INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

ARQUITETURA DE COMPUTADORES

LEIC

IST-TAGUSPARK

JOGO DE BATALHA ESPACIAL

(**A=1.0**) Excelente
(**B=0.8**) Muito Bom
(**C=0.6**) Bom
(**D=0.4**) Suficiente
(**E=0.2**) Fraco

PARTE	OBJECTIVO	NÍVEL	PESO	VALOR
Análise	Estrutura Geral do Projecto	0.9	2.0	1.8
Conceção	Detalhes de Implementação	0.9	2.0	1.8
Programa	Dados (Variáveis, Tabelas)	1.0	2.0	2.0
Programa	Comunicação entre Processos	1.0	4.0	4.0
Programa	Utilização de Interrupções	1.0	2.0	2.0
Programa	Estrutura das Rotinas	1.0	5.0	5.0
Qualidade	Originalidade e Variantes	0.5	2.0	1.0
Qualidade	Organização do Relatório	0.9	1.0	0.9
			TOTAL	18.5

Daniel Trindade, Nº7634

João Santos, Nº76363

André Faustino, Nº76510

1. Introdução

Este trabalho foi realizado para a disciplina de Arquitectura de Computadores do Instituto Superior Técnico – Taguspark, com o objectivo de testar os conteúdos dados em aulas teóricas como por exemplo: programação em linguagem Assembly, saber trabalhar com periféricos e interrupções, entre outros.

O trabalho consiste num jogo de batalha espacial onde existem vários aliens a tentarem aproximar-se da nave do jogador. O jogador pode controlar a sua nave através do teclado, assim como pode disparar um raio que destrói as naves alien, pode ainda suspender ou terminar o jogo quando quiser. O trabalho deve ser programado em linguagem Assembly usando o simulador adoptado pelos docentes da disciplina, o circuito que simula o hardware foi fornecido junto ao enunciado, sendo pedido aos alunos que programassem o jogo de forma a poder ser compilado e executado pelo processador adoptado (PEPE).

O enunciado refere também que é possível fazer alterações ao circuito do jogo ou à ideia do jogo em si, desde que as alterações consigam simplificar o circuito ou o jogo sem prejudicar a ideia do jogo.

Na secção 2 descrever-se-à as estratégias de implementação adoptadas assim como todos os detalhes referentes às rotinas criadas, e à comunicação entre processos.

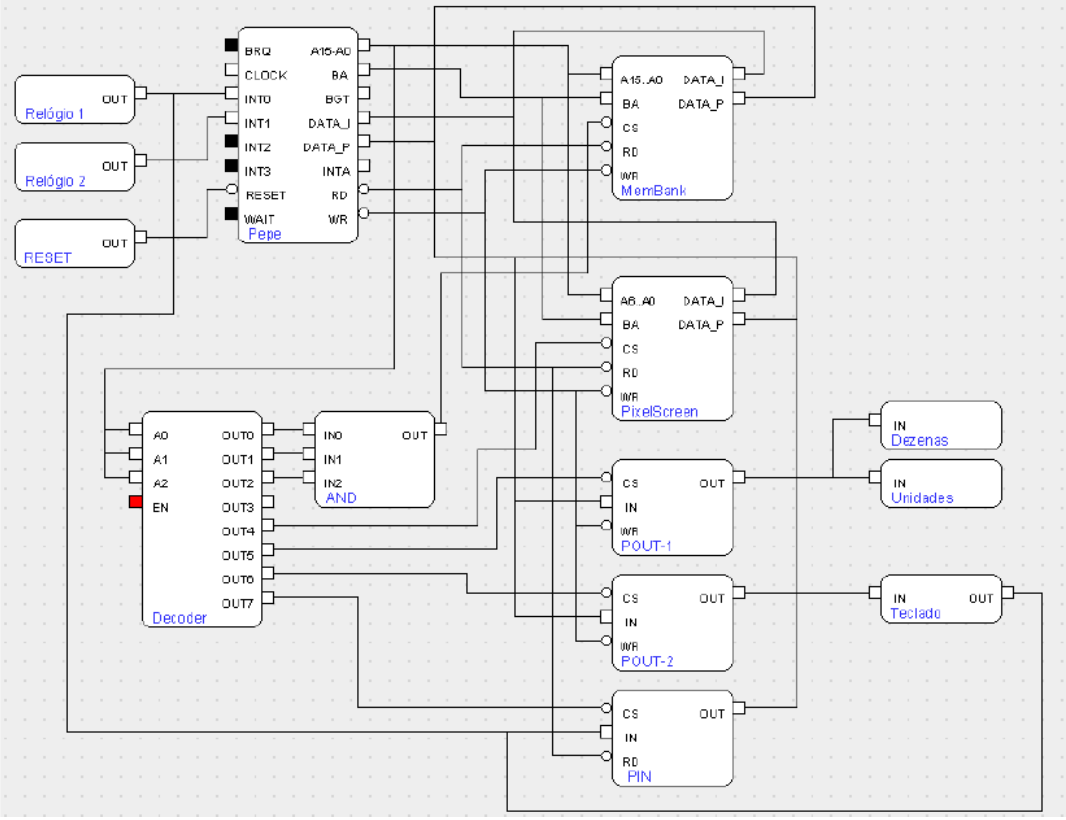
Na secção 3 poderá ser encontrada a conclusão que contém uma reflexão crítica sobre o trabalho realizado, dizendo claramente o que foi feito, quais as dificuldades encontradas e o foi feito para ultrapassar essas dificuldades. A conclusão conterá também sugestões de melhoramentos.

Por fim, a secção 4 contém todo o código em linguagem Assembly escrito para tentar resolver o problema colocado e implementar o jogo de batalha espacial.

2. Conceção e Implementação

2.1. Estrutura Geral

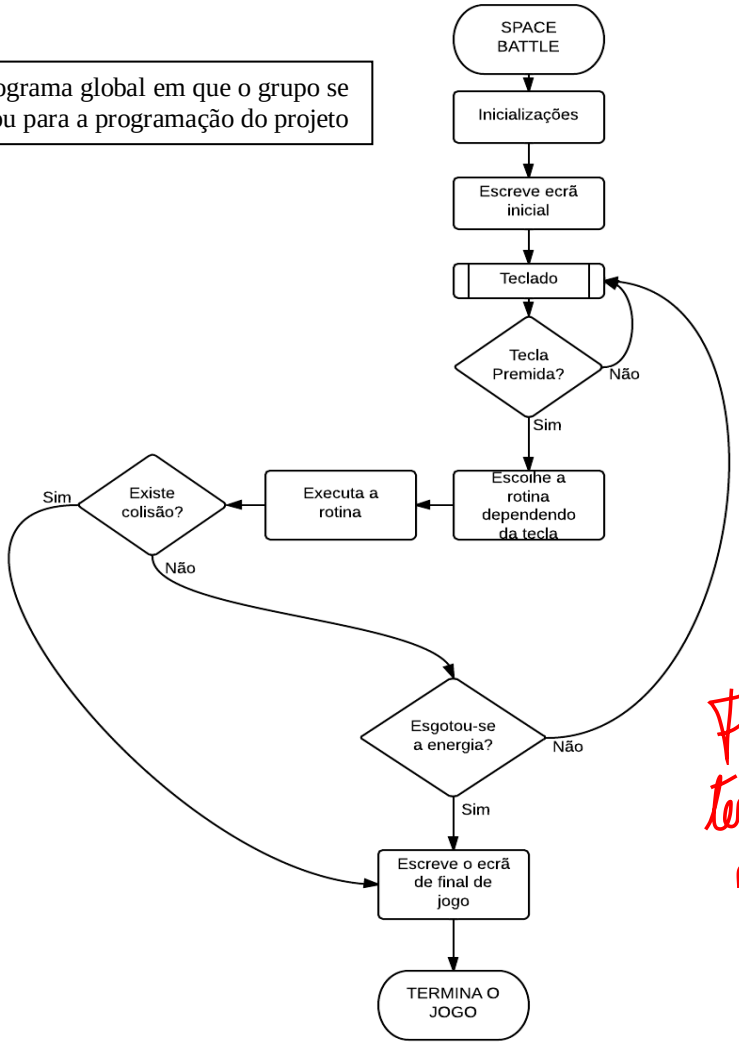
Em termos de hardware não foram feitas alterações ao circuito original fornecido com o enunciado do projecto. O circuito continua assim a ter este formato:



A ideia por trás do circuito passa em ter dois relógios ligados ao processador mais concretamente à parte das interrupções 0 e 1. Cada relógio tem um período diferente de forma a que cada interrupção se possa ajustar da melhor forma ao jogo. As saídas do processador estão ligadas a um ecrã (Pixelscreen) de tamanho 32x32, este ecrã funciona como uma memória de 128 bytes (4 bytes por linha em 32 linhas, dando assim 32 bits por cada linha), a uma memória de 24kbytes.

No circuito encontra-se também um teclado que vai ser muito útil para o jogador poder controlar a sua nave e dois displays hexadecimais ligados ao mesmo porto de saída, neste conjunto será mostrado o valor de energia da nave do jogador (pois o jogo acaba quando se esgotar a energia, podendo gastar-se energia com ao disparar o raio – 2 pontos; ou movimentar a nave – 1 ponto, também se pode recuperar energia – 10 pontos, ao matar naves alien).

Fluxograma global em que o grupo se baseou para a programação do projeto



Figuras devem ter legendas numeradas

2.1.1. Mapa de endereçamento escolhido

Em termos de endereçamento por hardware foi mantido todo o endereçamento original do circuito, que consistia em:

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

Em termos de endereçamento dentro do software (na RAM), pode-se destacar:

Tipo de dados	Endereços
Código	0000H
Stack	1000H a 114FH
Tabelas varidas	1200H a 140BH
Buffer (Teclado)	2000H

O código foi inicializado em 0000H pois, para além de ser sugerido no enunciado, é o endereço onde começa o PC, se o código começasse num endereço mais distante o PC teria de percorrer vários endereços vazios perdendo assim tempo.

RAM 2001H a 5FFFH serve para quê?

O stack foi inicializado em 1000H porque é um endereço que se encontra suficientemente longe de toda a zona ocupada pelo código, não havendo assim possibilidade de se causar algum conflito entre endereços.

A zona de tabelas foi inicializada em 1200H de forma a ser seguida ao stack de forma a não ocupar muita memória mas também para começar num endereço conhecido para fácil verificação das tabelas em questão, na implementação do código.

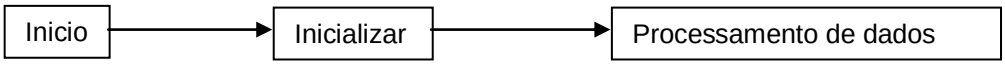
O buffer do teclado, que consiste num endereço de memória onde é guardada a tecla lida pelo teclado foi inicializado em 2000H por ser um endereço suficientemente distante dos restantes para fácil visualização durante a implementação do código.

2.1.2. Comunicação entre processos

A comunicação entre processos é feita de duas formas diferentes escolhidas de forma a proporcionar uma fácil implementação e um uso recorrente de certas funções mais importantes.

A forma de comunicação mais comum usada consiste em ter um processo que chama através de um call ou estados (secção 2.1.3) outros processos que por sua vez tratam de todo o processamento necessário, chamando se necessário outras rotinas.

Um exemplo deste tipo de comunicação é o processo **início** que não faz nenhum processamento para além do processamento relativo ao estado e chama outros processos que por sua vez inicializam as variáveis e efecutam as acções necessárias para o início do jogo.

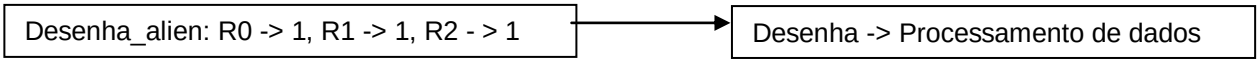


Exemplo de comunicação entre processo inicio e inicializar (que por sua vez faz o processamento de dados)

usar Registos como variáveis de Estado ! só restrito

A outra forma de comunicação presente no projecto consiste em usar rotinas de forma equivalente a usar funções em linguagens de programação como C ou Python, por exemplo. Antes de um processo chamar a rotina que faz o processamento afecta um conjunto de registos que funcionam como parâmetros de entrada. Isto permite que a mesma rotina funcione para objectos diferentes do jogo, o que evita ter rotinas muito semelhantes dentro do projecto.

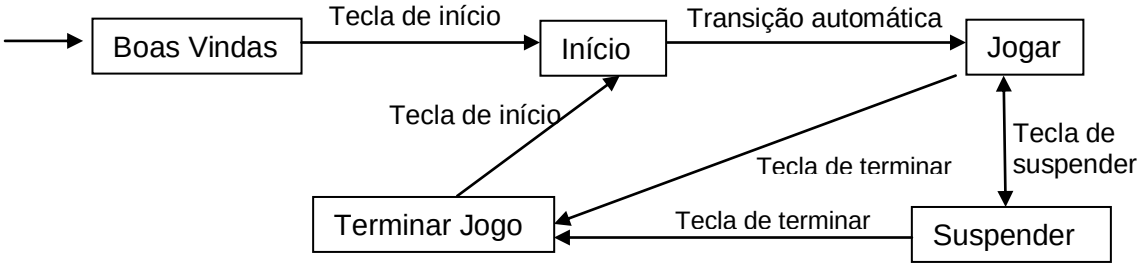
Um exemplo é a rotina **desenha** que recebe como parâmetros de entrada um valor a escrever (1 para escrever, 0 para apagar) em R0, e recebe uma linha e coluna onde escrever (em R1 e R2, respectivamente, cada coluna corresponde a um bit) escrevendo ou apagando assim no pixel seleccionado. Desta forma a mesma rotina é chamada para desenhar naves alien ou a nave do jogador ou o raio.



Exemplo de comunicação ao estilo de funções, a rotina desenha_alien passa parâmetros de entrada e a rotina desenha faz o processamento

2.1.3. Variáveis de Estado

Como forma de transitar entre estados como sugerido pelo enunciado, foi criada a seguinte máquina de estados:

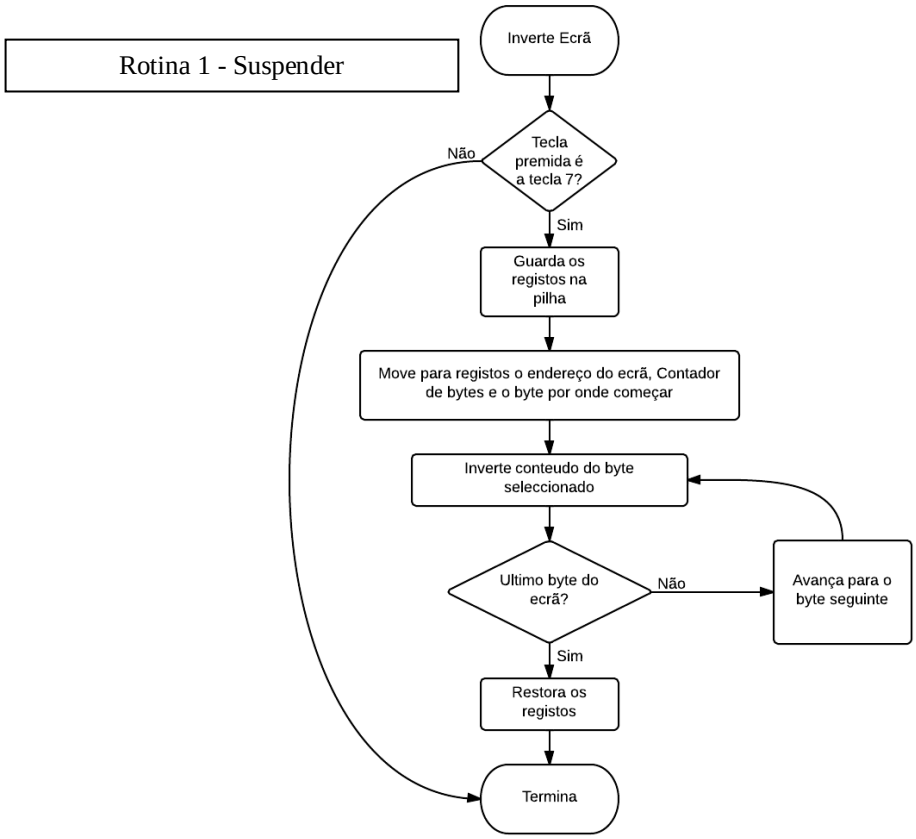


Como forma de implementar esta tabela de estados foi criada uma tabela de words chamada **TAB_ESTADO** onde cada word corresponde a um estado presente na máquina acima. Foi também criada uma pequena tabela só de uma string que corresponde ao estado actual (estado_ctrl). Em cada estado é colocado em R0 o endereço da tabela de estado actual e em R1 o estado para onde se quer transitar, transferindo depois R1 para a tabela de estado actual. Ao aceder à tabela TAB_ESTADO é fácil de se mudar de estado.

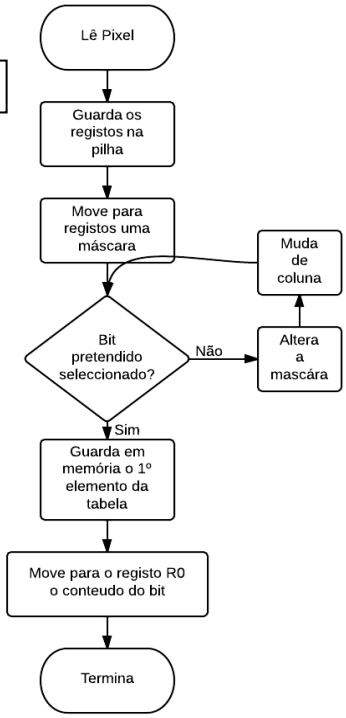
2.1.4. Interrupções

No programa usamos 2 interrupções principais (EI0 e EI1) na parte do inicio e no suspender para fazer o desenho das naves e dos aliens nas respectivas posições quando iniciamos ou reiniciamos o jogo. Em pormenor, o EI1 apenas decrementa a energia, enquanto que o EI0 muda uma flag.

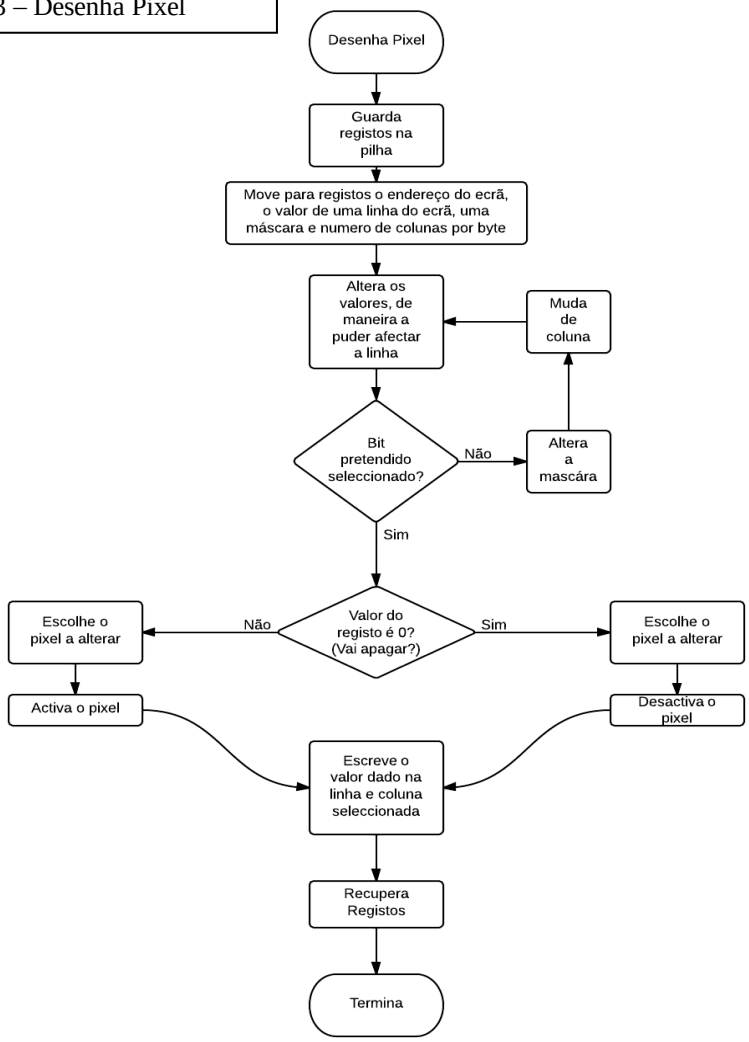
2.1.5. Rotinas


Rotina 1 - Suspender
Inverte Ecrã
Tecla premida é a tecla 7?
Não
Sim
Guarda os registos na pilha
Move para registos o endereço do ecrã, Contador de bytes e o byte por onde começar
Inverte conteudo do byte seleccionado
Ultimo byte do ecrã?
Não
Avança para o byte seguinte
Sim
Restora os registos
Termina

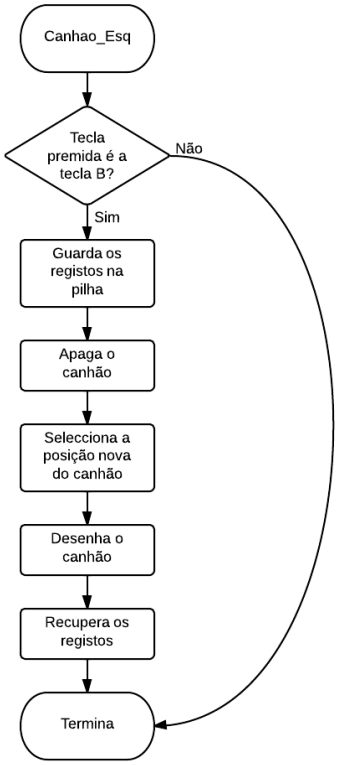
Rotina 2 - Lê Pixel



Rotina 3 – Desenha Pixel

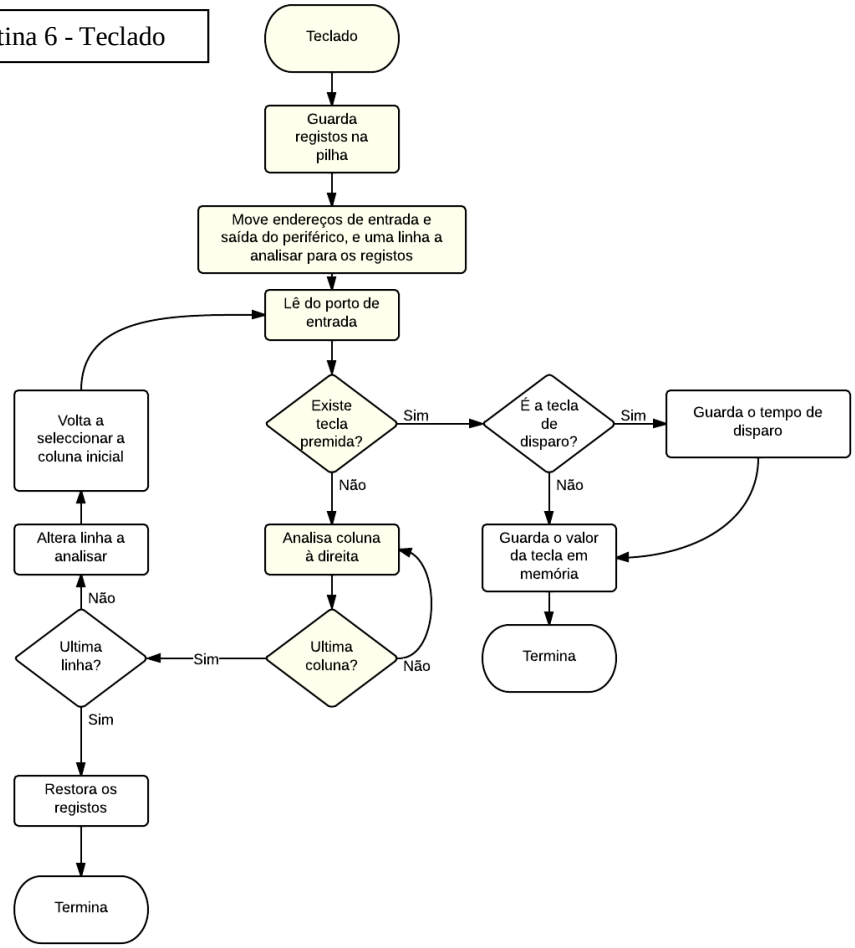


Rotina 4 – Canhão esquerdo

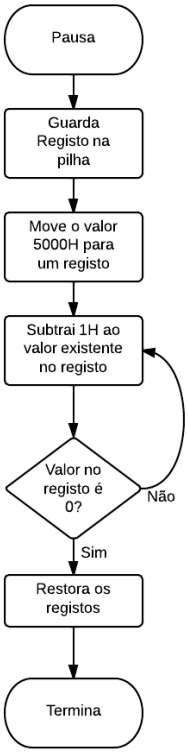


Rotina 5 – Canhão direito

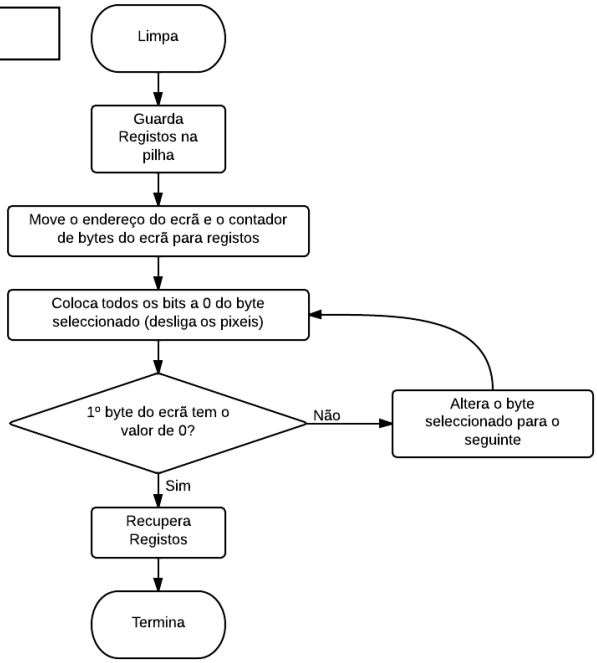
Rotina 6 - Teclado



Rotina 7 - Pausa



Rotina 8 - Limpa



3. Conclusões

Este trabalho foi-nos bastante útil e perspicaz no que diz respeito á vida profissional como engenheiros informáticos pois deu-nos a entender como olhar para as diferentes regras do programa Assembly e com elas criar bastantes processos.

Foi longo (devido ás extensas linhas de código que o grupo teve que programar e orientar-se entre si) e complicado, mas no final, acabou por resultar um programa bem estruturado e bem desenvolvido. Os objectivos de conhecimento do Assembly de cada elemento do grupo ficaram muito bem apurados tendo conseguido fazer um jogo complexo, completo, que cumpre quase todos os requisitos do enunciado (problemas no gasto do disparo - só gasta 2 pontos de energia se pressionarmos o botão 5 por 0.5 segundos) e original (referir que usamos efeitos como o gasto de 1 ponto energia quando embate numa parede).

O programa em si está feito de uma forma fácil de entender. Uma vez iniciado, basta carregar na tecla 3 para iniciar o jogo. As teclas “0”, “1”, “2”, ”4”, “6”, “8”, “9” e “A” servem para mover a nossa nave, sendo as teclas “C” e “E” as mudanças de direcção do canhão da nave. A tecla 5 tem como funcionalidade disparar o canhão e matar os aliens. Para pausar um jogo, basta clicar na tecla “7” e para acabar um jogo, basta clicar na tecla “B”. As teclas que não servem para este jogo são as teclas “D” e “F” que não fazem rigorosamente nada no desenrolar do jogo. Fora do teclado do jogo, optamos por fazer com que a nave ficasse com menos um de energia sempre que fosse contra uma parede (para dificultar um pouco o jogo).

No fim do trabalho, o grupo conclui que em termos de funcionamento, o jogo está bem formulado. Inicialmente tínhamos em mente um projecto que iria necessitar de muitas rotinas e algumas interrupções e foi o que se verificou ao longo do trabalho. A componente mais complicada de se fazer, na opinião do grupo foi o movimento dos aliens, mas que a partir de um fluxograma bem desenvolvido, conseguimos ultrapassar esse “obstáculo”. De resto, o grupo esteve bem distribuído, tendo todos os elementos colaborado e auxiliado uns aos outros.

Como melhoramento, para aqueles menos cultos da cadeira de AC, poderíamos fazer um gasto de energia de 0 a 99 por numeração decimal (o nosso programa tem uma numeração hexadecimal, ou seja, do 90 passa para o 8F), aumentar a vida da nave ou aumentar o desgaste do raio se este estiver ativado por muito tempo ou até mesmo reduzir o período dos aliens.

esta descrição devia estar antes e não nas Conclusões!

4. Código assembly

Podia estar melhor formatado!

```
; *********************************************************************
; JOGO BATALHA ESPACIAL
;
; Grupo:
; 76349 - Daniel Trindade
; 76363 - João Santos
; 76510 - André Faustino
; *********************************************************************

; *********************************************************************
; * Definições de constantes utilizadas no programa
; *********************************************************************

    ECRA        EQU 8000H    ; Endereço do ecrã
    LINHA       EQU 8H       ; Corresponde à última linha do teclado, para se iniciar os
testes
    COLUNA      EQU 8H       ; Corresponde à última coluna do teclado, para se iniciar os
testes
    PIN         EQU 0E000H   ; Endereço do porto de entrada do teclado
    POUT1       EQU 0A000H   ; Endereço do porto de saída 1
    POUT2       EQU 0C000H   ; Endereço do porto de saída 2
    TEC         EQU 0FH      ; Máscara que selecciona os bits do porto de entrada que
correspondem ao teclado
    MAX_ECRA    EQU 80H      ; Número de bytes do ecrã
    MAX_LINHA   EQU 3H       ; Número máximo de bits a 1 numa linha de uma nave (alien ou
do jogador)
    N_ALIEN     EQU 4H       ; Número de naves alien presentes no jogo
    ENERGIA     EQU 99H      ; Inicialização da energia máxima
    LIN_ECRA    EQU 4H       ; Número de bytes que formam uma linha do ecrã
    MAX_ELE     EQU 84       ; Contador de elementos na tabela de strings de ecrã final
(em bytes)
    ECRA_INT    EQU 14H      ; Número de linhas brancas antes das letras nos ecrãs de
início e fim
    MASCARA     EQU 80H      ; Máscara que selecciona cada bit a um individualmente
    LIN_NAVE    EQU 15       ; Linha inicial do centro da nave do jogador
    COL_NAVE    EQU 15       ; Coluna inicial do centro da nave do jogador
    LIN_CANHAO EQU 13        ; Linha do canhão na posição inicial
    T_INIC      EQU 03H      ; Tecla de começar jogo
    T_DISPARO   EQU 05H      ; Tecla de disparar o raio
    T_SUSP      EQU 07H      ; Tecla de suspender
    T_TERM      EQU 0BH      ; Tecla de terminar jogo
    T_ROD_ESQ   EQU 0CH      ; Tecla de rodar o canhão à esquerda
    T_IGNORE1   EQU 0DH      ; Tecla sem significado
    T_ROD_DIR   EQU 0EH      ; Tecla de rodar o canhão à direita
    T_IGNORE2   EQU 0FH      ; Tecla sem significado
    T_NEUTRA    EQU 0FFH     ; Tecla neutra (o valor assumido quando não existe uma tecla
premida)
    ENE_MOVE    EQU 1H       ; Valor de energia gasto quando a nave se move
    ENE_DISP    EQU 2H       ; Valor inicial de energia gasto a disparar o canhão
    ENE_CANHAO EQU 2H        ; Valor de energia gasto a disparar continuamente o canhão
        ENE_DEST    EQU 10H      ; Valor de energia ganho por destruir alien


; *********************************************************************************
; * Stack
; *********************************************************************************

PLACE       1000H
pilha:      TABLE 150H       ; Espaço reservado para a pilha

SP_inicial:                  ; Endereço com que o SP deve ser inicializado.

; *********************************************************************************
; * Tabela de Excepções
; *********************************************************************************

; Tabela de vectores de interrupção
tab:        WORD    int0
            WORD    int1

; *********************************************************************************
; * Tabelas e Variaveis
; *********************************************************************************
```

baixo!

```
PLACE       1200H

ALIEN:
    STRING 0A0H, 040H, 0A0H ; Tabela de strings que contém o formato das naves alien
NAVE:
    STRING 0E0H, 0A0H, 0E0H ; Tabela de strings que contém o formato da nave do jogador

BUFFER:                     ; Variavel onde se guarda a tecla premida
    WORD 0FFH               ; Valor da tecla inicial (sem tecla)
AUX_BUFFER:
    WORD 055H               ; Código de tecla inexistente

LINHA_NAVE:                 ; Variavel que guarda a linha da nave
    WORD  15                ; Linha inicial do centro da nave

COLUNA_NAVE:                ; Variavel que guarda a coluna actual da nave
    WORD  15                ; Coluna inicial do centro da nave

POS_CANHAO:                 ; Variavel que guarda a posição do canhão
    WORD  0                 ; Posição inicial do canhão

FLAG_DESENHA:               ; Variavel que guarda uma flag (0 ou 1) que diz se o canhão
deve ser ou não apagado
    WORD  0                 ; Flag iniciada a 0

FLAG_DISPARO:               ; Variavel que guarda uma flag (0 ou 1) que diz se o canhão
está a disparar ou não
    WORD  0                 ; Flag iniciada a 0

FLAG_ALIEN:                 ; Variavel que guarda uma flag (0 ou 1) que diz se os aliens
se movimentam ou não
    WORD  0                 ; Flag iniciada a 0

V_ENERGIA:                  ; Variavel que guarda o valor de energia actual
    WORD 99H                ; Valor de energia inicializado a 99H

PLACE 1300H

; DIR - Tabela de words que contém as 8 direcções possíveis
DIR_LINHAS:
    ; (-1, -1)
    WORD -1H                ; 0H
    ; (-1, 0)
    WORD -1H                ; 2H
    ; (-1, 1)
    WORD -1H                ; 4H
    ; (0, -1)
    WORD 0H                 ; 6H
    ; (0, 0)
    WORD 0H                 ; 8H
    ; (0, 1)
    WORD 0H                 ; AH
    ; (1, -1)
    WORD 1H                 ; CH
    ; (1, 0)
    WORD 1H                 ; EH
    ; (1, 1)
    WORD 1H                 ; 10H

DIR_COLUNAS:
    ; (-1, -1)
    WORD -1H                ; 0H
    ; (-1, 0)
    WORD 0H                 ; 2H
    ; (-1, 1)
    WORD 1H                 ; 4H
    ; (0, -1)
    WORD -1H                ; 6H
    ; (0, 0)
    WORD 0H                 ; 8H
    ; (0, 1)
    WORD 1H                 ; AH
    ; (1, -1)
    WORD -1H                ; CH
```

```
    ; (1, 0)
    WORD 0H                 ; EH
    ; (1, 1)
    WORD 1H                 ; 10H

; Teclas_Nave: Tabela que guarda as direcções que a nave tem a seguir a clicar numa
tecla
TECLAS_NAVE:
    STRING 00H              ; Tecla 0
    STRING 02H              ; Tecla 1
    STRING 04H              ; Tecla 2
    STRING 08H              ; Tecla 3
    STRING 06H              ; Tecla 4
    STRING 08H              ; Tecla 5
    STRING 0AH              ; Tecla 6
    STRING 08H              ; Tecla 7
    STRING 0CH              ; Tecla 8
    STRING 0EH              ; Tecla 9
    STRING 10H              ; Tecla A
    STRING 08H              ; Tecla B
    STRING 08H              ; Tecla C
    STRING 08H              ; Tecla D
    STRING 08H              ; Tecla E
    STRING 08H              ; Tecla F

; Linha_Alien: Tabela que contém a linha do centro de cada nave alien
LINHA_ALIEN:
    STRING 1                ; Linha do centro do 1º alien
    STRING 1                ; Linha do centro do 2º alien
    STRING 30               ; Linha do centro do 3º alien
    STRING 30               ; Linha do centro do 4º alien

; Coluna_Alien: Tabela que contém a coluna do centro de cada nave alien
COLUNA_ALIEN:
    STRING 1                ; Coluna do centro do 1º alien
    STRING 30               ; Coluna do centro do 2º alien
    STRING 1                ; Coluna do centro do 3º alien
    STRING 30               ; Coluna do centro do 4º alien

; Linha_Inicio: Tabela que contém a linha do centro de cada nave alien (posição de
início)
LINHA_INICIO:
    STRING 1                ; Linha do centro do 1º alien
    STRING 1                ; Linha do centro do 2º alien
    STRING 30               ; Linha do centro do 3º alien
    STRING 30               ; Linha do centro do 4º alien

; Coluna_Inicio: Tabela que contém a coluna do centro de cada nave alien (Posição de
início)
COLUNA_INICIO:
    STRING 1                ; Coluna do centro do 1º alien
    STRING 30               ; Coluna do centro do 2º alien
    STRING 1                ; Coluna do centro do 3º alien
    STRING 30               ; Coluna do centro do 4º alien

; Linha_Raio: Tabela que contém a linha do ponto onde acaba o raio
LINHA_RAIO:
    STRING 0

; Coluna_Raio: Tabela que contém a coluna do ponto onde acaba o raio
COLUNA_RAIO:
    STRING 0

; Tabela que contém a diferença entre a linha do centro da nave e o canhão em cada
posição
CANHAO_LIN:
    WORD -2
    WORD -2
    WORD  0
    WORD  2
    WORD  2
    WORD  2
    WORD  0
    WORD -2
```

```
; Tabela que contém a diferença entre a coluna do centro da nave e o canhão em cada
posição
CANHAO_COL:
    WORD  0
    WORD  2
    WORD  2
    WORD  2
    WORD  0
    WORD -2
    WORD -2
    WORD -2

; Tabela que contém a direcção a seguir pelo raio, seguindo a ordem das direcções do
canhão
RAIO:
    WORD 02H
    WORD 04H
    WORD 0AH
    WORD 10H
    WORD 0EH
    WORD 0CH
    WORD 06H
    WORD 00H

; ********************************************************************************
; * Ecrã de início
; ********************************************************************************

PLACE       1400H

INICIO:
    STRING 07DH, 0F3H, 0CFH, 0BEH
    STRING 045H, 012H, 048H, 0A0H
    STRING 041H, 012H, 048H, 020H
    STRING 041H, 012H, 048H, 020H
    STRING 07DH, 0F7H, 0ECH, 03EH
    STRING 00DH, 086H, 02CH, 030H
    STRING 00DH, 086H, 02CH, 030H
    STRING 04DH, 086H, 02CH, 0B0H
    STRING 07DH, 086H, 02FH, 0BEH

    STRING 000H, 000H, 000H, 000H
    STRING 000H, 000H, 000H, 000H
    STRING 000H, 000H, 000H, 000H

    STRING 0F1H, 0EFH, 07AH, 01FH
    STRING 091H, 024H, 022H, 010H
    STRING 091H, 024H, 022H, 010H
    STRING 091H, 024H, 022H, 010H
    STRING 0FBH, 0F6H, 033H, 01FH
    STRING 0CBH, 016H, 033H, 018H
    STRING 0CBH, 016H, 033H, 018H
    STRING 0CBH, 016H, 033H, 018H
    STRING 0FBH, 016H, 033H, 0DFH

; ********************************************************************************
; * Ecrã de fim
; ********************************************************************************

FIM:
    STRING 01FH, 09EH, 07FH, 07CH
    STRING 010H, 092H, 049H, 040H
    STRING 010H, 012H, 049H, 040H
    STRING 010H, 012H, 049H, 040H
    STRING 019H, 0BFH, 069H, 07CH
    STRING 018H, 0B1H, 069H, 060H
    STRING 018H, 0B1H, 069H, 060H
    STRING 018H, 0B1H, 069H, 060H
    STRING 01FH, 0B1H, 069H, 07CH

    STRING 000H, 000H, 000H, 000H
    STRING 000H, 000H, 000H, 000H
    STRING 000H, 000H, 000H, 000H

    STRING 01FH, 0B1H, 07DH, 0F0H
    STRING 011H, 0B1H, 041H, 010H
```

para quê?

```
    STRING 010H, 0B1H, 041H, 010H
    STRING 010H, 0B1H, 041H, 010H
    STRING 010H, 0B3H, 07DH, 0F0H
    STRING 010H, 092H, 061H, 088H
    STRING 010H, 092H, 061H, 088H
    STRING 010H, 092H, 061H, 088H
    STRING 01FH, 09EH, 07DH, 088H


; *********************************************************************************
; * Tabela de endereços de estados do loop de controlo
; *********************************************************************************
PLACE       1500H

TAB_ESTADO:
    WORD  boas_vindas       ; Boas_Vindas: Limpa o ecrã, desenha o ecrã de início de
jogo, espera uma tecla para começar
    WORD  inicio            ; Inicio:      Faz as inicializações, desenha as naves,
deixa tudo pronto a jogar
    WORD  jogar             ; Jogar:       O jogo em si, move naves, permite o movimento
da nave do jogador, disparo e rotação do canhão
    WORD  suspender         ; Suspender:   Após carregar numa tecla suspende o jogo,
voltando ao jogo carregando na mesma tecla
    WORD  fim_jogo          ; Fim_Jogo:    Após carregar em tecla ou o jogador perder,
limpa o ecrã e desenha o ecrã de final de jogo

estado_ctrl:                ; variavel que guarda o estado actual do controlo
    STRING 0H;

; *********************************************************************************
; * Código
; *********************************************************************************

PLACE 0H

inicio_programa:
    MOV   BTE, tab          ; Inicializa BTE
    MOV   SP, SP_inicial    ; Inicializa Stack Pointer
    MOV   R0, estado_ctrl   ; Comecar no inicio (estado 0 = boas_vindas)
    MOV   R1, 0             ; R1 com o estado actual
    MOVB  [R0], R1          ; Coloca na tabela de estado actual o primeiro estado
    MOV   R0, POUT1         ; R0 com o endereço do porto de saída 1
    MOV   R2, ENERGIA       ; R2 com o valor de energia inicial
    MOVB  [R0], R2          ; Coloca o valor de energia no display hexadecimal
loop_controlo:
    MOV   R0, estado_ctrl   ; Obter o estado actual
    MOVB  R1, [R0]          ; R1 com o endereço do estado actual
    SHL   R1, 1             ; Multiplicar por dois
    MOV   R0, TAB_ESTADO    ; Endereço base dos processos do jogo
    ADD   R1, R0            ; Agora R0 aponta para a rotina correspondente ao estado
actual
    MOV   R0, [R1]          ; Obter o endereço da rotina a chamar
    CALL  R0                ; invocar o processo correspondente ao estado
    JMP   loop_controlo     ; voltar a repetir até ao infinito


; *********************************************************************************
; * Rotinas que implementam o Processos Boas Vindas
; *********************************************************************************

boas_vindas:
    PUSH  R0                ; Guarda registos
    PUSH  R1
    CALL  limpa             ; Rotina que limpa o ecrã, colocando todos os bytes a 0,
para garantir que pode ser escrito sem problemas
    MOV   R0, INICIO        ; Base da tabela de strings que contém o ecrã de início
    MOV   R1, ECRA_INT      ; Contador de linhas em branco antes das palavras
    CALL  ecra_escreve      ; Desenha no ecrã o ecrã de início
espera_tecla:
    CALL  teclado           ; Chama o teclado para verificar que tecla está a ser
premida (em R0)
    MOV   R1, T_INIC        ; Tecla de início de jogo
    CMP   R0, R1            ; Verifica se a tecla premida é a de início de jogo
    JNZ   espera_tecla      ; Enquanto não for premida a tecla de inicio de jogo volta a
correr o ciclo
```

```
    MOV   R0, estado_ctrl   ; Obter o estado actual
    MOV   R1, 1             ; Avançar para o próximo estado (inicio)
    MOVB  [R0], R1          ; Actualiza o estado actual com o endereço do estado
seguinte
    POP   R1                ; Recupera registos
    POP   R0
    RET


; *****************************************************************************
; * Rotina que implementa o Processo Inicio
; *****************************************************************************

inicio:
    PUSH  R0                ; Guarda registos
    PUSH  R1
    CALL  limpa             ; Remove o ecrã de boas-vindas para poder desenhar as naves
no ecrã
    CALL  inicializar       ; Rotina que deixa o jogo pronto a começar, desenha as naves
no ecrã, e inicializa variáveis importantes
    MOV   R0, estado_ctrl   ; Obter o estado actual
    MOV   R1, 2             ; Avançar para o próximo estado (jogar)
    MOVB  [R0], R1          ; Actualiza o estado actual com o endereço do estado
seguinte
       EI1                     ; Enable interrupts
    EI0
    EI
    POP   R1                ; Recupera registos
    POP   R0
    RET

; *****************************************************************************
; * Rotina que implementa o Processo Jogar
; *****************************************************************************

jogar:
    PUSH  R0                ; Guarda registos
    PUSH  R1
    PUSH  R2
    PUSH  R3
    PUSH  R4
loop_jogar:
    CALL  teclado           ; Devolve em R0 a tecla premida (ou nenhuma)
    MOV   R1, T_SUSP        ; R1 com a tecla de suspender o jogo
    CMP   R0, R1            ; Verifica se a tecla premida é a de suspender o jogo
    JNZ   nao_susp          ; Se a tecla premida não for a de suspensão então o jogo
compara com outras teclas
    MOV   R1, 3             ; Mudar para o estado Suspender
    MOV   R0, estado_ctrl   ; Obter o estado actual
    MOVB  [R0], R1          ; Coloca o endereço do próximo estado na tabela de estado
actual
    JMP   sair_jogar        ; Sai para o estado seguinte
nao_susp:
    MOV   R1, T_TERM        ; R1 com a tecla de terminar
    CMP   R0, R1            ; Verifica se a tecla premida é a de terminar o jogo
    JNZ   nao_term          ; Se a tecla premida não for a de terminar então vai
comparar com outras teclas
    MOV   R1, 4             ; Mudar para o estado Terminar
    MOV   R0, estado_ctrl   ; Obter o estado actual
    MOVB  [R0], R1          ; Coloca o endereço do próximo estado na tabela de estado
actual
    JMP   sair_jogar        ; Sai para o estado seguinte
nao_term:
    MOV   R1, T_ROD_ESQ     ; R1 com a tecla de rodar o canhão à esquerda
    CMP   R0, R1            ; Verifica se a tecla premida é a de rodar à esquerda
    JNZ   nao_can_esq       ; Se não for vai comparar com outras teclas
    CALL  canhao_esq        ; Roda o canhão à esquerda
    JMP   jog_pos_tec       ; Volta ao ciclo de jogo
nao_can_esq:
    MOV   R1, T_ROD_DIR     ; R1 com a tecla de rodar o canhão à direita
    CMP   R0, R1            ; Verifica se a tecla premida é a de rodar à direita
    JNZ   nao_can_dir       ; Se não for vai comparar com outras teclas
    CALL  canhao_dir        ; Roda o canhão à direita
    JMP   jog_pos_tec       ; Volta ao ciclo de jogo
nao_can_dir:
    MOV   R1, T_DISPARO     ; R1 com a tecla de disparar o raio
```

```
    CMP   R0, R1            ; Verifica se a tecla premida é a de disparar
    JNZ   nao_disp          ; Se não for vai comparar com outras teclas
    JMP   jog_pos_tec       ; Volta ao ciclo de jogo
nao_disp:
    MOV   R1, T_NEUTRA      ; R1 com a tecla neutra
    CMP   R0, R1            ; Verifica se não há tecla premida
    JZ    jog_pos_tec       ; Se não houver tecla premida volta ao ciclo de jogo
    MOV   R1, T_INIC        ; R1 com a tecla de início
    CMP   R0, R1            ; Verifica se a tecla premida é a de início
    JZ    jog_pos_tec       ; Se for volta ao ciclo de jogo para varrer o teclado
    MOV   R1, T_IGNORE1     ; R1 com a tecla para ser ignorada
    CMP   R0, R1            ; Verifica se a tecla premida é para ser ignorada
    JZ    jog_pos_tec       ; Se for volta ao ciclo de jogo para varrer o teclado
    MOV   R1, T_IGNORE2     ; R1 com a tecla para ser ignorada
    CMP   R0, R1            ; Verifica se a tecla premida é para ser ignorada
    JZ    jog_pos_tec       ; Se for volta ao ciclo de jogo para varrer o teclado
    CALL  move_nave         ; Rotina que move a nave
jog_pos_tec:
    MOV   R2, FLAG_ALIEN    ; Endereço de uma flag que diz se os aliens se movimentam ou
não
    MOV   R1, [R2]          ; Valor da flag
    AND   R1, R1            ; Afectação de flags do processador
    JZ    nao_ataca         ; Se os aliens não se moverem, não os desenha
    CALL  ataque_alien      ; Rotina que movimenta os aliens no ecrã
    MOV   R1, 0             ; R1 com 0 para dar reset à flag
    MOV   [R2], R1          ; Reset na flag
nao_ataca:
    MOV   R2, FLAG_DISPARO  ; Endereço de uma flag que diz se o canhão está a disparar
    MOV   R0, [R2]          ; Valor da flag
    CALL  disparo           ; Chama a rotina que faz avançar o raio
    CALL  colisao           ; Chama a rotina que verifica se há colisão entre naves
verif_energia:
    MOV   R0, V_ENERGIA     ; R0 com a variável que contém o valor actual de energia
    MOV   R1, [R0]          ; R1 com o valor actual de energia
    MOV   R2, POUT1
    MOVB  [R2], R1          ; Mostrar o valor actual da energia
    CMP   R1, 0             ; Se R1 for 0 então o jogo termina
    JNZ   loop_jogar        ; Se não for 0 então volta para o ciclo de jogo
    MOV   R1, 4             ; Mudar para o estado Terminar
    MOV   R0, estado_ctrl   ; Obter o estado actual
    MOVB  [R0], R1          ; Coloca o endereço do próximo estado na tabela de estado
actual
sair_jogar:
    POP   R4                ; Recupera registos
    POP   R3
    POP   R2
    POP   R1
    POP   R0
    RET                     ; Termina rotina


; ***************************************************************************
; * Rotina que implementa o Processo Suspender
; ***************************************************************************

suspender:
    PUSH  R0                ; Guarda registo R0
    PUSH  R1                ; Guarda registo R1
        DI1                   ; Disable interrupts
    DI0                     basTava esta!
    DI
    CALLF inverte_ecra      ; Inverte o ecrã para diferenciar o jogo suspenso do jogo a
decorrer
ciclo_suspender:
    CALL  teclado           ; Devolve em R0 a tecla premida (ou nenhuma)
    MOV   R1, T_SUSP        ; R1 com a tecla de suspender
    CMP   R0, R1            ; Verifica se a tecla premida é a de suspender o jogo
    JNZ   ver_termina       ; Se a tecla premida for a de suspensão então o jogo volta
ao normal
    MOV   R1, 2             ; Avançar para o próximo estado (jogar)
    JMP   sai_suspensao     ; Termina a rotina e avança para o próximo estado
ver_termina:
    MOV   R1, T_TERM        ; R2 com a tecla de terminar jogo
    CMP   R0, R1            ; Se a tecla premida não for a de suspensão verifica se é a
de fim de jogo
```

```
    JNZ   ciclo_suspender   ; Se a tecla de fim de jogo for premida então o jogo acaba
imediatamente
    MOV   R1, 4             ; Avançar para o próximo estado (terminar_jogo)
sai_suspensao:              ; Quando a tecla de suspender for premida então trata de pôr
tudo a funcionar
    MOV   R0, estado_ctrl   ; Obter o estado actual
    MOVB  [R0], R1          ; Coloca o endereço do próximo estado na tabela de estado
actual
       EI1                     ; Enable interrupts
    EI0
    EI
    CALLF inverte_ecra      ; Volta a colocar o ecrã como aparece num jogo a decorrer
    POP   R1                ; Recupera registo R1
    POP   R0                ; Recupera registo R0
    RET                     ; Termina rotina

; *********************************************************************************
; * Rotina que implementa o Processo Fim Jogo
; *********************************************************************************

fim_jogo:
    PUSH  R0                ; Guarda registos
    PUSH  R1
    CALLF inverte_ecra      ; Inverte o ecrã, para dar a ideia de que algo aconteceu
    CALLF pausa             ; Rotina que faz uma pequena pausa para efeito visual
    CALLF inverte_ecra      ; Volta a colocar o ecrã como aparece num jogo a decorrer
    CALLF pausa             ; Rotina que faz uma pequena pausa para efeito visual
    CALL  limpa             ; Limpa o ecrã para desenhar o final de jogo
    MOV   R0, FIM           ; R0 com base da tabela de strings que contém o ecrã de fim
de jogo
    MOV   R1, ECRA_INT      ; Coloca em R1 as linhas em branco antes das palavras
    CALL  ecra_escreve      ; Desenha no ecrã o final de jogo
loop_fim_jogo:
    CALL  teclado           ; Devolve em R0 a tecla premida (ou nenhuma)
    MOV   R1, T_INIC        ; tecla de iniciar?
    CMP   R0, R1            ; Verifica se a tecla premida é a de iniciar o jogo
    JNZ   loop_fim_jogo     ; Se a tecla premida for a de iniciar então o jogo volta ao
inicio
    MOV   R1, 1             ; Mudar para o estado Inicio para reiniciar
    MOV   R0, estado_ctrl   ; Obter o estado actual
    MOVB  [R0], R1          ; Coloca o endereço do próximo estado na tabela de estado
actual
    POP   R1                ; Recupera registos
    POP   R0
    RET                     ; Termina rotina


; *********************************************************************
; * ROTINAS
; *********************************************************************

; *********************************************************************
; Inicializar
;   Rotina que inicializa várias variáveis
; Entradas
;   Nenhuma
; Saídas
;   Nenhuma
; Altera
;   Várias variáveis (posição de naves, energia, flags, ...)
; *********************************************************************
inicializar:
    PUSH  R0                ; Guarda registos
    PUSH  R1
    PUSH  R2
    PUSH  R3
    PUSH  R4
    PUSH  R5
    PUSH  R6
    MOV   R0, POUT1         ; R0 com o endereço do porto de saída 1
    MOV   R1, V_ENERGIA     ; R1 com o endereço da tabela que guarda o valor de energia
    MOV   R2, ENERGIA       ; R2 com o valor de energia inicial
    MOV   [R1], R2          ; Inicializa o valor de energia
    MOVB  [R0], R2          ; Coloca o valor de energia no display hexadecimal
    MOV   R2, 0             ; Flag que indica permissão de movimento dos aliens a 0
    MOV   R1, FLAG_ALIEN    ; Assinalar que aliens não se estão a mexer
```

```
    MOV   [R1], R2          ; Afecta a flag que diz se os aliens se podem mover
    MOV   R1, FLAG_DISPARO  ; Assinalar que o canhão não está a disparar
    MOV   [R1], R2          ; Afecta a flag que diz se o canhão está a disparar
    MOV   R1, FLAG_DESENHA  ; Assinalar que o canhão não deve ser apagado
    MOV   [R1], R2          ; Afecta a flag que diz se o canhão deve ser apagado
    CALL  inicializa_alien  ; Inicializações dos aliens
    CALL  aliens            ; Desenha as naves alien no ecrã
    MOV   R1, LINHA_NAVE    ; R1 com a tabela que contém a linha do centro da nave
    MOV   R2, COLUNA_NAVE   ; R2 com a tabela que contém a coluna do centro da nave
    MOV   R3, LIN_NAVE      ; R3 com a linha inicial do centro da nave
    MOV   R4, COL_NAVE      ; R4 com a coluna inicial do centro da nave
    MOV   [R1], R3          ; Inicializa a linha actual da nave com a linha inicial
    MOV   [R2], R4          ; Inicializa a coluna actual da nave com a coluna inicial
    MOV   R5, POS_CANHAO    ; R5 com a tabela que contém a posição actual do canhão
    MOV   R6, 0             ; R6 com 0 para inicializar a posição do canhão
    MOV   [R5], R6          ; Inicializa a posição do canhão a 0, para escrever o canhão
na posição inicial
    CALL  nave              ; Desenha a nave do jogador no ecrã
    MOV   R0, 1             ; R0 com 1 para ser passado como parâmetro à função que
desenha o canhão
    CALL  canhao            ; Coloca o canhão na posição inicial
    MOV   R3, BUFFER        ; R3 com o endereço do buffer
    MOV   R4, 0             ; R4 com 0 para limpar o buffer
    MOVB  [R3], R4          ; Coloca o buffer a 0 para não interferir com futuras
leituras do teclado
    MOV   R5, AUX_BUFFER    ; R5 com o endereço de aux_buffer
    MOV   R6, 055H          ; R6 com 55H, um valor diferente de tecla (verifica se ainda
há tecla premida)
    MOVB  [R5], R6          ; Inicializa o valor da tabela aux_buffer com 55H
    POP   R6                ; Recupera registos
    POP   R5
    POP   R4
    POP   R3
    POP   R2
    POP   R1
    POP   R0
    RET                     ; Termina rotina


; ***************************************************************************
; Inicializa Alien
;   Rotina que coloca as naves alien na sua posição inicial
; Entradas
;   Nenhuma
; Saídas
;   Nenhuma
; ***************************************************************************
inicializa_alien:
    PUSH  R0                ; Guarda registos
    PUSH  R1
    PUSH  R2
    PUSH  R3
    PUSH  R4
    PUSH  R5
    PUSH  R6
    PUSH  R7
    MOV   R0, 0             ; R0 com o alien actual a ser inicializado
    MOV   R1, N_ALIEN       ; R1 com contador do número de aliens
inicializa1:
    MOV   R2, LINHA_ALIEN   ; R2 com a tabela que contém a linha actual do centro de
cada alien
    ADD   R2, R0            ; Acede à linha actual do alien actual
    MOV   R3, COLUNA_ALIEN  ; R3 com a tabela que contém a coluna actual do centro de
cada alien
    ADD   R3, R0            ; Acede à coluna actual do alien actual
    MOV   R4, LINHA_INICIO  ; R4 com a tabela que contém a linha inicial do centro de
cada alien
    ADD   R4, R0            ; Acede à linha inicial do alien actual
    MOV   R5, COLUNA_INICIO ; R5 com a tabela que contém a coluna inicial do centro de
cada alien
    ADD   R5, R0            ; Acede à coluna inicial do alien actual
    MOVB  R6, [R4]          ; R6 com a linha inicial do centro do alien actual
    MOVB  R7, [R5]          ; R7 com a coluna inicial do centro do alien actual
    MOVB  [R2], R6          ; Linha actual do alien actualizada para a linha inicial
    MOVB  [R3], R7          ; Coluna actual do alien actualizada para a coluna inicial
    ADD   R0, 1             ; Avança para o alien seguinte
```

```
    SUB   R1, 1             ; Actualiza o contador
    JNZ   inicializa1       ; Inicializa as coordenadas do alien seguinte
    POP   R7                ; Restaura registos
    POP   R6
    POP   R5
    POP   R4
    POP   R3
    POP   R2
    POP   R1
    POP   R0
    RET                     ; Termina função

; *********************************************************************
; Escreve Ecrã
;   Rotina que desenha uma tabela de strings no ecrã
; Entradas
;   R0 - Tabela de strings a escrever
;   R1 - Linhas em branco antes de escrever
; Saídas
;   Nenhuma
; *********************************************************************
ecra_escreve:
    PUSH  R0                ; Guarda registos
    PUSH  R1
    PUSH  R2
    PUSH  R3
    PUSH  R4
    MOV   R3, ECRA          ; Endereço do ecrã
    ADD   R3, R1            ; Actualização do endereço onde começar a escrever
    MOV   R2, MAX_ELE       ; Número de elementos da tabela de strings
ciclo_ecra:
    MOVB  R4, [R0]          ; Elemento actual da tabela de strings
    MOVB  [R3], R4          ; Escreve o elemento da tabela de strings no ecrã
    ADD   R0, 1             ; Acede ao índice seguinte da tabela de strings
    ADD   R3, 1             ; Avança para o byte seguinte do ecrã
    SUB   R2, 1             ; Actualiza o contador
    JNZ   ciclo_ecra        ; Volta ao ciclo para escrever o que falta
    POP   R4                ; Recupera registos
    POP   R3
    POP   R2
    POP   R1
    POP   R0
    RET                     ; Termina rotina

; *********************************************************************
; Aliens
;   Rotina que desenha todas as naves alien
; Entradas
;   Nenhuma
; Saídas
;   Nenhuma
; *********************************************************************
aliens:
    PUSH  R0                ; Guarda registos
    PUSH  R1
    MOV   R1, N_ALIEN       ; R1 com contador do número de aliens presentes no jogo
    MOV   R0, 0             ; R0 com o alien actual a ser desenhado
ciclo_aliens:
    CALL  desenha_alien     ; Função que desenha uma nave no ecrã, tanto para nave alien
ou nave do jogador
    ADD   R0, 1             ; Avança para o alien seguinte
    SUB   R1, 1             ; Actualiza o contador para o número de aliens que falta
desenhar
    JNZ   ciclo_aliens      ; Vai desenhar a nave alien seguinte
    POP   R1                ; Recupera registos
    POP   R0
    RET                     ; Termina função

; *********************************************************************
; Desenha Alien
;   Rotina que desenha uma nave alien
; Entradas
;   R0 - número do alien
; Saídas
;   R0 - número do alien (não alterado)
; *********************************************************************
```

```
desenha_alien:
    PUSH  R0                ; Guarda registos
    PUSH  R1
    PUSH  R2
    PUSH  R3
    PUSH  R4
    PUSH  R5
    PUSH  R6
    PUSH  R7
    PUSH  R8
    PUSH  R9
    MOV   R3, LINHA_ALIEN   ; R3 com a tabela correspondente à linha do centro de cada
nave alien
    MOV   R4, COLUNA_ALIEN  ; R4 com a tabela correspondente à coluna do centro de cada
nave alien
    ADD   R3, R0            ; Acede à linha do alien a desenhar
    ADD   R4, R0            ; Acede à coluna do alien a desenhar
    MOV   R6, 1             ; R6 com 1, 0, -1, valor que permite calcular as linhas
adjacentes à do centro, para desenhar a nave
    MOV   R8, MAX_LINHA     ; Contador dos bits que são desenhados por linha
ciclo_desenha1:
    MOVB  R1, [R3]          ; R1 com a linha do centro do alien actual
    ADD   R1, R6            ; Soma que permite aceder à linha onde desenhar, a partir da
coordenada do centro
    MOV   R7, 1             ; R7 com 1, 0, -1, valor que permite calcular as colunas
adjacentes à do centro, para desenhar a nave
    MOV   R9, MAX_LINHA     ; Contador dos bits que são desenhados por linha
ciclo_desenha2:
    MOVB  R2, [R4]          ; R2 com a coluna do centro do alien centro
    ADD   R2, R7            ; Soma que permite aceder à coluna onde desenhar, a partir
da coordenada do centro
    PUSH  R1                ; Guarda registo R1
    PUSH  R2                ; Guarda registo R2
    MOV   R0, ALIEN         ; R0 com a tabela que guarda o formato de uma nave alien
    MOV   R1, R6            ; R1 com a linha onde desenhar, a partir do centro (para
poder ser passada como parâmetro à função Lê Pixel)
    ADD   R1, 1             ; Soma 1, porque a função Lê Pixel recebe como parâmetro um
valor entre 0 e 2
    MOV   R2, R7            ; R2 com a coluna onde desenhar, a partir do centro (para
poder ser passada como parâmetro à função Lê Pixel)
    ADD   R2, 1             ; Soma 1, porque a função Lê Pixel recebe como parâmetro um
valor entre 0 e 2
    CALLF le_pixel          ; Função Lê Pixel, que diz que bits devem ser escritos,
recebe como argumentos: R0 - Valor a escrever, R1 - Linha, R2 - Coluna
    POP   R2                ; Restaura registo R2 para poder desenhar na coluna certa
    POP   R1                ; Restaura registo R1 para poder desenhar na linha certa
    CALL  desenha           ; Função Desenha, que desenha um bit no ecrã, recebe como
argumentos: R0 - Valor a escrever, R1 - Linha, R2 - Coluna
    SUB   R7, 1             ; Avança para a linha anterior (contada a partir do centro)
    SUB   R9, 1             ; Actualiza o contador de colunas a desenhar
    JNZ   ciclo_desenha2    ; Percorre todos os valores de colunas adjacentes dentro de
uma linha
    SUB   R6, 1             ; Avança para a linha anterior (contada a partir do centro)
    SUB   R8, 1             ; Actualiza o contador de linhas a desenhar
    JNZ   ciclo_desenha1    ; Desenha todas as linhas de uma nave
    POP   R9                ; Restaura registos
    POP   R8
    POP   R7
    POP   R6
    POP   R5
    POP   R4
    POP   R3
    POP   R2
    POP   R1
    POP   R0
    RET                     ; Termina rotina

; ***************************************************************************
; Apaga Alien
;   Rotina que apaga uma nave alien
; Entradas
;   R0 - número do alien
; Saídas
;   R0 - número do alien (não alterado)
; ***************************************************************************
apaga_alien:
```

```
    PUSH  R0                ; Guarda registos
    PUSH  R1
    PUSH  R2
    PUSH  R3
    PUSH  R4
    PUSH  R5
    PUSH  R6
    PUSH  R7
    PUSH  R8
    PUSH  R9
    MOV   R3, LINHA_ALIEN   ; R3 com a tabela correspondente à linha do centro de cada
nave alien
    MOV   R4, COLUNA_ALIEN  ; R4 com a tabela correspondente à coluna do centro de cada
nave alien
    ADD   R3, R0            ; Acede à linha do alien a desenhar
    ADD   R4, R0            ; Acede à coluna do alien a desenhar
    MOV   R6, 1             ; R6 com 1, 0, -1, valor que permite calcular as linhas
adjacentes à do centro, para desenhar a nave
    MOV   R8, MAX_LINHA     ; Contador dos bits que são desenhados por linha
apaga1:
    MOVB  R1, [R3]          ; R1 com a linha do centro do alien actual
    ADD   R1, R6            ; Soma que permite aceder à linha onde desenhar, a partir da
coordenada do centro
    MOV   R7, 1             ; R7 com 1, 0, -1, valor que permite calcular as colunas
adjacentes à do centro, para desenhar a nave
    MOV   R9, MAX_LINHA     ; Contador dos bits que são desenhados por linha
apaga2:
    MOVB  R2, [R4]          ; R2 com a coluna do centro do alien centro
    ADD   R2, R7            ; Soma que permite aceder à coluna onde desenhar, a partir
da coordenada do centro
    MOV   R0, 0
    CALL  desenha           ; Função Desenha, que desenha um bit no ecrã, recebe como
argumentos: R0 - Valor a escrever, R1 - Linha, R2 - Coluna
    SUB   R7, 1             ; Avança para a linha anterior (contada a partir do centro)
    SUB   R9, 1             ; Actualiza o contador de colunas a desenhar
    JNZ   apaga2            ; Percorre todos os valores de colunas adjacentes dentro de
uma linha
    SUB   R6, 1             ; Avança para a linha anterior (contada a partir do centro)
    SUB   R8, 1             ; Actualiza o contador de linhas a desenhar
    JNZ   apaga1            ; Desenha todas as linhas de uma nave
    POP   R9                ; Restaura registos
    POP   R8
    POP   R7
    POP   R6
    POP   R5
    POP   R4
    POP   R3
    POP   R2
    POP   R1
    POP   R0
    RET                     ; Termina rotina

; ***************************************************************************
; Ataque Alien
;   Rotina que move todas as naves alien
; Entradas
;   Nenhuma
; Saídas
;   Nenhuma
; ***************************************************************************
ataque_alien:
    PUSH  R0                ; Guarda registos
    PUSH  R1
    MOV   R0, 0             ; R0 com o alien actual a ser movido
    MOV   R1, N_ALIEN       ; R1 com contador do número de aliens a serem movidos
ataque1:
    CALL  move_alien        ; Função que move uma nave no ecrã, tanto para nave alien ou
nave do jogador
    ADD   R0, 1             ; Avança para o alien seguinte
    SUB   R1, 1             ; Actualiza o contador para o número de aliens que falta
movimentar
    JNZ   ataque1           ; Vai mover a nave alien seguinte
    POP   R1                ; Recupera registos
    POP   R0
    RET                     ; Termina função
```

```
; *********************************************************************
; Move Alien
;   Rotina que move uma nave alien
; Entradas
;   R0 - número do alien
; Saídas
;   R0 - número do alien (não alterado)
; Altera
;   LINHA_ALIEN
;   COLUNA_ALIEN
; *********************************************************************
move_alien:
    PUSH  R0                ; Recupera registos
    PUSH  R1
    PUSH  R2
    PUSH  R3
    PUSH  R4
    PUSH  R5
    CALL  apaga_alien       ; Apaga o alien para o poder escrever na nova posição
    MOV   R3, LINHA_ALIEN   ; Endereço da tabela que contém a linha actual do centro de
cada alien
    ADD   R3, R0            ; Acede à linha do centro do alien actual
    MOVB  R1, [R3]          ; Linha do centro
    MOV   R4, LINHA_NAVE    ; Endereço da variavel que contém a linha actual do centro
da nave
    MOV   R5, [R4]          ; Linha da nave
    CMP   R5, R1            ; Compara a linha da nave com a linha do alien (linha nave -
linha alien)
    JGT   move1             ; Salta se nave mais abaixo do que o alien
    JEQ   move2             ; Salta se estiverem na mesma linha
    SUB   R1, 1             ; Nave mais acima do que o alien, este precisa de subir
    MOVB  [R3], R1          ; Guardar nova linha do alien
    JMP   move2             ; Agora ver as colunas
move1:
    ADD   R1, 1             ; nave está abaixo do alien, este precisa de descer
    MOVB  [R3], R1          ; Guardar nova linha do alien
move2:
    MOV   R3, COLUNA_ALIEN  ; Endereço da tabela que contém a coluna actual do centro de
cada alien
    ADD   R3, R0            ; Acede à coluna do centro do alien actual
    MOVB  R2, [R3]          ; Coluna do centro
    MOV   R4, COLUNA_NAVE   ; Endereço da tabela que contém a coluna actual do centro da
nave
    MOV   R5, [R4]          ; Coluna da nave
    CMP   R5, R2            ; Compara a coluna da nave com a coluna do alien (coluna
nave - coluna alien)
    JGT   move3             ; Salta se nave mais à direita do que o alien
    JEQ   move4             ; Salta se os dois na mesma coluna
    SUB   R2, 1             ; Nave mais à esquerda do alien, este precisa de ir para a
esquerda
    MOVB  [R3], R2          ; Guardar nova coluna do alien
    JMP   move4             ; Agora desenhar no nova posição
move3:
    ADD   R2, 1             ; Nave mais à direita do alien, este precisa de ir para a
direita
    MOVB  [R3], R2          ; Guardar nova coluna do alien
move4:
    CALL  desenha_alien     ; Desenha o alien
    POP   R5                ; Recupera registos
    POP   R4
    POP   R3
    POP   R2
    POP   R1
    POP   R0
    RET

; *********************************************************************
; Colisão
;   Rotina que verifica colisão entre naves inimigas e obstáculos
; Entradas
;   Nenhuma
; Saídas
;   Nenhuma
; *********************************************************************
colisao:
    PUSH  R0                ; Guarda registos
```

```
    PUSH  R1
        PUSH  R2
        PUSH  R3
    MOV   R0, 0             ; R0 com o alien actual a verificar
    MOV   R1, N_ALIEN       ; R1 com contador do número de aliens a serem verificados
colisao1:
    CALL  ver_colisao       ; Função que verifica se há colisão
        MOV   R2, FLAG_DISPARO  ; Variável que guarda uma flag que diz se o canhão
dispara ou não
        MOV   R3, [R2]          ; Flag em registo
        AND   R3, R3            ; Afectação de flags
        JZ    cont_colisao      ; Se a flag for zero então não vale a pena verificar se
os aliens colidem com o raio
    CALL  colisao_raio      ; Rotina que verifica se os aliens colidem com o raio
 cont_colisao:
    ADD   R0, 1             ; Avança para o alien seguinte
    SUB   R1, 1             ; Actualiza o contador para o número de aliens que falta
verificar
    JNZ   colisao1          ; Vai verificar a nave alien seguinte
    POP   R3                ; Recupera registos
    POP   R2
    POP   R1
    POP   R0
    RET                     ; Termina função

; ********************************************************************
; Verifica Colisão
;   Rotina que verifica colisão entre nave do jogador e naves inimigas
; Entradas
;   R0 - número do alien
; Saídas
;   R0 - número do alien (não é alterado durante a rotina)
; Altera
;   V_ENERGIA
; ********************************************************************
ver_colisao:
    PUSH  R0                ; Guarda registos
    PUSH  R1
        PUSH  R2
    PUSH  R3
    PUSH  R4
    PUSH  R5
    MOV   R3, LINHA_ALIEN   ; Endereço da tabela que contém a linha actual do centro de
cada alien
    ADD   R3, R0            ; Acede à linha do centro do alien actual
    MOV   R1, 0             ; Limpar registo (byte de maior peso)
    MOVB  R1, [R3]          ; Numero da Linha do centro do alien
    MOV   R2, LINHA_NAVE    ; Endereço da variável que contém a linha actual do centro
da nave
    MOV   R5, [R2]          ; Linha da nave
    CMP   R5, R1            ; Compara a linha da nave com a linha do alien
    JGT   ver_col1          ; Se a nave for maior vai verificar se a diferença é menor
que 3
    SUB   R1, R5            ; Como a linha do alien é menor, subtrai a linha do alien à
da nave
    CMP   R1, 3             ; Compara o valor obtido com 3
    JLT   ver_coluna        ; Se a diferença for menor que 3 pode haver colisão, falta
ver colunas
    JMP   fim_ver           ; Não pode haver colisão, logo terminar rotina
ver_col1:
    SUB   R5, R1            ; Calcula a diferença entre a linha da nave e a linha do
alien
    CMP   R5, 3             ; Compara a diferença com 3, pois qualquer nave ocupa 3
linhas
    JLT   ver_coluna        ; Se a diferença for menor que 3 pode haver colisão, falta
ver colunas
    JMP   fim_ver           ; Não pode haver colisão, logo terminar rotina
ver_coluna:
    MOV   R3, COLUNA_ALIEN  ; Endereço da tabela que contém a coluna actual do centro de
cada alien
    ADD   R3, R0            ; Acede à coluna do centro do alien actual
    MOV   R1, 0             ; Limpar registo (byte de maior peso)
    MOVB  R1, [R3]          ; Coluna do centro
    MOV   R4, COLUNA_NAVE   ; Endereço da variável que contém a coluna actual do centro
da nave
    MOV   R5, [R4]          ; Coluna da nave
```

```
    CMP   R5, R1            ; Compara a coluna da nave com a coluna do alien
    JGT   ver_col2          ; Se a nave for maior vai verificar se a diferença é menor
que 3
    SUB   R1, R5            ; Como a coluna do alien é menor, subtrai a coluna do alien
à da nave
    CMP   R1, 3             ; Compara a diferença com 3
    JLT   ha_colisao        ; Se a diferença for menor que 3 então há colisão
    JMP   fim_ver           ; Acabaram as verificações, a rotina vai terminar
ver_col2:
    SUB   R5, R1            ; Calcula a diferença entre a coluna da nave a a coluna do
alien
    CMP   R5, 3             ; Compara a diferença com 3, pois qualquer nave ocupa 3
colunas
    JLT   ha_colisao        ; Se a diferença for menor que 3 há colisão
    JMP   fim_ver           ; Salta para terminar a função
ha_colisao:
    MOV   R1, V_ENERGIA     ; Variável que guarda o valor actual de energia
    MOV   R5, 0             ; Actualizar com 0 pois se há colisão a energia da nave
acaba
    MOV   [R1], R5          ; Actualiza o valor de energia
    MOV   R1, POUT1         ; Endereço do display hexadecimal
    MOVB  [R1], R5          ; Actualiza o display hexadecimal
fim_ver:
    POP   R5                ; Guarda registos
    POP   R4
    POP   R3
       POP   R2
    POP   R1
       POP   R0
    RET                     ; Termina rotina

; *********************************************************************
; Colisão Raio
;   Rotina que verifica colisão entre raio e naves inimigas
; Entradas
;   R0 - número do alien
; Saídas
;   R0 - número do alien (não é alterado durante a rotina)
; Altera
;   LINHA_INICIO
;   COLUNA_INICIO
;   V_ENERGIA
; *********************************************************************
colisao_raio:
    PUSH  R0                ; Guarda registos
    PUSH  R1
       PUSH  R2
    PUSH  R3
    PUSH  R4
    PUSH  R5
       PUSH  R6
    MOV   R3, LINHA_ALIEN   ; Endereço da tabela que contém a linha actual do centro de
cada alien
    ADD   R3, R0            ; Acede à linha do centro do alien actual
    MOV   R1, 0             ; Limpar registo (byte de maior peso)
    MOVB  R1, [R3]          ; Numero da Linha do centro do alien
    MOV   R2, LINHA_RAIO    ; Endereço da variável que contém a linha actual da ponta do
raio
    MOVB  R5, [R2]          ; Linha do raio
    CMP   R5, R1            ; Compara a linha da nave com a linha do alien
    JGT   raio_col1         ; Se a nave for maior vai verificar se a diferença é menor
que 3
    SUB   R1, R5            ; Como a linha do alien é menor, subtrai a linha do alien à
da nave
    CMP   R1, 2             ; Compara o valor obtido com 3
    JLT   raio_coluna       ; Se a diferença for menor que 3 pode haver colisão, falta
ver colunas
    JMP   fim_raio          ; Não pode haver colisão, logo terminar rotina
raio_col1:
    SUB   R5, R1            ; Calcula a diferença entre a linha da nave e a linha do
alien
    CMP   R5, 2             ; Compara a diferença com 3, pois qualquer nave ocupa 3
linhas
    JLT   raio_coluna       ; Se a diferença for menor que 3 pode haver colisão, falta
ver colunas
    JMP   fim_raio          ; Não pode haver colisão, logo terminar rotina
```

```
raio_coluna:
    MOV   R3, COLUNA_ALIEN  ; Endereço da tabela que contém a coluna actual do centro de
cada alien
    ADD   R3, R0            ; Acede à coluna do centro do alien actual
    MOV   R1, 0             ; Limpar registo (byte de maior peso)
    MOVB  R1, [R3]          ; Coluna do centro
    MOV   R4, COLUNA_RAIO   ; Endereço da variável que contém a coluna actual do centro
da nave
    MOVB  R5, [R4]          ; Coluna da nave
    CMP   R5, R1            ; Compara a coluna da nave com a coluna do alien
    JGT   raio_col2         ; Se a nave for maior vai verificar se a diferença é menor
que 3
    SUB   R1, R5            ; Como a coluna do alien é menor, subtrai a coluna do alien
à da nave
    CMP   R1, 2             ; Compara a diferença com 3
    JLT   raio_colide       ; Se a diferença for menor que 3 então há colisão
    JMP   fim_raio          ; Acabaram as verificações, a rotina vai terminar
raio_col2:
    SUB   R5, R1            ; Calcula a diferença entre a coluna da nave a a coluna do
alien
    CMP   R5, 2             ; Compara a diferença com 3, pois qualquer nave ocupa 3
colunas
    JLT   raio_colide       ; Se a diferença for menor que 3 há colisão
    JMP   fim_raio          ; Salta para terminar a função
raio_colide:
    CALL  apaga_alien       ; Apaga o alien da posição onde houver colisão
    MOV   R1, V_ENERGIA     ; Variável que guarda o valor actual de energia
        MOV   R2, [R1]        ; Valor da energia actual em registo
    MOV   R5, ENE_DEST      ; Recompensa de energia por destruir um alien em registo
        ADD   R2, R5          ; Valor de energia actualizado com bónus por matar alien
    MOV   [R1], R2          ; Actualiza o valor de energia
    MOV   R2, LINHA_ALIEN   ; R2 com a tabela que contém a linha actual do centro de
cada alien
    ADD   R2, R0            ; Acede à linha actual do alien actual
    MOV   R3, COLUNA_ALIEN  ; R3 com a tabela que contém a coluna actual do centro de
cada alien
    ADD   R3, R0            ; Acede à coluna actual do alien actual
    MOV   R4, LINHA_INICIO  ; R4 com a tabela que contém a linha inicial do centro de
cada alien
    ADD   R4, R0            ; Acede à linha inicial do alien actual
    MOV   R5, COLUNA_INICIO ; R5 com a tabela que contém a coluna inicial do centro de
cada alien
    ADD   R5, R0            ; Acede à coluna inicial do alien actual
    MOVB  R6, [R4]          ; R6 com a linha inicial do centro do alien actual
    MOVB  R7, [R5]          ; R7 com a coluna inicial do centro do alien actual
    MOVB  [R2], R6          ; Linha actual do alien actualizada para a linha inicial
    MOVB  [R3], R7          ; Coluna actual do alien actualizada para a coluna inicial
fim_raio:
    POP   R6                ; Guarda registos
        POP   R5
    POP   R4
    POP   R3
        POP   R2
    POP   R1
        POP   R0
    RET                     ; Termina rotina

; Handler do interrupt int0 (mover aliens)
int0:
    PUSH  R0                ; Guarda registos
    PUSH  R1
    MOV   R1, FLAG_ALIEN    ; Indicador de que os aliens devem mexer-se
    MOV   R0, 1             ; Flag a 1 porque os alien devem mexer-se
    MOV   [R1], R0          ; Actualiza indicador
    POP   R1                ; Recupera registos
    POP   R0
    RFE                     ; Fim de rotina de interrupt

; Handler do interrupt int1 (energia gasta a disparar canhão)
int1:
    PUSH  R0                ; Guarda registos
        PUSH  R1
        PUSH  R2
        MOV   R0, FLAG_DISPARO  ; Indicador se o canhão está a disparar
        MOV   R1, [R0]          ; Flag se o canhão está a disparar em registo
        AND   R1, R1            ; Afectação de flags
```

```
        JZ    fim_int           ; Se o canhão não está a disparar então não desconta
energia
        MOV   R0, V_ENERGIA     ; Variável com o valor actual de energia
        MOV   R1, [R0]          ; Valor actual da energia em registo
        MOV   R2, ENE_CANHAO    ; Valor da energia perdida por disparar canhão
continuamente em registo
        SUB   R1, R2            ; Actualiza valor da energia
        MOV   [R0], R1          ; Guarda valor da energia na variável
fim_int:
    POP   R2                ; Recupera registos
    POP   R1
    POP   R0
    RFE


; ********************************************************************
; Nave
;   Rotina que desenha a nave do jogador
; Entradas
;   Nenhuma
; Saídas
;   Nenhuma
; ********************************************************************
nave:
    PUSH  R0                ; Guarda registos
    PUSH  R1
    PUSH  R2
    PUSH  R3
    PUSH  R4
    PUSH  R5
    PUSH  R6
    PUSH  R7
    PUSH  R8
    PUSH  R9
    MOV   R6, 1             ; R6 com 1, 0, -1, valor que permite calcular as linhas
adjacentes à do centro, para desenhar a nave
    MOV   R8, MAX_LINHA     ; Contador dos bits que são desenhados por linha
ciclo_nave1:
    MOV   R3, LINHA_NAVE    ; Variável da linha do centro da nave do jogador
    MOV   R1, [R3]          ; Linha do centro da nave em registo
    ADD   R1, R6            ; Soma que permite aceder à linha onde desenhar, a partir da
coordenada do centro
    MOV   R7, 1             ; R7 com 1, 0, -1, valor que permite calcular as colunas
adjacentes à do centro, para desenhar a nave
    MOV   R9, MAX_LINHA     ; Contador dos bits que são desenhados por linha
ciclo_nave2:
    MOV   R4, COLUNA_NAVE   ; Variável com a coluna do centro da nave do jogador
    MOV   R2, [R4]          ; Coluna do centro da nave em registo
    ADD   R2, R7            ; Soma que permite aceder à coluna onde desenhar, a partir
da coordenada do centro
    PUSH  R1                ; Guarda registo R1
    PUSH  R2                ; Guarda registo R2
    MOV   R0, NAVE          ; Tabela que contém o formato da nave
    MOV   R1, R6            ; Linha onde desenhar, a partir do centro (para poder ser
passada como parâmetro à função Lê Pixel)
    ADD   R1, 1             ; Soma 1, porque a função Lê Pixel recebe como parâmetro um
valor entre 0 e 2
    MOV   R2, R7            ; Coluna onde desenhar, a partir do centro (para poder ser
passada como parâmetro à função Lê Pixel)
    ADD   R2, 1             ; Soma 1, porque a função Lê Pixel recebe como parâmetro um
valor entre 0 e 2
    CALLF le_pixel          ; Função Lê Pixel, que diz que bits devem ser escritos,
recebe como argumentos: R0 - Valor a escrever, R1 - Linha, R2 - Coluna
    POP   R2                ; Restaura registo R2 para poder desenhar na coluna certa
    POP   R1                ; Restaura registo R1 para poder desenhar na linha certa
    CALL  desenha           ; Função Desenha, que desenha um bit no ecrã, recebe como
argumentos: R0 - Valor a escrever, R1 - Linha, R2 - Coluna
    SUB   R7, 1             ; Avança para a linha anterior (contada a partir do centro)
    SUB   R9, 1             ; Actualiza o contador de colunas a desenhar
    JNZ   ciclo_nave2       ; Percorre todos os valores de colunas adjacentes dentro de
uma linha
    SUB   R6, 1             ; Avança para a linha anterior (contada a partir do centro)
    SUB   R8, 1             ; Actualiza o contador de linhas a desenhar
    JNZ   ciclo_nave1       ; Desenha todas as linhas de uma nave
    POP   R9                ; Restaura registos
    POP   R8
```

```
    POP   R7
    POP   R6
    POP   R5
    POP   R4
    POP   R3
    POP   R2
    POP   R1
    POP   R0
    RET                     ; Termina rotina

; *********************************************************************
; Apaga Nave
;   Rotina que apaga a nave do jogador
; Entradas
;   Nenhuma
; Saídas
;   Nenhuma
; *********************************************************************
apaga_nave:
    PUSH  R0                ; Guarda registos
    PUSH  R1
    PUSH  R2
    PUSH  R3
    PUSH  R4
    PUSH  R5
    PUSH  R6
    PUSH  R7
    PUSH  R8
    PUSH  R9
    MOV   R3, LINHA_NAVE    ; Tabela correspondente à linha do centro de cada nave alien
    MOV   R4, COLUNA_NAVE   ; Tabela correspondente à coluna do centro de cada nave
alien
    MOV   R6, 1             ; R6 com 1, 0, -1, valor que permite calcular as linhas
adjacentes à do centro, para desenhar a nave
    MOV   R8, MAX_LINHA     ; Contador dos bits que são desenhados por linha
apaga_nave1:
    MOV   R1, [R3]          ; R1 com a linha do centro do alien actual
    ADD   R1, R6            ; Soma que permite aceder à linha onde desenhar, a partir da
coordenada do centro
    MOV   R7, 1             ; R7 com 1, 0, -1, valor que permite calcular as colunas
adjacentes à do centro, para desenhar a nave
    MOV   R9, MAX_LINHA     ; Contador dos bits que são desenhados por linha
apaga_nave2:
    MOV   R2, [R4]          ; R2 com a coluna do centro do alien centro
    ADD   R2, R7            ; Soma que permite aceder à coluna onde desenhar, a partir
da coordenada do centro
    MOV   R0, 0             ; R0 a 0 para poder apagar a nave na posição actual
    CALL  desenha           ; Função Desenha, que desenha um bit no ecrã, recebe como
argumentos: R0 - Valor a escrever, R1 - Linha, R2 - Coluna
    SUB   R7, 1             ; Avança para a linha anterior (contada a partir do centro)
    SUB   R9, 1             ; Actualiza o contador de colunas a desenhar
    JNZ   apaga_nave2       ; Percorre todos os valores de colunas adjacentes dentro de
uma linha
    SUB   R6, 1             ; Avança para a linha anterior (contada a partir do centro)
    SUB   R8, 1             ; Actualiza o contador de linhas a desenhar
    JNZ   apaga_nave1       ; Desenha todas as linhas de uma nave
    POP   R9                ; Restaura registos
    POP   R8
    POP   R7
    POP   R6
    POP   R5
    POP   R4
    POP   R3
    POP   R2
    POP   R1
    POP   R0
    RET                     ; Termina rotina

; *********************************************************************
; Move Nave
;   Rotina que movimenta a nave em função da tecla lida
; Entradas
;   Nenhuma
; Saídas
;   Nenhuma
; Altera
```

```
;   LINHA_NAVE
;   COLUNA_NAVE
;   V_ENERGIA
; ***************************************************************************
move_nave:
    PUSH  R0                ; Guarda registos
    PUSH  R1
    PUSH  R2
    PUSH  R3
    PUSH  R4
    PUSH  R5
    PUSH  R6
    PUSH  R7
    PUSH  R8
    CALL  apaga_nave        ; Apaga a nave para poder desenhar na nova posição
    MOV   R0, 0             ; R0 com 0 para apagar o canhão na posição anterior antes de
mover
    CALL  canhao            ; Rotina canhão, vai apagar o canhão na posição actual para
desenhar na nova depois do movimento
    MOV   R3, TECLAS_NAVE   ; R3 com a tabela que contém o índice da direcção a seguir
para cada tecla
    MOV   R4, DIR_LINHAS    ; R4 com a tabela de direcção a seguir em termos de linhas
    MOV   R5, DIR_COLUNAS   ; R5 com a tabela de direcção a seguir em termos de colunas
    MOV   R0, BUFFER        ; R0 com a variável que guarda a tecla premida
    MOVB  R6, [R0]          ; R6 com a tecla premida
    ADD   R3, R6            ; Acede ao índice de direcção seleccionada pela tecla
premida
    MOVB  R6, [R3]          ; R6 com a o índice de direcção para a nave seguir
    ADD   R4, R6            ; Acede à diferença da linha entre a nova direcção e a
actual
    ADD   R5, R6            ; Acede à diferença de coluna entre a nova direcção e a
actual
    MOV   R7, [R4]          ; R7 com a diferença de linha entre a nova direcção e a
actual
    MOV   R8, [R5]          ; R8 com a diferença de coluna entre a nova direcção e a
actual
    MOV   R4, LINHA_NAVE    ; R4 com a variável que guarda a linha da nave
    MOV   R5, COLUNA_NAVE   ; R5 com a variável que guarda a coluna da nave
    MOV   R1, [R4]          ; R1 com a linha actual da nave
    MOV   R2, [R5]          ; R2 com a coluna actual da nave
    ADD   R1, R7            ; R1 com a nova linha da nave
    MOV   R6, 2             ; Linha limite do centro da nave = 2
    CMP   R1, R6            ; Ver se nave está nos limites
    JGE   lin_test          ; Testa a última linha
    SUB   R1, R7            ; Repor valor anterior porque senão fica fora dos limites
lin_test:
    MOV   R6, 29            ; Linha limite do centro da nave = 29
    CMP   R1, R6            ; Ver se nave está nos limites
    JLE   lin_ok            ; Verifica se a nave não pode passar as colunas limite
    SUB   R1, R7            ; Repor valor anterior porque senão fica fora dos limites
lin_ok:
    ADD   R2, R8            ; R2 com a nova coluna da nave
    MOV   R6, 2             ; Coluna limite do centro da nave = 2
    CMP   R2, R6            ; Ver se nave está nos limites
    JGE   col_test          ; Verifica se a nave não pode passar a coluna máxima limite
    SUB   R2, R8            ; Repor valor anterior porque senão fica fora dos limites
col_test:
    MOV   R6, 29            ; Coluna limite do centro da nave = 29
    CMP   R2, R6            ; Ver se nave está nos limites
    JLE   col_ok            ; Salta se a nave não pode ultrapassar as colunas
    SUB   R2, R8            ; Repor valor anterior porque senão fica fora dos limites
col_ok:
    MOV   [R4], R1          ; Guarda a linha actual na variável
    MOV   [R5], R2          ; Guarda a coluna actual na variável
    CALL  nave              ; Chama a função que desenha a nave
    MOV   R0, 1             ; R0 a 1 para desenhar o canhão
    CALL  canhao            ; Desenha o canhão na nova posição
    MOV   R0, V_ENERGIA     ; R0 com a variável que guarda o valor actual de energia
    MOV   R1, [R0]          ; R1 com o valor actual de energia
    MOV   R2, POUT1         ; R2 com o display hexadecimal
    MOV   R3, ENE_MOVE      ; R3 com o valor de energia que se gasta ao mover a nave
    SUB   R1, R3            ; Subtrai à energia actual o valor de energia gasto a mover
    MOV   [R0], R1          ; Actualiza a energia actual na variável
    MOVB  [R2], R1          ; Actualiza a energia actual no display
    POP   R8                ; Restaura registos
    POP   R7
```

```
    POP   R6
    POP   R5
    POP   R4
    POP   R3
    POP   R2
    POP   R1
    POP   R0
    RET                     ; Termina rotina

; *********************************************************************
; Canhão
;   Rotina que desenha (ou limpa) o canhão no ecrã
; Entradas
;   R0:  0 - apaga canhão; 1 - desenha canhão
; Saídas
;   Nenhuma
; *********************************************************************
canhao:
    PUSH  R0                ; Guarda registos
    PUSH  R1
    PUSH  R2
    PUSH  R3
    PUSH  R4
    PUSH  R5
    MOV   R4, LINHA_NAVE    ; R4 com a variável que contém a linha da nave
    MOV   R1, [R4]          ; R1 com a linha da nave
    MOV   R4, COLUNA_NAVE   ; R4 com a variável que contém a coluna da nave
    MOV   R2, [R4]          ; R2 com a coluna da nave
    MOV   R4, POS_CANHAO    ; R4 com a variável que contém a posição actual do canhão
    MOV   R3, [R4]          ; R3 com a posição do canhão
    MOV   R4, CANHAO_LIN    ; R4 com a tabela que devolve a diferença da linha do canhão
para a nave em cada posição do canhão
    MOV   R5, CANHAO_COL    ; R5 com a tabela que devolve a diferença da coluna do
canhão para a nave em cada posição do canhão
    ADD   R4, R3            ; Acede ao elemento da tabela de linhas correspondente à
posição actual
    ADD   R5, R3            ; Acede ao elemento da tabela de colunas correspondente à
posição actual
    MOV   R3, [R4]          ; R3 com a diferença da linha do canhão para a linha da nave
    ADD   R1, R3            ; R1 com a linha onde desenhar o canhão
    MOV   R3, [R5]          ; R3 com a diferença da coluna do canhão para a coluna da
nave
    ADD   R2, R3            ; R2 com a coluna onde desenhar o canhão
    CALL  desenha           ; Chama a função que desenha o canhão dadas as suas
coordenadas
    POP   R5                ; Recupera registos
    POP   R4
    POP   R3
    POP   R2
    POP   R1
    POP   R0
    RET                     ; Termina rotina

; *********************************************************************
; Canhao_Esq
;   Rotina que roda o canhão à esquerda
; Entradas
;   Nenhuma
; Saídas
;   Nenhuma
; *********************************************************************
canhao_esq:
    PUSH  R0                ; Guarda registos
    PUSH  R1
    PUSH  R2
    PUSH  R3
    PUSH  R4
    MOV   R0, 0             ; R0 com 0 para poder apagar o canhão na posição actual
    CALL  canhao            ; Apaga o canhão para poder desenhar na posição seguinte
    MOV   R0, POS_CANHAO    ; R0 com a variável que guarda a posição actual do canhão
    MOV   R3, [R0]          ; R3 com a posição actual do canhão
    SUB   R3, 2             ; Acede à posição seguinte
    MOV   R4, 16            ; R4 com o valor a partir do qual as posições começam a
repetir-se
    MOD   R3, R4            ; Resto da divisão inteira para garantir que as posições
batem certo quando se dá mais que uma volta
```

```
    MOV   [R0], R3          ; Actualiza a posição actual para a nova
    MOV   R0, 1             ; R0 com 1 para poder desenhão o canhão na nova posição
    CALL  canhao            ; Desenha o canhão na nova posição
    POP   R4                ; Restaura registos
    POP   R3
    POP   R2
    POP   R1
    POP   R0
    RET                     ; Termina rotina

; ********************************************************************
; Canhao_Dir
;   Rotina que roda o canhão à direita
; Entradas
;   Nenhuma
; Saídas
;   Nenhuma
; ********************************************************************
canhao_dir:
    PUSH  R0                ; Guarda registos
    PUSH  R1
    PUSH  R2
    PUSH  R3
    PUSH  R4
    MOV   R0, 0             ; R0 com 0 para poder apagar o canhão na posição actual
    CALL  canhao            ; Apaga o canhão para poder desenhar na posição seguinte
    MOV   R0, POS_CANHAO    ; R0 com a variável que guarda a posição actual do canhão
    MOV   R3, [R0]          ; R3 com a posição actual do canhão
    ADD   R3, 2             ; Acede à posição seguinte
    MOV   R4, 16            ; R4 com o valor a partir do qual as posições começam a
repetir-se
    MOD   R3, R4            ; Resto da divisão inteira para garantir que as posições
batem certo quando se dá mais que uma volta
    MOV   [R0], R3          ; Actualiza a posição actual para a nova
    MOV   R0, 1             ; R0 com 1 para poder desenhão o canhão na nova posição
    CALL  canhao            ; Desenha o canhão na nova posição
    POP   R4                ; Restaura registos
    POP   R3
    POP   R2
    POP   R1
    POP   R0
    RET                     ; Termina rotina


; ********************************************************************
; Disparo
;   Rotina que desenha o raio no ecrã
; Entradas
;   R0: 0 - limpar, 1 - escrever
; Saídas
;   Nenhuma
; ********************************************************************
disparo:
    PUSH  R0                ; Guarda registos
    PUSH  R1
    PUSH  R2
    PUSH  R3
    PUSH  R4
    PUSH  R5
    PUSH  R6
    PUSH  R7
    PUSH  R8
    PUSH  R9
    AND   R0, R0            ; Afectação de flags
    JNZ   des_disp          ; Se a flag estiver a 1 vai desenhar o raio
    MOV   R1, FLAG_DESENHA  ; Verifica se a flag que diz se o canhão deve ser apagado
está activa
    MOV   R2, [R1]          ; Valor da flag em registo
    AND   R2, R2            ; Afectação de flags
    JZ    nao_disparo       ; Se o canhão não for apagado então não se desenha nem apaga
    MOV   R2, 0             ; Coloca a flag a 0
    MOV   [R1], R2          ; Actualiza a variável que guarda a flag
des_disp:
    MOV   R9, 3             ; Contador de pixels do raio
    MOV   R4, LINHA_NAVE    ; R4 com a variável que contém a linha da nave
    MOV   R1, [R4]          ; R1 com a linha da nave
```

```
    MOV   R4, COLUNA_NAVE   ; R4 com a variável que contém a coluna da nave
    MOV   R2, [R4]          ; R2 com a coluna da nave
    MOV   R4, POS_CANHAO    ; R4 com a variável que contém a posição actual do canhão
    MOV   R3, [R4]          ; R3 com a posição do canhão
    MOV   R4, CANHAO_LIN    ; R4 com a tabela que devolve a diferença da linha do canhão
para a nave em cada posição do canhão
    MOV   R5, CANHAO_COL    ; R5 com a tabela que devolve a diferença da coluna do
canhão para a nave em cada posição do canhão
    ADD   R4, R3            ; Acede ao elemento da tabela de linhas correspondente à
posição actual
    ADD   R5, R3            ; Acede ao elemento da tabela de colunas correspondente à
posição actual
    MOV   R3, [R4]          ; R3 com a diferença da linha do canhão para a linha da nave
    ADD   R1, R3            ; R1 com a linha onde desenhar o canhão
    MOV   R3, [R5]          ; R3 com a diferença da coluna do canhão para a coluna da
nave
    ADD   R2, R3            ; R2 com a coluna onde desenhar o canhão
    MOV   R3, RAIO          ; Tabela que contém a direcção para onde o raio deve ser
escrito
    MOV   R4, POS_CANHAO    ; Variável que contém a posição do canhão
    MOV   R5, [R4]          ; Posição actual do canhão em registo
    ADD   R3, R5            ; Acede à direcção a seguir pelo raio em relação à posição
actual do canhão
    MOV   R4, DIR_LINHAS    ; Tabela que contém as direcções em termos de linhas
    MOV   R5, DIR_COLUNAS   ; Tabela que contém as direcções em termos de colunas
    MOV   R6, [R3]          ; Direcção a seguir pelo raio em registo
    ADD   R4, R6            ; Selecciona a linha a seguir pelo raio
    ADD   R5, R6            ; Selecciona a coluna a seguir pelo raio
    MOV   R7, [R4]          ; Linha a seguir pelo raio em registo
    MOV   R8, [R5]          ; Coluna a seguir pelo raio em registo
disparo1:
    ADD   R1, R7            ; Linha onde escrever o raio actualizada
    ADD   R2, R8            ; Coluna onde escrever o raio actualizada
    CALL  desenha           ; Rotina que desenha o raio no ecrã
    SUB   R9, 1             ; Avança para o pixel seguinte do raio
    JNZ   disparo1          ; Enquanto que não forem escritos 3 pixels, o raio não está
completo
    MOV   R5, LINHA_RAIO    ; Variável que guarda a linha actual da ponta do raio
    MOVB  [R5], R1          ; Actualiza a linha actual da ponta do raio
    MOV   R5, COLUNA_RAIO   ; Variável que guarda a coluna actual da ponta do raio
    MOVB  [R5], R2          ; Actualiza a coluna actual da ponta do raio
nao_disparo:
    POP   R9                ; Recupera registos
    POP   R8
    POP   R7
    POP   R6
    POP   R5
    POP   R4
    POP   R3
    POP   R2
    POP   R1
    POP   R0
    RET

; ***************************************************************************
; Limpa
;   Rotina que limpa o ecrã
; Entradas
;   Nenhuma
; Saídas
;   Nenhuma
; ***************************************************************************
limpa:
    PUSH  R0              ; Guarda registo R0
    PUSH  R1              ; Guarda registo R1
    PUSH  R2              ; Guarda registo R2
    MOV   R0, ECRA        ; R0 com o endereço do ecrã
    MOV   R1, MAX_ECRA    ; R1 com contador de bytes do ecrã
    MOV   R2, 0H          ; R2 com 0, este valor permite limpar o ecrã
ciclo_l:
    MOVB  [R0], R2        ; Coloca o byte actual do ecrã a 0
    ADD   R0, 1           ; Avança para o byte seguinte
    SUB   R1, 1           ; Actualiza o contador
    JNZ   ciclo_l         ; Enquanto que o contador for diferente de 0 tem que limpar o
ecrã
    POP   R2              ; Recupera registo R2
```

```
    POP   R1              ; Recupera registo R1
    POP   R0              ; Recupera registo R0
    RET                   ; Termina rotina


; *********************************************************************
; Teclado
;   Guarda no [BUFFER] a tecla lida
; Entradas
;   Nenhuma
; Saídas
;   R0 o código da tecla premida
; *********************************************************************
teclado:
    PUSH  R1              ; Guarda registo R1
    PUSH  R2              ; Guarda registo R2
    PUSH  R3              ; Guarda registo R3
    PUSH  R4              ; Guarda registo R4
    PUSH  R5              ; Guarda registo R5
    MOV   R1, LINHA       ; Começar o teste pela linha 4
    MOV   R2, PIN         ; Endereço do periférico de entrada
    MOV   R3, POUT2       ; Endereço do periférico de saída
    MOV   R5, 16          ; código inicial de tecla
cic_lin:
    MOVB  [R3], R1        ; Escrever no porto de saída, teste a uma linha
    MOVB  R0, [R2]        ; Ler do porto de entrada
    MOV   R4, COLUNA      ; Começar o teste pela coluna 4
cic_col:
    SUB   R5, 1           ; Actualiza o código de tecla
    TEST  R0, R4          ; Selecciona apenas os bits do porto de entrada que
correspondem ao teclado, Afecta as flags
    JNZ   tecla           ; Se houver tecla premida vai guardar
    SHR   R4, 1           ; ver se esta coluna foi selecionada
    JNZ   cic_col         ; experimentar outra coluna
    SHR   R1, 1           ; Muda para a linha seguinte
    JNZ   cic_lin         ; Verifica se há uma tecla premida na linha seguinte
    MOV   R5, T_NEUTRA    ; Coloca o valor de uma tecla neutra em registo
tecla:
    MOV   R0, T_DISPARO   ; É a tecla de disparo?
    CMP   R0, R5          ; Verifica se é a tecla de disparo
    JNZ   tecla2          ; Se for avança para processar
    MOV   R4, FLAG_DISPARO  ; Ligar o raio
    MOV   R0, 1           ; Flag de disparo activa
    MOV   [R4], R0        ; Actualiza a variável com o novo valor da flag
    MOV   R4, FLAG_DESENHA  ; Pode ser apagado
    MOV   [R4], R0        ; Actualiza a variável com o novo valor da flag
    JMP   cont_tecl       ; O disparo é sempre considerado
tecla2:
    MOV   R4, FLAG_DISPARO  ; Tecla não é disparo, logo desligar raio
    MOV   R0, 0           ; Valor da flag a 0
    MOV   [R4], R0        ; Actualiza variável com o novo valor da flag
    MOV   R4, AUX_BUFFER  ; Verificar qual a tecla que estava premida anteriormente
    MOVB  R0, [R4]        ; Valor da tecla anterior em registo
    CMP   R0, R5          ; Verifica se ainda é a tecla de disparo
    JNZ   cont_tecl       ; Tecla diferent, guardar a nova e divulgar
    MOV   R5, T_NEUTRA    ; Tecla repetida, deve ser ignorada
    JMP   fim_tecl        ; Salta para o fim da rotina
cont_tecl:
    MOV   R4, AUX_BUFFER  ; Guardar a tecla que estava realmente premida em registo
    MOVB  [R4], R5        ; Guarda a tecla que estava realmente premida em memória
fim_tecl:
    MOV   R0, R5          ; tecla selecionada (0 ... 15 ou -1)
    MOV   R4, BUFFER      ; Endereço de memória onde guardar a tecla premida
    MOVB  [R4], R0        ; Guarda tecla premida em memória
    POP   R5              ; Recupera registo R5
    POP   R4              ; Recupera registo R4
    POP   R3              ; Recupera registo R3
    POP   R2              ; Recupera registo R2
    POP   R1              ; Recupera registo R1
    RET                   ; Termina rotina


; *********************************************************************
; Desenha Pixel
;   Desenha ou apaga um pixel no ecrã dadas as suas coordenadas
; Entradas:
```

```
;   R0 - Valor a escrever (1 - desenha, 0 - apaga)
;   R1 - Linha onde escrever
;   R2 - Coluna onde escrever
; Saídas:
;   Nenhuma
; ************************************************************************
desenha:
    PUSH  R0                ; Guarda registo R0
    PUSH  R1                ; Guarda registo R1
    PUSH  R2                ; Guarda registo R2
    PUSH  R3                ; Guarda registo R3
    PUSH  R4                ; Guarda registo R4
    PUSH  R5                ; Guarda registo R5
    PUSH  R6                ; Guarda registo R6
    PUSH  R7                ; Guarda registo R7
    PUSH  R8                ; Guarda registo R8
    MOV   R3, ECRA          ; R3 com o endereço do ecrã
    MOV   R4, LIN_ECRA      ; R4 com o valor equivalente a uma linha do ecrã
    MOV   R5, 80H           ; R5 com uma máscara que permite escrever em cada coluna
    MOV   R6, 8H            ; R6 com o número de colunas por byte
    MUL   R1, R4            ; Transforma a linha dada pelo utilizador num byte do ecrã
    ADD   R3, R1            ; Acede à linha do ecrã seleccionada pelo utilizador
    MOV   R7, R2            ; Cópia da coluna para poder ser separada no byte a que diz
respeito e no bit a escrever
    DIV   R7, R6            ; Transforma a coluna seleccionada pelo utilizador no byte
do ecrã correspondente
    ADD   R3, R7            ; Acede à coluna seleccionada pelo utilizador
    MOD   R2, R6            ; R1 com o bit a escrever/apagar dentro do byte certo
ciclo_desenha:
    AND   R2, R2            ; Afectação de flags,
    JZ    fim_ciclo         ; Quando o bit a escrever for o último avança directamente
para a fase de desenho
    SHR   R5, 1             ; Avança dentro da máscara para o bit seguinte
    SUB   R2, 1             ; Avança para o bit seguinte
    JMP   ciclo_desenha     ; Verifica todos os bits
fim_ciclo:
    MOVB  R8, [R3]          ; R8 com o conteúdo do ecrã no byte actual
    AND   R0, R0            ; Afectação de flags, se R0 = 1 escreve, se R0 = 0 apaga
    JZ    apaga_pix         ; Se R0 = 0, vai apagar o bit pretendido
escreve_pix:
    OR    R8, R5            ; Escolha do bit onde escrever
    JMP   fim_desenha       ; Acaba de desenhar o bit
apaga_pix:
    NOT   R5                ; Inverso do bit para poder apagar
    AND   R8, R5            ; Apaga o bit pretendido
fim_desenha:
    MOVB  [R3], R8          ; Escreve o valor dado pelo utilizador na linha e coluna
seleccionada
    POP   R8                ; Recupera registo R8
    POP   R7                ; Recupera registo R7
    POP   R6                ; Recupera registo R6
    POP   R5                ; Recupera registo R5
    POP   R4                ; Recupera registo R4
    POP   R3                ; Recupera registo R3
    POP   R2                ; Recupera registo R2
    POP   R1                ; Recupera registo R1
    POP   R0                ; Recupera registo R0
    RET                     ; Termina rotina

; ************************************************************************
; Lê Pixel
;   Recebe uma tabela e um ponto e devolve o conteúdo do ponto em R0
;   Lê o equivalente a uma matriz 3x8 (3 linhas e 8 colunas)
; Entradas:
;   R0 - Tabela de strings
;   R1 - Linha a ler
;   R2 - Coluna a ler
; Saidas:
;   R0 - Valor do Pixel, pode ser 0 ou diferente de 0 (1,2,4,8,etc)
; ************************************************************************
le_pixel:
    PUSH  R1                ; Guarda registos
    PUSH  R2
    PUSH  R3
    MOV   R3, MASCARA       ; Máscara para seleccionar apenas um bit de cada vez
    ADD   R0, R1            ; Soma da base de desenho com a linha a ler
```

```
ciclo_ler:
    AND   R2, R2            ; Afectação de flags
    JZ    fim_ciclo_ler     ; Se o bit estiver na última posição lê directamente
    SHR   R3, 1             ; Selecciona o bit seguinte em máscara
    SUB   R2, 1             ; Avança para a coluna seguinte
    JMP   ciclo_ler         ; Lê a coluna seguinte
fim_ciclo_ler:
    MOVB  R1, [R0]          ; R1 com o primeiro elemento da tabela passada
    AND   R3, R1            ; Verifica o conteúdo do bit seleccionado por R1
    MOV   R0, R3            ; R0 com o conteúdo do bit seleccionado
    POP   R3                ; Recupera registos
    POP   R2
    POP   R1
    RETF                    ; Termina rotina

; *********************************************************************
; Inverte Ecrã
;   Inverte o ecrã, diferenciando o estado de suspensão do de jogo
; Entradas:
;   Nenhuma
; Saidas:
;   Nenhuma
; *********************************************************************
inverte_ecra:
    PUSH  R0                ; Guarda registos
    PUSH  R1
    PUSH  R2
    MOV   R0, ECRA          ; Endereço do ecrã
    MOV   R1, MAX_ECRA      ; Contador de bytes do ecrã
ciclo_inverte:
    MOVB  R2, [R0]          ; Obtem o byte actual do ecrã
    NOT   R2                ; Inverte o conteúdo do byte actual
    MOVB  [R0], R2          ; Coloca no ecrã o inverso do que estava escrito
anteriormente
    ADD   R0, 1             ; Avança para o byte seguinte
    SUB   R1, 1             ; Actualiza o contador
    JNZ   ciclo_inverte     ; Enquanto que o contador for diferente de 0 tem que
percorrer o ecrã
    POP   R2                ; Recupera registos
    POP   R1
    POP   R0
    RETF                    ; Termina rotina

; *********************************************************************
; Pausa
;   Rotina que faz uma pequena pausa.
; Entradas:
;   Nenhuma
; Saidas:
;   Nenhuma
; *********************************************************************
pausa:
    PUSH  R0                ; Guarda registos
    MOV   R0, 5000          ; R0 com um valor grande para fazer uma contagem
decrescente, apenas para fazer uma pausa entre rotinas
ciclo_pausa:
    SUB   R0, 1             ; Subtrai R0
    JNZ   ciclo_pausa       ; Enquanto não for 0 continua a subtrair
    POP   R0                ; Recupera registos
    RETF
```