INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

ARQUITETURA DE COMPUTADORES

LETI

IST-TAGUSPARK

RELATORIO

Grupo 4

João Costa 84604

Daniel Fortunato 84586

João Pereira 85250

18/05/2016

Índice

1. Introdução

Este projeto, foi realizado para a cadeira de Arquitetura de Computadores com o objetivo de podermos aplicar a matéria das teóricas em algo mais concreto. Mostramos que aprendemos a dominar a linguagem de programação Assembly, o funcionamento do PEPE (Processador Especial Para Ensino) e do seu relacionamento com a memoria, o uso de periféricos, interrupções e de rotinas.

O objetivo deste projeto era realizar um jogo, o Pacman. Para isso usamos o PixelScreen onde representamos o campo, o Pacman, fantasmas e estrelas. O jogo consiste em avançarmos com o Pacman em qualquer direção e comermos as estrelas, se conseguirmos comer as estrelas todas ganhamos o jogo. Mas temos de ter cuidado pois não podemos deixar os fantasmas chegarem até nós, pois se formos comidos perdemos o jogo.

Para a realização do projeto a nossa maior restrição era o tamanho do campo pois era apenas 32x32, o que limitou os objetos que desenhamos e a nossa criatividade. Para além disso os objetos tinham de ter características especiais para fazer com que o jogo decorresse da melhor maneira:

- **Pacman**: pode deslocar-se em qualquer direção.
- **Estrelas**: são quatro, uma em cada canto do PixelScreen, quando o Pacman passa por cima delas, devem desaparecer.
- **Fantasmas**: deslocam-se em direção ao Pacman.
- **Obstáculos**: na eventualidade de haver uma colisão com o Pacman, este não mexe, os fantasmas também não colidem, são obrigados a encontrar outro caminho.
- **Moldura**: define o campo de jogo, não deixa passar nenhum objeto.

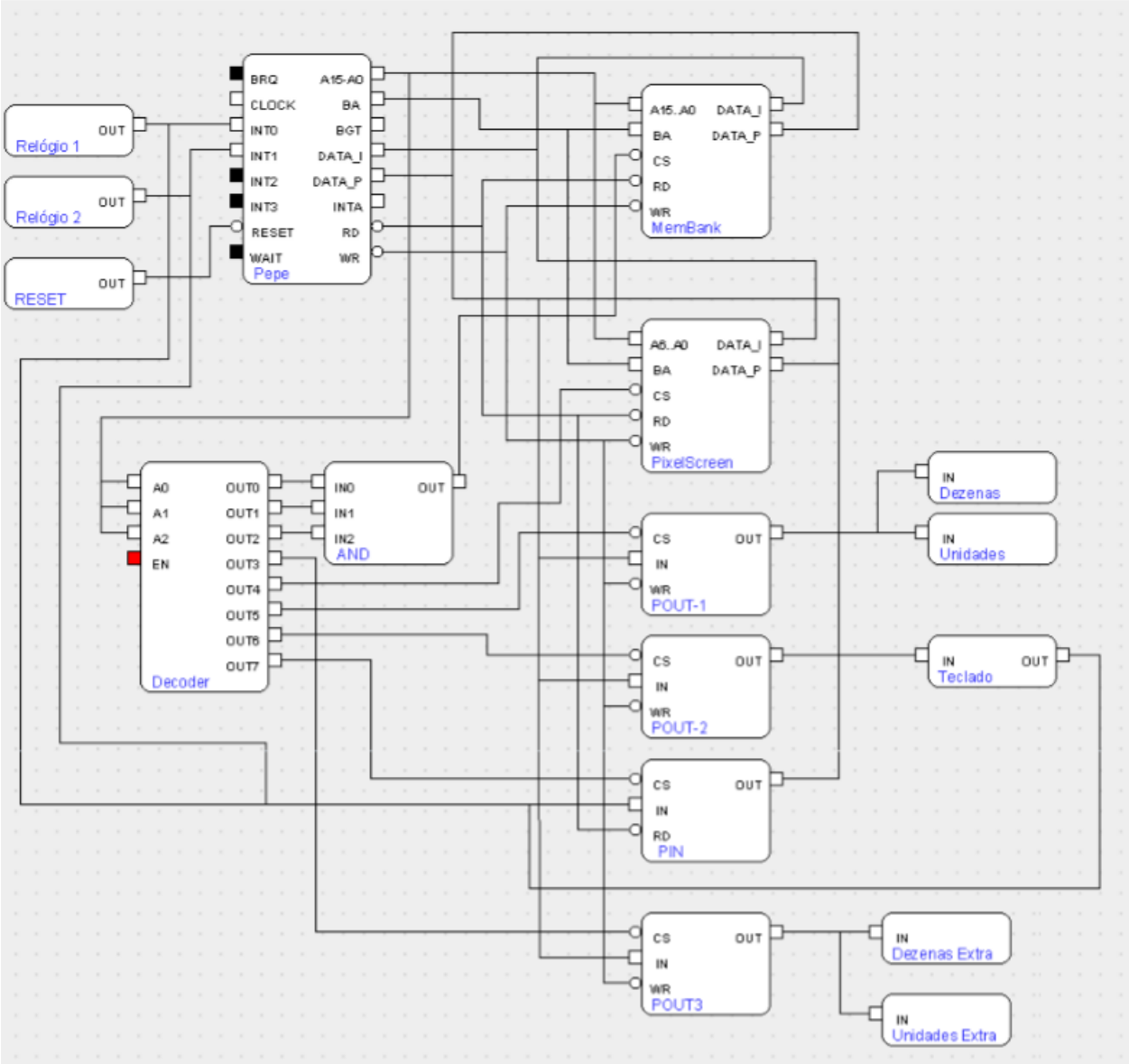
Todos estes objetos podem ser alterados, sem mudarmos o código. O que permite liberdade na configuração do jogo.

Neste relatório, poderá encontrar um resumo gráfico do projeto, do hardware e do software, na secção 2.1. Na secção 2.1.1 encontra-se o mapa de endereço utilizado para o trabalho. Na 2.1.2 descreve-se os mecanismos de comunicação entre os vários processos do trabalho enquanto que na 2.1.3 explicitamos as variáveis de estado das maquinas implementadas. A secção 2.1.4 descreve as interrupções utilizadas, a 2.1.5 descreve as rotinas graficamente. Finalmente fazendo as conclusões na secção 3, e listando o código na secção 4.

2. Conceção e Implementação

2.1. Estrutura Geral

Hardware



Relógios (1 e 2): um relógio de tempo real, para ser usado como base para a temporização da contagem dos segundos. Outro relógio de tempo real, para ser usado como base para a temporização do movimento dos Fantasmas.

PixelScreen: ecrã de 32 x 32 pixels. É acedido como se fosse uma memória de 128 bytes (4 bytes em cada linha, 32 linhas).

Teclado: Teclado, de 4 x 4 botões, com 4 bits ligados ao periférico POUT-2 e 4 bits ligados ao periférico PIN (bits 3-0). A deteção de qual botão está carregado é feita por varrimento.

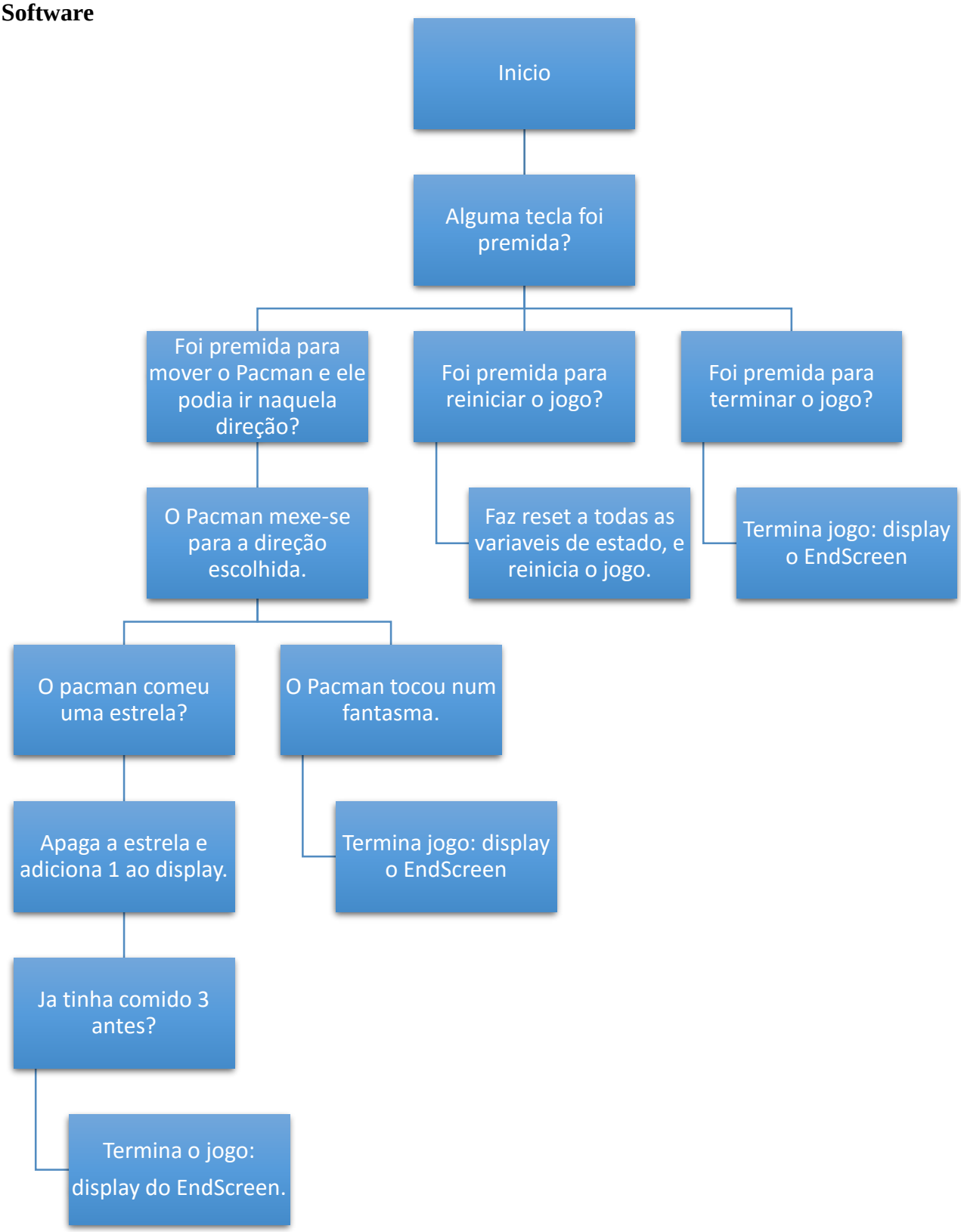
PIN: recebe a informação do teclado

POUT1: periférico de saída (8 bits) que envia o tempo que já passou desde o inicio do jogo para os displays (unidades e dezenas).

POUT2: periférico de saída (8 bits) que envia informação para o teclado.

POUT3: periférico de saída (8 bits) que envia o numero de estrelas já comidas pelo Pacman para os displays extra (unidades e dezenas).

Software


Inicio
Alguma tecla foi premida?
Foi premida para mover o Pacman e ele podia ir naquela direção?
Foi premida para reiniciar o jogo?
Foi premida para terminar o jogo?
O Pacman mexe-se para a direção escolhida.
Faz reset a todas as variaveis de estado, e reinicia o jogo.
Termina jogo: display o EndScreen
O pacman comeu uma estrela?
O Pacman tocou num fantasma.
Apaga a estrela e adiciona 1 ao display.
Termina jogo: display o EndScreen
Ja tinha comido 3 antes?
Termina o jogo: display do EndScreen.

2.1.1. Mapa de endereçamento escolhido

Para a realização deste projeto foi-nos proposta a utilização do seguinte mapa de endereços:

Dispositivo	Endereços
RAM (MemBank)	0000H a 5FFFH
POUT-3 (periférico de saída de 8 bits)	06000H
PixelScreen	8000H a 807FH
POUT-1 (periférico de saída de 8 bits)	0A000H
POUT-2 (periférico de saída de 8 bits)	0C000H
PIN (periférico de entrada de 8 bits)	0E000H

2.1.2. Comunicação entre processos

Neste trabalho, para a comunicação entre os vários processos usamos variáveis globais, registos e rotinas. Isto facilita bastante a implementação, pois permite uma abstração no trabalho o que acelera bastante o programa.

2.1.3. Variáveis de Estado

Usamos algumas variáveis de estado neste trabalho para simplificar o código:

- **keyboard_flag**: variável que indica se uma nova tecla foi premida.
- **pressed_key**: variável que indica a tecla premida.
- **key_processed**: variável que indica se uma tecla já foi processada, ou não.
- **JA_FOSTE_STAR**: variável que indica qual das estrelas acabou de ser comida.
- **STARS_EATEN**: variável que indica o numero de estrelas comidas.
- **STARS**: variável que indica quais estrelas já foram comidas.
- **ghost_move**: variável que, quando a 1, incita o movimento dos fantasmas.
- **time_flag**: variável que, quando a 1, anima.
- **time**: variável que guarda os segundos.
- **time_out**: variável que fica a 1 quando o tempo chega ao fim.
- **GHOST_NUMBERS**: variável que dá o numero de fantasmas.

- **JA_FOSTE**: variável que fica 1 quando o fantasma toca no pacman e 0 caso contrario.
- **ADDR_PACMAN**: variável com a posição do Pacman.
- **GHOST_ADDR_POS1/2/3/4:** variáveis com a posição dos fantasmas.

2.1.4. Interrupções

O código tem duas interrupções:

- **int_0**: anima o contador dos segundos atendo ao valor da flag.
- **int**_1: anima o movimento do fantasma atendendo ao valor da flag.

2.1.5. Rotinas

Maioritariamente no trabalho usamos rotinas, chamadas com a função CALL, muitas vezes melhor que usar a função JMP pois é mais rápido. Seguem as rotinas usadas, e o seu papel no projeto:

- **restart:** rotina que inicializa o jogo.
- **keyboard:** rotina que deteta a tecla acionada no teclado.
- **choose_action:** rotina que associa a cada tecla a respetiva accao no contexto do jogo.
- **move_ghosts:** rotina responsável pelo movimento dos fantasmas.
- **star_gone**: rotina que verifica se uma estrela foi comida e incrementa o display de 7 segmentos.
- **temporizador:** rotina que conta os segundos e os exibe no display de 7 segmentos.
- **time_over**: rotina que acaba o jogo quando o tempo chega a 99.
- **the_end**: rotina que processa o fim de jogo: ganho ou perdido.
- **keyboard_processed**: rotina que marca a tecla como processada.
- **keyboard_get_key**: rotina que retorna a tecla carregada atualmente ou -1 se não houver tecla carregada.
- **move_pacmanzao**: rotina que escolhe a direção na qual o Pacman se move.
- **secret_passage**: rotina que faz com que o Pacman se teletransporte de um lado para o outro do campo (cima para baixo, esquerda para a direita e vice-versa).

- **Compare_limits**: rotina que compara se uma imagem toca nos limites do display.
- **draw_pacmanzao**: rotina que desenha o Pacman.
- **touch_stars**: rotina que verifica se alguma estrela foi comida no momento, e identifica-a.
- **draw_imagem**: rotina que desenha uma imagem.
- **clear_display**: rotina que apaga todos os pixels do PixelScreen.
- **draw_moldura**: rotina que desenha o campo do jogo.
- **draw_star**: rotina que desenha uma estrela.
- **init_ghosts**: rotina que inicializa os fantasmas.
- **draw_pixel**: rotina escreve um pixel no display.
- **cmp_pos**: rotina que compara se um pixel toca nos limites do display.
- **mov_init_casper**: rotina que inicializa o movimento do fantasma.
- **choose_diretion**: rotina que decide qual direção o fantasma deve tomar (caminho mais curto até ao Pacman).
- **another_diretion**: rotina que escolhe outra direção no caso da direção escolhida inicialmente não se conseguir realizar.
- **draw_casper**: rotina que desenha um fantasma.
- **touch_pacman**: rotina que verifica se o fantasma tocou ou não no Pacman.
- **move_casper**: rotina que mexe o fantasma.
- **inverso**: rotina que quando se está a escolher a próxima direção do fantasma, verifica se duas direções são opostas.

3. Conclusões

Este projeto exigiu bastante trabalho por parte do grupo. Tivemos de aprender a dominar Assembly, o PEPE, e periféricos ao longo do semestre, por vezes com dificuldades, mas que sempre conseguimos ultrapassar.

Acabámos o semestre a dominar relativamente bem o que nos foi ensinado, e a conseguir fazer o que nos exigiram. Podendo dizer que acabámos o trabalho que nos era pedido na sua totalidade. Ao que nos pediam no enunciado ainda adicionamos certas funcionalidades ao jogo. Pusemos um Ending Screen no final do jogo, um se o jogador perdeu, e outro se ganhou. A moldura original para o campo também foi um pouco alterada pois acrescentamos-lhe uns obstáculos, e passagens secretas para adicionar mais

opções ao jogo e mais maneiras de o jogar. Ao display extra decidimos pô-lo a mostrar o número de estrelas comidas pelo Pacman, quando este chega a 4, o jogo acaba e o jogador ganhou. Finalmente decidimos acabar o jogo quando este já está a decorrer à 99 segundos (o jogador perde).

O jogo decorre normalmente sem grandes erros e faz tudo o que é suposto, parece nos que está rápido o suficiente para poder ser jogado sem nenhum problema.

Podíamos, no entanto, melhorar a evolução dos fantasmas em relação ao Pacman e melhor a performance do programa quando existe vários fantasmas em campo.

4. Código Assembly

```
;*********************************************************************************************

;*  Grupo 4:                                                                                 *

;*      Joao Costa        84604
*

;*      Joao Pereira      85250
          *

;*      Daniel Fortunato  84586
          *

;*                                                                                           *

;*********************************************************************************************



;*********************************************************************************************
****************

;***********************************      Definicao      de      constantes      -      I
****************************************

;*********************************************************************************************
****************



PIN_ADDR       EQU 0E000H                                          ;     Periferico     de
entrada de 8 bits

POUT1_ADDR      EQU 0A000H                                         ; Periferico de saida
de 8 bits

POUT2_ADDR      EQU 0C000H                                         ; Periferico de saida
de 8 bits

POUT3_ADDR      EQU 06000H                                         ; Periferico de saida
de 8 bits



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

```
O               EQU 1                                                   ; constante para os
desenhos das mascaras

_               EQU 0                                                   ; constante para os
desenhos das mascaras



;*******************************************************************************************
****************

;*********************************      Definicao      de      Objetos      -      Mask
*****************************************

;*******************************************************************************************
****************


PLACE 2000H


;Tabela de Mascaras:


bitmask:        STRING  80H
                 STRING         40H
                 STRING         20H
                 STRING         10H
                 STRING         08H
                 STRING         04H
                 STRING         02H
                 STRING         01H


the_end_mask:   STRING  _,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,_,O,O,_,_,_,_,_,_,_,_,O,O,_,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,_,O,O,_,_,_,_,_,_,_,_,O,O,_,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,_,_,O,O,_,_,_,_,O,O,_,_,_,_,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,_,_,O,O,_,_,_,_,O,O,_,_,_,_,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,_,O,O,O,O,O,O,O,O,O,O,O,O,O,O,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,_,O,O,O,O,O,O,O,O,O,O,O,O,O,O,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,O,O,O,O,_,_,O,O,O,O,_,_,O,O,O,O,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,O,O,O,O,_,_,O,O,O,O,_,_,O,O,O,O,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,_,_,_,_,_,_
```

```
                STRING  _,_,_,_,_,_,O,O,_,_,O,O,O,O,O,O,O,O,O,O,O,_,_,O,O,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,O,O,_,_,O,O,O,O,O,O,O,O,O,O,O,_,_,O,O,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,O,O,_,_,O,O,_,_,_,_,_,_,_,_,O,O,_,_,O,O,_,_,_,_,_
                STRING  _,_,_,_,_,_,O,O,_,_,O,O,_,_,_,_,_,_,_,_,O,O,_,_,O,O,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,_,_,O,O,O,_,_,O,O,O,_,_,_,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,_,_,O,O,O,_,_,O,O,O,_,_,_,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,O,O,_,_,_,_,_,O,_,_,_,_,O,_,_,_,O,_,O,O,O,O,_,_,_,_
                STRING  _,_,_,_,O,_,_,O,_,_,_,O,O,O,_,_,_,O,O,_,O,O,_,O,_,_,_,_,_,_,_
                STRING  _,_,_,_,O,_,_,_,_,_,_,O,_,O,_,_,_,O,_,O,_,O,_,O,O,O,_,_,_,_,_
                STRING  _,_,_,_,O,_,O,O,_,_,O,O,O,O,O,_,_,O,_,_,_,O,_,O,_,_,_,_,_,_,_
                STRING  _,_,_,_,O,_,_,O,_,_,O,_,_,_,O,_,_,O,_,_,_,O,_,O,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,O,O,_,_,O,O,_,_,_,O,O,_,O,O,O,O,O,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,O,O,_,_,O,O,_,_,_,O,O,_,O,O,O,O,_,O,O,O,_,_,_,_,_
                STRING  _,_,_,_,_,O,_,_,O,_,_,O,_,_,_,O,_,_,O,_,_,_,O,_,_,O,_,_,_,_,_
                STRING  _,_,_,_,_,O,_,_,O,_,_,O,O,_,O,O,_,_,O,O,O,_,_,O,O,O,_,_,_,_,_
                STRING  _,_,_,_,_,O,_,_,O,_,_,O,_,O,_,_,_,O,_,_,_,O,_,_,O,_,_,_,_,_,_
                STRING  _,_,_,_,_,O,_,_,O,_,_,O,_,O,_,_,_,O,_,_,_,O,_,O,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,O,_,_,O,_,_,O,O,O,_,_,_,O,_,_,_,O,_,_,O,_,_,_,_,_,_
                STRING  _,_,_,_,_,O,_,_,O,_,_,_,O,_,_,_,_,O,_,_,_,O,_,O,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,O,O,_,_,_,_,_,O,_,_,_,_,O,O,O,_,O,_,_,O,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_

winner_mask:    STRING  _,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,O,_,_,_,_,_,_,_,O,O,O,O,O,O,O,O,O,_,_,_,_,_,_,_,O,_,_,_
                STRING  _,_,O,O,O,_,_,_,_,_,_,O,O,_,_,_,_,_,_,_,O,O,_,_,_,_,_,O,O,O,_,_
                STRING  _,_,_,O,_,_,_,_,_,_,O,O,_,_,O,_,_,_,_,_,_,O,O,_,_,_,_,_,O,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,_,O,_,_,O,O,O,_,_,_,_,O,O,O,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,O,O,_,_,_,O,_,_,O,O,O,O,O,_,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,O,_,_,_,_,_,O,O,O,_,_,_,_,O,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,O,_,_,_,_,_,O,_,_,_,_,_,_,O,O,O,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,O,_,_,_,_,_,O,O,O,_,_,_,_,_,O,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,O,O,O,_,_,_,O,_,_,O,O,O,O,O,_,_,_,_,_,_,_,_,_
```

```
                STRING  _,_,_,_,_,_,_,_,_,_,O,_,_,_,_,_,_,_,_,O,O,O,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,_,O,O,_,_,_,_,_,_,_,_,_,_,O,_,_,_,_,_,_,_,_
                STRING  _,_,_,O,_,_,_,_,_,_,O,O,O,_,_,_,_,_,O,O,O,_,_,_,_,_,_,O,_,_,_
                STRING  _,_,O,O,O,_,_,_,_,_,_,_,O,O,O,O,O,O,O,_,_,_,_,_,_,_,O,O,O,_,_
                STRING  _,_,_,O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,O,_,_,_,O,_,_,_,_,O,O,O,_,_,_,_,O,_,_,_,O,_,_,_,_
                STRING  _,_,_,_,_,_,_,O,_,O,_,_,_,_,O,_,_,_,O,_,_,_,O,_,_,_,O,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,O,_,_,_,_,_,O,_,_,_,O,_,_,_,O,_,_,_,O,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,O,_,_,_,_,_,O,_,_,_,O,_,_,_,O,_,_,_,O,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,O,_,_,_,_,_,O,_,_,_,O,_,_,_,O,_,_,_,O,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,O,_,_,_,_,_,O,O,O,_,_,_,_,O,O,O,_,_,_,_,_,_,_
                STRING  _,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_
                STRING  _,_,_,_,O,_,_,_,O,_,_,_,O,_,_,O,O,O,_,_,O,_,_,_,_,O,_,O,_,O,_,_
                STRING  _,_,_,_,O,_,_,O,_,O,_,_,O,_,O,_,_,_,O,_,O,O,_,_,_,O,_,O,_,O,_,_
                STRING  _,_,_,_,_,O,_,O,_,O,_,O,_,_,O,_,O,_,_,_,O,_,O,_,O,_,_,O,_,O,_,_
                STRING  _,_,_,_,_,O,_,O,_,O,_,O,_,_,O,_,O,_,_,_,O,_,O,_,O,_,_,O,_,O,_,_
                STRING  _,_,_,_,_,_,O,_,_,_,O,_,_,_,O,_,O,_,_,_,O,O,_,_,_,_,_,O,_,_,_
                STRING  _,_,_,_,_,O,_,_,_,O,_,_,_,O,_,O,_,_,_,O,_,O,_,_,_,O,_,O,_,O,_,_
                STRING  _,_,_,_,_,O,_,_,_,O,_,_,_,O,_,O,_,_,_,O,_,O,_,_,_,O,_,O,_,O,_,_

moldura_mask:   STRING  O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,_,_,_,O,O,O,O,O,O,O,O,O,O,O,O,O
                STRING  O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
                STRING  O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
                STRING  O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
                STRING  O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
                STRING  O,_,_,_,_,O,O,_,_,_,_,_,_,O,O,O,O,O,O,O,_,_,_,_,_,O,O,_,_,_,_,O
                STRING  O,_,_,_,_,O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O,_,_,_,_,O
                STRING  O,_,_,_,_,O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O,_,_,_,_,O
                STRING  O,_,_,_,_,O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O,_,_,_,_,O
                STRING  O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
                STRING  O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
                STRING  O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
                STRING  O,_,_,_,_,_,_,_,_,_,_,_,_,O,O,_,_,_,O,O,_,_,_,_,_,_,_,_,_,_,O
```

```
                STRING O,_,_,_,_,_,_,_,_,_,_,_,_,O,_,_,_,_,_,O,_,_,_,_,_,_,_,_,_,_,_,O
                STRING O,_,_,_,_,_,_,_,_,_,_,_,_,O,_,_,_,_,_,O,_,_,_,_,_,_,_,_,_,_,_,O
                STRING _,_,_,_,_,O,_,_,_,_,_,_,_,O,_,_,_,_,_,O,_,_,_,_,_,_,O,_,_,_,_,_
                STRING _,_,_,_,_,O,_,_,_,_,_,_,_,O,_,_,_,_,_,O,_,_,_,_,_,_,O,_,_,_,_,_
                STRING _,_,_,_,_,O,_,_,_,_,_,_,_,O,_,_,_,_,_,O,_,_,_,_,_,_,O,_,_,_,_,_
                STRING O,_,_,_,_,_,_,_,_,_,_,_,_,O,O,O,O,O,O,O,_,_,_,_,_,_,_,_,_,_,_,O
                STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
                STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
                STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
                STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
                STRING O,_,_,_,_,O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O,_,_,_,_,O
                STRING O,_,_,_,_,O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O,_,_,_,_,O
                STRING O,_,_,_,_,O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O,_,_,_,_,O
                STRING O,_,_,_,_,O,O,_,_,_,_,_,_,O,O,O,O,O,O,O,_,_,_,_,_,O,O,_,_,_,_,O
                STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
                STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
                STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
                STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
                STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
                STRING  O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,_,_,_,O,O,O,O,O,O,O,O,O,O,O,O,O,O


pacmanzao_mask:         STRING  O,O,O
                STRING O,_,_
                STRING O,O,O


casper_mask:    STRING O,_,O
                STRING _,O,_
                STRING O,_,O


star_mask:      STRING _,O,_
                STRING O,O,O
                STRING _,O,_


limits_mask:    STRING  O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O
```

```
STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
STRING O,_,_,_,_,O,O,_,_,_,_,_,_,O,O,O,O,O,O,O,_,_,_,_,_,O,O,_,_,_,_,O
STRING O,_,_,_,_,O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O,_,_,_,_,O
STRING O,_,_,_,_,O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O,_,_,_,_,O
STRING O,_,_,_,_,O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O,_,_,_,_,O
STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
STRING O,_,_,_,_,_,_,_,_,_,_,_,_,O,O,O,O,O,O,O,_,_,_,_,_,_,_,_,_,_,_,O
STRING O,_,_,_,_,_,_,_,_,_,_,_,_,O,_,_,_,_,_,O,_,_,_,_,_,_,_,_,_,_,_,O
STRING O,_,_,_,_,_,_,_,_,_,_,_,_,O,_,_,_,_,_,O,_,_,_,_,_,_,_,_,_,_,_,O
STRING O,_,_,_,_,O,_,_,_,_,_,_,_,O,_,_,_,_,_,O,_,_,_,_,_,_,O,_,_,_,_,O
STRING O,_,_,_,_,O,_,_,_,_,_,_,_,O,_,_,_,_,_,O,_,_,_,_,_,_,O,_,_,_,_,O
STRING O,_,_,_,_,O,_,_,_,_,_,_,_,O,_,_,_,_,_,O,_,_,_,_,_,_,O,_,_,_,_,O
STRING O,_,_,_,_,_,_,_,_,_,_,_,_,O,O,O,O,O,O,O,_,_,_,_,_,_,_,_,_,_,_,O
STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
STRING O,_,_,_,_,O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O,_,_,_,_,O
STRING O,_,_,_,_,O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O,_,_,_,_,O
STRING O,_,_,_,_,O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O,_,_,_,_,O
STRING O,_,_,_,_,O,O,_,_,_,_,_,_,O,O,O,O,O,O,O,_,_,_,_,_,O,O,_,_,_,_,O
STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
STRING O,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,O
STRING O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O,O


;**********************************************

;************ Constantes - Display ************
```

```
;*********************************************


DISPLAY_ADDR          EQU 8000H                                ;  endereco  do  porto
dos displays hexadecimais

DISPLAY_END_ADDR      EQU 8080H

DISPLAY_BYTES_LIN     EQU 4

DISPLAY_SIZE_L        EQU 31

DISPLAY_SIZE_C        EQU 31

DISPLAY_SIZE          EQU 32

CONST1                EQU 8                                    ; valor para a formula



time_flag:            WORD   0                                 ; variavel que, quando
a 1, anima

time:                 STRING 0                                 ; variavel que guarda
os segundos

time_out:             STRING 0                                 ; variavel que fica a
1 quando o tempo chega ao fim



;*********************************************

;************ Constantes - PACMAN *************

;*********************************************


PACMAN_SIZE_L         EQU 2                                    ; 0 a 2

PACMAN_SIZE_C         EQU 2                                    ; 0 a 2



JA_FOSTE:             WORD 0                                   ; variavel que fica 1
quando o fantasma toca no pacman e 0 caso contrario



ADDR_POS_OLD:         WORD 27                                  ; endereco da posicao
do pacman

                      WORD 15



ADDR_POS_START:               WORD 27                                        ;  endereco  da
posicao inicial do pacman

                      WORD 15

;*********************************************

;************ Constantes - GHOSTS *************

;*********************************************
```

```
GHOST_SIZE_L           EQU 2

GHOST_SIZE_C           EQU 2

GHOST_SIZE             EQU 3

LINE_MIN               EQU 13

LINE_MAX               EQU 9

COLUNE                 EQU 15


GHOST_NUMBERS:         STRING 1                                ; numero de fantasmas


ghost_move:            STRING 0                                ; variavel que, quando
a 1, incita o movimento dos fantasmas


GHOST_ADDR_INIT:       WORD 13                                 ; endereco da posicao
inicial do fantasma

                       WORD 15


GHOST_ADDR_POS1:       WORD 13                                 ; endereco da posicao
inicial do fantasma 1

                       WORD 15


GHOST_ADDR_POS2:       WORD 13                                 ; endereco da posicao
inicial do fantasma 2

                       WORD 15


GHOST_ADDR_POS3:       WORD 13                                 ; endereco da posicao
inicial do fantasma 3

                       WORD 15


GHOST_ADDR_POS4:       WORD 13                                 ; endereco da posicao
inicial do fantasma 4

                       WORD 15


;*********************************************

;************ Constantes - STARS **************

;*********************************************
```

```
STAR_SIZE_L           EQU 2                                   ; 0 a 2

STAR_SIZE_C           EQU 2                                   ; 0 a 2


STAR_ADDR_POS1:              WORD 1                                   ; endereco da
posicao inicial da estrela 1

                      WORD 2


STAR_ADDR_POS2:              WORD 1                                   ; endereco da
posicao inicial da estrela 2

                      WORD 27


STAR_ADDR_POS3:              WORD 28                                  ; endereco da
posicao inicial da estrela 3

                      WORD 2


STAR_ADDR_POS4:              WORD 28                                  ; endereco da
posicao inicial da estrela 4

                      WORD 27


JA_FOSTE_STAR:        STRING 0                                ; variavel que indica
qual das estrelas acabou de ser comida


STARS_EATEN:          STRING 0                                ; variavel que indica
o numero de estrelas comidas


STARS:                STRING 0,0,0,0                          ; variavel que indica
quais estrelas ja foram comidas


;*********************************************

;************ Constantes - Teclado ************

;*********************************************


LINHA                 EQU 8                                   ;  posicao  do  bit
correspondente a linha a testar

CONST8                EQU 4                                   ;  constante  para  a
formula do teclado

CONST9                EQU 4                                   ; numero de linhas que
existe no teclado
```

```
keyboard_flag:         STRING 0                                    ; variavel que indica
se uma nova tecla foi premida

pressed_key:           STRING 0                                    ; variavel que indica
a tecla premida

key_processed:         STRING 0                                    ; variavel que indica
se uma tecla ja foi processada, ou nao



;*********************************************

;************ CASOS DO TECLADO ****************

;*********************************************


                       ;TECLA 0

KEYBOARD_KEYS:         WORD    -1

                       WORD    -1


                       ;TECLA 1

                       WORD    -1

                       WORD     0


                       ;TECLA 2

                       WORD    -1

                       WORD     0100H


                       ;TECLA 3

                       WORD    0

                       WORD    0


                       ;TECLA 4

                       WORD     0

                       WORD    -1


                       ;TECLA 5 -> INICIA O JOGO

                       WORD    0A00H

                       WORD    0A00H


                       ;TECLA 6
```

```
WORD    0
WORD    0100H

;TECLA 7
WORD    0
WORD    0

;TECLA 8
WORD    0100H
WORD    -1

;TECLA 9
WORD    0100H
WORD    0

;TECLA A
WORD    0100H
WORD    0100H

;TECLA B
WORD    0
WORD    0

;TECLA C
WORD    0
WORD    0

;TECLA D
WORD    0
WORD    0

;TECLA E
WORD    0
WORD    0
```

```
                        ;TECLA F -> TERMINA O JOGO
                        WORD    0D00H
                        WORD    0D00H

;*********************************************
;********* Corpo principal da Funcao **********
;*********************************************

tab:                    WORD  int_0
                        WORD  int_1

pilha:                  TABLE 300H
SP_inicial:

PLACE  0

                        ;inicializacoes do programa

inicio:                 MOV SP, SP_inicial                      ;   inicializacao   do
stack pointer
                        MOV BTE, tab                            ;   inicializacao   da
tabela de excepcoes
                        EI
                        EI0
                        EI1
                        CALL restart                            ;      rotina      que
inicializa o jogo
movimento:              CALL keyboard                           ; rotina que deteta a
tecla accionada no teclado
                        CALL choose_action                      ; rotina que associa a
cada tecla a respetiva accao no contexto do jogo
                        CALL move_ghosts                        ;  rotina  responsavel
pelo movimento dos fantasmas
                        CALL star_gone                          ; rotina que verifica
se uma estrela foi comida e incrementa o display de 7 segmentos
                        CALL temporizador                       ; rotina que conta os
segundos e os exibe no display de 7 segmentos
```

```
                    CALL time_over                          ; rotina que acaba o
jogo quando o tempo chega a 99

                    CALL the_end                            ; rotina que processa
o fim de jogo: ganho ou perdido

                    CALL keyboard_processed                 ; rotina que
marca a tecla como processada

                    JMP  movimento




; * * * * * * * * * * * Choose_action * * * * * * * * * * * * * * * * * * * * * * * *

; *

; * Descricao: Escolhe que comando a tecla faz.

; *

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


choose_action:      PUSH R0

                    PUSH R1

                    PUSH R2

                    PUSH R7

                    PUSH R8


                    CALL keyboard_get_key

                    CMP  R2, -1

                    JZ   skip_action


                    MOV  R3, 5                              ; se escolher tecla 5
(restart)

                    CMP  R2, R3

                    JZ   reset


                    MOV  R3, 0FH                            ; se escolher tecla f
(the_end)

                    CMP  R2, R3

                    JZ   end_game


                    MOV  R3, 0CH                            ; se escolher tecla c
(mais fantasmas)
```

```
                        CMP  R2, R3
                        JZ   add_ghost
                        CALL move_pacmanzao
                        JMP  skip_action

reset:                  CALL restart
                        JMP  skip_action

end_game:               MOV  R8, JA_FOSTE
                        MOV  R7, 1
                        MOV  [R8], R7                   ;  coloca  a  1  a
variavel que termina o jogo
                        CALL the_end
                        JMP  skip_action

add_ghost:              MOV  R8, GHOST_NUMBERS
                        MOVB R7, [R8]                   ;  guarda  em  R7  o
numero de fantasmas que existem
                        INC  R7                         ;
                        MOVB [R8], R7                   ;  adiciona  mais  um
fantasma ao jogo (no maximo 4)

skip_action:            POP  R8
                        POP  R7
                        POP  R2
                        POP  R1
                        POP  R0
                        RET

; * * * * * * * * * * * move_pacmanzao * * * * * * * * * * * * * * * * * * * * * * *
; *
; * Descricao: O pacman desloca em uma das 8 direcoes
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

move_pacmanzao:             PUSH R5
```

```
                        PUSH R4
                        PUSH R3
                        PUSH R2
                        PUSH R1
                        PUSH R0

                        MOV  R5, 00FFH                  ;  mascara  para  o
primeiro byte
                        MOV  R0, KEYBOARD_KEYS
                        SHL  R2, 2
                        ADD  R2, R0                     ; determina qual foi a
direcao escolhida
                        MOVB R0, [R2]                   ; guarda em R0 o valor
da direcao da linha
                        ADD  R2, 2                      ;  avanca  para  o
endereco seguinte
                        MOVB R1, [R2]                   ; guarda em R1 o valor
da direcao da coluna
                        CMP  R0, 0
                        JNZ  continua
                        CMP  R1, 0
                        JZ   dont_move                  ; se as direcoes for
igual a zero, nao se move
continua:               MOV  R3, ADDR_POS_OLD           ;  guarda  em  R3  o
endereco da posicao do pacman
                        MOVB R4, [R3]                   ;  guarda  em  R4  a
posicao da linha do pacman
                        ADD  R0, R4                     ; adiciona a direcao
da linha a posicao da linha
                        AND  R0, R5                     ; faz uma mascara ao
byte da direita
                        ADD  R3, 2                      ;  avanca  para  o
endereco seguinte
                        MOVB R4, [R3]                   ;  guarda  em  R4  a
posicao da coluna do pacman
                        ADD  R1, R4                     ; adiciona a direcao
da coluna a posicao da coluna
                        AND  R1, R5                     ; faz uma mascara ao
byte da direita

                        CALL secret_passage             ; rotina que verifica
se o pacman vai teletransportar se ou nao
```

```
                    MOV  R4, 1

                    CMP  R3, R4                         ;        se        for
teletransportar o R3 tem de ser igual a 1

                    JZ   move_on                        ; caso o R3 = 1, salta

                    MOV  R3, pacmanzao_mask                     ; guarda em R3
a mascara(imagem) do pacman

                    MOV  R4, PACMAN_SIZE_C              ;  guarda  em  R4  o
comprimento do pacman

                    MOV  R5, PACMAN_SIZE_L              ;  guarda  em  R5  a
largura do pacman

                    CALL Compare_limits                 ; verifica se o pacman
encontrou um obstaculo

                    CMP  R10, 1                         ;  se  encontrar  um
obstaculo R10 = 1

                    JZ   dont_move                      ; caso R10 = 1, salta


move_on:            MOV  R2, 0                          ; R2  =  0,  logo  vai
apagar

                    CALL draw_pacmanzao                 ;     rotina     que
desenha/apaga o pacman, neste caso apaga

                    MOV  R2, ADDR_POS_OLD               ;  guarda  em  R2  o
endereco da posicao do pacman

                    MOVB [R2], R0                       ;  guarda  em  R0  a
posicao da linha do pacman

                    ADD  R2, 2                          ;   avanca   para   o
endereco seguinte

                    MOVB [R2], R1                       ;  guarda  em  R1  a
posicao da coluna do pacman

                    MOV  R2, 1                          ; R2  =  1,  logo  vai
desenhar

                    CALL draw_pacmanzao                 ;     rotina     que
desenha/apaga o pacman, neste caso desenha

                    CALL touch_stars                    ; rotina que verifica
se o pacman tocou em alguma estrela e qual.


dont_move:          POP  R0

                    POP  R1

                    POP  R2

                    POP  R3

                    POP  R4

                    POP  R5

                    RET
```

```
;################################    INTERRUPCOES       ################################

;################################_____________________###############################
####


; * * * * * * * * * * * * int_0 * * * * * * * * * * * * * * * * * * * * * * * * * * * *

; *

; * Descricao: Anima o contador dos segundos atendo ao valor da flag.

; *

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


int_0:                  PUSH R1

                        PUSH R2


                        MOV  R1, 1

                        MOV  R2, time_flag

                        MOV  [R2], R1                               ; anima o contador dos
segundos


                        POP  R2

                        POP  R1

                        RFE


; * * * * * * * * * * * * int_1 * * * * * * * * * * * * * * * * * * * * * * * * * * * *

; *

; * Descricao: Anima o movimento do fantasma atendendo ao valor da flag.

; *

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


int_1:                  PUSH R1

                        PUSH R2


                        MOV  R1, 1

                        MOV  R2, ghost_move
```

```
                        MOVB [R2], R1                           ; anima o movimento do
fantasma


                        POP  R2

                        POP  R1

                        RFE


; * * * * * * * * * * * * draw_moldura* * * * * * * * * * * * * * * * * * * * * * * * * * *

; *

; * Descricao: Desenha os limites do Display.

; *

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


draw_moldura:           PUSH R2

                        PUSH R1

                        PUSH R0

                        PUSH R3

                        PUSH R4

                        PUSH R5


                        MOV  R2, 1                              ; inicializa o registo
a 1, valor que indica a accao de escrita de um pixel

                        MOV  R1, 0                              ; inicializa o registo
na coluna 0

                        MOV  R0, 0                              ; inicializa o registo
na linha 0

                        MOV  R3, moldura_mask                   ; inicializa o registo
no primeiro endereco da mascara da moldura

                        MOV  R4, DISPLAY_SIZE_C                         ; inicializa o
registo com o valor maximo das colunas

                        MOV  R5, DISPLAY_SIZE_L                         ; inicializa o
registo com o valor maximo das linhas

                        CALL draw_imagem                        ; chama a rotina que
vai escrever a moldura e a caixa no PixelScreen


                        POP  R5

                        POP  R4

                        POP  R3
```

```
                    POP  R0
                    POP  R1
                    POP  R2
                    RET

; * * * * * * * * * * * * draw_pacmanzao * * * * * * * * * * * * * * * * * * * * * * * * *
; *
; * Descricao: Desenha o pacman
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

draw_pacmanzao:          PUSH R0                                    ;R2  =  0  ->
apaga pacman
                    PUSH R1                                    ;R2  =  1  ->  escreve
pacman
                    PUSH R3
                    PUSH R4
                    PUSH R5

                    MOV  R1, ADDR_POS_OLD                      ; inicializa o registo
com o endereco da posicao do pacman antes de se mover
                    MOVB R0, [R1]                              ; guarda a linha do
pacman
                    ADD  R1, 2                                 ;  avanca  para  o
endereco seguinte
                    MOVB R1, [R1]                              ; guarda a coluna do
pacman
                    MOV  R3, pacmanzao_mask                          ; inicializa o
registo no primeiro endereco da mascara do pacman
                    MOV  R4, PACMAN_SIZE_C                     ; inicializa o registo
com o valor maximo das colunas
                    MOV  R5, PACMAN_SIZE_L                     ; inicializa o registo
com o valor maximo das linhas
                    CALL draw_imagem                           ; chama a rotina que
vai escrever o pacman no PixelScreen

                    POP  R5
                    POP  R4
                    POP  R3
                    POP  R1
```

```
                    POP  R0
                    RET

; * * * * * * * * * * * draw_star * * * * * * * * * * * * * * * * * * * * * * * * * *
; *
; * Descricao: Desenha uma estrela
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

draw_star:          PUSH R1                                 ;R2 = 0 -> apaga star
                    PUSH R0                                 ;R2  =  1  ->  escreve
star
                    PUSH R3
                    PUSH R4
                    PUSH R5

                    MOV  R0, [R5]                           ; guarda a coordenada
da linha
                    MOV  R1, [R5+2]                                 ;   guarda   a
coordenada da coluna
                    MOV  R3, star_mask                      ; inicializa o registo
no primeiro endereco da mascara da estrela
                    MOV  R4, STAR_SIZE_C                    ; inicializa o registo
com o valor maximo das colunas
                    MOV  R5, STAR_SIZE_L                    ; inicializa o registo
com o valor maximo das linhas
                    CALL draw_imagem                        ; chama a rotina que
vai escrever as estrelas no PixelScreen

                    POP  R5
                    POP  R4
                    POP  R3
                    POP  R1
                    POP  R0
                    RET

; * * * * * * * * * * * draw_casper * * * * * * * * * * * * * * * * * * * * * * * *
; *
```

```
; * Descricao: Desenha um fantasma
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

draw_casper:          PUSH R0    ;R2 = 0 -> apaga star
                      PUSH R1    ;R2 = 1 -> escreve star
                      PUSH R3
                      PUSH R4
                      PUSH R5

                      MOV  R0, [R5]                          ; guarda no registo o
valor da linha do fantasma
                      ADD  R5, 2                             ;   avanca   para   o
endereco seguinte
                      MOV  R1, [R5]                          ; guarda no registo o
valor da coluna do fantasma
                      MOV  R3, casper_mask                   ; inicializa o registo
no primeiro endereco da mascara do fantasma
                      MOV  R4, GHOST_SIZE_C                  ; inicializa o registo
com o valor maximo das colunas
                      MOV  R5, GHOST_SIZE_L                  ; inicializa o registo
com o valor maximo das linhas
                      CALL draw_imagem                       ; chama a rotina que
vai escrever os fantasmas no PixelScreen

                      POP  R5
                      POP  R4
                      POP  R3
                      POP  R1
                      POP  R0
                      RET

; * * * * * * * * * * * restart * * * * * * * * * * * * * * * * * * * * * * * * * *
; *
; * Descricao: Inicializa o jogo
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
restart:            PUSH R0

                    PUSH R1

                    PUSH R2

                    PUSH R3

                    PUSH R5



                    MOV  R3, 0                          ; inicialisa o registo
a 0

                    MOV  R0, POUT1_ADDR                 ; inicializa o registo
com o endereco do POUT_1

                    MOV  R1, time                       ; inicializa o registo
com o valor da variavel time

                    MOVB [R1], R3                       ;   faz   reset   ao
contador dos segundos (time)

                    MOVB [R0], R3                       ; faz reser ao display
dos segundos (time)

                    MOV  R0, STARS                      ; inicializa o registo
no primeiro endereco da variavel STARS

                    MOVB [R0], R3                       ;  coloca  a  zero  a
variavel que verifica se ja comeu a estrela ou nao (star1)

                    INC  R0                             ;  passa  ao  proximo
endereco

                    MOVB [R0], R3                       ;  coloca  a  zero  a
variavel que verifica se ja comeu a estrela ou nao (star2)

                    INC  R0                             ;  passa  ao  endereco
seguinte

                    MOVB [R0], R3                       ;  coloca  a  zero  a
variavel que verifica se ja comeu a estrela ou nao (star3)

                    INC  R0                             ; passa para o ultimo
endereco

                    MOVB [R0], R3                       ;  coloca  a  zero  a
variavel que verifica se ja comeu a estrela ou nao (star4)

                    CALL clear_display                  ; chama a rotina que
coloca todos os pixeis do display a 0

                    CALL draw_moldura                   ; chama a rotina que
redesenha a moldura, a caixa dos fantasmas e o pacman

                    MOV  R1, ADDR_POS_START                     ; inicializa o
registo com o endereco da linha na posicao inicial do pacman

                    MOV  R0, [R1]

                    ADD  R1, 2                          ;  incremneta  2  para
obter o endereco da coluna na posicao inicial do pacman

                    MOV  R1, [R1]

                    MOV  R2, ADDR_POS_OLD               ; inicializa o registo
com o endereco da linha na posicao anterior do pacman
```

```
                    MOVB [R2], R0

                    ADD  R2, 2                          ;  incremneta  2  para
obter o endereco da coluna na posicao anterior do pacman

                    MOVB [R2], R1

                    MOV  R2, 1                          ;  inicializa  R2  a  1
para indicar que vai ecrever no display

                    CALL draw_pacmanzao

                    MOV  R5, STAR_ADDR_POS1                     ; inicializa o
registo com o endereco da estrela 1

                    CALL draw_star

                    MOV  R5, STAR_ADDR_POS2                     ; inicializa o
registo com o endereco da estrela 2

                    CALL draw_star

                    MOV  R5, STAR_ADDR_POS3                     ; inicializa o
registo com o endereco da estrela 3

                    CALL draw_star

                    MOV  R5, STAR_ADDR_POS4                     ; inicializa o
registo com o endereco da estrela 4

                    CALL draw_star

                    CALL init_ghosts                    ;      rotina      que
inicializa os fantasmas

                    MOV  R0, STARS_EATEN                ; inicializa o registo
com a variavel que guarda o numero de estrelas comidas

                    MOV  R1, 0

                    MOVB [R0], R1

                    MOV  R0, POUT3_ADDR                 ; inicializa o registo
com o endereco do periferico de saida

                    MOVB [R0], R1

                    MOV  R0, POUT1_ADDR                 ; inicializa o registo
com o endereco do periferico de saida

                    MOVB [R0], R1

                    MOV  R0, GHOST_NUMBERS              ; inicializa o registo
com o numero de fantasmas que vao ser criados, inicialmente 1

                    MOV  R1, 1

                    MOVB [R0], R1



                    POP  R5

                    POP  R3

                    POP  R2

                    POP  R0
```

```
                        POP  R1
                        RET


; * * * * * * * * * * * the_end * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *
; * Descricao: Termina o jogo
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


the_end:                PUSH R0
                        PUSH R1
                        PUSH R2
                        PUSH R3
                        PUSH R4
                        PUSH R5


                        MOV  R5, JA_FOSTE                       ; inicializa o registo
com o valor da variavel JA_FOSTE
                        MOV  R5, [R5]
                        MOV  R4, 0
                        CMP  R5, R4                             ;   verifica   se   o
fantasma comeu o pacman
                        JZ   not_the_end


                        CALL clear_display                      ; limpa o pixelscreen
                        MOV  R2, 1                              ; inicializa o registo
a 1, valor que indica a accao de escrita de um pixel
                        MOV  R1, 0                              ; inicializa o registo
na coluna 0
                        MOV  R0, 0                              ; inicializa o registo
na linha 0
                        MOV  R5, 1
                        CMP  R6, R5
                        JNZ  lose
                        MOV  R3, winner_mask                    ; inicializa o registo
no primeiro endereco da mascara winner_mask
                        JMP  forward
```

```
lose:                   MOV  R3, the_end_mask                   ; inicializa o registo
no primeiro endereco da mascara the_end_mask

forward:                MOV  R4, DISPLAY_SIZE_C                 ; inicializa o
registo com o valor maximo das colunas

                        MOV  R5, DISPLAY_SIZE_L                 ; inicializa o
registo com o valor maximo das linhas

                        CALL draw_imagem                        ; chama a rotina que
vai escrever o ecran de fim de jogo no PixelScreen



the_end_cycle:          CALL keyboard_processed                 ; rotina que
marca a tecla como processada

                        CALL keyboard                           ; verifica se outra
tecla foi premida

                        MOV  R0, keyboard_flag                  ; inicializa o
registo com a variavel

                        MOVB R0, [R0]

                        CMP  R0, 1                              ; se R0 = 1 o jogo
reinicia

                        JNZ  the_end_cycle                      ; enquanto nenhuma
tecla for premida o jogo nao reinicia



                        CALL clear_display                      ; limpa o pixelscreen

                        CALL restart                            ; reinicia o jogo



                    MOV  R5, JA_FOSTE

                    MOV  R4, 0

                    MOV  [R5], R4                               ; coloca a variavel
que termina o jogo( = 1) a 0



not_the_end:        POP  R5

                    POP  R4

                    POP  R3

                    POP  R2

                    POP  R1

                    POP  R0

                    RET


; * * * * * * * * * * * draw_imagem * * * * * * * * * * * * * * * * * * * * * * * * * * *

; *

; * Descricao: Desenha uma imagem
```

```
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

draw_imagem:        PUSH R0                             ;R0 = LINHAS
                    PUSH R1                             ;R1 = COLUNAS
                    PUSH R2                             ;R2 = ESCREVE OU APAGA
                    PUSH R3                             ;R3 = mask
                    PUSH R4
                    PUSH R5
                    PUSH R6
                    PUSH R7
                    PUSH R8

                    ; colocar as coordenadas do vector tamanho da mascara em R4 e R5

                    MOV  R6, 0                          ; contador da coluna
                    MOV  R7, 0                          ; contador da linha

nextcolune:          MOVB R8, [R3]
                     CMP  R8, 0                         ; verifica se naquele
endereco da mascara e suposto escrever ou nao
                    JZ   not_draw
                    CALL draw_pixel                             ;      escreve
pixel
not_draw:           INC  R3                             ; endereco seguinte
                    CMP  R6, R4
                    JZ   nextline                       ;   salta   de   linha
quando percorrer todas as colunas dessa linha
                    INC  R6                             ;  passa   a   proxima
linha
                    INC  R1                             ;  passa   a   proxima
coluna
                    JMP  nextcolune                             ; salta para a
proxima coluna
nextline:            MOV  R6, 0
                    SUB  R1, R4
                    INC  R0                             ;  passa   a   proxima
linha
```

```
                    CMP  R7, R5
                    JZ   skip
                    INC  R7                                 ;  passa  a  proxima
coluna
                    JMP  nextcolune


skip:                POP  R8
                    POP  R7
                    POP  R6
                    POP  R5
                    POP  R4
                    POP  R3
                    POP  R2
                    POP  R1
                    POP  R0
                    RET

; * * * * * * * * * * * * Draw_pixel * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *
; * Descricao: Escreve um pixel no display
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

draw_pixel:          PUSH R0
                    PUSH R1
                    PUSH R3
                    PUSH R4

                    ; encontrar o endereco da linha
                    MOV  R3, DISPLAY_BYTES_LIN              ;  guarda  em  R3  o
numero de bytes em cada linha (= 4)
                    MUL  R0, R3                             ; multiplica o numero
de linhas pelo numero de bytes
                    MOV  R3, DISPLAY_ADDR                   ;  guarda  em  R3  o
endereco base do display
                    ADD  R0, R3                             ;  soma  o  numero  de
bytes ao endereco base
```

```
                    ; encontrar o byte da coluna

                    MOV  R3, R1                         ;  guarda  em  R3  a
posicao da coluna do pixel

                    MOV  R4, CONST1                             ; CONST1 = 8

                    DIV  R3, R4                         ;  para  descobrir  em
que byte da coluna estamos

                    ADD  R0, R3                         ;  soma  o  byte  da
coluna ao endereco



                    ; encontrar a mascara correspondente ao bit a acender

                    MOD  R1, R4                         ;  resto  da  divisao
inteira

                    MOV  R4, bitmask                    ;  guarda  em  R4  o
endereco base da bitmask

                    ADD  R1, R4                         ; determina a mascara
correspondente ao bit a escrever

                    MOVB R1, [R1]                       ;  guarda  em  R1  a
mascara do bit

                    ; buscar o byte atual do display

                    MOVB R4, [R0]                       ; guarda em R4 o byte
correspondente

                    CMP  R2, 0                          ;   0->   APAGA   1->
ESCREVE

                    JZ   clear_pixel

                    OR   R4, R1                         ; faz uma mascara para
escrever o bit correspondente

                    JMP  draw_pix_end

clear_pixel:          XOR  R4, R1                       ; faz uma mascara para
apagar o bit correspondente

draw_pix_end:         MOVB [R0], R4                     ;  coloca  no  mesmo
endereco do display o byte alterado



draw_pix_skip:        POP R4

                      POP R3

                    POP R1

                    POP R0

                    RET



; * * * * * * * * * * Clear_display * * * * * * * * * * * * * * * * * * * * * *

; *
```

```
; * Descricao: Apaga todos os pixeis do Display

; *

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


clear_display:          PUSH R0

                        PUSH R1

                        PUSH R2


                        MOV  R0, DISPLAY_ADDR                       ; da a R0 o endereco
dos primeiros 8 pixeis

                        MOV  R1, DISPLAY_END_ADDR                   ; da a R1 o endereco
do conjunto de 8 pixeis finais

                        MOV  R2, 0                                  ; da a R2 aquilo que
se vai enviar para o pixel

clear_cycle:            MOVB [R0], R2                               ; apaga os pixeis do
endereco

                        INC  R0                                     ; muda para o proximo
endereco

                        CMP  R0, R1                                 ;   verifica   se   o
pixelscreen ja esta todo limpo

                        JNZ  clear_cycle                            ; vai para o inicio do
ciclo de limpeza


                        POP  R2

                        POP  R1

                        POP  R0

                        RET


; * * * * * * * * * * * Compare_limits * * * * * * * * * * * * * * * * * * * * * * * * * *

; *

; * Descricao: Compara se uma imagem toca nos limites do display.

; *

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


Compare_limits:             PUSH R0

                        PUSH R1

                        PUSH R2

                        PUSH R3
```

```
                        PUSH R4
                        PUSH R5
                        PUSH R6
                        PUSH R7
                        PUSH R8
                        PUSH R9

                        ;R0 = LINHAS
                        ;R1 = COLUNAS
                        ;R2 = ESCREVE OU APAGA
                        ;R3 = mask
                        MOV  R10, 0                             ; 0 -> nao tocou, 1 -
> tocou
                        ; colocar as coordenadas do vector tamanho da mascara em R4 e R5
                        ;MOV  R3, pacmanzao_mask
                        ;MOV  R4, PACMAN_SIZE_C
                        ;MOV  R5, PACMAN_SIZE_L
                        MOV  R6, 0                              ; contador da coluna
                        MOV  R7, 0                              ; contador da linha

next_col:               MOVB R8, [R3]
                         CMP  R8, 0
                        JZ   clean
                        CALL cmp_pos                            ; verifica se o pixel
esta a sobrepor outro
                        CMP  R9, 1                              ; esta a sobrepor se
R9 = 1
                        JZ   touch
clean:                  INC  R3                                 ; endereco seguinte
                        CMP  R6, R4
                        JZ   next_li
                        INC  R6                                 ;   passa   a   coluna
seguinte
                        INC  R1                                 ;   passa   a   coluna
seguinte
                        JMP  next_col
next_li:                MOV  R6, 0
```

```
                    SUB  R1, R4
                    INC  R0                                 ;   passa   a   linha
seguinte
                    CMP  R7, R5
                    JZ   not_touch
                    INC  R7                                 ;   passa   a   linha
seguinte
                    JMP  next_col
touch:              MOV  R10, 1                             ; se estiver a tocar,
R10 = 1


not_touch:          POP R9
                    POP R8
                    POP R7
                    POP R6
                    POP R5
                    POP R4
                    POP R3
                    POP R2
                    POP R1
                    POP R0
                    RET

; * * * * * * * * * * * * cmp_pos * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *
; * Descricao: Compara se um pixel toca nos limites do display.
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

cmp_pos:            PUSH R5
                    PUSH R4
                    PUSH R3
                    PUSH R1
                    PUSH R0

                    ;R6=colunas
```

```
                    ;R7=linhas

                    MOV  R9, 1

                    MOV  R2, limits_mask                    ; inicializa o registo
com o primeiro endereco da mascara com os limites

                    MOV  R3, DISPLAY_SIZE                   ; inicializa o registo
com o tamanho do display

                    MUL  R0, R3                             ; forma de calcular

                    ADD  R0, R1                             ; o endereco do

                    ADD  R2, R0                             ; pixel nessa posicao

                    MOVB R1, [R2]

                    CMP  R1, 1                              ;   verifica   se   os
pixeis estao sobrepostos

                    JZ   ups

                    MOV  R9, 0                              ; R9 = 0, nao estao
sobrepostos


ups:                POP  R0

                    POP  R1

                    POP  R3

                    POP  R4

                    POP  R5

                    RET


; * * * * * * * * * * * secret_passage * * * * * * * * * * * * * * * * * * * * * *

; *

; * Descricao: Rotina que faz com que o Pacman se teletransporte de um lado

; *            para o outro do campo (cima para baixo, esquerda para a

; *            direita e vice-versa).

; *

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


secret_passage:             PUSH R2

                    PUSH R4

                    PUSH R5
```

```
PASSAGEM_DIREITA:     MOV  R3, 14
                      CMP  R0, R3
                      JNZ  PASSAGEM_ESQUERDA
                      MOV  R3, 29
                      CMP  R1, R3
                      JNZ  PASSAGEM_ESQUERDA

                      MOV  R3, 14
                      MOV  R0, R3
                      MOV  R3, 0
                      MOV  R1, R3
                      MOV  R3, 1
                      JMP  PASSAGE

PASSAGEM_ESQUERDA:    MOV  R3, 14
                      CMP  R0, R3
                      JNZ  PASSAGEM_CIMA
                      MOV  R3, 0
                      CMP  R1, R3
                      JNZ  PASSAGEM_CIMA

                      MOV  R3, 14
                      MOV  R0, R3
                      MOV  R3, 29
                      MOV  R1, R3
                      MOV  R3, 1
                      JMP  PASSAGE

PASSAGEM_CIMA:        MOV  R3, 0
                      CMP  R0, R3
                      JNZ  PASSAGEM_BAIXO
                      MOV  R3, 15
                      CMP  R1, R3
                      JNZ  PASSAGEM_BAIXO
```

```
                    MOV  R3, 29
                    MOV  R0, R3
                    MOV  R3, 15
                    MOV  R1, R3
                    MOV  R3, 1
                    JMP  PASSAGE

PASSAGEM_BAIXO:          MOV  R3, 29
                    CMP  R0, R3
                    JNZ  NO_PASSAGE
                    MOV  R3, 15
                    CMP  R1, R3
                    JNZ  NO_PASSAGE
                    MOV  R3, 0
                    MOV  R0, R3
                    MOV  R3, 15
                    MOV  R1, R3
                    MOV  R3, 1
                    JMP  PASSAGE

NO_PASSAGE:         MOV  R3, 0
PASSAGE:            POP  R5
                    POP  R4
                    POP  R2
                    RET

; * * * * * * * * * * * mov_init_casper * * * * * * * * * * * * * * * * * * * * * * *
; *
; * Descricao: Rotina que inicializa o movimento do fantasma.
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

mov_init_casper:    PUSH R2
```

```
                        PUSH R3
                        PUSH R4

                        MOV  R9, 0
                        MOV  R1, R3                         ; guarda o endereco da
posicao do fantasma
                        MOV  R0, [R1]                       ; guarda a linha do
fantasma
                        ADD  R1, 2                          ;   avanca   para   o
endereco seguinte
                        MOV  R1, [R1]                       ; guarda a coluna do
fantasma
                        MOV  R5, LINE_MIN
                        MOV  R4, COLUNE
                        CMP  R1, R4                         ; compara a coluna do
fantasma com a coluna inicial (na caixa)
                        JNZ  iniciado
                        MOV  R4, LINE_MAX
                        CMP  R0, R5                         ; verifica se o
                        JGT  iniciado                       ; fantasma ainda nao
                        CMP  R0, R4                         ;  subiu  as  5  linhas
                        JLT  iniciado                       ; para sair da caixa
                        SUB  R0, 1
                        MOV  R9, 1

iniciado:               POP  R4
                        POP  R3
                        POP  R2
                        RET

; * * * * * * * * * * * touch_pacman * * * * * * * * * * * * * * * * * * * * * *
; *
; * Descricao: Rotina que verifica se o fantasma tocou ou nao no Pacman.
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
touch_pacman:          PUSH R0                                  ; linha do fantasma
                       PUSH R1                                  ; coluna do fantasma
                       PUSH R2
                       PUSH R3
                       PUSH R4


                       MOV  R4, ADDR_POS_OLD                    ; inicializa o registo
com o endereco da posicao do pacman
                       MOVB R3, [R4]                            ; guarda a linha do
pacman no registo
                       ADD  R4, 2
                       MOVB R4, [R4]                            ; guarda a coluna do
pacman no registo
                       CMP  R0, R3                              ; compara a linha do
pacman com a linha do fantasma
                       JNZ  dont_touch
                       CMP  R1, R4                              ; compara a coluna do
pacman com a coluna do fantasma
                       JNZ  dont_touch
                       MOV  R0, JA_FOSTE
                       MOV  R2, 1
                       MOV  [R0], R2                            ; mete a variavel
JA_FOSTE a 1


dont_touch:            POP  R4
                       POP  R3
                       POP  R2
                       POP  R1
                       POP  R0
                       RET


; * * * * * * * * * * * choose_direction * * * * * * * * * * * * * * * * * * * *
; *
; * Descricao: Rotina que decide qual direcao o fantasma deve tomar
; *            (caminho mais curto ate ao pacman).
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
choose_direction:      PUSH R2
                       PUSH R3


                       MOV  R1, ADDR_POS_OLD                    ; guarda no registo o
endereco da posicao do pacman
                       MOVB R0, [R1]                            ; guarda no registo a
linha do pacman
                       ADD  R1, 2
                       MOVB R1, [R1]                            ; guarda no registo a
coluna do pacman
                       MOV  R2, [R3]                            ; guarda a linha do
fantama
                       ADD  R3, 2
                       MOV  R3, [R3]                            ; guarda a coluna do
fantasma


                       CMP  R0, R2                              ; ve se a linha do
pacman e menor do que a linha do fantasma
                       JLT  _up_
                       CMP  R0, R2                              ; ve se a linha do
pacman e maior do que a linha do fantasma
                       JGT  _down_
_hold_:                MOV  R0, 0
                       JMP  _verif_col
_up_:                  MOV  R0, -1
                       JMP  _verif_col
_down_:                MOV  R0, 1


_verif_col:            CMP  R1, R3                              ; ve se a coluna do
pacman e menor do que a coluna do fantasma
                       JLT  _left_
                       CMP  R1, R3                              ; ve se a coluna do
pacman e maior do que a coluna do fantasma
                       JGT  _right_
_hold:                 MOV  R1, 0
                       JMP  _verif_end
_left_:                MOV  R1, -1
                       JMP  _verif_end
```

```
_right_:            MOV  R1, 1


_verif_end:         POP  R3
                    POP  R2
                    RET

; * * * * * * * * * * * * move_casper * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *
; * Descricao: Rotina que mexe o fantasma.
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

move_casper:        PUSH R10
                    PUSH R9
                    PUSH R8
                    PUSH R7
                    PUSH R6
                    PUSH R5
                    PUSH R4
                    PUSH R3
                    PUSH R2
                    PUSH R1                          ;coluna a deslocar
                    PUSH R0                          ;linha a deslocar



                    MOV  R6, R3                      ;R3  =  endereco  do
casper
                    CALL mov_init_casper
                    CMP  R9, 1
                    JZ   inicializado
                    CALL choose_direction
                    MOV  R7, R0                      ;COPIA DIR LIN
                    MOV  R8, R1                      ;COPIA DIR COL
                    MOV  R4, [R3]
                    ADD  R0, R4
```

```
                    ADD  R3, 2
                    MOV  R4, [R3]
                    ADD  R1, R4
                    MOV  R3, limits_mask
                    MOV  R4, GHOST_SIZE_L
                    MOV  R5, GHOST_SIZE_C
                    CALL Compare_limits
                    CMP  R10, 0
                    JZ   inicializado
                    CALL another_diretion

inicializado:       MOV  R2, 0
                    MOV  R5, R6
                    CALL draw_casper
                    MOV  [R6], R0
                    ADD  R6, 2
                    MOV  [R6], R1
                    MOV  R2, 1
                    CALL draw_casper
                    CALL touch_pacman

not_move:           POP  R0
                    POP  R1
                    POP  R2
                    POP  R3
                    POP  R4
                    POP  R5
                    POP  R6
                    POP  R7
                    POP  R8
                    POP  R9
                    POP  R10
                    RET
```

```
; * * * * * * * * * * * * move_ghosts * * * * * * * * * * * * * * * * * * * * * * * * *

; *

; * Descricao: Rotina responsavel pelo movimento dos fantasmas.

; *

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *



move_ghosts:          PUSH R3

                      PUSH R10



                      MOV  R10, ghost_move                    ; guarda no registo a
variavel que decide se os fantasmas se mexem ou nao

                      MOVB R10, [R10]

                      CMP  R10, 0                             ;  verifica  se  os
fantasmas se devem mexer ou nao

                      JZ   _not_move

                      MOV  R10, GHOST_NUMBERS                       ;  guarda  no
registo o numero de fantasmas

                      MOVB R10, [R10]

                      CMP  R10, 1                             ;  verifica  se  e  o
fantasma 1

                      JZ   ONE_CASPER

                      CMP  R10, 2                             ;  verifica  se  e  o
fantasma 2

                      JZ   TWO_CASPER

                      CMP  R10, 3                             ;  verifica  se  e  o
fantasma 3

                      JZ   THREE_CASPER

FOUR_CASPER:          MOV  R3, GHOST_ADDR_POS4                ; inicializa o registo
com o endereco do fantasma 4

                      CALL move_casper

THREE_CASPER:         MOV  R3, GHOST_ADDR_POS3                ; inicializa o registo
com o endereco do fantasma 3

                      CALL move_casper

TWO_CASPER:           MOV  R3, GHOST_ADDR_POS2                ; inicializa o registo
com o endereco do fantasma 2

                      CALL move_casper

ONE_CASPER:           MOV  R3, GHOST_ADDR_POS1                ; inicializa o registo
com o endereco do fantasma 1

                      CALL move_casper
```

```
                        MOV  R3, 0

                        MOV  R10, ghost_move

                        MOVB [R10], R3                          ; coloca a variavel,
que decide se os fantasmas se mexem ou nao, a zero



_not_move:              POP  R10

                        POP  R3

                        RET



; * * * * * * * * * * * * init_ghosts * * * * * * * * * * * * * * * * * * * * * * * * *

; *

; * Descricao: Rotina que inicializa os fantasmas.

; *

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *



init_ghosts:            PUSH R0

                        PUSH R1

                        PUSH R2

                        PUSH R5



                        MOV  R1, GHOST_ADDR_INIT                ; iniciaiza o registo
com o endereco da linha onde o fantasma vai ser criado

                        MOV  R0, [R1]

                        ADD  R1, 2                              ; incrementa 2 para
avancar dois bytes e obter o endereco da coluna onde o fantasma vai ser criado

                        MOV  R1, [R1]



ghost_1:                MOV  R5, GHOST_ADDR_POS1                ; inicializa o registo
com o endereco da linha do fantasma 1

                        MOV  R2, R5

                        MOV  [R2], R0

                        ADD  R2, 2                              ; incrementa 2 para
avancar dois bytes e obter o endereco da coluna do fantasma

                        MOV  [R2], R1

                        CALL draw_casper                        ; chama a roitina que
vai desenhar o fantasma no display
```

```
ghost2:                 MOV  R5, GHOST_ADDR_POS2                    ; inicializa o registo
com o endereco da linha do fantasma 2

                        MOV  R2, R5

                        MOV  [R2], R0

                        ADD  R2, 2                                  ;  incrementa  2  para
avancar dois bytes e obter o endereco da coluna do fantasma

                        MOV  [R2], R1

                        CALL draw_casper                            ; chama a roitina que
vai desenhar o fantasma no display



ghost_3:                MOV  R5, GHOST_ADDR_POS3                    ; inicializa o registo
com o endereco da linha do fantasma 3

                        MOV  R2, R5

                        MOV  [R2], R0

                        ADD  R2, 2                                  ;  incrementa  2  para
avancar dois bytes e obter o endereco da coluna do fantasma

                        MOV  [R2], R1

                        CALL draw_casper                            ; chama a roitina que
vai desenhar o fantasma no display



ghost_4:                MOV  R5, GHOST_ADDR_POS4                    ; inicializa o registo
com o endereco da linha do fantasma 4

                        MOV  R2, R5

                        MOV  [R2], R0

                        ADD  R2, 2                                  ;  incrementa  2  para
avancar dois bytes e obter o endereco da coluna do fantasma

                        MOV  [R2], R1

                        CALL draw_casper                            ; chama a roitina que
vai desenhar o fantasma no display



                        POP  R5

                        POP  R2

                        POP  R1

                        POP  R0

                        RET


;*****************************************************************************

;*******************************  KEYBOARD  ********************************

;*****************************************************************************
```

```
; * * * * * * * * * * * * keyboard * * * * * * * * * * * * * * * * * * * * * * * * *
; *
; * Descricao: Rotina que le o teclado e verifica se alguma tecla foi premida
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

keyboard:               PUSH R0
                        PUSH R1
                        PUSH R2
                        PUSH R3
                        PUSH R4
                        PUSH R5
                        PUSH R6
                        PUSH R7

                        MOV  R0, LINHA
                        MOV  R1, PIN_ADDR
                        MOV  R2, POUT2_ADDR
                        MOV  R4, CONST9                                 ; linha actual
keyb_next:              CMP  R0, 0
                        JZ   keyb_no_key
                        MOVB [R2], R0                                   ; coloca na entrada a
linha atual
                        MOVB R3, [R1]                                   ; guarda em R3 o valor
da coluna
                        MOV  R7, 0FH
                        AND  R3, R7
                        SHR  R0, 1                                      ; muda para a linha
seguinte
                        DEC  R4
                        CMP  R3, 0                                      ; se nao houver tecla
carregada
                        JZ   keyb_next
                        ; se a tecla pressionada ja foi analisada, sai
                        MOV  R6, key_processed                          ;
```

```
                    MOVB R6, [R6]                           ; verifica se a tecla
ja foi processada
                    CMP  R6, 1
                    JZ   keyb_processed                     ; se ja foi salta
                    ; calcular a tecla carregada com base na linha
                    MOV  R5, 0                              ; coluna actual
calc_col:            CMP  R3, 0001H
                    JZ   calc_key
                    SHR  R3, 1
                    INC  R5
                    JMP  calc_col
calc_key:            MOV  R6, CONST8
                    MUL  R4, R6
                    ADD  R4, R5                             ; determina o valor da
tecla
                    MOV  R0, pressed_key
                    MOVB [R0], R4                           ; guarda o valor da
tecla em pressed_key
                    MOV  R0, keyboard_flag
                    MOV  R1, 1                              ;       ativa       a
keybboard_flag a 1
                    MOVB [R0], R1
                    JMP  keyb_end
keyb_no_key:         MOV  R0, key_processed                 ; se nao foi
carregada nenhuma tecla
                    MOV  R1, 0
                    MOVB [R0], R1                           ; deixa de haver tecla
processada, fica a 0
keyb_processed:      MOV  R0, keyboard_flag
                    MOV  R1, 0                              ;    e    coloca    a
keyboard_flag a 0
                    MOVB [R0], R1

keyb_end:           POP  R7
                    POP  R6
                    POP  R5
                    POP  R4
                    POP  R3
```

```
                    POP  R2
                    POP  R1
                    POP  R0
                    RET

; * * * * * * * * * * * keyboard_get_key * * * * * * * * * * * * * * * * * * * * *
; *
; * Descricao: Rotina que retorna a tecla carregada actualmente ou -1 se nao
; *            houver tecla carregada
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
keyboard_get_key:       ; verifica se ha uma tecla pressionada
                    MOV  R2, keyboard_flag                      ; guarda no registo a
variavel que indica se uma nova tecla foi premida
                    MOVB R2, [R2]
                    CMP  R2, 1                                  ; verifica se uma nova
tecla foi premida
                    JZ   keyboard_has_key
                    MOV  R2, -1
                    JMP  _sai
keyboard_has_key:     MOV  R2, pressed_key                      ; guarda no registo a
tecla premida
                    MOVB R2, [R2]
_sai:                 RET

; * * * * * * * * * * * keyboard_processed * * * * * * * * * * * * * * * * * * * *
; *
; * Descricao: Rotina que, no caso de haver uma tecla pressionada,
; *            a marca como processada no fim do ciclo
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
keyboard_processed:     PUSH R0
                    PUSH R1

                    MOV  R0, keyboard_flag                      ; guarda no registo a
variavel que indica se uma nova tecla foi premida
```

```
                        MOVB R0, [R0]

                        CMP  R0, 0                              ; verifica se uma nova
tecla foi premida

                        ; se nenhuma tecla foi pressionada, sai

                        JZ   kprocessed_end

                        MOV  R0, key_processed                  ; guarda no registo a
informacao que a tecla ja foi processada ou nao

                        MOV  R1, 1

                        MOVB [R0], R1                           ; actualiza a variavel
kye_processed a 1 o que indica que a tecla ja foi processada


kprocessed_end:          POP  R1

                        POP  R0

                        RET


; * * * * * * * * * * * * temporizador * * * * * * * * * * * * * * * * * * * * * * *

; *

; * Descricao: Rotina que conta os segundos e os exibe no display de 7 segmentos.

; *

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


temporizador :          PUSH R0                                 ; endereco do display

                        PUSH R1                                 ; time

                        PUSH R2                                 ; flag

                        PUSH R3                                 ; time incrementado

                        PUSH R4                                 ; 10

                        PUSH R5

                        PUSH R6


                        MOV  R2, time_flag                      ; poe a flag do time
no R2

                        MOV  R2, [R2]

                        CMP  R2, 0                              ; verifica se houve um
flanco de relogio

                        JZ   exit                               ; se nao houve ou ja
foi incrementado, sai

                        MOV  R0, POUT1_ADDR                     ; endereco do display
```

```
                        MOV  R1, time
                        MOVB R3, [R1]                   ; guarda em R3 o tempo
atual
                        INC  R3                         ; incrementa mais um
no contador do tempo


                        MOV  R5, R3
                        MOV  R6, R3
                        MOV  R4, 10                     ; variavel que vai
servir pa dividir
                        MOD  R5, R4                     ; resto da divisao
inteira por 10 (unidades)
                        DIV  R6, R4                     ; divisao inteira por
10 (dezenas)
                        SHL  R6, 4                      ; passa as dezenas
para os bits 7-4
                        OR   R6, R5                     ; junta dezenas e
unidades num registo
                        MOVB [R1], R3
                        MOVB [R0], R6
                        MOV  R3, 153
                        CMP  R6, R3                     ; verifica se ja
acabou o tempo (99 segundos)
                        JNZ  running
                        MOV  R3, time_out
                        MOV  R6, 1
                        MOVB [R3], R6                   ; ativa a variavel
(acabou o tempo) a 0
running:                MOV  R2, time_flag
                        MOV  R1, 0
                        MOV  [R2], R1                   ; reset da flag


exit:                   POP  R6
                        POP  R5
                        POP  R4
                        POP  R3
                        POP  R2
                        POP  R1
                        POP  R0
```

```
                    RET


; * * * * * * * * * * * touch_stars * * * * * * * * * * * * * * * * * * * * * * * *
; *
; * Descricao: Rotina que verifica se alguma estrela foi comida no momento,
; *            e identifica-a.
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


touch_stars:        PUSH R0                                 ;linha atual
                    PUSH R1                                 ;coluna atual
                    PUSH R2
                    PUSH R3                                 ;linha
                    PUSH R4                                 ;coluna
                    PUSH R5
                    PUSH R6


                    MOV  R4, ADDR_POS_OLD                   ; inicializa o registo
com o endereco da linha do pacman na posicao antiga
                    MOVB R3, [R4]
                    ADD  R4, 2                              ; incrementa 2 para
avancar dois bytes e obter o endereco da coluna do pacman na posicao antiga
                    MOVB R4, [R4]


estrela_1:          MOV  R1, STAR_ADDR_POS1                         ; inicializa o
registo com o endereco da linha da estrela 1
                    MOV  R0, [R1]
                    ADD  R1, 2                              ; incrementa 2 para
avancar dois bytes e obter o endereco da coluna da estrela 1
                    MOV  R1, [R1]
                    CMP  R0, R3                             ; compara a linha da
estrela com a linha do pacman na posicao antiga
                    JNZ  estrela_2
                    CMP  R1, R4                             ; compara a coluna da
estrela com a coluna do pacman na posicao antiga
                    JNZ  estrela_2
                    MOV  R5, JA_FOSTE_STAR                  ; se as posicoes
coincidirem, inicializamos o rregisto com a variavel JA_FOSTE_STAR
```

```
                    MOV  R6, 1                              ; inicializa o registo
com  o numero da estrela que foi comida pelo pacman

                    MOVB [R5], R6



estrela_2:          MOV  R1, STAR_ADDR_POS2                 ; inicializa o
registo com o endereco da linha da estrela 2

                    MOV  R0, [R1]

                    ADD  R1, 2                              ; incrementa 2 para
avancar dois bytes e obter o endereco da coluna da estrela 2

                    MOV  R1, [R1]

                    CMP  R0, R3                             ; compara a linha da
estrela com a linha do pacman na posicao antiga

                    JNZ  estrela_3

                    CMP  R1, R4                             ; compara a coluna da
estrela com a coluna do pacman na posicao antiga

                    JNZ  estrela_3

                    MOV  R5, JA_FOSTE_STAR                  ; se as posicoes
coincidirem, inicializamos o rregisto com a variavel JA_FOSTE_STAR

                    MOV  R6, 2                              ; inicializa o registo
com  o numero da estrela que foi comida pelo pacman

                    MOVB [R5], R6



estrela_3:          MOV  R1, STAR_ADDR_POS3                 ; inicializa o
registo com o endereco da linha da estrela 3

                    MOV  R0, [R1]

                    ADD  R1, 2                              ; incrementa 2 para
avancar dois bytes e obter o endereco da coluna da estrela 3

                    MOV  R1, [R1]

                    CMP  R0, R3                             ; compara a linha da
estrela com a linha do pacman na posicao antiga

                    JNZ  estrela_4

                    CMP  R1, R4                             ; compara a coluna da
estrela com a coluna do pacman na posicao antiga

                    JNZ  estrela_4

                    MOV  R5, JA_FOSTE_STAR                  ; se as posicoes
coincidirem, inicializamos o rregisto com a variavel JA_FOSTE_STAR

                    MOV  R6, 3                              ; inicializa o registo
com  o numero da estrela que foi comida pelo pacman

                    MOVB [R5], R6
```

```
estrela_4:            MOV  R1, STAR_ADDR_POS4                          ; inicializa o
registo com o endereco da linha da estrela 4

                      MOV  R0, [R1]

                      ADD  R1, 2                                ; incrementa 2 para
avancar dois bytes e obter o endereco da coluna da estrela 4

                      MOV  R1, [R1]

                      CMP  R0, R3                               ; compara a linha da
estrela com a linha do pacman na posicao antiga

                      JNZ  dont_touch_2

                      CMP  R1, R4                               ; compara a coluna da
estrela com a coluna do pacman na posicao antiga

                      JNZ  dont_touch_2

                      MOV  R5, JA_FOSTE_STAR                    ;  se  as  posicoes
coincidirem, inicializamos o rregisto com a variavel JA_FOSTE_STAR

                      MOV  R6, 4                                ; inicializa o registo
com  o numero da estrela que foi comida pelo pacman

                      MOVB [R5], R6



dont_touch_2:         POP  R6

                      POP  R5

                      POP  R4

                      POP  R3

                      POP  R2

                      POP  R1

                      POP  R0

                      RET


; * * * * * * * * * * * star_gone * * * * * * * * * * * * * * * * * * * * * * * * *

; *

; * Descricao: Rotina que verifica se uma estrela foi comida e incrementa

; *            o display de 7 segmentos.

; *

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


star_gone:            PUSH R0
```

```
                        PUSH R1

                        PUSH R2

                        PUSH R3

                        PUSH R4

                        PUSH R6


                        MOV  R0, JA_FOSTE_STAR

                        MOVB R0, [R0]


STAR_1:                 MOV  R1, 1

                        CMP  R0, R1                             ;   verifica   se   a
estrela 1 ja foi comida

                        JNZ  STAR_2

                        MOV  R3, STARS

                        MOVB R3, [R3]

                        CMP  R3, R1                             ;   verifica   se   a
estrela 1 ja tinha sido comida (so pode ser comida 1 vez)

                        JZ   STAR_2

                        MOV  R3, STARS                          ; guarda no registo o
estado das estrelas (comidas/1 ou nao/0)

                        MOVB [R3], R1

                        MOV  R3, STARS_EATEN

                        MOVB R4, [R3]

                        INC  R4                                 ; incrementa o numero
de estrelas comidas

                        MOVB [R3], R4                           ; coloca na variavel
STARS a estrela comida


STAR_2:                 MOV  R1, 2

                        CMP  R0, R1                             ;   verifica   se   a
estrela 2 ja foi comida

                        JNZ  STAR_3

                        MOV  R3, STARS

                        ADD  R3, 1

                        MOVB R3, [R3]

                        MOV  R1, 1

                        CMP  R3, R1                             ;   verifica   se   a
estrela 2 ja tinha sido comida (so pode ser comida 1 vez)
```

```
                    JZ   STAR_3

                    MOV  R3, STARS                          ; guarda no registo o
estado das estrelas (comidas/1 ou nao/0)

                    ADD  R3, 1

                    MOVB [R3], R1

                    MOV  R3, STARS_EATEN

                    MOVB R4, [R3]

                    INC  R4                                 ; incrementa o numero
de estrelas comidas

                    MOVB [R3], R4                           ; coloca na variavel
STARS a estrela comida



STAR_3:             MOV  R1, 3

                    CMP  R0, R1                             ;   verifica   se   a
estrela 3 ja foi comida

                    JNZ  STAR_4

                    MOV  R3, STARS

                    ADD  R3, 2

                    MOVB R3, [R3]

                    MOV  R1, 1

                    CMP  R3, R1                             ;   verifica   se   a
estrela 3 ja tinha sido comida (so pode ser comida 1 vez)

                    JZ   STAR_4

                    MOV  R3, STARS                          ; guarda no registo o
estado das estrelas (comidas/1 ou nao/0)

                    ADD  R3, 2

                    MOVB [R3], R1

                    MOV  R3, STARS_EATEN

                    MOVB R4, [R3]

                    INC  R4                                 ; incrementa o numero
de estrelas comidas

                    MOVB [R3], R4                           ; coloca na variavel
STARS a estrela comida



STAR_4:             MOV  R1, 4

                    CMP  R0, R1                             ;   verifica   se   a
estrela 4 ja foi comida

                    JNZ  NO_STAR

                    MOV  R3, STARS
```

```
                        ADD  R3, 3

                        MOVB R3, [R3]

                        MOV  R1, 1

                        CMP  R3, R1                         ;   verifica   se   a
estrela 4 ja tinha sido comida (so pode ser comida 1 vez)

                        JZ   NO_STAR

                        MOV  R3, STARS                      ; guarda no registo o
estado das estrelas (comidas/1 ou nao/0)

                        ADD  R3, 3

                        MOVB [R3], R1

                        MOV  R3, STARS_EATEN

                        MOVB R4, [R3]

                        INC  R4                             ; incrementa o numero
de estrelas comidas

                        MOVB [R3], R4                       ; coloca na variavel
STARS a estrela comida



NO_STAR:                MOV  R0, JA_FOSTE_STAR

                        MOV  R1, 0

                        MOVB [R0], R1                       ;   mete   a   variavel
JA_FOSTE_STAR a 1



                        MOV  R0, STARS_EATEN

                        MOVB R4, [R0]

                        MOV  R3, POUT3_ADDR

                        MOVB [R3], R4                       ;   manda   para   o
periferico POUT_3 o numero de estrelas comidas

                        MOV  R1, 4

                        CMP  R4, R1                         ;   verifica   se   as
quatro estrelas foram comidas

                        JNZ  SKIP_STAR



                        MOV  R0, JA_FOSTE

                        MOV  R1, 1

                        MOVB [R0], R1                       ;   mete   a   variavel
JA_FOSTE a 1

                        MOV  R6, 1

                        CALL the_end
```

```
                    MOV  R4, 0
                    MOVB [R3], R4                    ;    reinicializa    o
periferico POUT_3 a 0


SKIP_STAR:          POP  R6
                    POP  R4
                    POP  R3
                    POP  R2
                    POP  R1
                    POP  R0
                    RET


; * * * * * * * * * * * time_over * * * * * * * * * * * * * * * * * * * * * * * * *
; *
; * Descricao: Rotina que acaba o jogo quando o tempo chega a 99.
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


time_over:          PUSH R0
                    PUSH R1


                    MOV  R0, time_out
                    MOVB R0, [R0]
                    MOV  R1, 1
                    CMP  R0, R1                      ; verifica se o tempo
ja chegou ao fim
                    JNZ  RUN
                    MOV  R0, JA_FOSTE
                    MOV  R1, 1
                    MOVB [R0], R1                    ;  mete  a  variavel
JA_FOSTE a 1
                    CALL the_end                     ; termina o jogo
                    MOV  R0, time_out
                    MOV  R1, 0
                    MOVB [R0], R1                    ;  mete  a  variavel
time_out a 0
```

```
RUN:                    POP  R1
                        POP  R0
                        RET

; * * * * * * * * * * * another_diretion * * * * * * * * * * * * * * * * * * * * *
; *
; * Descricao: Rotina que escolhe outra direcao no caso da direcao escolhida
; *            inicialmente nao se conseguir realizar.
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

another_diretion:       PUSH  R7                                    ;DIR LIN
                        PUSH  R8                                    ;DIR COL
                        PUSH  R9
                        PUSH  R2                                    ;COLGHOST
                        PUSH  R10
                        PUSH  R3
                        PUSH  R4
                        PUSH  R5
                        PUSH  R6

                        MOV   R3, R6
                        MOV   R6, [R3]
                        ADD   R3, 2
                        MOV   R2, [R3]

_1:                     MOV  R0, R7
                        CMP  R0, 0
                        JZ   _2
                        MOV  R1, 0
                        CALL inverso
                        MOV  R3, 1
                        CMP  R9, R3
```

```
                    JZ   _2
                    ADD  R0, R6
                    ADD  R1, R2
                    MOV  R3, limits_mask
                    MOV  R4, GHOST_SIZE_L
                    MOV  R5, GHOST_SIZE_C
                    CALL Compare_limits
                    CMP  R10, 0
                    JZ   new_diretion

_2:                 MOV  R0, 0
                    MOV  R1, R8
                    CMP  R1, 0
                    JZ   _3
                    CALL inverso
                    MOV  R3, 1
                    CMP  R9, R3
                    JZ   _3
                    ADD  R0, R6
                    ADD  R1, R2
                    MOV  R3, limits_mask
                    MOV  R4, GHOST_SIZE_L
                    MOV  R5, GHOST_SIZE_C
                    CALL Compare_limits
                    CMP  R10, 0
                    JZ   new_diretion

_3:                 MOV  R0, 0
                    MOV  R1, R8
                    CMP  R1, 1
                    JZ   pos
                    JMP  cont_
pos:                MOV  R1, -1
cont_:              CALL inverso
```

```
                    MOV  R3, 1
                    CMP  R9, R3
                    JZ   _4
                    ADD  R0, R6
                    ADD  R1, R2
                    MOV  R3, limits_mask
                    MOV  R4, GHOST_SIZE_L
                    MOV  R5, GHOST_SIZE_C
                    CALL Compare_limits
                    CMP  R10, 0
                    JZ   new_diretion

_4:                 MOV  R0, R7
                    MOV  R1, 0
                    CMP  R0, -1
                    JZ   _neg
                    MOV  R0, 1
_neg:               CALL inverso
                    MOV  R3, 1
                    CMP  R9, R3
                    JZ   _5
                    ADD  R0, R6
                    ADD  R1, R2
                    MOV  R3, limits_mask
                    MOV  R4, GHOST_SIZE_L
                    MOV  R5, GHOST_SIZE_C
                    CALL Compare_limits
                    CMP  R10, 0
                    JZ   new_diretion

_5:                 MOV  R0, 1
                    MOV  R1, 1
                    CALL inverso
                    MOV  R3, 1
```

```
                    CMP  R9, R3
                    JZ   _6
                    ADD  R0, R6
                    ADD  R1, R2
                    MOV  R3, limits_mask
                    MOV  R4, GHOST_SIZE_L
                    MOV  R5, GHOST_SIZE_C
                    CALL Compare_limits
                    CMP  R10, 0
                    JZ   new_diretion

_6:                 MOV  R0, -1
                    MOV  R1, -1
                    CALL inverso
                    MOV  R3, 1
                    CMP  R9, R3
                    JZ   _7
                    ADD  R0, R6
                    ADD  R1, R2
                    MOV  R3, limits_mask
                    MOV  R4, GHOST_SIZE_L
                    MOV  R5, GHOST_SIZE_C
                    CALL Compare_limits
                    CMP  R10, 0
                    JZ   new_diretion

_7:                 MOV  R0, 1
                    MOV  R1, -1
                    CALL inverso
                    MOV  R3, 1
                    CMP  R9, R3
                    JZ   _8
                    ADD  R0, R6
                    ADD  R1, R2
```

```
                        MOV  R3, limits_mask
                        MOV  R4, GHOST_SIZE_L
                        MOV  R5, GHOST_SIZE_C
                        CALL Compare_limits
                        CMP  R10, 0
                        JZ   new_diretion

_8:                     MOV  R0, -1
                        MOV  R1, 1
                        CALL inverso
                        MOV  R3, 1
                        CMP  R9, R3
                        JZ   new_diretion
                        ADD  R0, R6
                        ADD  R1, R2
                        MOV  R3, limits_mask
                        MOV  R4, GHOST_SIZE_L
                        MOV  R5, GHOST_SIZE_C
                        CALL Compare_limits

new_diretion:           POP   R6
                        POP   R5
                        POP   R4
                        POP   R3
                        POP   R10
                        POP   R2
                        POP   R9
                        POP   R8
                        POP   R7
                        RET

; * * * * * * * * * * * inverso * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *
; * Descricao: Rotina que quando se esta a escolher a proxima direcao do
```

```
; *            fantasma, verifica se duas direcoes sao opostas.
; *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

inverso:            PUSH  R0
                    PUSH  R1
                    PUSH  R7                                    ;DIR LIN
                    PUSH  R8                                    ;DIR COL
                    PUSH  R2

                    MOV   R2, 1
                    CMP   R0, R2
                    JNZ   negative
                    MOV   R0, -1
                    JMP   _col_
negative:           MOV   R2, -1
                    CMP   R0, R2
                    JNZ   _col_
                    MOV   R0, 1
_col_:              MOV   R2, 1
                    CMP   R1, R2
                    JNZ   nega
                    MOV   R1, -1
                    JMP   suivant
nega:               MOV   R2, -1
                    CMP   R1, R2
                    JNZ   suivant
                    MOV   R1, 1

suivant:            CMP   R0, R7
                    JNZ   col_una
                    MOV   R9, 1
col_una:            CMP   R1, R8
                    JZ    _skip_
```

```
                        MOV   R9, 0


_skip_:                 POP   R2
                        POP   R8
                        POP   R7
                        POP   R1
                        POP   R0
                        RET
```