSISTENCIA
STA R0, .CTIME #limpa tempo
RET


VERIFICA_CONSISTENCIA:
LDA R0, @0 #carrega unidade de segundo
```

```
CLT R0, @14 #verifica se menor 10
JLT .PASSA_UN_S
LDI R0, $9 #carrega 9
STA R0, @0 #limita para 9
PASSA_UN_S:
LDA R0, @1 #carrega dezena de segundo
CLT R0, @16 #verifica se menor 6
JLT .PASSA_D_S
LDI R0, $5 #carrega 5
STA R0, @1 #limita para 5
PASSA_D_S:
LDA R0, @2 #carrega unidade de minuto
CLT R0, @14 #verifica se menor 10
JLT .PASSA_UN_M
LDI R0, $9 #carrega 9
STA R0, @2 #limita para 9
PASSA_UN_M:
LDA R0, @3 #carrega dezena de minuto
CLT R0, @16 #verifica se menor 6
JLT .PASSA_D_M
LDI R0, $5 #carrega 5
STA R0, @3 #limita para 5
PASSA_D_M:
LDA R0, @4 #carrega unidade de hora
CLT R0, @18 #verifica se menor 4
JLT .PASSA_UN_H
LDI R0, $3 #carrega 3
STA R0, @4 #limita para 3
PASSA_UN_H:
LDA R0, @5 #carrega dezena de hora
CLT R0, @19 #verifica se menor 3
JLT .PASSA_D_H
LDI R0, $2 #carrega 2
```

```
STA R0, @5 #limita para 2
PASSA_D_H:
RET

CHECA_60:
LDA R0, @0 #carrega unidade de segundo
CEQ R0, @15 #compara com 0
JEQ .DEPOIS_0_SEG #desvia se unidade for 0 seg ou retorna
RET
DEPOIS_0_SEG:
LDA R1, @1 #carrega dezena de segundo
CEQ R1, @16 #compara com 6
JEQ .MINUTO #desvia se dezena for 6 ou retorna
RET
MINUTO:
LDI R0, $0 #carrega zero
STA R0, @0 #zera unidade segundo
STA R0, @1 #zera dezena segundo
JSR .VERIFICA_ESTOURO_MINUTO #soma 1 no minuto e confere estouro
LDA R0, @2 #carrega unidade minuto
CEQ R0, @15 #compara com 0
JEQ .DEPOIS_0_MIN #desvia se unidade for 0 min ou retorna
RET
DEPOIS_0_MIN:
LDA R1, @3 #carrega dezena de minuto
CEQ R1, @16 #compara com 6
JEQ .HORA #desvia se dezena for 6 ou retorna
RET
HORA:
LDI R0, $0 #carrega zero
STA R0, @2 #zera unidade min
STA R0, @3 #zera dezena min
JSR .VERIFICA_ESTOURO_HORA #soma 1 na hora e confere estouro
```

```
LDA R0, @4 #carrega unidade de hora

CEQ R0, @18 #compara com 4

JEQ .DEPOIS_4_HOR #desvia se unidade for 4 horas ou retorna

RET

DEPOIS_4_HOR:

LDA R1, @5 #carrega dezena de hora

CEQ R1, @17 #compara com 2

JEQ .ZERA_HORA #desvia se dezena for 6 ou retorna

RET

ZERA_HORA:

LDI R0, $0 #carrega zero

STA R0, @4 #zera unidade hora

STA R0, @5 #zera dezena hora

RET



MUDA_BASE:

STA R1, .CK1 #limpa k1

ESPERA:

LDA R2, .K1 #carrega k1

CEQ R2, @15 #compara com zero para saber se esta pressionado

JEQ .ESPERA

LDA R1, .SWS #carrega chaves

STA R1, .TIME_DIV #armazena o numero de vezes mais rapido que a base de tempo alternativa eh

STA R1, .CK1 #limpa k1

LDI R1, $100 #carrega 100

RET
```