Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Trabalho Prático 4 - Jogo Squash para 1

47206 : Tiago Pardal (47206@alunos.isel.pt)

47202 : Manuel Henriques (47202@alunos.isel.pt)

48052 : Manuel Fonseca (48052@alunos.isel.pt)

Relatório para a Unidade Curricular de Arquitetura de Computadores
da Licenciatura em Engenharia Informática e de Computadores

Professor : Engenheiro Jorge Fonseca

16 – 06 – 2022

Índice

Lista de Figuras

Lista de Listagens

1

Introdução

Este trabalho tem por objetivo consolidar todos os pontos centrais lecionados ao longo da cadeira.

Passando pela programação em *assembly*, o acesso a memória, a comunicação com portos de saída e de entrada, assim como o uso da rotina de interrupções.

O código desenvolvido encontra-se disponível em anexo, assim sendo este relatório parte do pressuposto do acesso por parte do leitor ao código desenvolvido no âmbito do mesmo, não sendo assim necessário enunciá-lo em extensão, bastando apenas mencionar trechos do mesmo.

O código disponível em anexo encontra-se ligeiramente modificado em relação ao entregue, isto por se ter reparado em pequenas falhas, que nesta nova versão se encontram corrigidas.

As falhas principais são as seguintes:

- era efetuada um *branch link* para a rotina game_continue, que corretamente deveria ser apenas um *branch*
- eram efetuadas paragens do *picoTimer*, entre jogos, tal como nos foi chamado à atenção pelo professor, tal não deveria ocorrer
- era mantido o valor do registo r2 na rotina de interrupção, através de um push e um pop no final para o *stack*, que era desnecessário, visto não o utilizarmos nesta rotina

2

Problema

2.1 Squash para Um

O problema proposto é o da realização do jogo "Squash para Um".

O objetivo do jogo é do jogador é manter a bola em movimento entre ele mesmo e a parede.

Na versão que se pretende implementar o jogo é unidimensional, sendo representado graficamente por 6 LEDS, representando cada um uma posição distinta entre o jogador e a parede.

O objetivo do jogador é manter a bola em movimento, durante o jogo, quando a bola se posicionar em frente ao jogador, o mesmo tem de efetuar uma tacada, antes que o tempo dessa jogada acabe.

Existem 2 interruptores, para alternar entre os 4 níveis distintos existentes: velocidade lenta, velocidade média, velocidade rápida, e velocidade variável.

O jogador faz uso de um interruptor para efetuar uma tacada.

O deslocamento da bola é visível pelos leds.

2.2 Sistema usado

O hardware usado é a placa de desenvolvimento SDP16, o circuito Pico Timer/Couter (pTC), implementado sobre uma pal e a ATB, tal como demonstrado na figura 2.1.

A placa SDP16, é o processador em si, responsável por armazenar o programa, todas as suas variáveis e por o correr.

A ATB providencia o *clock* para o Pico Timer.

E por fim o *Pico Timer*, é um contador, responsável pela contagem de tempo, um elemento essencial para a implementação do projeto. É através do *Pico Timer* que será feita a gestão de pedidos de interrupção ao processador, consoante as configurações do dispositivo. De acordo com a nossa implementação do projeto, considerámos que as configurações mais adequadas para o resultado que pretendíamos, seria estabelecer um *clock* de 1Khz, juntamente com um limite de contagem de 50. Assim, o dispositivo *Pico Timer* irá realizar um pedido de interrupção a cada 50 *clocks*, o que dado o *clock* 1khz, se traduz em um pedido de interrupção a cada 50 ms, unidade de tempo que utilizamos para realizar a contagem de tempo.

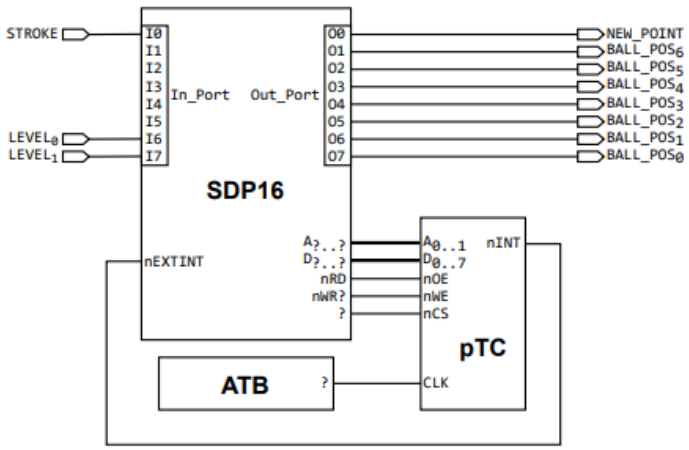


Figura 2.1: Diagram de Blocos do Sistema

3

Detalhes Implementação

3.1 Lógica programa

A lógica do programa encontra-se descrita num asm chart no final deste documento.

O desenvolvimento deste programa focou se em três grandes pontos:

- Ciclo de jogo
- Contagem e apresentação indicador novo ponto
- Movimento da bola
- Verificação de jogada, aquando da bola se encontrar em frente ao jogador
- Game Over
- Implementação dos níveis distintos
- Rotina de Interrupção

Para além dos pontos previamente referidos existem outros, que consideramos, no entanto, de menor relevo, e assim não serão descritos com o mesmo detalhe que os mencionados.

3.1.1 Ciclo de jogo

O Ciclo de jogo baseia-se em 3 verificações. Passou 1 segundo, passou o tempo do nível, se a bola está no player.

- Caso tenha passado 1 segundo aumentamos o *score*, ligamos o led de pontos, atualizamos o a variável do temporizador. Caso o led esteja ligado, e tenham passado 0.25s desligamos.
- Caso o tempo o nível tenha passado, se a bola ainda estiver no jogador perdemos o jogo, caso não, verificamos se está na parede se sim invertemos a direção, e finalmente movemos a bola para a posição seguinte, e atualizamos o temporizador.
- Caso a bola está no jogador verificamos a tacada e caso exista, invertemos a direção, movemos a bola e atualizamos o temporizador do nível.

3.1.2 Contagem e apresentação indicador novo ponto

Tal como é visível na imagem seguinte, começamos por verificar através da nossa variável timer_1s se já passou 1 segundo, se tal se tiver sucedido procedemos a incrementar o score, ligar o led indicador de score e reiniciar essa variável.

Passados 250ms do led estar ligado apagamo-lo.

O valor usado para contabilizar 1 segundo em função do pico timer é 20, pois a cada 20 interrupções causadas pelo pico timer, obtemos 1 segundo, visto cada interrupção ter a duração de 50ms.

3.1.3 Movimento da bola

A rotina set_ball_leds passa como parâmetro a máscara BALL_LEDS_MASK que corresponde a 0xFE, pois não afeta o bit de menor peso do porto de saída, visto este estar destinado ao indicador de ponto.

A direção é guardada em memória como uma variável de tipo byte, que pode possuir o valor 0 ou 1.

Sendo que o valor 0 representa a direção de jogador para a parede, e o valor 1 a direção oposta, da parede para o jogador.

O invert_dir procede à inversão da direção da bola, com base num xor entre a direção atual com 1, o que vai necessariamente alterar o valor de direção.

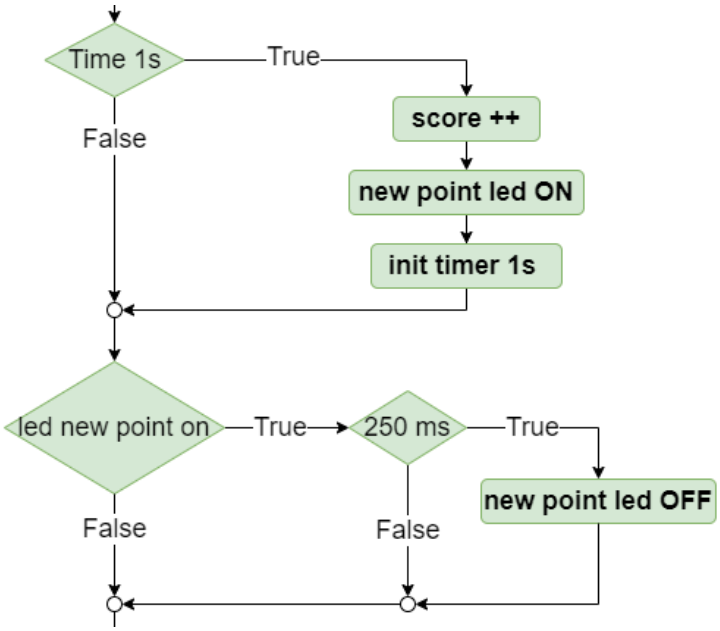


Figura 3.1: Indicador ponto

3.1.4 Verificação de jogada, aquando da bola se encontrar em frente ao jogador

Na presente lógica do programa começamos por verificar se o tempo do nível, ou seja, o tempo para a bola se movimentar, já passou.

Se sim procedemos à verificação de se a bola se encontra em frente ao jogador, se este for o caso, o jogo acabou, pois, o jogador não deu a tacada durante o tempo que tinha para o fazer.

Caso contrário verificamos se a bola se encontra na parede e se esse for o caso mudamos de direção.

Independentemente da bola se encontrar ou não na parede movimentamos a bola.

Caso o tempo de nível ainda não tenha terminado, verificamos se a bola se encontra no jogador, e se este for o caso verificamos se houve tacada.

Se esta ocorrer o jogo prossegue, alterando se a bola de direção, para esta passar a ir outra vez em direção à parede, e movimentando a mesma.

3.1.5 Game Over

No caso do jogador perder, invertemos a direção da bola, para a preparar para uma nova jogada, apresentamos, apresentamos o score durante 5 segundos e após estes 5 segundos voltamos ao inicio, onde aguardamos uma nova jogada.

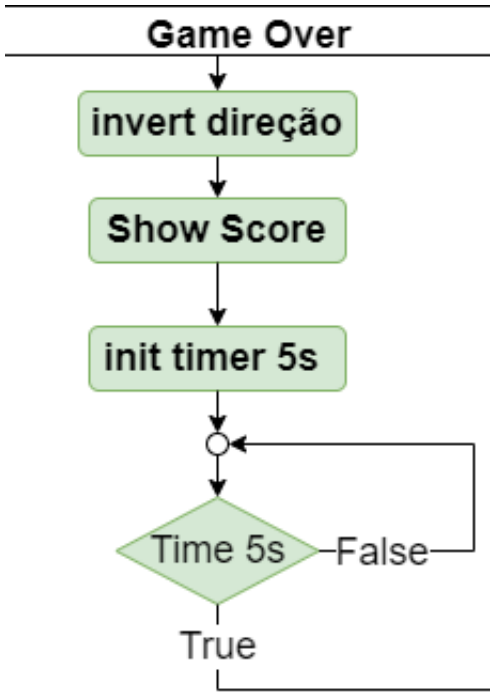


Figura 3.2: Esquema Game Over

3.1.6 Implementação dos niveis distintos

Os tempos usados para cada nivel são os seguintes:

- Nivel 0 Velocidade lenta - 1s (20 ticks)
- Nivel 1 Velocidade média - 0,5s (10 ticks)
- Nivel 2 Velocidade rápida - 0,25s (5 ticks)
- Nivel 3 Velocidade variável - Esta não se encontra implementada, no entanto deixamos preparado de modo a que basta alterarmos uma variável com um valor e o programa funciona.

O jogo implementado permite que antes do início da partida, se escolha o nível de dificuldade pretendido de entre os 4 disponíveis. De modo a permitir a escolha do nível, foram disponibilizados os PINs I6 e I7 do porto de entrada para a seleção.

Visto que cada um dos vários níveis possui o nível de ticks correspondente guardado num array em memória, os valores obtidos nos dois bits do porto entrada são utilizados como index do array e assim é selecionada a velocidade da bola durante a partida.

3.1.7 Rotina Interrupção

A rotina de interrupções é a seguinte:

```
;-------------------------------------------------------------------------
; Rotina:    isr
; Descricao: Rotina responsavel pelo processamento do pedido de
   interrupcao.
; Entradas:  -
; Saidas:    -
; Efeitos:   Incrementa o valor da variavel global ticks
; void isr() {
;   ticks++;
;   //clear Interrupt Request
;}
isr:
    ; Prologo
    push    r0
    push    r1
    ; Corpo da rotina
    ldr     r0, ticks_addr_ext
    ldr     r1, [r0, #0]
    add     r1, r1, #1
    str     r1, [r0, #0]
    ; clear Interrupt Request
    mov r1, 0xFF
    ldr  r0, ptc_addr
    strb r1, [ r0, #pTC_TIR ]
    ; Epilogo
    pop     r1
    pop     r0
    movs    pc, lr


timer_clearInterrupt:
    mov r0, 0
    ldr r1, timer_addr
    strb r0, [ r1, #pTC_TIR ]
    mov pc, lr
```

Listagem 3.1: Rotina Interrupção

A rotina é relativamente simples, na medida em que cada interrupção aceite se limita a

incrementar uma variável de memória, que conta o número de interrupções ocorridas, e de seguida, limpa o pedido de interrupção executando um comando store em pTC-TIR.

Na presente rotina não salvaguardamos lr, pois o mesmo não é usado dentro da interrupção, visto que não efetuamos branches de qualquer tipo dentro desta rotina. Ou seja, não usamos braches with link e por isso não alteramos o valor de lr, sendo assim não temos necessidade de o salvaguardar.

4

Resposta a questões colocadas no enunciado

4.1 Solução adotada para ligar o circuito pTC à placa SDP16

A solução adotada para ligar o circuito pTC à placa SDP16 encontra se descrita na seguinte imagem.

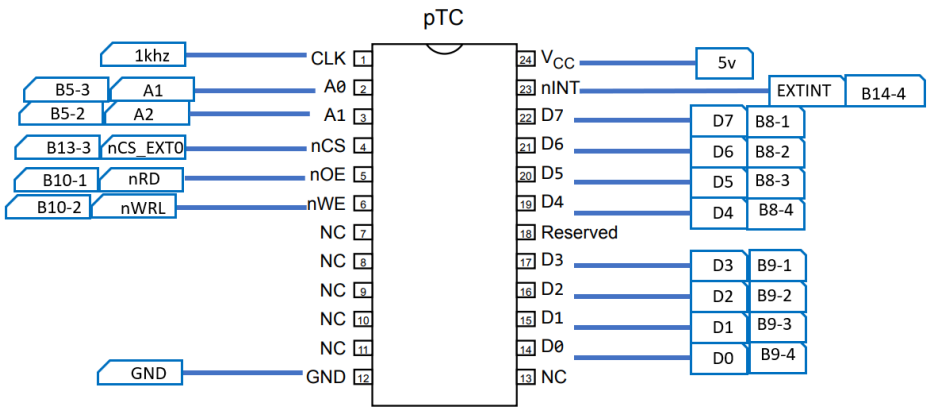


Figura 4.1: Esquema Ligações

4.2 Cálculos para determinar as temporizações envolvidas no trabalho

Ocorre uma interrupção a cada 50 milissegundos.

Visto usarmos 50 para configurar o pico timer.

fin_pTC = 1kHz com o ptc configurado a 50 ficamos com:

$$fout\_ptc = \frac{1khz}{50} = 20Hz$$

Assim os tempos usados no programa obtêm se a partir do seguinte cálculo:

$$ticks = \frac{tempo}{0,05s}$$

Que resulta nos seguintes ticks para cada tempo

- 5s - 100
- 1s - 20
- 0,5s - 10
- 0,25s - 5

4.3 Latência máxima do sistema no atendimento dos pedidos de interrupção gerados pelo circuito pTC

No *worst case scenario*, no caso de nos encontrarmos no inicio de uma instrução com acesso à memoria temos 6 ciclos de *clock*, um ciclo para tratar das *flags*, e colocar o PC a 2 e como no nosso programa temos:

```
1 ldr pc, isr_addr
2 isr_addr:
3     .word   isr
```

Listagem 4.1: Acesso Rotina Interrupção

Devido a não termos a nossa rotina de interrupção logo na linha 2, e em lugar disso fazermos o termos de carregar a mesma de memória, efetuamos mais 6 *clocks* levando a um total de 13 ciclos de *clock* isto com 50khz ficamos com uma latência de 0,26 ms.

$$n\_ciclos\_relogio = 6 + 1 + 6 = 13$$

$$lat\_max = 0,02ms \times 13 = 0,26ms$$

4.4 Tempo de execução no pior caso da rotina da atendimento da interrupção externa

A nossa função de interrupção tem 12 instruções (incluindo o prologo e o epilogo) em que 9 destas acedem a memoria.

Assim o tempo de execução é calculado seguinte modo:

$$n\_ciclos = (3 \times 3) + (9 \times 6) = 63$$

Com 50khz ficamos com um tempo de execução de 1.26 ms.

$$max\_time\_isr = 0,02 \times 63 = 1,26ms$$

5

Conclusão

A realização deste trabalho permitiu-nos consolidar os conhecimentos recentemente obtidos relativos a interrupções, assim como a todos os tópicos, já abrangidos nos trabalhos anteriores, pertencentes a esta uc.

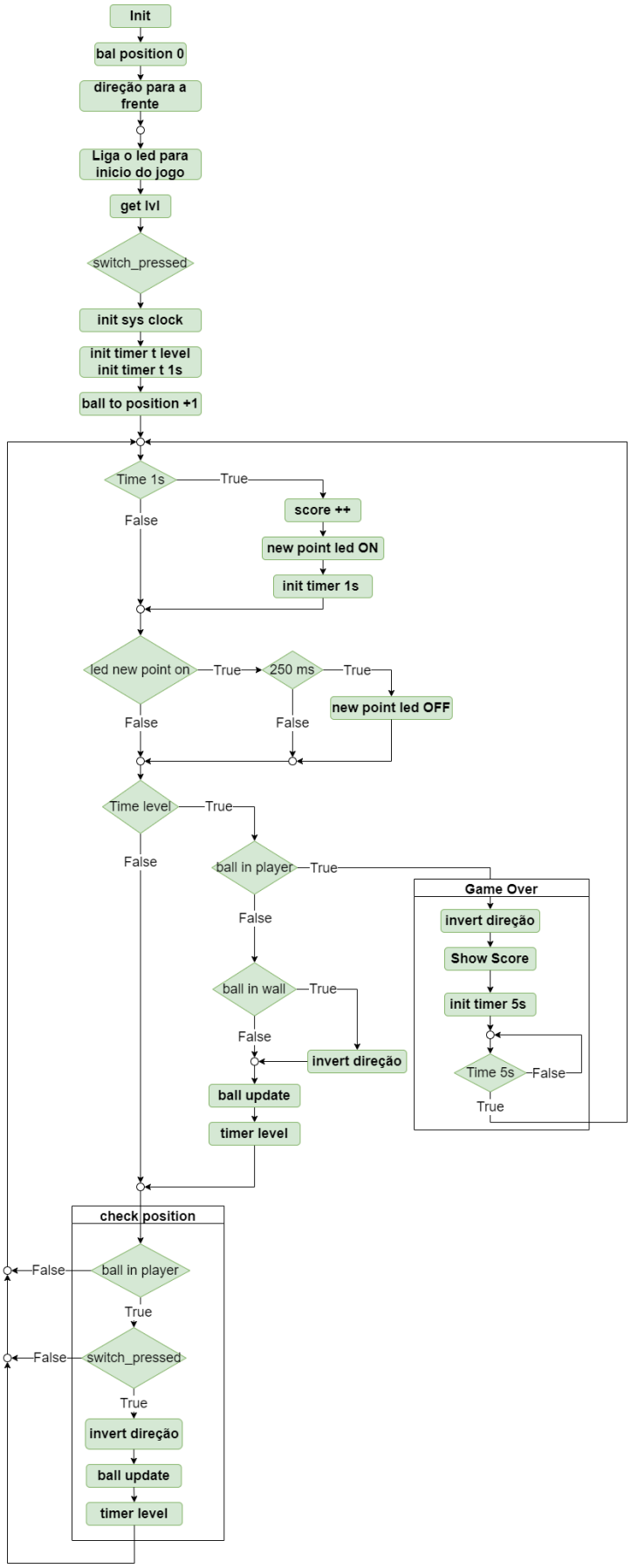
Compreendemos assim a utilidade das interrupções como ferramenta onde em algumas situações é vantajoso o seu uso para acesso a dados de um periférico, visto a maior simplicidade do código e a garantia de sincronismo dada pela rotina de interrupção.

No presente programa não desenvolvemos o último nível, com velocidade variável, visto não termos tido tempo para o mesmo.

Appendice

Aqui encontra se na integra o *ASM chart* do programa, e o código na integra tal como foi exigido.

O *ASM chart* só se apresenta na página seguinte, de modo a permitir a presentara-lo na integra sem cortes.


Init
bal position 0
direção para a frente
Liga o led para inicio do jogo
get lvl
switch_pressed
init sys clock
init timer t level
init timer t 1s
ball to position +1
Time 1s
True
False
score ++
new point led ON
init timer 1s
led new point on
True
250 ms
True
new point led OFF
False
False
Time level
True
False
ball in player
True
Game Over
invert direção
Show Score
init timer 5s
Time 5s
False
True
False
ball in wall
True
False
invert direção
ball update
timer level
check position
ball in player
False
True
switch_pressed
False
True
invert direção
ball update
timer level

```
1
2  ; Autor:      Manuel Fonseca   n: 48052
3  ; Autor:      Manuel Henriques n: 47202
4  ; Autor:      Tiago Pardal     n: 47206
5
6
7
8
9  ; Definicao dos valores dos simbolos utilizados no programa
10 ; valores calculados para o pico timer ligado a 1khz
11
12     .equ    STACK_SIZE, 64           ; Dimensao do stack (em bytes)
13
14     .equ    INPORT_ADDRESS, 0xFF00  ; Endereço do porto de entrada da placa
       SDP16
15     .equ    OUTPORT_ADDRESS, 0xFF00 ; Endereço do porto de saida da placa
       SDP16
16
17     .equ    CPSR_BIT_I, 0x10          ; Mascara para a flag I do registo
       CPSR
18     .equ    PTC_VALUE, 50           ; Intervalo de contagem do circuito pTC
       ; valores calculados para o pico timer ligado a 1khz
19
20                                       ; que suporta a implementação do
       sysclk
21                                       ; fin_pTC = 1kHz fout_ptc=20Hz => T
       =50ms
22
23     .equ    LED0_MASK, 0x01
24     .equ    OUTPORT_INIT_VALUE, 0x00
25
26     .equ    IE_MASK,0x10
27
28     .equ    pTC_ADDRESS, 0XFF40
29     .equ    pTC_TCR, 0
30     .equ    pTC_TMR, 2
31     .equ    pTC_TC,  4
32     .equ    pTC_TIR, 6
33
34     .equ    pTC_CMD_STOP,  1
35     .equ    pTC_CMD_START, 0
36
37     .equ    PLAYER_MASK, 0X80
```

```
38      .equ    WALL_MASK,    0X02
39      .equ    NEW_POINT_LED_MASK, 0X1
40      .equ    LEVEL_INPUT_MASK, 0xc0
41      .equ    BALL_LEDS_MASK, 0xfe
42      .equ    RAKET_MASK, 0x01
43      .equ    LVL_MASK, 0xc0
44      .equ    VALUE_OF_1S, 20 ;0.05 * 20 = 1s
45      .equ    VALUE_OF_5S, 100    ;0.05 * 100 = 5s
46      .equ    VALUE_OF_25, 5      ;0.05 * 5 = 2.5s
47
48      .equ    VARIANT_LEVEL, 3
49
50  ; Seccao:    .startup
51  ; Descricao: Guarda o código de arranque do sistema
52  ;
53      .section .startup
54      b   _start
55      ldr pc, isr_addr
56
57  _start:
58      ldr sp, tos_addr
59      bl SYS_init
60      ldr pc, main_addr
61
62  tos_addr:
63      .word   tos
64  main_addr:
65      .word   main
66  isr_addr:
67      .word   isr
68
69  ;---------------------------------------
70  ;# define OUTPORT_INIT_VALUE 0
71  ;# define LED0_MASK 1
72  ;
73  ;uint16_t ticks = 0;
74  ;
75  ;void main() {
76  ;uint16_t t;
77  ;   outport_init ( OUTPORT_INIT_VALUE );
78  ;   timer_init ( SYSCLK_FREQ );
79  ;   //Habilitar o atendimento das interrupcoes
80  ;   while(1) {
```

```
;       outport_set_bits(LED0_MASK);
;       t = sysclk_get_value ();
;       while ( sysclk_elapsed ( t ) < LED_TOGGLE_TIME );
;       outport_clear_bits(LED0_MASK);
;       t = sysclk_get_value ();
;       while ( sysclk_elapsed ( t ) < LED_TOGGLE_TIME );
;   }
;}
;----------------------------------------
    .text
SYS_init:
    push lr
    bl outport_init
    ;bl timer_stop
    ldr     r0, ticks_addr
    ldr     r1, [r0, #0]
    mov     r1,  #0
    str     r1, [r0, #0]
    mov r0, PLAYER_MASK ;posição de inicio do jogo
    ldr r1, ball_pos_addr
    strb r0, [r1]

    mov r0, PTC_VALUE
    bl timer_init

    mov r0, IE_MASK
    msr cpsr, r0
    pop pc

ball_pos_addr:
    .word   ball_pos

ticks_addr:
    .word ticks

main:
    push lr
    ;bl timer_stop
    ldr r0, direction_addr
    mov r1, 0
    strb r1, [r0]
    b main_while

```

```
124 direction_addr:
125     .word   direction
126 
127 main_while:
128     bl reset_all
129     bl set_ball_leds
130 
131 wait_for_init_stroke:
132     mov r0, RAKET_MASK
133     bl sw_is_pressed
134     add r0,r0,0
135     bzc start_game
136     b    wait_for_init_stroke
137 
138 start_game:
139     bl set_level_dif
140     bl timer_start
141     bl init_timer_lvl
142     bl init_timer_1s
143     bl mov_ball
144     bl set_ball_leds
145 
146 game_loop:
147     mov r0, RAKET_MASK
148     bl sw_is_pressed
149     bl get_timer_1s
150     bl sysclk_elapsed
151     mov r1, VALUE_OF_1S
152     cmp r0, r1 ;20
153     blo one_second_pass_spik
154     bl one_second_pass
155 
156 one_second_pass_spik:
157     ldr r0, new_point_led_addr
158     ldrb r0,[r0]
159     sub r0,r0,0
160     bzs time_lvl
161 
162     bl get_timer_1s
163     bl sysclk_elapsed
164     mov r1, VALUE_OF_25
165     cmp r0, r1      ;5
166     blo time_lvl
```

```
167     mov r0, 0
168     bl set_led_newpoint
169 
170 time_lvl:
171     bl get_timer_lvl
172     bl sysclk_elapsed
173     mov r1, r0
174     bl get_level_dif
175     cmp r1, r0
176     blo await_time_or_player
177 
178     bl init_timer_lvl
179     bl get_ball_position
180     mov r2, PLAYER_MASK
181     sub r0, r0, r2
182     bzc next_move_dir
183     b game_over
184 
185 ; averigua direção em que a bola se deve movimentar
186 next_move_dir:
187     bl get_ball_position
188     sub r0, r0, WALL_MASK
189     bzc continue_game
190     bl invert_dir
191 
192 continue_game:
193     bl mov_ball
194     bl set_ball_leds
195     bl init_timer_lvl
196 
197 ; verifica se o jogador já jogou no caso de a bola se encontrar à frente
        dele
198 await_time_or_player:
199     ;verifica se a bola se encontra no jogador
200     bl get_ball_position
201     mov r2, PLAYER_MASK
202     sub r0, r0, r2
203     bzc game_loop
204 
205     ;verifica se butão foi premido
206     ;se o jogador moveu a raquete
207     mov r0, RAKET_MASK
208     bl sw_is_pressed
```

```
209     add r0,r0,0
210     bzs  game_loop
211
212     bl invert_dir
213     b continue_game
214     b game_loop
215
216
217 ;apresenta do score no porto de saída durante 5 segundos
218 game_over:
219     bl invert_dir
220     mov r0, 0xff
221     bl outport_clear_bits
222     bl get_score
223     bl outport_set_bits
224     bl init_timer_5s
225     mov r4, VALUE_OF_5S
226
227 game_over_loop:
228     bl get_timer_5s
229     bl sysclk_elapsed
230     cmp r0, r4          ;wait 5s
231     blo game_over_loop
232     ;bl timer_stop
233     mov r0, 0xff
234     bl outport_clear_bits
235     b  main_while
236
237 new_point_led_addr:
238     .word   new_point_led
239
240
241 ; incrementa score e apresenta indicador de ponto
242 one_second_pass:
243     ;SCORE ++
244     ;LED ON NEW POINT
245     ;INIT TIMER
246     push lr
247     bl init_timer_1s
248     mov r0, 1
249     bl set_led_newpoint
250     bl add_score
251     pop pc
```

```
252
253
254 ; set led new point to the valu of r0
255 set_led_newpoint:
256     push lr
257     ldr r1, new_point_led_addr_ext
258     strb r0, [r1]
259     mov r1,r0
260     mov r0, NEW_POINT_LED_MASK
261     bl  outport_write_bits
262     pop pc
263
264 new_point_led_addr_ext:
265         .word   new_point_led
266
267
268 ;-------------------------------------------------------------------------
269 ; Funçes relacionadas com a bola
270 ;-------------------------------------------------------------------------
271
272 set_ball_leds:
273     push lr
274     ldr r1, ball_pos_addr_ext1
275     ldrb r1, [r1]
276     mov r0, BALL_LEDS_MASK
277     bl  outport_write_bits
278     pop pc
279
280 invert_dir:
281     push lr
282     ldr r0, direction_addr_ext
283     ldrb r1, [r0]
284     mov r2, 1
285     eor r1, r1, r2
286     strb r1, [r0]
287     pop pc
288
289 get_direction:
290     ldr r0, direction_addr_ext
291     ldrb r0, [r0]
292     mov pc, lr
293
294 get_ball_position:
```

```
295     ldr r0, ball_pos_addr_ext1
296     ldrb r0, [r0]
297     mov pc, lr
298 
299 ball_pos_addr_ext1:
300     .word   ball_pos
301 
302 direction_addr_ext:
303     .word   direction
304 
305 ;-------------------------------------------------------------------------
306 ; Funçes relacionadas com timers
307 ;-------------------------------------------------------------------------
308 
309 init_timer_1s:
310     push lr
311     bl sysclk_get_value
312     ldr r1, timer_1s_addr
313     str r0, [r1]
314     pop pc
315 
316 get_timer_1s:
317     ldr r0, timer_1s_addr
318     ldr r0, [r0]
319     mov pc, lr
320 
321 timer_1s_addr:
322     .word   timer_1s
323 
324 
325 init_timer_5s:
326     push lr
327     bl sysclk_get_value
328     ldr r1, timer_5s_addr
329     str r0, [r1]
330     pop pc
331 
332 get_timer_5s:
333     ldr r0, timer_5s_addr
334     ldr r0, [r0]
335     mov pc, lr
336 
337 timer_5s_addr:
```

```
338     .word   timer_5s
339
340 ;-------------------------------------------------------------------------
341 ; Funçes relacionadas com nivel
342 ;-------------------------------------------------------------------------
343
344 init_timer_lvl:
345     push lr
346     bl sysclk_get_value
347     ldr r1, timer_level_addr
348     str r0, [r1]
349     pop pc
350
351 get_timer_lvl:
352     ldr r0, timer_level_addr
353     ldr r0, [r0]
354     mov pc, lr
355
356 get_level_dif:
357     ldr r0, current_lvl_addr
358     ldrb r0, [r0]
359     mov pc, lr
360
361 set_level_dif:
362     push lr
363     bl inport_read
364     mov r1, LEVEL_INPUT_MASK
365     and r0, r1, r0
366     lsr r0, r0, #6
367     mov r1, VARIANT_LEVEL
368     cmp r0, r1
369     beq set_level_dif //TODO
370
371     ldr r1, lvl_list_addr
372     ldrb r0, [r1, r0]   //lvl_list + input lvl as offset
373     ldr r1, current_lvl_addr
374     strb r0, [r1]
375     pop pc
376
377
378 timer_level_addr:
379     .word timer_level
380
```

```
381 lvl_list_addr:
382     .word lvl_in_time
383 
384 current_lvl_addr:
385     .word current_lvl_dificult_in_time
386 
387 ;-------------------------------------------------------------------------
388 ; Funçes relacionadas com score
389 ;-------------------------------------------------------------------------
390 
391 get_score:
392     push lr
393     ldr r1, score_addr
394     ldr r0, [r1]
395     pop pc
396 
397 add_score:
398     push lr
399     ldr r0, score_addr
400     ldr r1, [r0]
401     add r1, r1, 1
402     str r1, [r0]
403     pop pc
404 
405 
406 ;-------------------------------------------------------------------------
407 ; Rotina:    mov_ball
408 ; Descricao: R
409 ; Entradas:  -
410 ; Saidas:    -
411 ; Efeitos:   Move bola em função de direção
412 ;             Na direção do player ou da parede
413 ; void mov_ball() {
414 ;
415 ;}
416 mov_ball:
417     push lr
418     bl get_direction
419     mov r1, r0
420     bl get_ball_position
421     mov r2, 1
422     and r1,r1,r2
423     sub r1, r1, 0
```

```
424     bzs mov_away
425     ;move from wall to player (BALL_POS6)01 -> (BALL_POS0)07
426     lsl r0, r0,1
427     b finish_mov
428 mov_away:
429     lsr r0, r0,1
430 
431 finish_mov:
432     ldr r1, ball_pos_addr_ext2
433     strb r0, [r1]
434     pop pc
435 
436 
437 ball_pos_addr_ext2:
438         .word ball_pos
439 
440 ;-------------------------------------------------------------------------
441 ; Funcao para preparar o inicio de um novo jogo
442 ; Para o contador, limpa o e limpa o score
443 ;-------------------------------------------------------------------------
444 reset_all:
445     push    lr
446     ;bl timer_stop
447     ;timer sysclk  = 0
448     ldr     r1, ticks_addr_ext
449     mov r0, 0
450     str     r0, [r1, #0]
451     ;score = 0
452     ldr     r1, score_addr
453     mov r0, 0
454     str     r0, [r1, #0]
455 
456     pop     pc
457 
458 
459 score_addr:
460     .word   score
461 
462 ;-------------------------------------------------------------------------
463 ; Rotina:    isr
464 ; Descricao: Rotina responsavel pelo processamento do pedido de
    interrupcao.
465 ; Entradas:  -
```

```
466 ; Saidas:    -
467 ; Efeitos:   Incrementa o valor da variavel global ticks
468 ; void isr() {
469 ;   ticks++;
470 ;   //clear Interrupt Request
471 ;}
472 isr:
473     ; Prologo
474     push    r0
475     push    r1
476     ; Corpo da rotina
477     ldr     r0, ticks_addr_ext
478     ldr     r1, [r0, #0]
479     add     r1, r1, #1
480     str     r1, [r0, #0]
481     ; clear Interrupt Request
482     mov r1, 0xFF
483     ldr  r0, ptc_addr
484     strb r1, [ r0, #pTC_TIR ]
485     ; Epilogo
486     pop     r1
487     pop     r0
488     movs    pc, lr
489
490
491 timer_clearInterrupt:
492     mov r0, 0
493     ldr r1, timer_addr
494     strb r0, [ r1, #pTC_TIR ]
495     mov pc, lr
496
497
498 ;-------------------------------------------------------------------------
499 ;Funcao para devolver o valor corrente da variável global ticks.
500 ;uint16_t sysclk_get_value ( void );
501 ;   return ticks;
502 ;-------------------------------------------------------------------------
503 sysclk_get_value:
504     ldr     r1, ticks_addr_ext
505     ldr     r0, [r1, #0]    ; r0 = ticks
506     mov     pc, lr
507
508 ;-------------------------------------------------------------------------
```

```
;Funcao para devolver o tempo decorrido desde o instante last_read.
;O tempo e medido em unidades de contagem ( ticks ).
;uint8_t sysclk_elapsed ( uint16_t last_read ){
;   return ( ticks - last_read )
;}
;-------------------------------------------------------------------------
sysclk_elapsed:
    ldr  r1, ticks_addr_ext
    ldr  r2, [r1, #0]   ; r0 = ticks
    sub  r0, r2, r0
    mov  pc,lr

ticks_addr_ext:
    .word ticks

;-------------------------------------------------------------------------
;Funcao para iniciar a contagem no periferico.
;void timer_start ( void );
;-------------------------------------------------------------------------
timer_start:
    mov  r1, #pTC_CMD_START
    ldr  r0, ptc_addr
    strb r1, [ r0, #pTC_TCR ]
    mov  pc, lr


timer_write:
    ldr     r2, timer_addr
    add     r0, r0, r0
    strb    r1, [r2,r0]
    mov     pc,lr

timer_addr:
    .word  pTC_ADDRESS
;-------------------------------------------------------------------------
;Funcao para parar a contagem no periferico.
;Colocando o contador com o valor zero.
;void timer_stop ( void );
;-------------------------------------------------------------------------
timer_stop:
    mov  r1, #pTC_CMD_STOP
    ldr  r0, ptc_addr
    strb r1, [ r0, #pTC_TCR ]
```

```
552     mov  pc, lr
553
554 ;-------------------------------------------------------------------------
555 ;Funcao que faz a iniciacao do periferico para habilitar o
556 ;funcionamento em modo continuo e com intervalo de contagem
557 ;interval, em ticks.
558 ;void timer_init ( uint8_t interval );
559 ;-------------------------------------------------------------------------
560 timer_init:
561     push lr
562     push r0
563     ; Parar contagem
564     bl   timer_stop
565     ; Programar intervalo de contagem
566     pop  r0
567     ldr  r1, ptc_addr
568     strb r0, [ r1, #pTC_TMR ]
569     ; Clear Interrupt Request
570     ldr  r1, ptc_addr
571     strb r0, [ r1, #pTC_TIR ]
572     pop  pc
573
574 ptc_addr:
575     .word pTC_ADDRESS
576
577 ;
    ------------------------------------------------------------------------------

578 ;uint8_t sw_is_pressed(uint8_t pin_mask) {
579 ;uint8_t sw_new_state;
580 ;   sw_new_state = inport_test_bits( pin_mask );
581 ;   if ( sw_state == sw_new_state )
582 ;       return 0;
583 ;   sw_state = sw_new_state;
584 ;   if ( sw_new_state == 0 )
585 ;       return 0;
586 ;   return 1;
587 ;}
588 ;
    ------------------------------------------------------------------------------

589 ; Rotina:    sw_is_pressed
590 ; Descricao:
```

```
591 ; Entradas:  pins_mask
592 ; Saidas:    devolve 1 se detecta uma transição 0 -> 1 no pino identificado
        em pin_mask
593 ;            e 0 se não detecta.
594 ; Efeitos:
595 ;
    ------------------------------------------------------------------------

596 sw_is_pressed:
597     push    lr
598     bl      inport_test_bits
599     ; r0 = sw_new_state = inport_test_bits(pins_mask)
600     ldr     r1, sw_state_address
601     ldrb    r2, [r1, #0]    ; r2 = sw_state
602     cmp     r0, r2          ; sw_state == sw_new_state
603     beq     sw_is_pressed_0
604     strb    r0, [r1, #0]    ; sw_state = sw_new_state;
605     sub     r0, r0, #0
606     beq     sw_is_pressed_0
607     mov     r0, #1
608     b       sw_is_pressed_1
609 sw_is_pressed_0:
610     mov     r0, #0
611 sw_is_pressed_1:
612     pop     pc
613
614 sw_state_address:
615     .word   sw_state
616
617 ;
    ------------------------------------------------------------------------

618 ;uint16_t inport_test_bits(uint16_t pins_mask) {
619 ;   return ((inport_read() & pins_mask) == pins_mask);
620 ;}
621 ;
    ------------------------------------------------------------------------

622 ; Rotina:    inport_test_bits
623 ; Descricao: Devolve um se todos dos pinos do porto de entrada
       identificados com o valor um
624 ; em pins_mask tomaremm o valor logico um , ou zero no caso contrario .
625 ; Entradas:  Mascara com os bits a testar
```

```
626 ; Saidas:    Devolve um ou zero conforme a descrição.
627 ; Efeitos:
628 ;
    ------------------------------------------------------------------------

629 inport_test_bits:
630     push    lr
631     push    r4
632     mov     r4, r0
633     bl      inport_read
634     and     r0, r0, r4
635     cmp     r0, r4
636     beq     end_inport_test_bit_1
637     mov     r0, #0
638     b       end_inport_test_bit
639 end_inport_test_bit_1:
640     mov     r0, #1
641 end_inport_test_bit:
642     pop     r4
643     pop     pc
644
645 ;
    ------------------------------------------------------------------------

646 ;uint16_t inport_read() {
647 ;   return [INPORT_ADDRESS];
648 ;}
649 ;
    ------------------------------------------------------------------------

650 ; Rotina:    inport_read
651 ; Descricao: Devolve o valor corrente do estado dos pinos do porto de
    entrada.
652 ; Entradas:
653 ; Saidas:    Valor corrente do porto
654 ; Efeitos:
655 ;
    ------------------------------------------------------------------------

656 inport_read:
657     ldr     r0, inport_address_local
658     ldrb    r0, [r0, #0]
659     mov     pc, lr
```

```
660
661 inport_address_local:
662     .word   INPORT_ADDRESS
663 ;
    ---------------------------------------------------------------------------

664 ;uint8_t outport_init(uint8_t initial_value) {
665 ;   outport_img = initial_value;
666 ;   outport_write(outport_img);
667 ;}
668 ;
    ---------------------------------------------------------------------------

669 ; Rotina:    outport_init
670 ; Descricao: Inicia o porto de saida, atribuindo-lhe o valor do argumento
    passado
671 ;            a rotina.
672 ; Entradas:  Valor para iniciar o porto de saida
673 ; Saidas:
674 ; Efeitos:   Atualiza o valor da variavel imagem do porto
675 ;
    ---------------------------------------------------------------------------

676 outport_init:
677     push    lr
678     mov r0 , #0
679     ldr     r1, outport_img_address
680     strb    r0, [r1, #0]
681     bl      outport_write
682     pop     pc
683
684 ;
    ---------------------------------------------------------------------------

685 ;void outport_set_bits(uint8_t pins_mask) {
686 ;   outport_img |= pins_mask;
687 ;   ourport_write(outport_img);
688 ;}
689 ;
    ---------------------------------------------------------------------------

690 ; Rotina:    outport_set_bits
```

```
691 ; Descricao: Atribui o valor logico '1' aos pinos do porto de saida
    identificados
692 ;            com o valor 1 no argumento passado a rotina. O estado dos
    restantes
693 ;            bits nao e alterado.
694 ; Entradas:  Mascara com os bits a alterar
695 ; Saidas:
696 ; Efeitos:   Atualiza o valor da variavel imagem do porto
697 ;
    ------------------------------------------------------------------------

698 outport_set_bits:
699     push    lr
700     ldr     r1, outport_img_address
701     ldrb    r2, [r1, #0]
702     orr     r0, r2, r0
703     strb    r0, [r1, #0]
704     bl      outport_write
705     pop     pc
706
707 ;
    ------------------------------------------------------------------------

708 ;void outport_clear_bits(uint8_t pins_mask) {
709 ;   outport_img &= ~pins_mask ;
710 ;   ourport_write(outport_img);
711 ;}
712 ;
    ------------------------------------------------------------------------

713 ; Rotina:    outport_clear_bits
714 ; Descricao: Atribui o valor logico '0' aos pinos do porto de saida
    identificados
715 ;            com o valor 1 no argumento passado a rotina. O estado dos
    restantes
716 ;            bits nao e alterado.
717 ; Entradas:  Mascara com os bits a alterar
718 ; Saidas:
719 ; Efeitos:   Atualiza o valor da variavel imagem do porto
720 ;
    ------------------------------------------------------------------------

721 outport_clear_bits:
```

```
722     push    lr
723     ldr     r1, outport_img_address
724     ldrb    r2, [r1, #0]
725     mvn     r0, r0
726     and     r0, r2, r0
727     strb    r0, [r1, #0]
728     bl      outport_write
729     pop     pc
730
731 ;   ------------------------------------------------------------------------

732 ;void outport_write_bits(uint8_t pins_mask, uint8_t value) {
733 ;   value &= pins_mask;
734 ;   outport_img &= ~pins_mask;
735 ;   outport_img |= value;
736 ;   ourport_write(outport_img)
737 ;}
738 ;
    ------------------------------------------------------------------------

739 ; Rotina:    outport_write_bits
740 ; Descricao: Atribui aos pinos do porto de saida identificados com o valor
    lógico
741 ;            um em pins_mask o valor dos bits correspondentes de value. O
    estado
742 ;            dos restantes bits nao e alterado.
743 ; Entradas:  Mascara com os bits a alterar
744 ;         :  valor dos bits a alterar
745 ; Saidas:
746 ; Efeitos:   Atualiza o valor da variavel imagem do porto
747 ;
    ------------------------------------------------------------------------

748 outport_write_bits:
749     push    lr
750     and     r1, r0, r1              ; r1 = pins_mask & value
751     ldr     r2, outport_img_address
752     ldrb    r3, [r2, #0]
753     mvn     r0, r0                  ; ~pins_mask
754     and     r3, r3, r0              ; outport_img &= ~pins_mask;
755     orr     r0, r3, r1              ; outport_img |= pins_mask & value;
756     strb    r0, [r2, #0]
```

```
757     bl      outport_write
758     pop     pc
759
760 ;
    ------------------------------------------------------------------------

761 ;void outport_write(uint8_t value) {
762 ;   outport_img = value;
763 ;   [OUTPORT_ADDRESS] = outport_img;
764 ;}
765 ;
    ------------------------------------------------------------------------

766 ; Rotina:    outport_write
767 ; Descricao: Atribui aos pinos do porto de saida o valor dos bits
    correspondentes de value.
768 ; Entradas:  Valor a escrever no porto
769 ; Saidas:
770 ; Efeitos:   Atualiza o valor da variavel imagem do porto
771 ;
    ------------------------------------------------------------------------

772 outport_write:
773     ldr     r1, outport_addr
774     strb    r0, [r1, #0]
775     mov     pc, lr
776
777 outport_img_address:
778     .word   outport_img
779
780 outport_addr:
781     .word   OUTPORT_ADDRESS
782
783 ; Seccao:    .data
784 ; Descricao: Guarda as variáveis globais com valor inicial definido
785 ;
786     .data
787 timer_level:
788     .word   0
789
790 timer_1s:
791     .word   0
792
```

```
793 timer_5s:
794     .word   0
795
796 current_lvl_dificult_in_time:
797     .word   10
798
799 score:
800     .word   0
801
802 ticks:
803     .word   0       ; uint16_t ticks;
804
805 ball_pos:
806     .byte   0x80
807 lvl_in_time:
808     .byte   20, 10, 5       ; 1s / 0.5s / 0.25s
809
810 new_point_led:
811     .byte   0x00
812
813 direction: ; 0 away from player 1 into the player
814     .byte   0x00
815 sw_state:
816     .byte   0
817 ; Seccao:    .bss
818 ; Descricao: Guarda as variáveis globais sem valor inicial definido
819 ;
820     .section .bss
821 outport_img:            ; Imagem do porto de saida no programa
822     .space  1
823
824 ; Seccao:    .stack
825 ; Descricao: Implementa a pilha com o tamanho definido pelo simbolo
    STACK_SIZE
826 ;
827     .section .stack
828     .space STACK_SIZE
829 tos:
```

Listagem A.1: Código Fonte