TÉCNICO LISBOA

Introdução à Arquitetura de Computadores

LEIC IST-UL

RELATÓRIO - PROJETO BEYOND MARS

Grupo 25:

João Miguel Figueiredo Barros	106063
Guilherme Vaz Rocha	106171
Gabriel dos Reis Fonseca Castelo Ferreira	107030

2022/2023 – 2º Semester, P4

Contents

1 Manual de Utilizador

O mapeamento das teclas é o seguinte:

Tecla 0:	Atira Sonda Esquerda
Tecla 1:	Atira Sonda Meio
Tecla 2:	Atira Sonda Direita
Tecla C:	Começa/Renicia o jogo
Tecla D:	Pausa/Resume o jogo
Tecla E:	Termina o jogo

2 Arquitetura

2.1 Processos

O projeto recorre ao uso de processos cooperativos, são um total de cinco a contar com o principal, que desempenham as suas respetivas tarefas.

2.1.1 MAIN

Este processo é reponsável por executar os comandos associados a cada tecla. Foi feito um mapa de rotinas para cada estado do jogo de modo a evitar ações indesejadas (*E.g.:* ativação de uma sonda no estado *pausado*). O mapa em efeito está escrito na variável `MAPA_ROTINAS`, e pode ser uma das três seguintes, representando o estado do jogo:

- `MAPA_ROTINAS_TERMINADO`: Durante o ínicio ou depois do fim de um jogo
- `MAPA_ROTINAS_JOGO`: Durante o jogo
- `MAPA_ROTINAS_PAUSA`: Durante pausa

Depois a partir do valor da tecla em questão, é possível chamar uma rotina com uma complexidade de tempo constante $\theta(1)$, e de espaço linear $\theta(n)$ ao ir buscar o endereço da rotina a uma tabela de rotinas usando o valor da tecla como índice, da seguinte forma:

Código 2.1: Escolha da rotina

```
SHL   R0, 1     ; WORD são 2 bytes (Multiplica incremento por 2)
ADD   R1, R0    ; salta para o comando correspondente
MOV   R0, [R1] ; R0 = endereço da rotina a executar
CALL  R0        ; call lista_rotinas[valor]
```

2.1.2 TECLADO

O processo `teclado` trata do varrimento das teclas linha a linha, escrevendo para a variável LOCK `tecla_premida`, que resulta no desbloqueio do processo `MAIN` para qual é deferido o que fazer com a tecla premida.

No ciclo `espera_tecla` para evitar o uso de mais uma condição, em vez de ser usado um `SHL` ou `SHR` e o R1 ter de ser reposto a cada 4 ciclos de volta ao estado inicial, é usado um `ROL` e o valor inicial de `R1` é 1111H, rodando entre 1111H-2222H-4444H-8888H-1111H de forma a não precisar de nenhuma condição. De facto, isto vai tornar com que o PEPE-16 escreva para o periférico das linhas do teclado os bytes 11H em vez de 01H, 22H em vez de 02H, etcetera, mas o chip Teclado apenas tem uma entrada de 4 bits de largura, logo apenas o nibble low é relevante.

Código 2.2: Ciclo espera_tecla

```
espera_tecla: ; Ciclo enquanto a tecla NÃO estiver a ser premida
  ...
  ROL   R1,   1  ; Roda o valor da linha
  MOVB  [R2], R1 ; Escreve no periférico das linhas do teclado
  ...
```

2.1.3 DISPLAY

O processo `display` lê a variável LOCK `valor_display`, e quando esta é atualizada converte o valor de hexadecimal para decimal e representa-o no display.

2.1.4 GERA_ASTERÓIDES

Para evitar a sobreposição de asteróides, o processo `gera_asteróides` bloqueia a ler a variável LOCK `atualiza_ecrã` várias vezes de forma a apenas gerar um asteróide a cada poucas atualizações de ecrã.

Para a geração do asteróide é lido um número aleatório do periférico de leitura do teclado como descrito no guião e depois usando MOD é selecionado um elemento de uma lista que guarda as posições e incremento direções possíveis, e de outra lista se for minerável ou não (lista com 4 elementos, 1 dele sendo o minerável).

2.1.5 GRÁFICOS

As interrupções estão desativadas para prevenir artefactos e movimento de objetos durante verificação de colisões. O processo está dividido em 4 partes:

- `gráficos_painel:` Desenha o painel da nave, é executado só ao ínicio do jogo
- `gráficos_luzes:` Desenha as luzes no painel a cada 200ms
- `gráficos_asteroides:` Desenha os asteroides ativos, verifica se estão dentro dos limites do ecrã e trata das colisões asteróide-sonda e asteróide-nave.
- `gráficos_sondas:` Desenha as sondas as ativas, e verifica se estão dentro dos limites do ecrã.

2.2 Interrupções

Foi optado que as interrupções fazem determinadas ações diretamente, invés de apenas acionar uma variável LOCK, para evitar fazer mais quatro processos, uma vez que isso significaria uma desnecessária alocação de memória com mais 4 STACKs, e uma dupla verificação de se as interrupções já foram ativas, visto que o processador já faz isso. Isto também traz mais prioridade às interrupções e possivelmente evita interrupções serem ativas duas vezes e o programa apenas reagir a uma dessas ativações.

Para as funcionalidades de pausa e terminar o jogo, todas as rotinas de interrupção antes de executar as suas ações verificam se a variável `estado_jogo` é == 0, e sendo retornam imediatamente, de forna a não executar as suas funcionalidades normais.

3 Comentários

Esta secção inclui detalhes sobre a abordagem tomada na implementação de algumas funcionalidades, cujo funcionamento ou não é óbvio e pode ter tomado uma direção diferente do que sugerido nos guiões de laboratório, ou que suscitam algum interesse por outra razão.

3.1 Rotina valor_teclado

No processo de obtenção do valor da tecla premida a partir dos valores de linha e de coluna, é preciso converter primeiro estes valores de 1-2-4-8 para o seu bit ativo: 0-1-2-3.

Olhemos então para a transformação que se efetua:

X		Ret
0001b 1	$\rightarrow$	0000b 0
0010b 2	$\rightarrow$	0001b 1
0100b 4	$\rightarrow$	0010b 2
1000b 8	$\rightarrow$	0011b 3

Isto trata-se obviamente da operação $Ret = log_2(X)$ (porque X é potência de 2) ou o número de *trailing zeroes*, e é possível implementar essa operação para fazer a conversão, como indicado no guião do lab3, tendo um loop em que é incrementado um registo e feito SHR até sair um 1 pelo Carry.

Mas olhando mais perto, como são apenas estes valores, é possível fazer algo mais eficiente que não recorra ao uso de loops. Reparemos que de 1,2,4 para 0,1,2 apenas ocorre um SHR, mas um SHR de 8 é 4 e queremos 3. O que é possível fazer para apenas subtrair 1 quando acabamos com 4 é reparar que fazendo um SHR, 2 de 0,1,2,4 obtemos 0,0,0,1. Então o resultado do primeiro SHR seguido da subtração desse mesmo resultado após SHR, 2 é igual ao que a conversão quer devolver.

O processo pode então ser descrito da seguinte forma:

X		Y		Z		Ret
Input		SHR X, 1		SHR Y, 2		Y-Z
0001b 1	→	0000b 0	→	0000b 0	→	0000b 0
0010b 2	→	0001b 1	→	0000b 0	→	0001b 1
0100b 4	→	0010b 2	→	0000b 0	→	0010b 2
1000b 8	→	0100b 4	→	0001b 1	→	0011b 3

Na linguagem de Assembly P4, sendo RX um registo onde least significant nibble é X e o resto 0, ficam apenas 4 instruções:

Código 3.1: Conversão 1,2,4,8 para 0,1,2,3

```
SHR RX, 1   ; RX = Y
MOV R2, RX  ; R2 = Y
SHR R2, 2   ; R2 = Z
SUB RX, R2  ; RX = Y-Z
```