INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

ARQUITETURA DE COMPUTADORES

LETI/LEE

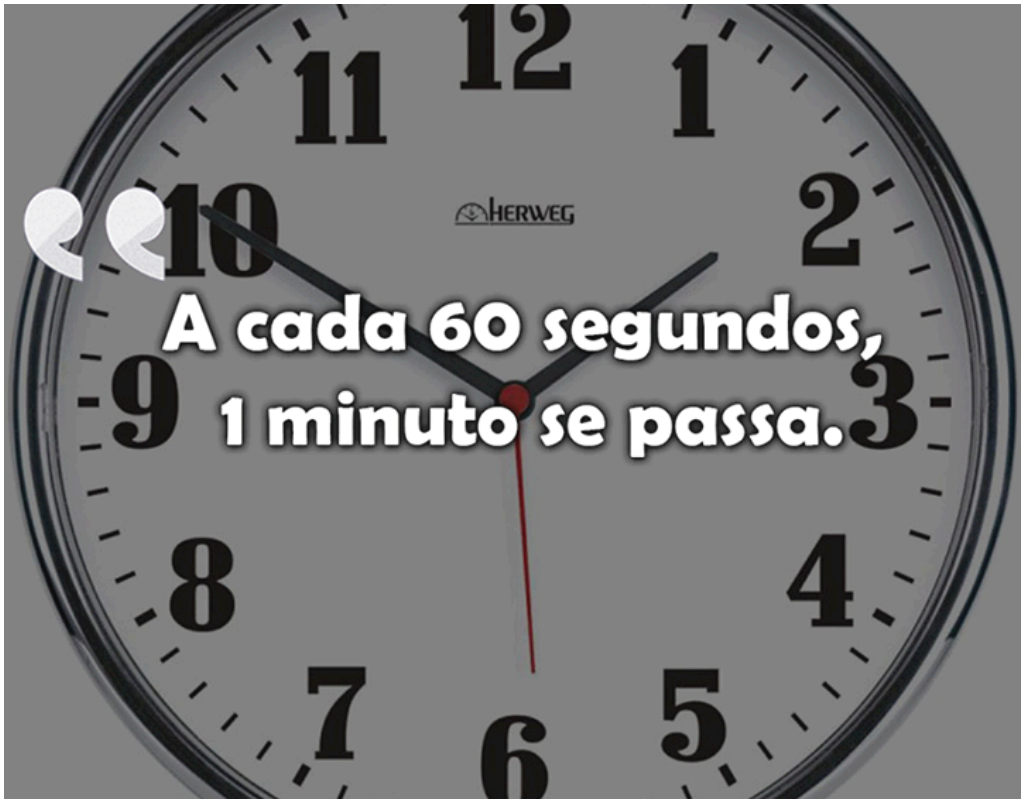
IST-TAGUSPARK

JOÃO BERNARDINO – 78022

TOMÁS MARTINS – 78034

FILIPE FERNANDES - 78083

RELATÓRIO DO PROJETO



1. Introdução

- Este trabalho tem como base fazer em linguagem *assembly* e com auxilio de um simulador, um relógio digital capaz de contar em tempo real (através do RTC) usando como mostrador o pixel screen. Para este relógio funcionar está na base do circuito o microprocessador PEPE 16 a que é ligada uma memória e vários periféricos o pixel screen já referido anteriormente e também um teclado que é usado para escolher o modo de funcionamento do relógio e fazer o acertos destes mesmos modos. Uma vez que o PEPE 16 também se encontra ligado por meio de uma interrupção a duas MUART, é também possível aceder a estes mesmos modos por meio de um terminal. Também ligado por interrupção ao PEPE 16 está o RTC que controla o tempo (mais precisamente os segundos). Para se obter uma maior precisão de tempo é possível usar um outro RTC ligado a uma outra interrupção que aumenta a precisão de tempo em 5 centésimos de segundo. No entanto não é necessário tanta precisão para o relógio funcionar corretamente. Para além do modo relógio existem também os modos cronómetro e alarme que funcionam em simultâneo com o relógio e que têm um funcionamento semelhante.

- Este projecto é feito no âmbito da cadeira de Arquitectura de Computadores lecionada nos cursos de LETI e LEE no Taguspark e aborda na sua essência a linguagem assembly, uma linguagem de baixo nível. No projeto exercita-se fundamentalmente o acesso a periféricos e interrupções com recurso a uma programação cooperativa que tem como objectivo alcançar o correto funcionamento do relógio/cronómetro/alarme.

- Neste projeto é essencial o uso de uma programação cooperativa em que existem várias rotinas que se vão chamando de forma a que o relógio tenha um funcionamento normal e a rotina de interrupção a que está ligada o Real Time Clock, que controla os segundos , se restrinja apenas ao incremento em memória de uma unidade por cada vez que esta é executada.
  O programa deve também estar organizado em três processos sendo estes varrimento do teclado (acima de tudo gere o modo de funcionamento do programa), a interação com o terminal remoto e a gestão de alarmes.

Na secção 2 descreve-se o funcionamento geral do programa concebido, onde é possível ver por que processos o programa passa até chegar ao seu funcionamento final. Nesta secção está também presente um diagrama de blocos que explica resumidamente a orgânica do projeto concebido.

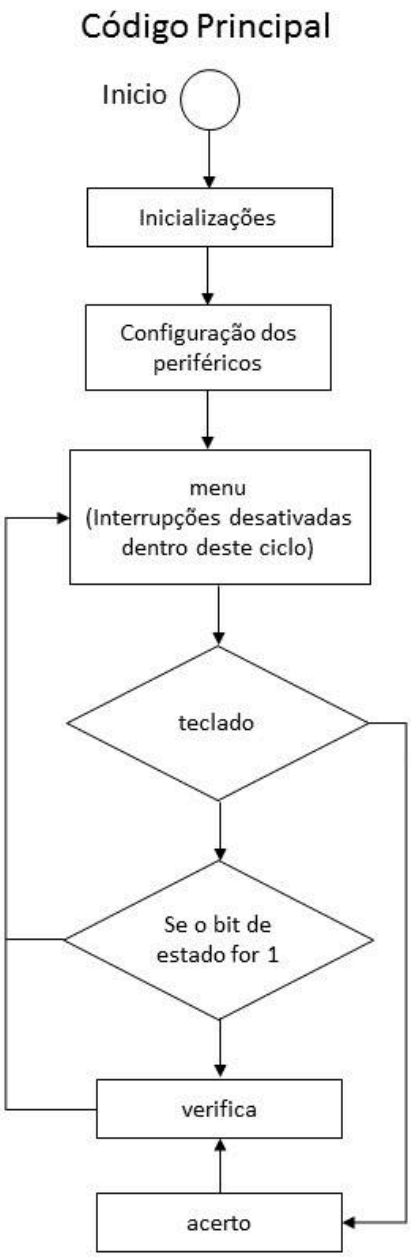
Na secção 3 apresentam-se as conclusões finais do projeto fazendo-se a distinção entre os objectivos iniciais e os objectivos alcançados.

Na secção 4 está o código produzido na concepção deste projeto.

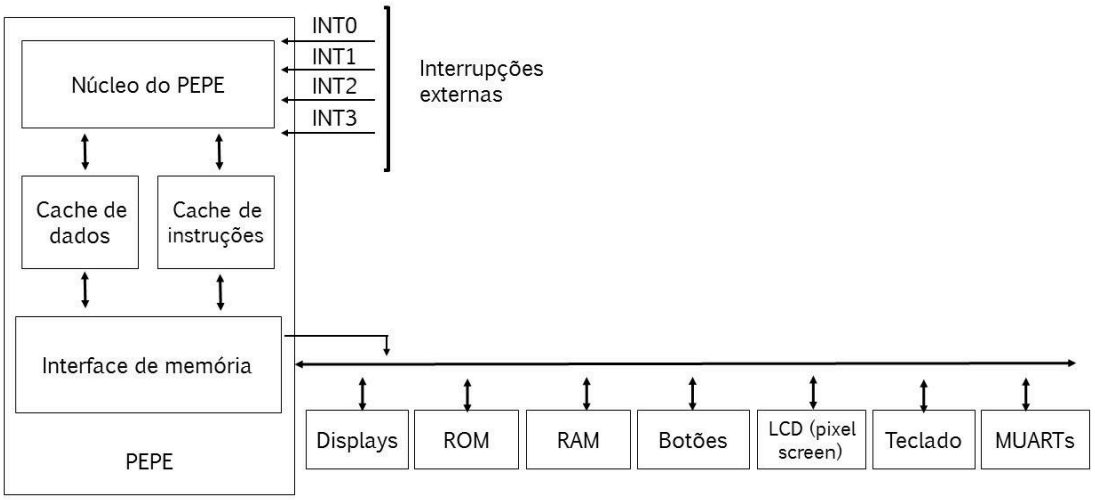
2. Concepção e Implementação

2.1. Estrutura Geral

Fluxograma de *software,* corresponde ao funcionamento do nosso código.



Fluxograma de *hardware*



2.1.1. Mapa de endereçamento escolhido

Descodificação de endereços

Dispositivo	Endereços
RAM (MemBank)	0000H a 7FFFH
Buttons&Clock	8000H a 8FFFH
LEDs	9000H a 9FFFH
Seven Segment Display	A000H a AFFFH
Keypad In/Out	B000H a BFFFH
LCD(PixelScreen)	C000H a C07FH
MUART	D000H a DFFFH

2.1.2. Comunicação entre processos

Explicitar as opções do grupo no que respeita aos mecanismos de comunicação entre os vários processos envolvidos no trabalho.

O trabalho tem dois principais processos, denominados de “verifica” e “menu”.

O processo “menu” está sempre a chamar o processo “teclado” (que devolve em R5 a tecla carregada) e dependentemente da tecla carregada reencaminha o programa para o processo que trata do modo correspondente. No caso de ser o modo relógio vai chamar um processo de acerto de teclado (acerto_rel) e de seguida o processo “verifica” que atualiza o relógio no écran LCD com o formato HH:MM:SS.

O processo “verifica” lê o endereço de memória que é incrementado e atualizado na rotina 0, e recorre a diferentes processos que escrevem no local certo do LCD o digito a atualizar. Estes processos recorrem ao processo “write” que chama o processo “pixel” que acende um pixel nas coordenadas indicadas (x,y)=(R1, R2).

2.1.3. Variáveis de Estado

O nosso código foi desenvolvido de forma a apenas necessitar de uma variável de estado, que serviu para controlar o acesso à função que trata do display e atualização do relógio. Ou seja, apenas depois de a interrupção ser acedida, o código permite que a atualização do display ocorra, limitando assim a uma por ciclo, impedindo erros.

Idem no que toca às variáveis de estado das máquinas implementadas.

2.1.4. Interrupções

Na concepção do projeto usamos exclusivamente a interrupção 0, a qual usamos apenas para incrementar um registo que vai ser guardado num endereço de memoria e copiar para um registo uma variável de comunicação que nos permite saber se a interrupção ocorreu ou não (funcionando com ON=1/OFF=0).

2.1.5. Rotinas

Rotina de Interrupção 0:

Variável dos segundos += 1

-> Verifica:
Se registo_horas = 24 :
  Vai para change24
Se digitodireito_horas > 9 :
  digitoesquerdo_horas += 6
Se registo_minutos > 59 :

digitodireito_horas += 1
Se digitodireito_minutos > 9 :
digitoesquerdo_minutos += 6
Se registo_segundos > 59 :
digitodireito_minutos += 1
Se digitodireito_segundos > 9 :
digitoesquerdo_segundos += 6
Caso contrário:
digitodireito_segundos += 1
change24:
registohoras = 0
registominutos = 0
registosegundos = 0

-> Write:

Esta função vai à memória buscar o desenho de cada linha de um numero, usa o bit mais à esquerda para decidir se liga o pixel numa coordenada, por cada shiftleft aumenta a coordenada, o que permite escrever sucessivamente. O acesso à memoria onde está o desenho depende do numero que pretendermos escrever, sendo que cada um tem o seu espaço.

-> Reset:
endereço_x = 0
x += 1
Vai para reset

(Todas as funções dentro de update_time são semelhantes, exceção para p2)
-- update_time --
-> p2:
x = <variável>
y = <variável>
Acende um pixel nas coordenadas x, y

-> seg2, seg1, min2, min1, hor2, hor1
x = <variável>
y = <variável>
l = 0 (0 para desligar pixels, 1 para ligar)
n = 8 (numero a escrever)
Vai para write (desliga pixels correspondentes a um 8 a começar nas coordenadas dadas)
l = 1
n = <variável>
Vai para write (liga pixels correspondentes ao numero pretendido a começar nas coordenadas dadas)

```
-> acerto_rel:
Varre teclado
Se nada_premido
   Volta para Varre teclado
tecla_premida = <variável>
digitodireito_horas = tecla_premida
Varre teclado
Se nada_premido
   Volta para Varre teclado
tecla_premida = <variável>
digitoesquerdo_horas = tecla_premida
Varre teclado
Se nada_premido
   Volta para Varre teclado
tecla_premida = <variável>
digitodireito_minutos = tecla_premida
Varre teclado
Se nada_premido
   Volta para Varre teclado
tecla_premida = <variável>
digitoesquerdo_minutos = tecla_premida
Varre teclado
Se nada_premido
   Volta para Varre teclado
tecla_premida = <variável>
Se tecla_premida = f
   Volta para o menu
```

3. Conclusões

- Este trabalho tinha como objectivos a produção de um programa em *assembly* capaz de fazer funcionar um relógio em conjunto com 3 alarmes e também com um modo de cronómetro, sendo que o modo principal de relógio em si passava pelo seu acerto através do uso do teclado e pela contagem de tempo usando uma interrupção apenas para incrementar um unidade num endereço de memória.

- Foi concretizado a contagem de tempo do relógio com a atualização correta do écran LCD no formato HH:MM:SS em que também é possível fazer o acerto do relógio continuando este depois a realizar a contagem a partir do tempo para que foi acertado. Foi também produzido o código para o funcionamento do alarme 1 no entanto por falta de tempo não foi possível fazer com que o mesmo alarme funcionasse devidamente.

- Tendo em conta todos os objectivos iniciais apresentados, os objectivos alcançados foram ligeiramente reduzidos no entanto à exceção do

alarme que não foi concluído, o restante funciona na perfeição não tendo sido verificado o mínimo erro na sua execução.

- Durante a realização do projeto tudo decorreu de forma mais ou menos expectada excepto o simulador que é necessário usar, que aparenta ter vários *bugs* o que faz com que seja quase sempre necessário reiniciá-lo entre diferentes compilações/execuções do código. Aconselhamos a que o simulador seja substituído ou optimizado em futuros anos, se possível. De forma a que durante as execuções do programa não haja dúvidas sobre a origem dos possíveis erros que possam aparecer.

4. Código assembly

```
;--------------------------------

; Projeto de AC

; Grupo 4, 3ª Feira 11:00

; João Bernardino - 78022

; Tomás Martins - 78034

; Filipe Fernandes - 78083

;--------------------------------


;--------------------------------

; Constantes

;--------------------------------


mem_write             EQU           03000H  ;

lcd                   EQU           0C000H  ;

lcd_max               EQU           0C07FH  ;


t_segundos            EQU     0500H   ;

t_minutos             EQU     0600H   ;

t_horas               EQU     0700H   ;


sem_relogio           EQU     0FFH    ;

ir_relogio            EQU     0FH     ;

ir_cronometro         EQU     0CH     ;

ir_alarme             EQU     0AH     ;
```

```
ir_asc                  EQU     00H     ;


tal_1                   EQU     01H     ;
tal_2                   EQU     02H     ;
tal_3                   EQU     03H     ;
sem_tecla               EQU     0FFH    ;


ON                      EQU     01H     ;
OFF                     EQU     00H     ;


estado_al1              EQU     0100H   ;
led                     EQU     9000H   ;
t_al_minutos            EQU     0200H   ;
t_al_horas              EQU     0300H   ;


tec                     EQU     0B000H  ;
mem_tec                 EQU             0801H   ;
mem_tec_0               EQU             0800H   ;


; Esta tabela corresponde ao desenho pixelado de cada numero.
; Cada endereço vai ter correspondência a uma linha de um numero.
        PLACE   3000H   ;
d_0:    STRING  07H, 05H, 05H, 05H, 05H, 05H, 07H, 0, 0, 0, 0, 0, 0, 0, 0, 0        ;
d_1:    STRING  01H, 01H, 01H, 01H, 01H, 01H, 01H, 0, 0, 0, 0, 0, 0, 0, 0, 0        ;
d_2:    STRING  07H, 01H, 01H, 07H, 04H, 04H, 07H, 0, 0, 0, 0, 0, 0, 0, 0, 0        ;
d_3:    STRING  07H, 01H, 01H, 07H, 01H, 01H, 07H, 0, 0, 0, 0, 0, 0, 0, 0, 0        ;
d_4:    STRING  05H, 05H, 05H, 07H, 01H, 01H, 01H, 0, 0, 0, 0, 0, 0, 0, 0, 0        ;
d_5:    STRING  07H, 04H, 04H, 07H, 01H, 01H, 07H, 0, 0, 0, 0, 0, 0, 0, 0, 0        ;
d_6:    STRING  07H, 04H, 04H, 07H, 05H, 05H, 07H, 0, 0, 0, 0, 0, 0, 0, 0, 0        ;
d_7:    STRING  07H, 01H, 01H, 01H, 01H, 01H, 01H, 0, 0, 0, 0, 0, 0, 0, 0, 0        ;
d_8:    STRING  07H, 05H, 05H, 07H, 05H, 05H, 07H, 0, 0, 0, 0, 0, 0, 0, 0, 0        ;
d_9:    STRING  07H, 05H, 05H, 07H, 01H, 01H, 07H, 0, 0, 0, 0, 0, 0, 0, 0, 0        ;


;---------------------------------
```

```
; Stack
;---------------------------------

              PLACE   1000H          ;
pilha:  TABLE   100H    ;
ZONA:   TABLE   200H
FIM_PILHA:

;---------------------------------
; Dados
;---------------------------------

PLACE           2200H

; Tabela de vectores de interrupção
tab:            WORD    rot0



;---------------------------------
; Código Principal
;---------------------------------

inic:   PLACE   0H

              MOV     BTE, tab              ;
              MOV     SP, FIM_PILHA  ;
              EI0
              EI

              CALL    reset                 ;
              CALL    p2                    ;
              MOV     R10, 00H              ;
              MOV     R9, 59H               ;
              MOV     R8, 23H               ;
```

```
                MOV     R1, t_segundos
                MOV     R2, t_minutos
                MOV     R3, t_horas
                MOVB    [R1], R10               ;
                MOVB    [R2], R9                ;
                MOVB    [R3], R8                ;

                CALL    seg2
                CALL    seg1
                CALL    min2
                CALL    min1
                CALL    hor2
                CALL    hor1
                MOV     R7, OFF
                MOV     R5, 0


C_ALL:  CMP     R7, ON
                JNZ     main_jump
                DI0
                DI
                CALL    verifica
                CALL    alarme
                EI0
                EI
                MOV     R7, OFF
main_jump:
                CALL    menu
                JMP     C_ALL


;---------------------------------
; Rotinas
;---------------------------------
```

```
; --- Rotina de Interrupção 0 ----------------------------------------

;

; Esta interrupção trata dos impulsos do RTC(Real Time Clock)

; Adiciona 1 a um registo que corresponde aos segundos do relógio, num formato HH:MM:SS

;


rot0:

        PUSH    R1                                      ; Guarda registos de trabalho

        PUSH    R10                                     ;

        MOV     R1, t_segundos          ; Endereço do porto dos segundos

        MOVB    R10, [R1]                       ; Actualiza registo dos segundos com o seu
        valor anterior guardado no endereço t_segundos

        ADD     R10, 1                          ; Incrementa contador

        MOVB    [R1], R10                       ; Actualiza endereço de memoria com os
        segundos

        MOV     R7, ON                          ; Esta variável comunica ao menu se a
        interrupção ocorreu ou não

        POP     R10                             ; Restaura registos de trabalho

        POP     R1                              ;

        RFE                                     ;


; ----- menu ------

;

; Esta função funciona como um menu, onde cada tecla pressionada no teclado vai
direcionar o programa para uma função

;

; Destroí:      R5(registo onde fica a tecla premida)

;               R7(registo de verificacao de RTC, so actualiza o pixelscreen se tiver
                havido aumento no RTC)

;

; ----------------------


menu:

                DI0                             ; Desativa as interrupções para não haver
                alterações do RTC durante o update do display
```

```
DI

PUSH    R0

PUSH    R1

PUSH    R2

PUSH    R3

PUSH    R4


MOV     R0, sem_tecla                ;   Guarda   nos   registos   valores
correspondente à tecla a pressionar

MOV     R1, ir_relogio               ; para aceder a determinada função

MOV     R2, ir_cronometro

MOV     R3, ir_alarme


CALL    teclado                              ;   Chama   a   função   que
percorre o teclado ( apenas uma vez )


CMP     R5, R0                               ; Caso o teclado retorne FF
(valor por defeito para quando

JZ      sai_menu                             ;  nada  é  premido)  saí  do
menu e regressa ao principal


CMP     R5, R1                               ; Caso retorne F vai seguir
para a função que permite acertar o relógio

JZ      entra_acerto

JMP     sai_menu


; As linhas que se seguem, socorrem-se do mesmo processo

; para seguir para a função alarme, que não se encontra finalizada

; como tal ficam aqui declaradas como comentário

;MOV    R1, tal_1

;CMP    R5, R1

;JZ     entra_acerto_al1

;JMP    sai_menu
```

```
entra_acerto:
        ; De forma a evitar erros, é "exigido" ao utilizador que páre de premir a tecla F
        ; antes de prosseguir de vez para a função que permite acertar o relógio
                CALL    teclado
                CMP     R5, R0                  ;  Espera  que  regresse  FF  (nada
premido) do teclado antes de seguir
                JZ      modo_relogio
                JMP     entra_acerto
modo_relogio:
                CALL    acerto_rel              ;  Chamada  da  função  que  permite
acertar
                JMP     sai_menu


; O mesmo processo da função anterior se aplica aqui
; no entanto a função não se encontra finalizada
;entra_acerto_al1:
                CALL    teclado
;               CMP     R5, R0
;               JZ      modo_al1
;               JMP     entra_acerto_al1
;modo_al1:      CALL    acerto_al1
;               JMP     sai_menu


sai_menu:
                POP     R4                      ; Restaura os registos usados
                POP     R3
                POP     R2
                POP     R1
                POP     R0
                EI0
                EI                                      ;     Volta     a     permitir
interrupções para que o relógio
                RET                             ; possa voltar a receber os impulsos
do RTC
```

```
;------------ Verifica --------------------

;

; A função verifica é uma função muito importante no panorama do projeto

; uma vez que é a função que leva os digitos do relógio a aumentar

; apenas quando suposto.

; Ex: No momento imediatamente a seguir aos minutos serem 59, passar a hora a 1 e os
minutos a 0

;

; Para a aplicação desta rotina, o processo passa por detetar se já é altura de mudar os
digitos,

;  se  for,  salta  para  um  momento  que  vai  tratar  somente  dos  digitos  que  necessitam
alteração.

;

; Parâmetros:  R8 - Horas, R9 - Minutos, R10 - Segundos

; São efetuados PUSH's deste registos na mesma, uma vez que o seu valor

; vai ser retirado de um endereço de memória

;

;-----------------------------------------



verifica:

                PUSH    R1                              ; Guarda o valor dos registos usados

                PUSH    R2

                PUSH    R3

                PUSH    R4

                PUSH    R5

                PUSH    R6

                PUSH    R8

                PUSH    R9

                PUSH    R10


                MOV     R1, t_segundos

                MOV     R2, t_minutos

                MOV     R3, t_horas

                MOVB    R10, [R1]               ;

                MOVB    R9, [R2]                ;

                MOVB    R8, [R3]                ;
```

```
; Se forem 23:59:59, vai passar a 00:00:00

                MOV     R4, 24H

                CMP     R8, R4

                JEQ     change24


; Se o 2° digito das horas for maior que 9, vai adicionar um ao 1° digito.

                MOV     R4, 0A0H

                MOV     R5, 0FFH

                SHL     R8, 4                           ; Isolamento do digito da direita do
registo das horas

                AND     R8, R5

                CMP     R8, R4                          ;   Para   efetuar   a   comparação
corretamente

                JEQ     salta_hor1


; Se o 1° registo dos minutos for maior que 59, vai adicionar um ao 2° digito das horas

                MOV     R6, 060H

                CMP     R9, R6

                JEQ     salta_hor2


; Se o 2° digito dos minutos for maior que 9, vai adicionar um ao 1° digito

                MOV     R4, 0A0H

                MOV     R5, 0FFH

                SHL     R9, 4                   ;   Isolamento   do   digito   da   direita   do
registo dos minutos

                AND     R9, R5

                CMP     R9, R4                  ; Para efetuar a comparação corretamente

                JEQ     salta_min1


; Se o registo dos segundos for maior que 59, vai adicionar um ao 2 digito dos minutos

                MOV     R6, 5AH

                CMP     R10, R6

                JEQ     salta_min2
```

```
; Se o 2 digito dos segundos for maior que 9, vai adicionar um ao 1 digito

                MOV     R4, 0A0H

                MOV     R5, 0FFH

                SHL     R10, 4                  ;  Isolamento  do  digito  da  direita  do
registo dos minutos

                AND     R10, R5

                CMP     R10, R4                 ; Para efetuar a comparação corretamente

                JEQ     salta_seg2


; Caso não seja necessária nenhuma alteração aos digitos para além do mais à direita

                MOVB    R10, [R1]               ;  Recebe  o  valor  que  já  encontrava  nos
segundos

                CALL    seg2                    ;  Chama  a  função  que  desenha  o  numero
correspondente ao valor

                JMP     sai_verifica    ;


; As "funções" que se seguem são as responsáveis pelas alterações

; necessárias aos registos quando estes atingem os valores máximos (Ex: 59 nos minutos)

; Tambem chamam as funções que atualizam visualmente os valores


salta_seg2:

                MOVB    R10, [R1]               ;

                ADD     R10, 6                  ; Adiciona 6 ao registo, de forma a impedir
que este tome valores hexadecimais

                MOVB    [R1], R10               ;

                CALL    seg2                    ;  Atualiza  os  digitos  correspondentes  aos
segundos

                CALL    seg1                    ;

                JMP     sai_verifica    ;


salta_seg1:

                MOV     R10, 0                  ; Atribui o valor 0 aos segundos

                MOVB    [R1], R10               ;

                ADD     R9, 1                   ; E adiciona 1 aos minutos

                CALL    seg2                    ; Atualiza os segundos

                CALL    seg1                    ;

                CALL    min2                    ; E o 2° digito dos minutos
```

```
                JMP     sai_verifica   ;


salta_min2:

                MOV     R4, 0AH                ;

                MOV     R5, 0FH                ;

                MOV     R10, 0                 ; Atribui o valor 0 aos segundos

                MOVB    [R1], R10              ;

                MOVB    R9, [R2]               ;

                ADD     R9, 1                  ; Adiciona 1 aos minutos

                MOVB    [R2], R9               ;

                AND     R9, R5                 ;

                CMP     R9, R4                 ; Caso o digito da direita dos minutos
passe o valor 9, vai ser posto o valor 10 nos minutos

                JEQ     salt_min2_spc  ;

                JMP     salt_min2_reg

salt_min2_spc:

                MOV     R9, 010H               ; Coloca-se o valor 10 nos minutos

                JMP     salta_min1             ; Chama-se a função que permite verificar
se os minutos não excedem já os 59

salt_min2_reg:

                CALL    seg2                   ; Caso contrário

                CALL    seg1                   ; atualiza-se os segundos

                CALL    min2                   ; e o digito à direita dos minutos

                JMP     sai_verifica   ;


salta_min1:

                MOV     R4, 5AH                ;

                MOVB    R9, [R2]               ;

                CMP     R9, R4                 ; Caso o registo já ultrapasse os 59
minutos,

                JEQ     salta_hor2             ; vai ser abordado pela função que coloca
os minutos a 0 e adiciona uma hora

                ADD     R9, 6                  ; Caso contrário, é adiciona 6 aos minutos,
para evitar novamente os valores hexadecimais

                MOVB    [R2], R9               ;

                CALL    seg2                   ; Atualiza-se todos os segundos

                CALL    seg1
```

```
                CALL    min2                    ; E todos os minutos

                CALL    min1

                JMP     sai_verifica    ;


salta_hor2:

                MOV     R9, 0                   ; Em chegando aqui é porque se está a
aumentar as horas, como tal coloca-se os minutos a 0

                MOVB    [R2], R9

                MOVB    R8, [R3]

                ADD     R8, 1                   ; E adiciona-se 1 às horas

                MOVB    [R3], R8

                MOV     R4, 0AH                 ; Se o 2° digito for nove

                CMP     R8, R4                  ; Vai para a função que adiciona 6 às horas
( Ex: Saltar de 9->10 )

                JEQ     salta_hor1

                MOV     R4, 24H                 ; Caso o registo tome o valor de 24h, vai
para a função que aborda este caso especifico

                CMP     R8, R4

                JEQ     change24

                CALL    seg2                    ; Atualiza os segundos

                CALL    seg1

                CALL    min2                    ; Os minutos

                CALL    min1

                CALL    hor2                    ; e o digito da direita da horas


                JMP     sai_verifica


salta_hor1:

                ADD     R8, 6                   ; No ultimo caso são adicionadas 6 horas
impedir os valores hexadecimais

                MOVB    [R3], R8

                CALL    seg2                    ;  E  atualizam-se  todos  os  digitos  do
relógio

                CALL    seg1

                CALL    min2

                CALL    min1

                CALL    hor2
```

```
                CALL    hor1


                JMP     sai_verifica


change24:

                MOV     R8, 0                   ; Quando o relógio atinge as 24h, coloca as
horas novamente a 0, reeiniciando o processo

                MOVB    [R3], R8

                CALL    seg2                    ; Atualiza iguamente todos os digitos do
relógio

                CALL    seg1

                CALL    min2

                CALL    min1

                CALL    hor2

                CALL    hor1

                JMP     sai_verifica


sai_verifica:                                   ; Restaura os registos e sai da rotina

                POP     R10

                POP     R9

                POP     R8

                POP     R6

                POP     R5

                POP     R4

                POP     R3

                POP     R2

                POP     R1

                RET




;----- Write ---------------

;

; Esta função escreve no pixelscreen o numero carregado (R3)

; nas coordenadas pretendidas (x,y) -> (R1, R2)
```

```
;
; Parâmetros: R1, R2 e R3
;
; Esta função acede ao desenho de cada numero (na memória)
; e acede o pixel na posição correspondente através dos bits a 1
; de cada linha desse numero.
;
;---------------------------

write:  PUSH    R4                      ;
                PUSH    R5              ;
                PUSH    R6              ;
                PUSH    R7              ;
                PUSH    R8              ;
                PUSH    R9              ;
                PUSH    R10             ;

                MOV     R1, 3                   ; Inicio do desenho na linha 3
                SUB     R2, 2                   ;
                MOV     R10, R2                 ;
                MOV     R4, mem_write   ; Endereço da memória com os desenhos
                MOV     R7, 7           ; Contador para correr 7 linhas
                MOV     R8, 5                   ; Contador para correr as 4 colunas

                MOV     R6, R3                  ; Copiar o numero a desenhar
                SHL     R6, 4                   ; x10 para aceder ao endereço correto
                ADD     R4, R6                  ;   Endereço   correspondente   ao   numero
pretendido

                MOVB    R9, [R4]                ; Lê a 1ª linha do numero
                SHL     R9, 3                   ; Elimina as primeiras 3 posições por serem
sempre zeros
cic_c:          ADD     R2, 1                   ; Adiciona 1 à coluna a ser desenhada
                MOV     R5, 80H                 ; Coloca 1000 0000 no registo
                SUB     R8, 1                   ; Subtraí 1 ao contador de colunas
```

```
                JZ      cic_l               ; Se estivermos na ultima coluna, vai
passar à linha seguinte e repetir

                SHL     R9, 1               ; Atualiza a flag Z com o ultimo bit do
numero

                AND     R5, R9              ; Permite assim analisar se é ou não para
acender um pixel nessa coordenada

                JZ      cic_c               ; Caso seja 0, regressa ao ciclo que vai
passar para a coluna seguinte

                CALL    pixel               ; Caso seja 1, acende um pixel nessa
coordenada

                JMP     cic_c               ; E continua o ciclo para a coluna seguinte


cic_l:          ADD     R1, 1               ; Linha seguinte

                ADD     R4, 1               ; Aumentar o endereço

                MOVB    R9, [R4]            ; Receber o desenho

                SHL     R9, 3               ; Elimina as primeiras 3 posições por serem
sempre zeros

                MOV     R8, 5               ; Recomeçar o contador das colunas

                MOV     R2, R10             ; Recomeçar a coluna

                SUB     R7, 1               ; Subtrair o contador de linhas

                JZ      w_end               ; Quando forem percorridas 4 linhas e 7
colunas, termina

                JMP     cic_c               ;


w_end:  POP     R10     ;       Restaura os registos usados

                POP             R9              ;

                POP             R8              ;

                POP             R7              ;

                POP             R6      ;

                POP             R5              ;

                POP             R4              ;

                RET



;----- pixel -------

;

; Liga/desliga um pixel nas coordenadas escolhidas

;
```

```
; Parâmetros:  R0 - Liga/Desliga
;              R1 - Coordenada X
;              R2 - Coordenada Y
;
; Usada pela função write para acender um pixel numa localização
; exata do display
; --------------------


pixel:          PUSH    R0              ;
                PUSH    R1              ;
                PUSH    R2              ;
                PUSH    R3              ;
                PUSH    R4              ;
                PUSH    R9              ;
                PUSH    R10             ;


                        MOV             R10, lcd        ; Registo que vai receber o endereço
                        MOV             R9, 0           ; Registo que vai receber o valor a
colocar no endereço
                        MOV             R3, 80H         ; Valor para colocar no registo
                        MOV             R4, 0           ; Registo de leitura do valor que já
está no registo


; Determinar a linha
linha:          SHL     R1, 2           ; Multiplicar por 4
                ADD     R10, R1         ; Adicionar ao endereço do bite


; Determinar a coluna
zero:           CMP     R2, 0           ; Se coluna for 0, passa para o display na linha
pretendida
                JZ      onoffz          ;


coluna:         SUB     R2, 1           ; Subtraí 1 à coluna por causa das coordenadas
começarem em 0 e não em 1
                CMP     R2, 7           ; Se a coluna desejada já estiver nas sete mais à
esquerda, só falta determinar o bit e enviar
                JLE     menor_sete      ; Caso contrário, vai ser tratada
```

```
maior_sete:     SUB     R2, 7           ; Subtrai 7 à coluna
                ADD     R10, 1          ; Mas adiciona 1 ao endereço
                JMP     coluna          ; E repete a verificação

menor_sete:     CMP     R2, 0           ; Caso y seja 0, o valor a colocar será sempre 80H
(1000 0000)
                JZ      onoff           ; E vai verificar se é para acender ou apagar

                SUB     R2, 1           ; Usando y como um contador, podemos decrescer o
valor a colocar no endereço.
                SHR     R3, 1           ; Valor a colocar
                CMP     R3, 1           ; Se for para acender o pixel, no ultimo ponto do
byte, tem um tratamento diferente
                JEQ     igual_sete
                JMP     menor_sete      ; Caso contrário terá um bit desviado para a
direita as vezes ditadas pela coluna

igual_sete:     ADD     R10, 1          ; Caso seja para acender o ultimo bit de um byte
                MOV     R3, 80H         ; é só colocar 80H no endereço
                JMP     onoffz          ;

onoff:          SHR     R3, 1           ;
onoffz:         CMP     R0, 0           ; Verificar se é para ligar ou desligar o pixel
                JZ      off             ;
                JNZ     on              ;

on:             MOVB    R4, [R10]       ; Lê o que já se encontra no endereço
                OR      R4, R3          ; Permite adicionar o pixel aos que já lá se
encontram
                MOVB    [R10], R4       ; Guarda as alterações
                JMP     term            ;

off:            MOVB    R4, [R10]       ; Lê o que já se encontra no endereço
                CMP     R4, 0           ;
                JZ      term            ; Caso não haja nada no endereço, termina
```

```
                CMP     R4, R3          ; Caso seja igual, vai desligar todos os bits desse
endereço

                JZ      term_ig         ;

                NOT     R3                      ; Nos restantes casos, a combinação NOT e
AND permite desligar apenas o pretendido

                AND     R4, R3          ; sem afetar outros bits nesse byte

                MOVB    [R10], R4       ; Guarda as alterações

                JMP     term            ;


term_ig:        MOV     R4, 0           ;

                MOVB    [R10], R4       ;


term:           POP     R10             ; Restaura os registos

                POP     R9              ;

                POP     R4              ;

                POP     R3              ;

                POP     R2              ;

                POP             R1              ;

                POP             R0              ;

                RET


;----- reset ------

;

; Limpa o pixel screen

; Esta função toma um papel importante, dado por vezes haver

; lixo no display logo no inicio da simulação

;

; São percorridos todos os bytes do display e colocados a 0.

;

;------------------------------


reset:  PUSH    R1

        PUSH    R2

        PUSH    R3              ;
```

```
        MOV             R1, lcd         ;
        MOV             R2, 0           ;
        MOV             R3, lcd_max     ;

res_cic:        MOVB    [R1], R2        ;
                ADD             R1, 1           ;
                CMP             R1, R3          ;
                JNZ             res_cic         ;

                POP             R3
                POP             R2
                POP             R1              ;
                RET




;----- Rotinas update_time ------
;
; Escreve nas posições certas os numeros do relógio
;
; Parâmetros: R8 - Horas, R9 - Minutos, R10 - Segundos
;
; Estas funções atualizam os digitos desejados com o valor que estiver nos registos
correspondentes
;
;--------------------------------------------------


; 2 Pontos
p2:             PUSH    R1                      ;
                PUSH    R2                      ;

                MOV     R0, 1           ; Assume as coordenadas
                MOV     R1, 05          ;
                MOV     R2, 21          ;
                CALL    pixel           ; e desenha o 1º ponto
```

```
                MOV     R1, 07          ;
                MOV     R2, 21          ;
                CALL    pixel           ; e o 2° ponto

                MOV     R1, 05          ; repete para outros dois pontos
                MOV     R2, 11          ;
                CALL    pixel           ;

                MOV     R1, 07          ;
                MOV     R2, 11          ;
                CALL    pixel           ;

                POP     R2                      ;
                POP     R1                      ;
                RET

; Segundos - 2o digito (o da direita)
seg2:
                PUSH    R1                      ;
                PUSH    R2                      ;
                PUSH    R3                      ;
                PUSH    R5                      ;

; Primeiramente vai ser eliminado um eventual numero que
; se encontrasse na posição
                MOV     R1, 3           ; linha pretendida
                MOV     R2, 27          ; Coluna pretendida
                MOV     R3, 8           ;
                MOV     R0, 0           ; Desligar o pixel
                CALL    write           ;

                MOV     R0, 1           ; Ligar o pixel
                MOV     R1, 3           ; Linha pretendida
```

```
                MOV     R2, 27          ; Coluna pretendida
                MOV     R5, 0FH         ;
                AND     R5, R10         ; Isola o numero da direita do registo
                MOV     R3, R5          ; Numero para escrever
                CALL    write           ; Usa a função write com o numero pretendido
                JMP     term_time


; Segundos - 1o digito
seg1:
                PUSH    R1                      ;
                PUSH    R2                      ;
                PUSH    R3                      ;
                PUSH    R5                      ;

                MOV     R1, 3           ; Linha pretendida
                MOV     R2, 23          ; Coluna pretendida
                MOV     R3, 8           ;
                MOV     R0, 0           ; Desligar o pixel
                CALL    write           ;

                MOV     R1, 3           ; Linha pretendida
                MOV     R2, 23          ; Coluna pretendida
                MOV     R0, 1           ; Ligar o pixel
                MOV     R5, 0F0H        ;
                AND     R5, R10         ; Isola o numero da esquerda do registo
                MOV     R3, R5          ;
                SHR     R3, 4           ;
                CALL    write           ; Usa a função write com o numero pretendido
                JMP     term_time


; Minutos - 2o digito
min2:
                PUSH    R1                      ;
                PUSH    R2                      ;
```

```
                PUSH    R3                      ;
                PUSH    R5                      ;

                MOV     R1, 3           ; Linha pretendida
                MOV     R2, 17          ; Coluna pretendida
                MOV     R3, 8           ;
                MOV     R0, 0           ; Desligar o pixel
                CALL    write           ;

                MOV     R1, 3           ; Linha pretendida
                MOV     R2, 17          ; Coluna pretendida
                MOV     R0, 1           ; Ligar o pixel
                MOV     R5, 0FH         ;
                AND     R5, R9          ; Isola o numero da direita do registo
                MOV     R3, R5          ;
                CALL    write           ; Usa a função write com o numero pretendido
                JMP     term_time

; Minutos - 1o digito
min1:
                PUSH    R1                      ;
                PUSH    R2                      ;
                PUSH    R3                      ;
                PUSH    R5                      ;

                MOV     R1, 3           ; Linha pretendida
                MOV     R2, 13          ; Coluna pretendida
                MOV     R3, 8           ;
                MOV     R0, 0           ; Desligar o pixel
                CALL    write           ;

                MOV     R1, 3           ; Linha pretendida
                MOV     R2, 13          ; Coluna pretendida
                MOV     R0, 1           ; Ligar o pixel
```

```
                MOV     R5, 0F0H        ;
                AND     R5, R9          ; Isola o numero da esquerda do registo
                MOV     R3, R5          ;
                SHR     R3, 4           ;
                CALL    write           ; Usa a função write com o numero pretendido
                JMP     term_time


; Hora - 2o digito
hor2:
                PUSH    R1                      ;
                PUSH    R2                      ;
                PUSH    R3                      ;
                PUSH    R5                      ;

                MOV     R1, 3                   ; Linha pretendida
                MOV     R2, 7           ; Coluna pretendida
                MOV     R3, 8           ;
                MOV     R0, 0           ; Desligar o pixel
                CALL    write           ;

                MOV     R1, 3           ; Linha pretendida
                MOV     R2, 7           ; Coluna pretendida
                MOV     R0, 1           ; Ligar o pixel
                MOV     R5, 0FH         ;
                AND     R5, R8          ; Isola o numero da direita do registo
                MOV     R3, R5          ;
                CALL    write           ; Usa a função write com o numero pretendido
                JMP     term_time

; Hora - 1o digito
hor1:
                PUSH    R1                      ;
                PUSH    R2                      ;
```

```
                PUSH    R3                      ;
                PUSH    R5                      ;

                MOV     R1, 3           ; Linha pretendida
                MOV     R2, 3           ; Coluna pretendida
                MOV     R3, 8           ;
                MOV     R0, 0           ; Desligar o pixel
                CALL    write           ;

                MOV     R1, 3           ; Linha pretendida
                MOV     R2, 3           ; Coluna pretendida
                MOV     R0, 1           ; Ligar o numero que queremos

                MOV     R5, 0F0H        ;
                AND     R5, R8          ; Isola o numero da esquerda do registo
                MOV     R3, R5          ;
                SHR     R3, 4           ;
                CALL    write           ; Usa a função write com o numero pretendido
                JMP     term_time


term_time:      POP     R5      ; Restara os registos e retorna
                POP     R3      ;
                POP     R2      ;
                POP     R1      ;
                RET


; ---- teclado ------
;
; Esta funão vai varrer uma vez o teclado e devolver o valor da tecla pressionada
;
; Retorna R5 com esse valor.
;
;------------------------------
```

```
teclado:        PUSH    R0              ; Guarda os registos de trabalho
                PUSH    R2
                PUSH    R3
                PUSH    R4
                PUSH    R6
                PUSH    R7
                PUSH    R8
                PUSH    R9
                PUSH    R10


; A função guarda na memória os valores 0,1,2 e 3, nas posições
; correspondentes ao formato de entrada dos dados do teclado: 1,2,4 e 8
                 MOV    R10, mem_tec
                 MOV    R9, 0
                 MOV    R8, 1
mem_c:          MOVB    [R10], R9               ; Escreve na memória o valor
                ADD     R9, 1
                ADD     R10, R8                 ; Aumentar o endereço
                SHL     R8, 1                   ; Aumentar o valor a somar ao endereço
                CMP     R9, 5                   ; Repetir 4 vezes
                JEQ     init
                JMP     mem_c


init:           MOV     R3, 1
                MOV     R4, tec
                MOV     R5, 0
                MOV     R0, mem_tec
                MOV     R6, 0FH


ver_press:      MOVB    [R4], R3                ; Escrever no porto de saída
                MOVB    R5, [R4]                ; Ler do porto de entrada
                AND     R5, R6                  ; Isolar o bit proveniente do RTC e ativar
as flags
                JNZ     pres_y                  ; Se pressionada, vai descodificar o valor
```

```
pres_n:         SHL     R3, 1                   ; Se não pressionada vai repetir para as
linhas seguintes

                MOV     R7, 09H

                CMP     R3, R7

                JLT     ver_press


                MOV     R7, 0FFH                ; Caso termine o ciclo sem ser pressionada
nenhuma tecla escreve FF no registo

                JMP     tec_end


pres_y:         MOV     R0, mem_tec_0  ; Se pressionada

                ADD     R0, R3                  ; Vai à memória correspondente ver o valor
descodificado

                MOVB    R7, [R0]

                SHL     R7, 2                   ; Multiplica por 4

                MOV     R0, mem_tec_0

                ADD     R0, R5

                MOVB    R8, [R0]

                ADD     R7, R8


tec_end:        MOV     R5, R7                  ; Recupera os registos e sai

                POP     R10

                POP     R9

                POP     R8

                POP     R7

                POP     R6

                POP     R4

                POP     R3

                POP     R2

                POP     R0

                RET
```

```
; ---- acerto_rel ------

;

; Acerta o relogio

;




acerto_rel:

               PUSH    R0

               PUSH    R1

               PUSH    R2

               PUSH    R3

               PUSH    R4

               PUSH    R8

               PUSH    R9

               PUSH    R10



               MOV     R0, t_segundos        ;Carrega diferentes registos com endereços
               onde estão guardados os segundos, minutos e horas

               MOV     R2, t_minutos

               MOV     R3, t_horas

               MOV     R4, 9

               MOV     R1, sem_tecla         ;Carrega registo com o valer que sai do
               processo teclado quando nenhuma tecla é premida



tec_1: CALL    teclado               ;Ciclo que actualiza o LCD com o digito esquerdo
das horas

               CMP     R5, 2          ;

               JGT     tec_1          ;

               MOV     R8, R5

               SHL     R8, 4

               MOVB    [R3], R8

               CALL    hor1



tec_2:         CALL    teclado        ;Ciclo que actualiza o LCD com o digito direito das
horas e guarda o valor das horas em memoria

               CMP     R5, R4         ;
```

```
                JGT     tec_2           ;
                MOVB    R8, [R3]
                ADD     R8, R5
                MOVB    [R3], R8
                CALL    hor2

tec_3:          CALL    teclado         ;Ciclo que actualiza o LCD com o digito esquerdo
dos minutos
                CMP     R5, 5           ;
                JGT     tec_3           ;
                MOV     R9, R5
                SHL     R9, 4
                MOVB    [R2], R9
                CALL    min1

tec_4:          CALL    teclado         ;Ciclo que actualiza o LCD com o digito direito dos
minutos e guarda o valor dos minutos em memoria
                CMP     R5, R4          ;
                JGT     tec_4           ;
                MOVB    R9, [R2]
                ADD     R9, R5
                MOVB    [R2], R9
                CALL    min2

                MOV     R10, 00H
                MOV     R0, t_segundos
                MOVB    [R0], R10
                CALL    seg1
                CALL    seg2

                MOV     R1, ir_relogio

tec_5: CALL     teclado                 ;certifica se que foi premida a tecla para parar o
acerto de relogio(tecla f=0FH)

                CMP     R5, R1
```

```
                JZ      sai_confirm
                JMP     tec_5


sai_confirm:    CALL    teclado
                MOV     R1, sem_tecla
                CMP     R5, R1
                JZ      sai_acerto_rel
                JMP     sai_confirm


sai_acerto_rel:                         ;Restaura os registos de trabalho e
actualiza o endereço de memoria dos segundos para 00
                MOV     R7, ON
                MOV     R10, 00H
                MOVB    [R0], R10
                POP     R10
                POP     R9
                POP     R8
                POP     R4
                POP     R3
                POP     R2
                POP     R1
                POP     R0
                RET


; ---- acerto_al1 ------
;
; Acerta o alarme 1
;
acerto_al1:                     ;Rotina faz o mesmo que a rotina acerto_rel mas
referente ao acerto do alarme 1


                PUSH    R0
                PUSH    R1
                PUSH    R2
                PUSH    R3
```

```
                PUSH    R4
                PUSH    R8
                PUSH    R9
                PUSH    R10


                MOV     R0, t_al_minutos
                MOV     R2, t_al_horas
                MOV     R4, 9
                MOV     R1, sem_tecla

tec_al1_1:      CALL    teclado

                CMP     R5, 2                   ;
                JGT     tec_al1_1               ;

                MOV     R8, R5          ;escreve o valor da tecla pressionada nas horas
                SHL     R8, 4
                MOVB    [R2], R8
                ;CALL   al_hor1
tec_al1_2:
                CALL    teclado
                CMP     R5, R4                  ;
                JGT     tec_al1_2               ;

                MOVB    R8, [R2]
                ADD     R8, R5

                MOVB    [R2], R8
                ;CALL   al_hor2

tec_al1_3:      CALL    teclado         ;escreve o valor da tecla pressionada nos minutos
                CMP     R5, 5           ;
                JGT     tec_al1_3       ;
```

```
                MOV     R9, R5
                SHL     R9, 4
                MOVB    [R0], R9
                ;CALL   al_min1

tec_al1_4:      CALL    teclado
                CMP     R5, R4          ;
                JGT     tec_al1_4               ;

                MOVB    R9, [R0]
                ADD     R9, R5

                MOVB    [R0], R9
                ;CALL   al_min2
                MOV     R1, tal_1

tec_al1_5:      CALL    teclado                 ;certifica se que foi premida a tecla para
parar o acerto de relogio

                CMP     R5, R1
                JZ      sai_al1_confirm
                JMP     tec_al1_5

sai_al1_confirm:        CALL    teclado
                        MOV     R1, sem_tecla
                        CMP     R5, R1
                        JZ              sai_acerto_al1
                        JMP     sai_al1_confirm

sai_acerto_al1:

                MOV     R10, estado_al1
                MOV     R9, ON
                MOVB    [R10], R9
```

```
                MOV     R1, led                         ;actualiza os leds dando indicação
que o alarme está activo
                MOV     R2, 2
                MOVB    [R1], R2
                POP     R10
                POP     R9
                POP     R8
                POP     R4
                POP     R3
                POP     R2
                POP     R1
                POP     R0
                RET


; ---- alarme ------
;
; Verifica se  o relógio ja alcançou o tempo de alarme e nesse caso dispara o alarme
;
alarme:


                PUSH    R0
                PUSH    R1
                PUSH    R2
                PUSH    R3
                PUSH    R4
                PUSH    R9
                PUSH    R10



                MOV     R10, 0
                MOV     R0, t_al_minutos
                MOV     R1, t_al_horas
                MOVB    R2, [R0]                ;registo com minutos do alarme
                MOVB    R3, [R1]                ;registo com horas do alarme
                MOV     R0, t_minutos
```

```
                MOV     R1, t_horas


                MOV     R4, [R0]                ;registo com minutos do relogio

                MOV     R9, [R1]                ;registo com horas do relogio


                CMP     R2, R4                  ;se os minutos no alarme forem iguais aos
minutos no relogio, vai comparar as horas

                JZ      igual

                JMP     sai_alarme


igual:          CMP     R3, R9                  ;se as horas do relogio e do alarme tambem
forem iguais vai disparar o alarme, se nao for sai

                JZ      disparou                ;e volta ao código principal

                JMP     sai_alarme


disparou:

                MOV     R1, led                 ;quando o alarme dispara actualiza os leds
com esta indicação, sai do alarme e volta à main

                MOV     R2, 4

                MOVB    [R1], R2

                JMP     sai_alarme

sai_alarme:


                POP     R10

                POP     R9

                POP     R4

                POP     R3

                POP     R2

                POP     R1

                POP     R0

                RET
```