Arquitectura e Sistemas de Computadores I

Interrupções no 80x86. Programação de entradas/saídas na porta série usando interrupções.

Objectivos: Este texto introduz o mecanismo de interrupções para as arquitecturas *i80x86*, em particular para o tratamento de I/O. A porta série é usada como exemplo.

1 O mecanismo de interrupções do Intel 80x86

Vector de interrupções ♦ Os processadores 80x86 assumem que as primeiras posições da memória física contêm um vector de 256 posições — **vector de interrupções**. Cada uma destas posições corresponde a um *tipo de interrupção* e ocupa 4 bytes. Estes bytes contêm um endereço *com 32 bits* da rotina a ser chamada quando ocorre uma interrupção desse tipo. Este tipo de rotina é designado por *rotina de tratamento*, *de serviço*, ou *de atendimento da interrupção*. Parte do vector de interrupções é apresentado na figura 1, tal como existe nos IBM/PC e compatíveis.

nº da Int.	Causa	Tipo	End. memória
0	divisão por zero	Interna	`000h - 003h`
1	exec. passo a passo	Interna	`004h - 007h`
2	N.M.I.	Externa	`008h - 00bh`
3	debug	Interna	`00ch - 010h`
...	...	...	`...`
11	**COM2 (IRQ3)**	**Externa**	**`02ch - 02fh`**
12	**COM1 (IRQ4)**	**Externa**	**`030h - 033h`**
...	...	...	`...`
33 (21h)	chamada ao DOS	Software	`084h - 087h`
...	...	...	`...`
255	...	...	`3fch - 3ffh`

Figura 1: Parte do vector de interrupções

Tipos de interrupções ♦ As interrupções podem ter diversas origens:

- interrupção por *software* (usando a instrução `INT`; ex: `INT 21h`);
- condição excepcional na execução de uma instrução (por exemplo uma divisão por zero quando executa a instrução `DIV`);
- uma interrupção exterior ao processador. Existem duas linhas externas de interrupção do processador (ver figura 2): a linha ***NMI*** (*Non Maskable Interrupt*) e a linha ***INTR*** (*Interrupt Request*) proveniente do controlador de interrupções (***PIC*** — *Programmable Interrupt Controler*). A interrupção NMI indica a ocorrência de erros no *hardware*, e as interrupções recebidas por essa linha não podem ser inibidas

(i.e. não podem ser ignoradas). Pelo contrário, as interrupções submetidas pela linha INTR podem ser ignoradas pelo CPU.

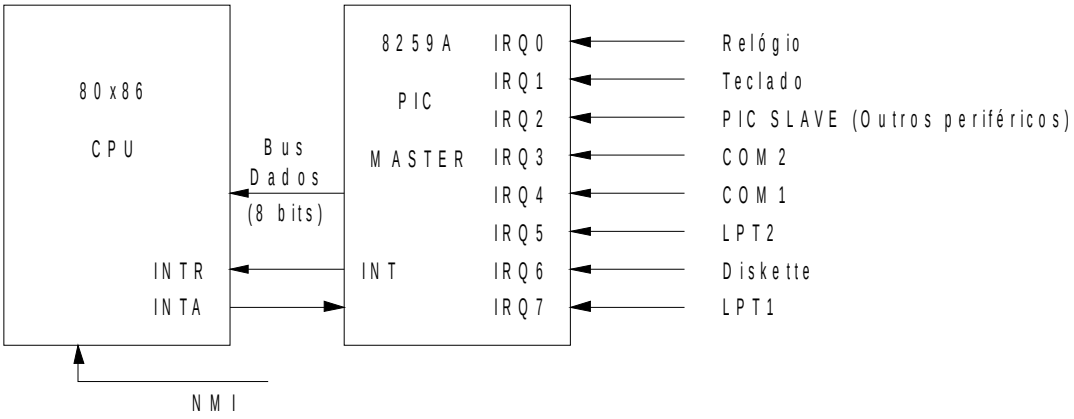

Figura 2: Controlador de Interrupções e CPU

Ignorar/permitir interrupções pela linha INTR♦ A instrução *assembly* **`cli`** faz com que o CPU ignore as interrupções externas provenientes do *PIC*. Após a instrução *assembly* **`sti`** o CPU passa de novo a atender as interrupções submetidas pela linha INTR.

Número da interrupção ♦ Nas interrupções por *software*, o *número* (ou **código**) *da interrupção* é dado explicitamente na instrução (p. ex. `int 21h`). No caso das condições excepcionais existe um código pré-definido para cada caso (p.ex. a interrupção número zero ocorre na execução de `DIV` se o divisor for zero). A interrupção NMI tem o número 2. No caso das interrupções externas (linha INTR), o **código (ou número) é indicado pelo PIC** através do *bus* de dados.

Atendimento da interrupção♦ A ***aceitação*** de uma interrupção pelo CPU é equivalente à sequência:

1. as flags são guardadas na pilha
2. todas as interrupções são inibidas (passam a ser ignoradas)
3. é efectuado um *call* usando o endereço que está no vector de interrupções na posição correspondente ao número da interrupção (endereço: 4xnum.intr.)

A instrução **`iret`** (*interrupt return*) tem de ser usada para o ***retorno de uma rotina de tratamento de interrupções***, pois além de desempilhar o endereço de retorno (deslocamento e segmento), é também necessário repor o registo das *flags* usando a cópia que está na pilha.

Como se pode observar desta descrição, sempre que o CPU atende uma interrupção desliga as interrupções externas. Para permitir de novo a ocorrência desse tipo de interrupções a ***interrupt flag*** deve ser colocada a **1**: a) com a instrução **`sti`** (por exemplo quando se quer continuar a atender essas interrupções, mesmo que a rotina de atendimento ainda não tenha terminado); b) com a instrução **`iret`** que, ao repôr o estado anterior das *flags*, volta a permitir a aceitação de interrupções.

2 Interrupções externas

Níveis de interrupção ♦ O ***controlador de interrupções (PIC)*** é um dos componentes centrais do IBM-PC e permite que os vários periféricos interrompam o CPU.
O *PIC* associa a cada uma das suas linhas de entrada (***IRQ***) um código de interrupção (o número da interrupção) que é transmitido ao processador através do *bus* de dados.
As várias linhas de entrada possuem diferentes prioridades, também chamadas de *níveis de interrupção*, sendo o IRQ 0 o mais prioritário e o IRQ 7 o menos. O controlador só envia novas interrupções ao CPU quando este não está a tratar nenhuma interrupção de prioridade igual ou superior. Mesmo que dentro de uma rotina de tratamento se liguem as interrupções (`sti`), esta rotina só será interrompida por outras de nível superior, ou seja de *IRQ* **menor**. Para que o PIC saiba que o tratamento de uma interrupção terminou tem de ser explicitamente informado dando-lhe um comando (ver mais à frente).

Programação do controlador PIC ♦ O *PIC* ocupa os endereços 0020h-23h do *mapa de I/O* (ver figura 1 do enunciado do trabalho 4) e a sua utilização[1] envolve dois tipos de comandos:

- Comandos de inicialização, que definem o modo de operação do PIC, incluindo a associação entre os níveis de interrupção (IRQs) e códigos de interrupção. Estes comandos são executados pela BIOS, durante o arranque da máquina e não vão ser discutidos.
- Comandos de operação, usados durante o normal funcionamento do computador.

Este último tipo de comandos permite assinalar ao controlador que a interrupção foi tratada e mascarar ou desmascarar (ligar ou desligar) certas entradas de IRQ. Para tal são usados 2 portos de *I/O*:

Registo de comandos no endereço 20H

Registo de máscara no endereço 21H

No registo de comandos apenas se pode escrever. O único comando que interessa referir é o ***EOI*** (*End Of Interrupt*, valor `20h`) que assinala ao controlador que a interrupção pendente foi tratada, para voltar a permitir ao PIC a emissão de novas interrupções de qualquer prioridade ou nível.

O registo de máscara pode ser lido e escrito e permite inibir ou activar certos níveis de interrupção (IRQs); um bit a 1 indica que esse nível está inibido (desligado). Por exemplo: se o registo de máscara tem o valor 10111110, isto significa que o PIC apenas vai aceitar interrupções de IRQ 6 e IRQ 0.
Se se pretender modificar o estado em relação a um destes níveis é recomendável desligar as interrupções (através da instrução `cli`) sempre que se efectua uma operação de escrita neste registo, e voltar a ligar (`sti`) logo de seguida.

[1] Os IBM/PC e compatíveis possuem dois PIC em cascata para permitir mais de 8 fontes de interrupção. Neste documento só nos referimos ao que se encontra directamente ligado ao CPU (o *maste*r)

3 Interrupções da porta série

Na ***UART***, além dos registos já referidos no documento anterior, há outros **registos** relacionados com a forma como o controlador da porta série pode interromper o CPU:

- A colocação a 1 do bit 3 do **registo *MCR*** indica que a UART deve gerar interrupções.

- O **registo *IER*** especifica em que situações a UART deve interromper o CPU:

  bit 0 — interrupção de recepção (chegou um carácter)

  bit 1 — interrupção de transmissão (THR ficou vazio)

- Quando ocorre uma interrupção, podemos consultar o **registo *IIR*** para identificar a causa da interrupção (quando ambas estão ligadas no IER):

  bit 0 — Esta porta série causou uma interrupção que aguarda tratamento.

  bits 1-2 — A causa da interrupção: 01B - registo de transmissão (THR) vazio; 10B carácter disponível no registo de recepção (RBR); 11B erro na recepção (consultar LSR para saber o motivo).

Para ver a relação entre IRQs, números de interrupção (ou códigos) e a porta série, consultar as figuras 1 e 2.

4 O Turbo C e as interrupções

Apresentam-se agora algumas das extensões à linguagem C que são suportadas por este compilador, e algumas funções das bibliotecas do Turbo C que são úteis neste trabalho.

Rotina de tratamento de interrupção ♦ O compilador Turbo C permite a utilização de uma palavra chave **`interrupt`** para indicar que determinada função vai ser chamada através de uma interrupção em vez de ser chamada pelo programa.
Esta palavra chave diz ao compilador para salvaguardar todos os registos no inicio e para terminar a função com a instrução **`iret`** em vez da tradicional instrução **`ret`** usada nas funções normais. Exemplo:

```
void interrupt f(void) {
     ...
}
```

Esta será traduzida pelo compilador em algo equivalente a:

```
f proc far
     push ax       ; salva todos os registos
     push bx
     push ...
     ...
     pop ...       ; repõe todos os registos
     pop bx
     pop ax
```

```
        iret    ; interrupt return
    f endp
```

Nota: Tenha em atenção que, para a compilação correcta de código C com rotinas de atendimento interrupções, **ambas as opções** "Use register variables" (em *Options->Compiler->Optimization*) e "Stack warning" (em *Options->Linker*) têm de estar a "**OFF**".

Definir ou alterar a rotina de atendimento de interrupção ♦ O Turbo C permite efectuar chamadas às funções no MS-DOS. Entre estas encontram-se as que permitem consultar e alterar os valores no vector de interrupções e podem ser acedidas pelas seguintes funções C:

```
void setvect(int interruptno, void interrupt (*isr)());
void interrupt (*getvect(int interruptno))();
```

Estas encontram-se declaradas no *header* **`dos.h`** e correspondem, respectivamente, às funções 25h e 35h do MS-DOS (`int 21h`).

Lembre-se que em C podemos obter o endereço de uma função usando simplesmente o seu nome, sem a invocar. Por exemplo, para alterar o tratamento de uma interrupção:

```
#include <dos.h>

    void interrupt f(void) {
        .../* nova função */ ...
    }
    . . .
    void interrupt (*old_isr)();

    . . .
        old_isr = getvect(numInt);
        setvect(numInt, f);
```

Ligar e desligar as interrupções♦ No *header* **`dos.h`** encontram-se também os protótipos de rotinas C correspondentes às instruções *assembly* **`cli`** e **`sti`**:

```
void disable ( void );   /* cli- Desliga as interrupções. */
void enable ( void );   /* sti- Liga as interrupções. */
```

5 Uso de interrupções na recepção de caracteres

Esta secção discute de que forma é que o mecanismo de interrupções pode ser usado na recepção de caracteres. Os caracteres recebidos serão guardados num *buffer* temporário para tratamento de forma assíncrona pelo programa principal. Evita-se assim que se percam os caracteres que cheguem enquanto o programa está a realizar outra actividade.

5.1 Recepção por interrupções

Agora a funcionalidade de receber o carácter da porta série passa a ser assegurada pela *rotina de tratamento de interrupções*, em vez do modelo de *espera activa* onde se ficava a verificar a chegada de mais um carácter. O carácter recebido deve ser colocado num *buffer circular*, e é função do programa principal retirar de lá os caracteres e escrevê-los no ficheiro.

5.2 Um buffer circular

Um *buffer* circular é uma estrutura de dados que implementa um reservatório temporário de dados, onde podem ser colocados e retirados elementos, usando uma política "FIFO" (First In First Out). Este pode ser implementado através de um vector e, pelo menos, dois índices — um indica a posição do primeiro elemento a sair (e que foi o primeiro a entrar) e outro que indica a posição onde vai entrar o próximo elemento (ou onde se encontra o último que entrou). Opcionalmente podemos manter o número de caracteres no *buffer*, para que seja mais fácil saber quando ele está cheio ou vazio.
Se tivermos a seguinte definição em C:

```
typedef struct {
    char buffer[MAX];
    int count;  /* Número de caracteres no buffer */
    int head;   /* Próximo carácter a ser retirado */
    int tail;   /* Posição onde se colocará
                   um novo caracter */
} queue;
```

Uma imagem do *buffer* para MAX=16, pode ser encontrada na figura seguinte. 'A' será o próximo carácter a ser retirado e o próximo carácter que entrar, será colocado na posição 1.

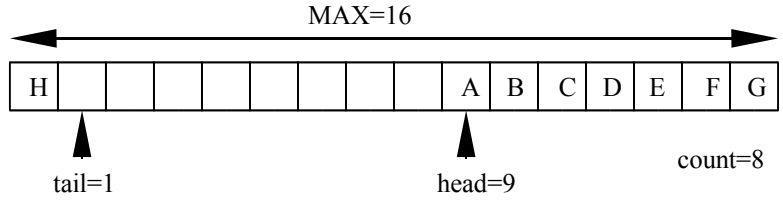

Figura 3: Exemplo de um *buffer* de 16 elementos

Repare que pode acontecer que este *buffer* fique cheio ou vazio, dependendo da velocidade relativa entre quem coloca e quem retira elementos.

6 Componentes para a implementação da recepção com interrupções

6.1 Preparação para a recepção por interrupções

A preparação para a recepção com interrupções inclui os seguintes passos:

a. programação do controlador de interrupções (PIC) para aceitar interrupções da porta série usada. Quando esta programação é feita as interrupções devem estar desligadas no CPU.

b. programação do controlador da porta série (UART) para gerar interrupções no caso da recepção de um carácter;
c. inclusão do processamento da chegada de um carácter numa rotina de tratamento de interrupções e seu registo no vector de interrupções . Antes de mudar o conteúdo de uma entrada do vector de interrupções o valor que lá se encontra deve ser salvaguardado; quando o seu programa acaba este valor salvaguardado deve ser reposto no vector. guarde sempre o estado das interrupções antes de cada uma destas alterações e de modo a que esse estado possa ser reposto no fim do seu programa;

6.2 Programação do *buffer circular*

Terão de ser definidas as seguintes operações sobre o *buffer* circular:
- inicialização do *buffer;*
- inserção de um elemento no *buffer*; suponha que quando esta rotina é chamada o *buffer* não está cheio;
- remoção de um elemento do *buffer*; suponha que quando esta rotina é chamada o *buffer* não está vazio;
- testa se o *buffer* está vazio
- testa se o *buffer* está cheio.

7 Anexo: *Inline Assembly*

[LEITURA OPCIONAL]

No Turbo C também podemos usar código *assembly* embutido ("***inline***") no código C desde que:
- o *pragma inline* esteja previamente definido;
- *cada linha assembly* seja *precedida* da instrução **`asm`**;

Assim, caso quisesse incluir as instruções *assembly* **`cli`** ("clear interrupts") e **`sti`** ("set interrupts"), em vez de utilizar as funções C correspondentes (**`disable()`** e **`enable()`**), faria:

```
#pragma inline

    void DisableInterrupts(void) {
              asm cli;
    }

    void EnableInterrupts(void) {
              asm sti;
    }
```