INTRODUÇÃO À ARQUITETURA DE COMPUTADORES

LEIC

IST-TAGUSPARK

INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

RELATÓRIO DO PROJETO

2014/2015

GRUPO 2

81900 - Nuno Anselmo

81938 - Mariana Silva

82583 - Miguel Elvas

Índice

1. Introdução

Este projecto foi realizado no âmbito da cadeira de Introdução à Arquitectura de Computadores, com o objectivo de exercitar as componentes fundamentais desta cadeira, nomeadamente a programação em linguagem *Assembly*, os periféricos e as interrupções.

Este consiste na criação de um jogo de simulação de treino de ténis, em que dois robôs vão disparando bolas em diversas direcções e o tenista têm de apanhá-las com a raquete. Se conseguir ganha um ponto por cada bola, caso contrário perde um ponto por cada bola que deixa escapar. O jogo tem duas especificidades face a um treino real: as bolas fazem ricochete nas paredes (topo e fundo do ecrã) e desaparecem quando são apanhadas pela raquete. O jogo termina quando se quiser e a pontuação final, mostrada em displays de 7 segmentos, mede a qualidade do tenista.

Este relatório está dividido em 4 secções (sendo a Introdução a secção **1.**) e está organizado da seguinte maneira:

- A secção **2.** consiste na Conceção e Implementação o projeto, estando esta secção dividida em 2 secções;
  - Na secção **2.1.** descreve-se a Estrutura Geral do projecto, onde falamos da estrutura envolvente, quer em termos de Hardware quer de Software, e são apresentados um diagrama de blocos e um fluxograma;
  - Na secção **2.2.** apresenta-se o mapa de endereçamento escolhido;
    - Na secção **2.2.1.** é descrito o raciocínio lógico por detrás dos processos responsáveis pelo funcionamento do jogo;
    - Na secção **2.2.2.** encontra-se a informação relativa às variáveis de estado;
    - Na secção **2.2.3.** exibe-se as duas únicas interrupções no programa, com respectivas descrições e fluxogramas;
    - Na secção **2.2.4.** encontram-se as principais rotinas e a sua descrição.
- Na secção **3.** apresentam-se as conclusões sobre todo o projeto e algumas ideias de melhoramento;
- Na secção **4.** encontra-se o código, formatado justificado e devidamente comentado.

2. Conceção e Implementação

2.1. Estrutura Geral

Existem duas partes essenciais na elaboração deste projeto, a parte referente ao **Hardware** (parte física do jogo; circuito montado) e a parte referente ao **Software** (parte lógica do jogo; código implementado).

Quanto ao **Hardware**, este foi-nos fornecido utilizando o Simulador disponibilizado pela cadeira. Aqui encontramos uma explicação sucinta do Hardware utilizado e do funcionamento e objetivo de cada componente:

- **PEPE (16 bits) -** Componente responsável pela lógica, aritmética e ainda pela interpretação do código e controlo de todos os componentes;
- **Relógio 1 -** Relógio de tempo real, é utilizado como base para a temporização do movimento dos robôs (a funcionalidade de cada relógio foi trocada, tendo em conta o que nos foi fornecido);
- **Relógio 2 -** Relógio de tempo real, é utilizado como base para a temporização do movimento das bolas (a funcionalidade de cada relógio foi trocada, tendo em conta o que nos foi fornecido);
- **Pixel Screen -** Ecrã de 32×32 pixéis, onde decorre jogo;
- **Teclado –** Teclado 4x4, utilizado para controlar o tenista e para parar e começar o jogo. Tem 4 bits ligados ao periférico POUT-2 e 4 bits ligados ao periférico PIN e a deteção de qual botão está a ser carregado é feita por varrimento.
- **Dois displays de 7 segmentos –** Ligados ao periférico POUT-1, são utilizados para mostrar o contador de bolas apanhadas.

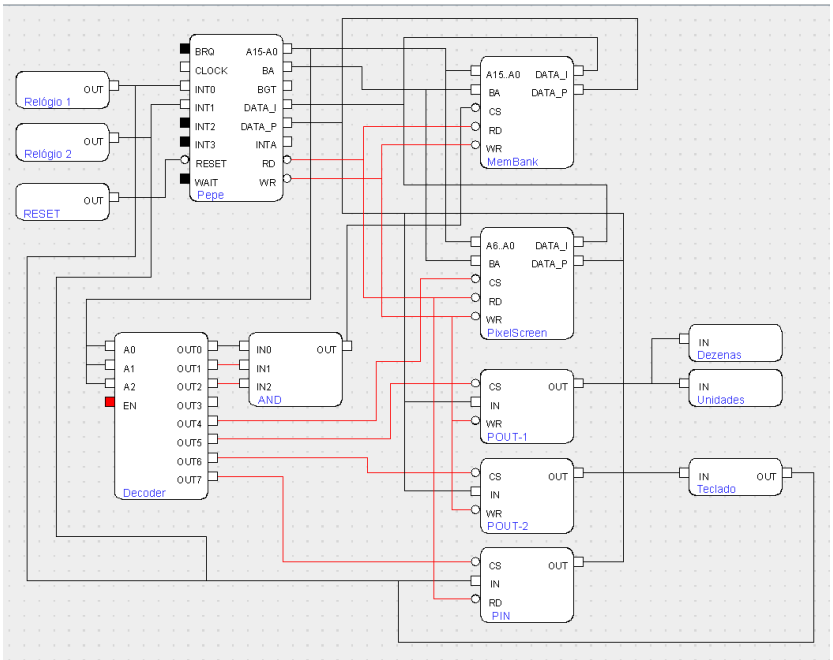
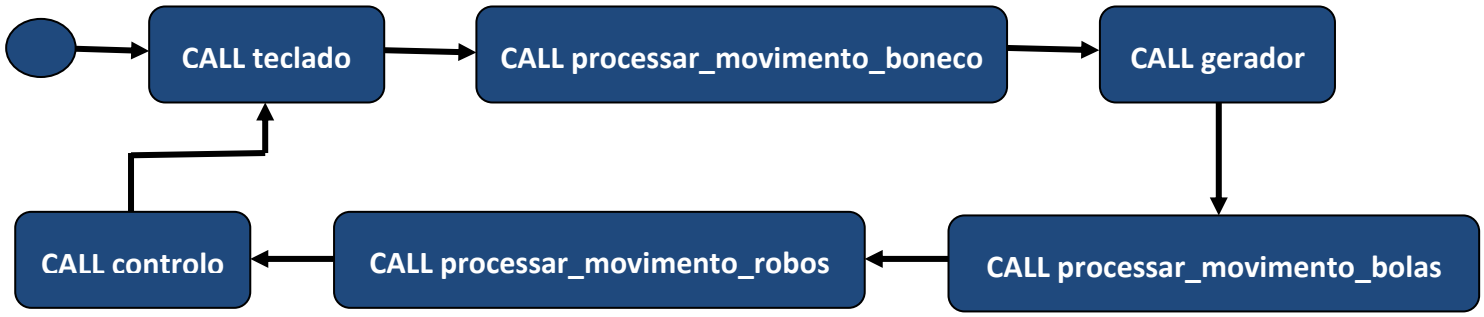


Figura 1 – Diagrama de blocos do Hardware

Quanto ao **Software**, o **Fluxograma 1** mostra o funcionamento base do programa.

O programa começa por chamar a função teclado, que vai detetar que tecla é que foi premida e guardar esse input no registo de memória BUFFER. De seguida é chamada a função processar_movimento_boneco, que vai realizar o movimento do boneco na direção escolhida com o teclado. Depois a função gerador é chamada e esta vai alterar o gerador para que este seja pseudo-aleatório, pois de seguida é chamada a função processar_movimento_bolas, que executa a rotina de movimento de bola para cada bola. De seguida é chamada a função processar_movimento_robos, que para cada um dos robôs, chama a rotina de processamento de movimento de robô. E, por fim, a função controlo é chamada, esta função vai detetar se o jogo foi terminado e entra depois num loop infinito até o jogo ser terminado novamente.

Resumindo, o Software é concebido de forma cíclica entre os vários processos, desencadeando um conjunto de respostas consoante a tecla que foi premida.



Fluxograma 1 – Fluxograma da lógica da implementação do Software

2.2. Mapa de endereçamento escolhido

O mapa de endereçamento escolhido encontra-se na **Tabela 1**. Este mapa é referente aos endereços em que os dispositivos podem ser acedidos pelo PEPE (16 bits).

Dispositivo	Endereços
RAM (MemBank)	0000H a 5FFFH
PixelScreen	8000H a 807FH
POUT-1 (periférico de saída de 8 bits)	0A000H
POUT-2 (periférico de saída de 8 bits)	0C000H
PIN (periférico de entrada de 8 bits)	0E000H

Tabela 1 – Mapa de endereçamento do Hardware.

2.2.1. Comunicação entre processos

A comunicação entre processos é feita de forma a proporcionar uma fácil implementação e um uso recorrente das funções mais importantes.

A forma de comunicação mais utilizada consiste em ter um processo que, através de um CALL, chama outros processos, que por sua vez tratam de todo o processamento necessário, chamando, se preciso, outras rotinas.

Os principais processos são:

- Processo Teclado
- Processo Escrever Pixel
- Processo Desenhar Figura
- Processo Movimento do Boneco
- Processo Movimento do Robô
- Processo Movimento da Bola

2.2.2. Variáveis de Estado

No desenvolvimento do projeto foram utilizadas flags, como a FLAG_RS e a FLAG_BS e foram implementadas quatro tabelas (desenhar o boneco, desenhar os robos, interrupções e teclado). É importante referir que os valores de todos os registos foram sempre 'limpos', visto que, no início e no fim de cada função, são sempre utilizados PUSHs e POPs, respetivamente.

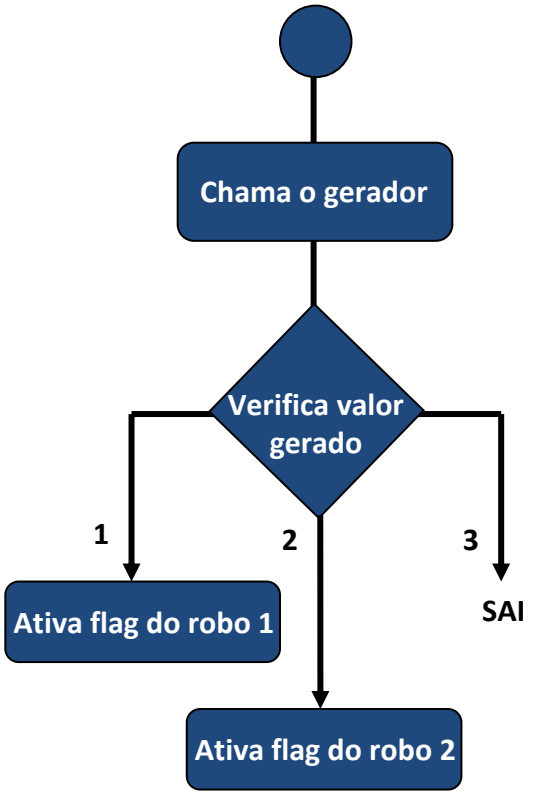
2.2.3. Interrupções

O projeto é constituído por duas rotinas de interrupção, a Rotina de Serviço de Interrupção 1 e a Rotina de Serviço de Interrupção 2:

- A Rotina de Serviço de Interrupção 1 trata da interrupção gerada pelo primeiro relógio, alterando a *flag* FLAG_RS.
- A Rotina de Serviço de Interrupção 2 trata da interrupção gerada pelo segundo relógio, alterando a *flag* FLAG_BS.

Os fluxogramas, **Fluxograma 2** e **Fluxograma 3** mostram as duas rotinas de interrupção:

Rotina de Interrupção 1



Fluxograma 2 - Fluxograma da Rotina de Serviço de Interrupção 1

Rotina de Interrupção 2

Fluxograma 3 - Fluxograma da Rotina de Serviço de Interrupção 2

2.2.4. Rotinas

Descrição das principais rotinas implementadas no projeto:

- **teclado:** Esta rotina detecta o input e guarda esse input no registo de memória BUFFER, guardando a tecla anterior em BUFFER+1.
- **escrever_pixel:** Esta rotina altera o pixel escolhido para o estado dado, isto é, acende ou apaga o pixel.
- **desenhar_figura:** Esta rotina desenha a figura pedida nas coordenadas dadas.
- **processar_movimento_boneco:** Esta rotina realiza o movimento do boneco na direção escolhida com o teclado.
- **gerador:** Esta rotina altera o gerador para que este seja pseudo-aleatório. Pega no valor anteriormente presente e soma 1, caso seja 4 fica 1 (pois vai apenas de 1 a 3).
- **processar_movimento_robo:** Esta rotina executa o movimento do robô, quando é dada uma referência a esse mesmo robô.
- **processar_movimento_bola:** Esta rotina executa o movimento de uma bola, quando é dada uma referência a essa mesma bola.
- **controlo:** Esta rotina deteta se o jogo foi ou não terminado, o que acontece quando a tecla F é premida. Depois entra num loop infinito até ser premida novamente.

3. Conclusões

De acordo com o que foi referido na Introdução, os objetivos iniciais do projecto eram desenvolver as capacidades referentes à programação em linguagem Assembly e ficarmos a perceber melhor as interrupções e os periféricos utilizados. Assim, podemos concluir que, depois da realização deste projeto, estes objectivos foram cumpridos e todas as dificuldades encontradas foram ultrapassadas.

Tudo o que era necessário para a elaboração do projeto foi concretizado e foram feitas duas alterações em relação ao que era pedido. Estas alterações são relativas ao movimento dos robôs, pois não os limitámos a duas posições, podendo estes andar para cima e para baixo desde que não haja colisão com outro robô e não saiam do ecrã; e o movimento dos robôs pode ocorrer mesmo que a bola correspondente se encontre em andamento. Estas alterações não tornam a implementação do código mais fácil mas achámos que faziam sentido, visto que tornam o jogo mais realista e um pouco mais desafiante.

No que toca a melhoramentos, a adição de um número dinâmico de robôs e bolas e o melhoramento do gerador aleatório (visto que atualmente está pouco aleatório) são aspectos que poderiam ter sido adicionados/melhorados no projeto.

4. Código assembly

```
; *********************************************************************
; * P R O J E C T O
; *********************************************************************

; *********************************************************************
; GRUPO 2:
; 81900 - Nuno Anselmo
; 81938 - Mariana Silva
; 82583 - Miguel Elvas
; *********************************************************************

; *********************************************************************
; ESPECIFICACOES ALTERADAS:
; Movimento de robos (1)
; O movimento nao se encontra limitado a 2 posicoes, podendo andar para cima e
; para baixo desde que nao haja colisao com outro robo, e nao saia do ecra
; Movimento de robos (2)
; O movimento pode ocorrer mesmo que se encontre a bola correspondente em andamento
; *********************************************************************

; *********************************************************************
; NOTAS:
; Todos os comentarios se encontram sem acentos, devido a problemas de compatibilidade
; *********************************************************************

; *********************************************************************
; * Constantes
; *********************************************************************

BUFFER   EQU     5600H          ; endereco onde e guardada a tecla

POS_JG   EQU 5650H              ; endereco onde e guardada posicao do boneco (linha, coluna)
POS_RS   EQU     5700H          ; endereco onde e guardada a posicao dos robos (linha robo1,
                                  linha robo2)
COL_RS   EQU     5710H          ; endereco onde e guardada a coluna dos robos
FLAG_RS  EQU     5720H          ; endereco onde e guardada a flag dos robos
GERADOR  EQU     5750H          ; endereco onde e guardado o valor gerado pelo gerador
                                  pseudo-aleatorio
POS_BS   EQU     5800H          ; endereco onde e guardada a posicao das bolas
DIR_BS   EQU     5810H          ; endereco onde e guardada a direccao das bolas (B1, B2)
FLAG_BS  EQU     5820H          ; endereco onde e guardada a flag das bolas
FLAG_DS  EQU     5830H         ; endereco onde e guardada a flag de "disparar"
PONT     EQU     5900H         ; endereco onde e guardada a pontuacao actual (0-99)

FLAG_T   EQU     5910H          ; endereco onde e guardada a flag se indica se o jogo deve
                                  ser pausado ou terminado

NULL     EQU     010H           ; correspondente a tecla "nula"
LINHA    EQU     8000H          ; correspondente a linha 1 antes do ROL
RAQUETE EQU      3              ; correspondente ao tamanho da raquete, comecando no canto
                                  superior direito do boneco
BONUS    EQU     7             ; correspondente a coluna a partir da qual existe bonus de pontos
                                  (indo de 0 a 31!)

PSCR_I   EQU 8000H
PSCR_F   EQU 807FH
POUT1    EQU 0A000H             ; endereco dos scores
POUT2    EQU 0C000H             ; endereco do porto de E/S do teclado
PIN      EQU 0E000H            ; endereco do porto de E/S do teclado
```

```
; ********************************************************************************
; * Stack
; ********************************************************************************

PLACE           5D00H
STACK:          TABLE 100H      ; espaco reservado para o stack
SP_inicial:                     ; endereco para inicializar SP

; ********************************************************************************
; * Dados
; ********************************************************************************

imagem_hexa:    STRING  00H     ; imagem em memoria dos displays hexadecimais

PLACE    2000H
boneco:
         STRING         4, 5            ; Numero de colunas, linhas
         STRING         1000b           ; Desenho (invertido)
         STRING         1010b
         STRING         1111b
         STRING         0010b
         STRING         0101b

PLACE    2100H
robo:
         STRING         3, 3            ; Numero de colunas, linhas
         STRING         110b            ; Desenho (invertido)
         STRING         111b
         STRING         110b
bol_DS:
         STRING         2               ; Altura do robo (a contar de cima) a que dispara a bola

PLACE    2200H
tab_interrup:
         WORD           interrup1       ; Rotina de interrupcao 1
         WORD           interrup2       ; Rotina de interrupcao 2

PLACE    2300H
teclado_movimento:                      ;teclado_movimento com alteracoes linha, coluna
         WORD           0FFFFH          ;0
         WORD           0FFFFH          ;0
         WORD           0FFFFH          ;1
         WORD           0               ;1
         WORD           0FFFFH         ;2
         WORD           1               ;2
         WORD           0               ;3
         WORD           0              ;3
         WORD           0               ;4
         WORD           0FFFFH          ;4
         WORD           0               ;5
         WORD           0               ;5
         WORD           0               ;6
         WORD           1               ;6
         WORD           0               ;7
         WORD           0               ;7
         WORD           1               ;8
         WORD           0FFFFH          ;8
         WORD           1               ;9
         WORD           0               ;9
         WORD           1               ;a
         WORD           1               ;a
         WORD           0               ;b
         WORD           0               ;b
         WORD           0               ;c
         WORD           0               ;c
         WORD           0               ;d
         WORD           0               ;d
         WORD           0               ;e
         WORD           0               ;e
         WORD           0               ;f
         WORD           0               ;f
```

```
; *********************************************************************
; * Codigo
; *********************************************************************

PLACE           0H
 MOV     SP, SP_inicial
 MOV     BTE, tab_interrup
 CALL    reset
 EI0
 EI1
 EI
ciclo_principal:
 CALL    teclado
 CALL    processar_movimento_boneco
 CALL    gerador
 CALL    processar_movimento_bolas
 CALL    processar_movimento_robos
 CALL    controlo
 JMP     ciclo_principal

; *********************************************************************
; * Rotinas
; *********************************************************************

;* -- Rotina de Varrimento de Teclado ----------------------------------------
;*
;* Description: Detecta input, e guarda esse input no registo de memoria BUFFER.
;*              Guarda a tecla anterior em BUFFER+1
;*
;* Parameters: --
;* Return: --
;* Destroy: none
;* Notes: --

teclado:
 PUSH    R1
 PUSH    R2
 PUSH    R3
 PUSH    R4
 PUSH    R5
 PUSH    R6
 PUSH    R7
 PUSH    R8
 MOV     R1, BUFFER            ; R1 com endereco de memoria BUFFER
 MOV     R2, POUT2            ; R2 com o endereco do periferico
 MOV     R3, PIN               ; R3 com endereco de input do teclado
 MOV     R4, 0                 ; R4 vazio, indica a coluna
 MOV     R5, LINHA             ; R5 guarda a linha verificada anteriormente
 MOV     R6, NULL              ; R6 indica o caracter premido, 10 indica 'vazio'
 MOV     R7, 10                ; R7 com o valor a comparar
 MOV     R8, 0FH               ; R8 com mascara que isola bits de entrada do teclado
teclado_ciclo:
 ROL     R5, 1                 ; Alterar linha para verificar a seguinte
 CMP     R5, R7                ; Comparar para saber se ainda "existe" a linha
 JGE     teclado_fim           ; Se a linha a verificar for maior que 4, terminar
 MOVB    [R2], R5              ; Escrever no periferico de saida
 MOVB    R4, [R3]              ; Ler do periferico de entrada
 AND     R4, R8                ; Afectar as flags e isolar os bits de entrada do teclado
 JZ      teclado_ciclo         ; Nenhuma tecla premida
teclado_linha:
 ADD     R6, 4
 SHR     R5, 1
 JNZ     teclado_linha         ; Se ainda nao for zero, ainda ha mais a incrementar
teclado_coluna:
 ADD     R6, 1
 SHR     R4, 1
 JNZ     teclado_coluna        ; Se ainda nao for zero, ainda ha mais a incrementar
 MOV     R7, 15H
 SUB     R6, R7                ; Incrementamos 1x4 e 1x1 a mais, e o 1 inicial de 'vazio'
```

```
teclado_fim:
 MOVB    R8, [R1]              ; Guardar tecla premida anteriormente (ou 10 caso vazia)
 MOVB    [R1], R6              ; Escrever para memoria a tecla que pode ser nulo (10)
 ADD     R1, 1
 MOVB    [R1], R8              ; Escrever para memoria a tecla premida anteriormente
 POP     R8
 POP     R7
 POP     R6
 POP     R5
 POP     R4
 POP     R3
 POP     R2
 POP     R1
 RET


;* -- Rotina de Limpar Ecra ------------------------------------------
;*
;* Description: Apaga todo o ecra (apaga todos os pixeis).
;*
;* Parameters:  --
;* Return:      --
;* Destroy: none
;* Notes: --

limpar_ecra:
 PUSH    R1
 PUSH    R2
 PUSH    R3
 PUSH    R4
 MOV     R1, PSCR_I            ; Primeiro endereco do ecra
 MOV     R2, PSCR_F            ; Ultimo endereco do ecra
 MOV     R3, 0
 MOV     R4, 1
ciclo_limpeza:
 MOVB    [R1], R3              ; Apagar o byte
 ADD     R1, R4                ; Avancar para o proximo byte
 CMP     R1, R2                ; Comparar o endereco actual com o ultimo
 JLE     ciclo_limpeza         ; Caso nao seja o ultimo, continuar a limpar
 POP     R4
 POP     R3
 POP     R2
 POP     R1
 RET

;* -- Rotina de Pintar Ecra ------------------------------------------
;*
;* Description: Pinta todo o ecra (acende todos os pixeis)
;*
;* Parameters:  --
;* Return:      --
;* Destroy: none
;* Notes: --

pintar_ecra:
 PUSH    R1
 PUSH    R2
 PUSH    R3
 PUSH    R4
 MOV     R1, PSCR_I            ; Primeiro endereco do ecra
 MOV     R2, PSCR_F            ; Ultimo endereco do ecra
 MOV     R3, 0FFFFH
 MOV     R4, 1
ciclo_pintura:
 MOVB    [R1], R3              ; Apagar o byte
 ADD     R1, R4                ; Avancar para o proximo byte
 CMP     R1, R2                ; Comparar o endereco actual com o ultimo
 JLE     ciclo_pintura         ; Caso nao seja o ultimo, continuar a limpar
 POP     R4
 POP     R3
 POP     R2
 POP     R1
 RET
```

```
;* -- Rotina de Escrever Pixel ------------------------------------------
;*
;* Description: Altera o pixel escolhido, para o estado dado (aceso ou apagado)
;*
;* Parameters:  R1 (linha), R2 (coluna), R3 (valor, aceso ou apagado)
;* Return:      --
;* Destroy: none
;* Notes: --

escrever_pixel:
  PUSH    R1                    ; Guarda a linha
  PUSH    R2                    ; Guarda a coluna
  PUSH    R3                    ; Guarda o valor (aceso ou apagado)
  PUSH    R4                    ; Registo auxiliar 1
  PUSH    R5                    ; Registo auxiliar 2
  PUSH    R6                    ; Registo auxiliar 3

                ; Byte a alterar = L*4 + C/8 + 8000H
  MOV     R5, 4
  MUL     R1, R5                ; L*4

  MOV     R5, 8
  MOV     R4, R2
  DIV     R4, R5                ; C/8
  ADD     R1, R4                ; L*4 + C/8

  MOV     R5, 8000H
  ADD     R1, R5                ; L*4 + C/8 + 8000H

                ; Fazer modulo, visto que MOD causa problemas (acusa negativo quando
                  resto 0)
                ; a%b = a - (a/b)*b
  MOV     R5, 8
  MOV     R6, R2
  DIV     R6, R5
  MUL     R6, R5
  SUB     R2, R6

  MOV     R5, 1
  MOV     R4, 80H
  AND      R2, R2
  JZ       escrever_memoria
escrever_ciclo:
  SHR     R4, 1
  SUB     R2, 1
  JNZ     escrever_ciclo
escrever_memoria:
  MOVB    R6, [R1]              ; Guardar o valor anterior do byte
  AND     R3, R3
  JZ      escrever_desligar
escrever_ligar:
  OR      R4, R6                ; Mascara para preservar o valor anterior
  JMP     escrever_fim
escrever_desligar:
  MOV     R5, 0FFH
  XOR     R4, R5
  AND     R4, R6                ; Mascara para preservar o valor anterior
escrever_fim:
  MOVB    [R1], R4
  POP     R6
  POP     R5
  POP     R4
  POP     R3
  POP     R2
  POP     R1
  RET
```

```
;* -- Rotina de Desenho de Figura ----------------------------------------
;*
;* Description: Desenha a figura pedida, nas coordenadas dadas.
;*
;* Parameters:  R1 (linha), R2 (coluna), R10 (endereco de memoria de figura a desenhar)
;* Return:      --
;* Destroy: none
;* Notes: --

desenhar_figura:
  PUSH    R1                    ; Linha para rotina escrever_pixel
  PUSH    R2                    ; Coluna para rotina escrever_pixel
  PUSH    R3                    ; Aceso ou apagado para rotina escrever_pixel
  PUSH    R4                    ; Guardar coluna canto superior esquerdo
  PUSH    R5                    ; Valor de auxilio
  PUSH    R6                    ; Contador de colunas
  PUSH    R7                    ; Contador de linhas
  PUSH    R8                    ; Mascara isoladora de ultimo bit, bem como controlador de apenas
                                  apagar
  PUSH    R9                    ; Posicao na memoria de linha a desenhar
  PUSH    R10                   ; Posicao na memoria de figura a desenhar (primeiros dois
                                  enderecos sao dimensoes, depois desenho)
  SUB     R1, 1
  MOV     R4, R2
  MOV     R7, R10
  ADD     R7, 1
  MOVB    R7, [R7]              ; Numero maximo de linhas
  ADD     R7, 1                 ; Adicionar um porque se realiza uma subtraccao (de 1) a mais
  MOV     R9, R10
  ADD     R9, 2
desenhar_linha:
  SUB     R7, 1                 ; Subtrair 1 ao contador para verificar se existem mais linhas
  JZ      desenhar_fim          ; Caso tenham acabado as linhas, terminar
  MOVB    R5, [R9]
  ADD     R9, 1
  MOV     R6, R10
  MOVB    R6, [R6]              ; Numero maximo de colunas
  ADD     R1, 1
  MOV     R2, R4
desenhar_coluna:
  MOV     R3, R5
  AND     R3, R8                ; Isolar bit de menor valor
  CALL    escrever_pixel
  ADD     R2, 1
  SHR     R5, 1                 ; Avancar para o proximo bit
  SUB     R6, 1                 ; Subtrair 1 ao contador para verificar se existem mais colunas
  JNZ     desenhar_coluna       ; Caso haja mais colunas para desenhar, continuar
  JMP     desenhar_linha        ; Caso tenha terminado a linha, terminar ciclo
desenhar_fim:
  POP     R10
  POP     R9
  POP     R8
  POP     R7
  POP     R6
  POP     R5
  POP     R4
  POP     R3
  POP     R2
  POP     R1
  RET
```

```
;* -- Rotina de Processamento de Movimento de Boneco ------------------------------------------
;*
;* Description: Realiza o movimento do boneco na direccao escolhida com o teclado.
;*
;* Parameters:  --
;* Return:      --
;* Destroy: none
;* Notes: --

processar_movimento_boneco:
  PUSH    R1
  PUSH    R2
  PUSH    R3
  PUSH    R4
  PUSH    R5
  PUSH    R6
  PUSH    R7
  PUSH    R8
  PUSH    R9
  PUSH    R10
  MOV     R10, boneco
  MOV     R3, BUFFER
  MOVB    R3, [R3]              ; R3 possui a tecla carregada actualmente
  MOV     R2, BUFFER
  ADD     R2, 1                 ; R2 possui a tecla carregada anteriormente
  MOVB    R2, [R2]
  CMP     R3, R2                ; Apenas se processa se a tecla premida anteriormente for maior
                                  que a premida
  JGE     movimento_fim         ; pois 10H (nula) e maior que todas as teclas premidas.
                                ; Se a tecla premida for nula, sera maior ou igual, logo jump
                                ; Se a tecla premida nao mudar, sera igual, logo jump.
  MOV     R1, POS_JG
  MOVB    R1, [R1]              ; Guardar em R1 a linha actual
  MOV     R2, POS_JG
  ADD     R2, 1
  MOVB    R2, [R2]              ; Guardar em R2 a coluna actual

  MOV     R7, R1                ; Guardar em R7 a linha actual (para limpeza)
  MOV     R8, R2                ; Guardar em R8 a coluna actual (para limpeza)

  MOV     R4, teclado_movimento ; Endereco onde se guarda tabela de movimentos
  SHL     R3, 2                 ; Multiplicar por 4 (2 words, 2 bytes por word)
  ADD     R4, R3                ; Somar para obter endereco de movimento para a tecla carregada
  MOV     R4, [R4]              ; Guardar deslocamento para linha
  ADD     R1, R4                ; Aplicar o deslocamento da linha
  ADD     R3, 2                 ; Preparar para avancar para proximo word
  MOV     R4, teclado_movimento ; Repetir passos anteriores mas agora para o endereco seguinte
                                 (+2)
  ADD     R4, R3
  MOV     R4, [R4]
  ADD     R2, R4                ; Aplicar o deslocamento da coluna

  MOV     R5, 21H               ; 33 em hexadecimal, dimensao horizontal maxima do ecra (31) +
                                  erros na subtracção (subtrai +2)
  MOV     R6, boneco
  MOVB    R6, [R6]
  SUB     R5, R6                ; Obter coluna mais a direita possivel para canto superior
                                  esquerdo
  MOV     R6, robo
  MOVB    R6, [R6]
  SUB     R5, R6
  CMP     R2, R5
  JZ      falha_ver_horizontal  ; Se exceder à direita, terminar
  CMP     R2, 0
  JGE     termina_ver_horizontal
falha_ver_horizontal:
  MOV     R2, R8


termina_ver_horizontal:
```

```
  MOV     R5, 21H               ; 33 em hexadecimal, dimensao vertical maxima do ecra (31) + erros
                                  na subtracção (subtrai +2)
  MOV     R6, boneco
  ADD     R6, 1
  MOVB    R6, [R6]
  SUB     R5, R6                ; Obter linha mais a baixo possivel para canto superior esquerdo
  CMP     R1, R5
  JZ      falha_ver_vertical    ; Se exceder à direita, terminar
  CMP     R1, 0
  JGE     termina_ver_vertical
falha_ver_vertical:
  MOV     R1, R7
termina_ver_vertical:
  MOV     R9, R1                ; Trocar R1 com R7
  MOV     R1, R7
  MOV     R7, R9
  MOV     R9, R2                ; Trocar R2 com R8
  MOV     R2, R8
  MOV     R8, R9
  PUSH    R8
  MOV     R8, 0
  CALL    desenhar_figura       ; Limpar boneco actual (para redesenhar)
  POP     R8
  MOV     R9, R1                ; Trocar R1 com R7
  MOV     R1, R7
  MOV     R7, R9
  MOV     R9, R2                ; Trocar R2 com R8
  MOV     R2, R8
  MOV     R8, R9
  PUSH    R8
  MOV     R8, 1
  CALL    desenhar_figura       ; Escrever o novo boneco apos os deslocamentos
  POP     R8
  MOV     R3, POS_JG
  MOVB    [R3], R1              ; Guardar a linha actual em memoria
  ADD     R3, 1
  MOVB    [R3], R2              ; Guardar a coluna actual em memoria
movimento_fim:
  POP     R10
  POP     R9
  POP     R8
  POP     R7
  POP     R6
  POP     R5
  POP     R4
  POP     R3
  POP     R2
  POP     R1
  RET


;* -- Rotina de Reset ------------------------------------------
;*
;* Description: Reinicializa todas as variaveis ao seu estado nulo, e chama as rotinas de
inicializacao
;*
;* Parameters:  --
;* Return:      --
;* Destroy: none
;* Notes: --

reset:
  PUSH    R1
  PUSH    R2
  CALL    limpar_ecra           ; Executar a limpeza de ecra para reiniciar
  CALL    inicializar_boneco
  CALL    inicializar_robos
  CALL    inicializar_bolas
  CALL    inicializar_pontuacao
  MOV     R2, NULL
  MOV     R1, BUFFER
  MOVB    [R1], R2              ; Limpar o buffer para tecla nula
```

```
 MOV     R2, 1
 MOV     R1, GERADOR
 MOVB    [R1], R2              ; Inicializar o gerador a 1
 MOV     R1, FLAG_T
 MOV     R2, 0
 MOVB    [R1], R2              ; Limpar a flag de reiniciar/pausar a 0
 POP     R2
 POP     R1
 RET


;* -- Rotina de Inicializacao de Boneco ----------------------------------------
;*
;* Description: Inicializa o boneco na posicao 0.
;*
;* Parameters:  --
;* Return:      --
;* Destroy: none
;* Notes: --

inicializar_boneco:
 PUSH    R1
 PUSH    R2
 PUSH    R8
 PUSH    R10
 MOV     R2, 0                 ; Reinicializar posicao do boneco para 0,0
 MOV     R1, POS_JG
 MOV     [R1], R2
 MOV     R1, 0
 MOV     R8, 1
 MOV     R10, boneco
 CALL    desenhar_figura       ; Desenhar o boneco para inicializar
 POP     R10
 POP     R8
 POP     R2
 POP     R1
 RET


;* -- Rotina de Inicializacao de Robos -----------------------------------------
;*
;* Description: Inicializa a posicao dos robos para o seu ponto de comeco.
;*
;* Parameters:  --
;* Return:      --
;* Destroy: none
;* Notes: --

inicializar_robos:
 PUSH    R1
 PUSH    R2
 PUSH    R8
 PUSH    R10
 MOV     R10, robo             ; Figura a utilizar e a do robo
 MOV     R2, 0717H             ; Reinicializar posicao dos robos para 7(07H) e 23(17H)
 MOV     R1, POS_RS
 MOV     [R1], R2

 MOV     R2, 20H               ; Reinicializar coluna dos robos, dependendo do seu tamanho
 MOV     R1, robo
 MOVB    R1, [R1]
 SUB     R2, R1
 MOV     R1, COL_RS
 MOVB    [R1], R2

 MOV     R1, POS_RS
 MOVB    R1, [R1]
 MOV     R8, 1
 CALL    desenhar_figura       ; Desenhar o boneco para inicializar
 MOV     R1, POS_RS
 ADD     R1, 1
 MOVB    R1, [R1]
```

```
 CALL    desenhar_figura        ; Desenhar o boneco para inicializar
 POP     R10
 POP     R8
 POP     R2
 POP     R1
 RET


;* -- Rotina de Inicializacao de Bolas ----------------------------------------
;*
;* Description: Inicializa a direccao das bolas a 2 (definida como a direccao "nula").
;*
;* Parameters:   --
;* Return:       --
;* Destroy: none
;* Notes: --

inicializar_bolas:
 PUSH    R1
 PUSH    R2
 MOV     R1, DIR_BS
 MOV     R2, 02H
 MOV     [R1], R2
 ADD     R1, 2
 MOV     [R1], R2
 POP     R2
 POP     R1
 RET


;* -- Rotina de Inicializacao de Pontuacao ------------------------------------
;*
;* Description: Inicializa a pontuacao a 0.
;*
;* Parameters:   --
;* Return:       --
;* Destroy: none
;* Notes: --

inicializar_pontuacao:
 PUSH    R1
 PUSH    R2
 MOV     R1, PONT
 MOV     R2, 0
 MOVB    [R1], R2
 MOV     R1, POUT1
 MOVB    [R1], R2
 POP     R2
 POP     R1
 RET


;* -- Rotina de Gerador -------------------------------------------
;*
;* Description: Altera o gerador para que este seja pseudo-aleatorio.
;*              Pega no valor anteriormente presente e soma 1. Caso seja 4 (pois apenas vai de 1
;*              a 3), fica 1
;*
;* Parameters:   --
;* Return:       --
;* Destroy: none
;* Notes: --

gerador:
 PUSH    R1
 PUSH    R2
 MOV     R2, GERADOR
 MOVB    R1, [R2]
 SUB     R1, 1
 JNZ     gerador_fim            ; Caso nao seja 0, ainda pertence ao intervalo [1, 3] logo e
valido
 MOV     R1, 3                  ; Caso tenha dado 0, volta a 3
```

```
gerador_fim:
  MOVB    [R2], R1
  POP     R2
  POP     R1
  RET


;* -- Rotina de Processamento de Movimento de Robos ----------------------------------------
;*
;* Description: Para cada um dos robos, chama a rotina de processamento de movimento de robo.
;*
;* Parameters:  --
;* Return:      --
;* Destroy: none
;* Notes: --

processar_movimento_robos:
  PUSH    R4
  PUSH    R5
  PUSH    R6
processar_movimento_robo1:
  MOV     R4, FLAG_RS
  MOVB    R4, [R4]
  AND     R4, R4
  JZ      processar_movimento_robo2   ; Caso a flag do robo1 estiver a 0, avancar para robo2
  MOV     R4, POS_RS
  MOV     R5, FLAG_RS
  MOV     R6, POS_RS
  ADD     R6, 1
  MOVB    R6, [R6]
  CALL    processar_movimento_robo
processar_movimento_robo2:
  MOV     R4, FLAG_RS
  ADD     R4, 1
  MOVB    R4, [R4]
  AND     R4, R4
  JZ      processar_movimento_robos_fim ; Caso a flag do robo2 estiver a 0, terminar
  MOV     R4, POS_RS
  ADD     R4, 1
  MOV     R5, FLAG_RS
  ADD     R5, 1
  MOV     R6, POS_RS
  MOVB    R6, [R6]
  CALL    processar_movimento_robo
processar_movimento_robos_fim:
  POP     R6
  POP     R5
  POP     R4
  RET


;* -- Rotina de Processamento de Movimento de Robo ----------------------------------------
;*
;* Description: Sendo dada uma referencia a um robo, executa o seu movimento
;*
;* Parameters:  R4 (endereco da linha do robo), R5 (endereco da flag do robo), R6 (linha do outro
robo)
;* Return:      --
;* Destroy: none
;* Notes: --

processar_movimento_robo:
  PUSH    R1
  PUSH    R2
  PUSH    R3
  PUSH    R4                  ; Endereco onde e guardada linha do robo
  PUSH    R5                  ; Endereco onde e guardada a flag do robo
  PUSH    R6                  ; Linha do outro robo
  PUSH    R7                  ; Altura do robo, -1 (para obter localizacao e nao altura)
  PUSH    R8                  ; Indica se apaga ou desenha, para a rotina desenhar_figura
  PUSH    R9                  ; Valor de auxilio
  PUSH    R10                 ; Valor de auxilio, figura a desenhar
```

```
  MOV     R7, robo
  ADD     R7, 1
  MOVB    R7, [R7]
  SUB     R7, 1
  MOV     R10, robo             ; Figura a desenhar/apagar sera o robo
processar_movimento_robo_limpar:
  MOV     R1, R4
  MOVB    R1, [R1]              ; Linha actual
  MOV     R2, COL_RS
  MOVB    R2, [R2]              ; Coluna dos robos
  MOV     R8, 0
  CALL    desenhar_figura
processar_movimento_robo_colisao:
  MOV     R3, GERADOR
  MOVB    R3, [R3]
  SUB     R3, 2                 ; Gerador vai de 1 a 3, e neste caso da jeito ir de -1 a 1, para
                                  os movimentos aleatorios
  MOV     R1, R4
  MOVB    R1, [R1]              ; Ir buscar linha actual
  ADD     R1, R3                ; Aplicar movimento (-1 [subir], 0 [neutro] ou 1 [descer])

  MOV     R9, R6
  ADD     R9, R7                ; Posicao do outro robo + altura = "linha" maxima para canto
                                  superior esquerdo do robo actual
  CMP     R1, R9                ; Verificar colisao no limite inferior
  JZ      processar_movimento_robo_nada
  MOV     R9, R6
  SUB     R9, R7                ; Posicao do outro robo - altura = "linha" minima para canto
                                  superior esquerdo do robo actual
  CMP     R1, R9                ; Verificar colisao no limite superior
  JZ      processar_movimento_robo_nada
  MOV     R9, 0H                ; Verificar se nao sai do ecra pelo limite superior (por cima do
                                    ecra)
  CMP     R1, R9
  JLT     processar_movimento_robo_nada
  MOV     R9, 1FH               ; Verificar se nao sai do ecra pelo limite superior (por baixo do
                                    ecra)
  SUB     R9, R7
  CMP     R1, R9
  JLE     processar_movimento_robo_desenhar
processar_movimento_robo_nada:
  SUB     R1, R3                ; Reverter a alteracao a linha
processar_movimento_robo_desenhar:
  MOV     R2, COL_RS
  MOVB    R2, [R2]
  MOV     R8, 1
  CALL    desenhar_figura
  MOV     R2, R4
  MOVB    [R2], R1
  MOV     R1, R5
  MOV     R2, 0
  MOVB    [R1], R2
  POP     R10                   ; POP POP
  POP     R9                    ; POP POP POP POP
  POP     R8                    ; POP
  POP     R7                    ; POP POP POP
  POP     R6                    ; POP POP
  POP     R5                    ; POP POP POP POP POP
  POP     R4                    ; POP POP POP POP
  POP     R3                    ; POP
  POP     R2                    ; POP POP POP POP
  POP     R1                    ; POP POP POP
  RET
```

```
;* -- Rotina de Processamento de Movimento de Bolas ----------------------------------------
;*
;* Description: Executa a rotina de movimento de bola para cada bola
;*
;* Parameters:  --
;* Return:      --
;* Destroy: none
;* Notes: --

processar_movimento_bolas:
  PUSH    R4
  PUSH    R5
  PUSH    R6
  PUSH    R7
  MOV     R4, FLAG_BS
  MOVB    R4, [R4]
  AND     R4, R4
  JZ      processar_movimento_bolas_fim ; Caso a flag de bolas esteja a 0, nao mover
processar_movimento_bola1:
  MOV     R4, POS_BS
  MOV     R5, FLAG_BS
  MOV     R6, DIR_BS
  MOV     R7, [R6]
  SUB     R7, 2
  CALL    processar_movimento_bola      ; Processar bola 1
processar_movimento_bola2:
  MOV     R4, POS_BS
  ADD     R4, 2
  MOV     R5, FLAG_BS
  ADD     R5, 1
  MOV     R6, DIR_BS
  ADD     R6, 2
  CALL    processar_movimento_bola      ; Processar bola 2
processar_movimento_bolas_fim:
  POP     R7
  POP     R6
  POP     R5
  POP     R4
  RET


;* -- Rotina de Processamento de Movimento de Bola ----------------------------------------
;*
;* Description: Sendo dada uma referência a uma bola, realiza o seu movimento.
;*
;* Parameters:  R4 (endereco da posicao da bola), R5 (endereco da flag da bola), R6 (endereco da
direccao da bola)
;* Return:      --
;* Destroy: none
;* Notes: --

processar_movimento_bola:
  PUSH    R1                            ; Linha da bola, para rotina escrever_pixel
  PUSH    R2                            ; Coluna da bola, para rotina escrever_pixel
  PUSH    R3                            ; Acender ou apagar, para rotina escrever_pixel, e valor
                                         de auxilio
  PUSH    R7                            ; Valor de auxilio
  PUSH    R8
  MOV     R8, [R6]
  SUB     R8, 2
  JNZ     processar_movimento_bola_continuar  ; Caso a direccao esteja a 2, "nulo", nao processar
  CALL    disparar_bola
processar_movimento_bola_continuar:
  MOV     R1, R4
  MOVB    R1, [R1]
  MOV     R2, R4
  ADD     R2, 1
  MOVB    R2, [R2]
  MOV     R3, 0
  CALL    escrever_pixel                ; Apagar a bola da sua posicao actual
```

```
  MOV     R3, 1
  SUB     R2, 1
  MOV     R7, boneco
  MOVB    R7, [R7]
  SUB     R7, 1
  MOV     R8, POS_JG
  ADD     R8, 1
  MOVB    R8, [R8]
  ADD     R7, R8
  CMP     R2, R7
  JLE     processar_ponto               ; Atingiu a coluna do tenista, logo comecar processamento
                                         do ponto
  MOV     R8, [R6]
  ADD     R1, R8
  MOV     R7, 0
  CMP     R1, R7
  JLT     processar_movimento_bola_reflexo
  MOV     R7, 1FH
  CMP     R1, R7
  JLE     desenhar_movimento_bola
processar_movimento_bola_reflexo:
  NEG     R8
  MOV     [R6], R8
  ADD     R1, R8                        ; Primeira vez para reverter o 'erro'
  ADD     R1, R8                        ; Segunda vez para reflectir
  JMP     desenhar_movimento_bola
processar_ponto:
  MOV     R7, 2
  MOV     [R6], R7                      ; "Anular" direccao, para nao processar novamente depois
                                          de atingir esta coluna
  CALL    ponto                         ; Caso tenha passado pela coluna do tenista, chamar a
                                          rotina dos pontos
  JMP     fim_movimento_bola
desenhar_movimento_bola:
  CALL    escrever_pixel                ; Escrever a bola na nova posicao
fim_movimento_bola:
  MOV     R7, R4
  MOVB    [R7], R1
  ADD     R7, 1
  MOVB    [R7], R2
  MOV     R7, 0
  MOVB    [R5], R7
  POP     R8
  POP     R7
  POP     R3
  POP     R2
  POP     R1
  RET


;* -- Rotina de Disparo de Bola ----------------------------------------
;*
;* Description: Dispara a bola caso nao tenha sido disparada
;*
;* Parameters:  R4 (endereco da posicao da bola), R6 (endereco da direccao da bola)
;* Return:      --
;* Destroy: none
;* Notes: --

disparar_bola:
  PUSH    R1
  PUSH    R2
  PUSH    R4
  PUSH    R5
  PUSH    R6
  ; Processar direccao (-1 [cima], 0 [horizontal], 1 [baixo])
  MOV     R1, GERADOR
  MOVB    R1, [R1]
  SUB     R1, 2
  MOV     [R6], R1
```

```
  ; Processar linha (altura) de disparo
  MOV     R1, bol_DS
  MOVB    R1, [R1]
  SUB     R1, 1
  MOV     R2, POS_RS
  MOV     R5, DIR_BS
  SUB     R6, R5
  SHR     R6, 1
  ADD     R2, R6
  MOVB    R2, [R2]                    ; Obter a linha do robo que dispara a bola "pedida"
  ADD     R2, R1
  MOVB    [R4], R2                    ; Definir linha da bola a disparar

  ; Processar coluna de disparo
  MOV     R2, COL_RS
  MOVB    R2, [R2]
  SUB     R2, 1                       ; Disparar a frente do canhao, e nao no seu "ultimo" pixel
  ADD     R4, 1
  MOVB    [R4], R2
disparar_fim:
  POP     R6
  POP     R5
  POP     R4
  POP     R2
  POP     R1
  RET


;* -- Rotina de Calculo de Pontos ----------------------------------------
;*
;* Description: Adiciona ou subtrai um ponto (nao subtrai caso pontuacao seja nula)
;*
;* Parameters: R1 (Linha da bola)
;* Return:       --
;* Destroy: none
;* Notes: --

ponto:
  PUSH    R1
  PUSH    R2
  PUSH    R3

  MOV     R3, POS_JG
  MOVB    R3, [R3]

  CMP     R1, R3
  JLT     ponto_perder

  ADD     R3, RAQUETE
  SUB     R2, 1                       ; O se o comprimento da raquete for 3, e suposto avancar
                                        apenas 2 linhas, pois a primeira ja foi verificada
  CMP     R1, R3
  JGT     ponto_perder

ponto_marcar:
  MOV     R3, BONUS
  CMP     R2, R3
  JGE     ponto_marcar_bonus
  MOV     R3, 1
  CALL    alterar_pontuacao
  JMP     ponto_fim
ponto_marcar_bonus:
  MOV     R3, 3
  CALL    alterar_pontuacao
  JMP     ponto_fim
ponto_perder:
  MOV     R3, 0FFFFH
  CALL    alterar_pontuacao
ponto_fim:
  POP     R3
```

```
 POP     R2
 POP     R1
 RET

;* -- Rotina de Alteracao de Pontuacao ----------------------------------------
;*
;* Description: Adiciona ou subtrai pontuacao ao score actual, e faz update aos displays
;*
;* Parameters: R3 (Pontos a somar)
;* Return:      --
;* Destroy: none
;* Notes: --

alterar_pontuacao:
 PUSH    R1
 PUSH    R2
 PUSH    R3
 PUSH    R4
 MOV     R1, PONT
 MOVB    R1, [R1]                    ; Guardar no R1 a pontuacao actual

 MOV     R2, 0FH
 AND     R2, R1                      ; Guardar em R2 as unidades actuais

 MOV     R4, 0F0H
 AND     R1, R4                      ; Guardar em R1 as dezenas actuais

 ADD     R2, R3
 MOV     R3, 0AH
 CMP     R2, R3
 JGE     alterar_pontuacao_soma      ; Caso as unidades sejam superiores ou iguais a A,
                                       incrementar dezenas
 MOV     R3, 0H
 CMP     R2, R3
 JLT     alterar_pontuacao_subtrai   ; Caso sejam inferiores a 0, decrementar dezenas
 JMP     alterar_pontuacao_fim       ; Caso nao seja necessario alterar dezenas, terminar
alterar_pontuacao_soma:
 MOV     R3, 0AH
 SUB     R2, R3
 MOV     R3, 10H
 ADD     R1, R3
 MOV     R3, 0A0H
 CMP     R1, R3
 JLT     alterar_pontuacao_fim
 MOV     R1, 90H
 JMP     alterar_pontuacao_fim
alterar_pontuacao_subtrai:
 MOV     R2, 0
 AND     R1, R1
 JZ      alterar_pontuacao_fim
 MOV     R3, 10H
 SUB     R1, R3
alterar_pontuacao_fim:
 ADD     R1, R2
 MOV     R2, POUT1
 MOVB    [R2], R1                    ; Escrever para os ecras a pontuacao actual
 MOV     R2, PONT
 MOVB    [R2], R1                    ; Guardar em memoria a pontuacao actual
 POP     R4
 POP     R3
 POP     R2
 POP     R1
 RET
```

```
;* -- Rotina de Controlo ------------------------------------------
;*
;* Description: Detecta caso o jogo tenha sido terminado
;*              Isto acontece quando a tecla F e premida
;*              Entra depois um loop infinito ate ser premida novamente
;*
;* Parameters:  --
;* Return:      --
;* Destroy: none
;* Notes: --

controlo:
 PUSH   R1
 PUSH   R2
 PUSH   R3
 PUSH   R4
 MOV    R4, 0
 JMP    controlo_teste
controlo_ciclo:
 CALL   teclado
controlo_teste:
 MOV    R1, BUFFER
 MOVB   R1, [R1]
 MOV    R2, BUFFER
 ADD    R2, 1
 MOVB   R2, [R2]
 MOV    R3, 0FH
 CMP    R1, R3
 JNZ    controlo_fim                ; Caso tecla actual seja nula, terminar
 CMP    R1, R2
 JZ     controlo_fim                ; Caso tecla acutal seja igual a anterior, terminar
controlo_termina:
 AND    R4, R4
 JNZ    controlo_reset              ; Caso tenha sido pressionado "F", terminar jogo
 CALL   pintar_ecra
 MOV    R4, 1
 JMP    controlo_ciclo
controlo_reset:
 CALL   reset                       ; Quando e premida pela segunda vez, dar reset ao jogo
 MOV    R4, 0
controlo_fim:
 AND    R4, R4                      ; Enquanto que R4 estiver a 1, prender no loop
 JNZ    controlo_ciclo
 POP    R4
 POP    R3
 POP    R2
 POP    R1
 RET

;* -- Rotina de Serviço de Interrupção 1 --------------------------------------
;*
;* Description: Trata da interrupcao gerada pelo primeiro relogio, alterando a flag FLAG_RS
;*
;* Parameters:  --
;* Return:      --
;* Destroy: none
;* Notes: --

interrup1:
 CALL   gerador             ; Para aumentar a aleatoriedade do gerador, vamos executa-lo
                              tambem nas interrupcoes
                            ; Corre-se aqui e nao usando a flag da interrupcao para maximizar
                              a aleatoriedade
                            ; Assim podem ocorrer varios "geradores" por loop
 PUSH   R1
 PUSH   R2
 MOV    R1, GERADOR
 MOVB   R1, [R1]            ; Buscar o valor gerado aleatoriamente
 SUB    R1, 1
 JZ     interrup1_R1        ; Caso esse valor seja 1, activa a flag do robo 1
```

```
 SUB     R1, 1
 JZ      interrup1_R2          ; Caso seja 2, activa flag do robo 2
 JMP     interrup1_fim         ; Caso seja 3, nao activa flag
interrup1_R1:
 MOV     R1, FLAG_RS
 MOV     R2, 1
 MOVB    [R1], R2
 JMP     interrup1_fim
interrup1_R2:
 MOV     R1, FLAG_RS
 ADD     R1, 1
 MOV     R2, 1
 MOVB    [R1], R2
 JMP     interrup1_fim
interrup1_fim:
 POP     R2
 POP     R1
 RFE


;* -- Rotina de Serviço de Interrupção 2 ------------------------------------------
;*
;* Description: Trata da interrupcao gerada pelo segundo relogio, alterando a flag FLAG_BS
;*
;* Parameters:  --
;* Return:      --
;* Destroy: none
;* Notes: --

interrup2:
 CALL    gerador               ; Para aumentar a aleatoriedade do gerador, vamos executa-lo
                                 tambem nas interrupcoes
                               ; Corre-se aqui e nao usando a flag da interrupcao para maximizar
                                 a aleatoriedade
                               ; Assim podem ocorrer varios "geradores" por loop
 PUSH    R1
 PUSH    R2
 MOV     R1, FLAG_BS
 MOV     R2, 1
 MOVB    [R1], R2              ; Activa a flag das bolas
 POP     R2
 POP     R1
 RFE
```