Arquitectura e Sistemas de Computadores I

Manipulação da porta série de um computador pessoal IBM PC

Objectivos: Este texto introduz o sistema de entradas/saídas do 80x86 e o seu uso na arquitectura dos IBM/PC e compatíveis para acesso aos periféricos. A porta série é usada como exemplo de um periférico, e o acesso a ela é feito por *espera activa*. O envio de um byte através da porta série faz-se com acesso directo ao *hardware* da porta série através de operações de *Entrada/Saída* (*In/Out*). É utilizada *espera activa* para determinar quando já pode ser transmitido o próximo carácter e para saber quando está disponível mais um carácter.

Este texto assume que os programas são implementados maioritariamente na linguagem C mas incluem partes em *assembly*.

1 Instruções para entrada e saída no *i80x86*

O acesso às interfaces dos periféricos existentes em computadores com uma arquitectura compatível com a família dos processadores *Intel 80x86* é realizado através do *mapa de Input/Output* (*I/O* ou seja, *entrada/saída*). Este mapa corresponde a um espaço de endereçamento separado da memória central, no qual cada periférico ocupa um conjunto de endereços (ver figura 1). Em cada conjunto, os endereços permitem o acesso a zonas de memória, ou registos, do periférico respectivo.

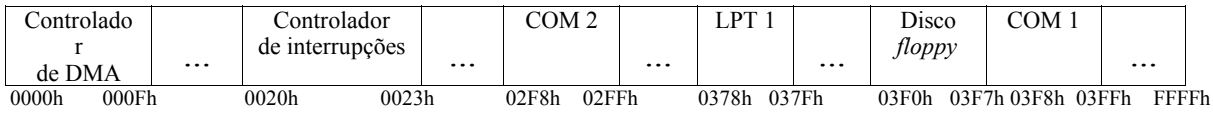

Figura 1: *Mapa de I/O* de um PC representando um subconjunto dos periféricos possíveis.

Operações *in* e *out* ♦ A família de processadores do *Intel 80x86* utiliza instruções específicas para aceder à memória ou aos registos dos periféricos (quer para a troca de dados, quer para lhes enviar comandos). Estas operações são chamadas de *In/Out* e designam-se, em *assembly*, pelas mnemónicas ***in*** e ***out***, respectivamente. Em ambos os casos, o valor a ler ou a escrever pode ser de 8 bits (1 byte) ou de 16 bits (2 bytes). Assim, a instrução ***in*** permite ler o valor de um porto (ou porta) num periférico para o registo `al` (ou `ax`) e a instrução ***out*** permite escrever o conteúdo do registo `al` (ou `ax`) para um porto de I/O.

De igual modo, os endereços dos portos de I/O podem ser dados por 8 ou 16 bits. Para indicar um porto com endereço de 16 bits é necessário usar o registo `dx` para conter esse endereço. Por sua vez, os endereços de 8 bits são colocados na própria instrução. A tabela seguinte representa as diferentes variantes no acesso aos portos:

Entrada	Saída	
in al, *end8bits* in al, dx	out *end8bits*, al out dx, al	Porto de 8 bits, endereçado por 8 bits Porto de 8 bits, endereçado por 16 bits
in ax, *end8bits* in ax, dx	out *end8bits*, ax out dx, ax	Porto de 16 bits, endereçado por 8 bits Porto de 16 bits, endereçado por 16 bits

2 Utilização das instruções de I/O em C

Como o seu programa C vai aceder a um periférico, é necessário disponibilizar funções para acesso a portos de I/O, Por exemplo, usando funções com os seguintes protótipos em C:

```
unsigned char inByte( unsigned int portid );

void outByte(    unsigned int portid,
                 unsigned char value );
```

Seguem-se exemplos da utilização em C dessas funções e as acções aproximadas em *assembly*:

C	*Assembly*

`outByte(0x3fc, 3);`	`mov dx, 3fCH`
	`mov al, 3`
	`out dx, al`

`i = inByte(40);`	`mov dx, 40` `in al, dx`

Dado que por vezes é necessário testar o estado de bits específicos de alguns portos, podemos tirar partido das operações lógicas binárias sobre inteiros, disponíveis na linguagem C e equivalentes às operações `and`, `or`, `xor` e `not` em *assembly*:

Op. em C	
`a & b`	`and` (**e** binário)
`a | b`	`or` (**ou** binário)
`a ^ b`	`xor` (**ou exclusivo** binário)
`~a`	`not` (**negação** binária)

Exemplos de testes de alguns portos I/O usando "máscaras de bits":

C	*Assembly*

`lsr = inByte(0x3fd);`
`thrEmpty = lsr & 0x20;`
	`mov dx, 3fDH`
	`in al, dx`
	`and al, 20H`

`i = inByte(64);` `j = i | 0x10;`	`in al, 64` `or al, 10H`

3 Comunicação série RS-232C

Na comunicação série os bits são transmitidos um a um (i.e. sequencialmente, como o nome indica), através de uma linha de dados. A ligação pode ser SIMPLEX (apenas num sentido), HALF-DUPLEX (em ambos os sentidos mas de forma alternada) ou FULL-DUPLEX (ambos os sentidos em simultâneo).

Os vários bits (0 e 1) são convertidos em sinais eléctricos e enviados de acordo com norma RS-232C. Esta norma define como efectuar este tipo de ligações entre dois equipamentos, como seja entre um computador e um MODEM (MOdulator/DEModulator) ou qualquer outro periférico (como por exemplo outro computador, uma impressora, um rato, etc).

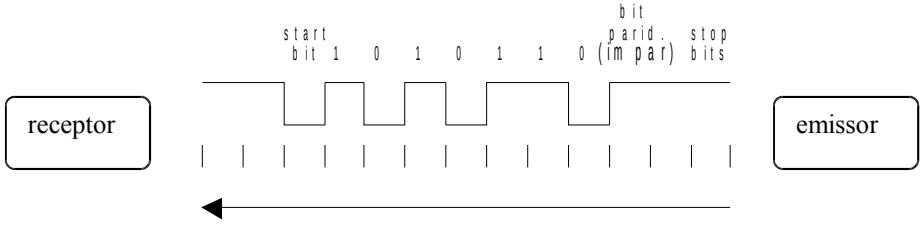

Figura 2: Exemplo de transmissão de 1010110B

A transmissão dos dados (podem ser entre 5 a 8 bits por caracter) é precedida por um *start* bit. Após os bits de dados são também enviados um *bit de controlo de paridade* (opcional) e um ou mais *stop bits* (ver figura 2). Antes da troca de dados, ambos os extremos da linha série têm de ser configurados para a mesma forma de comunicação e velocidade de transferência, para que se possam entender. A velocidade de transmissão é dada em bits/s (bits por segundo)[1] e é normalmente designada por *baud*.

3.1 Controlador da porta série nos IBM/PC: a UART

Cada porta série nos PCs é **controlada** por uma **UART** (Universal Asynchronous Receiver/Transmiter) integrado *NS 8250* ou compatível (por exemplo: 16450, 16550, 16C552). A porta série é operada através de um conjunto de **registos** neste controlador (ver figura 3), acessíveis por ***portos de I/O*** colocados a partir do endereço inicial definido pelo respectivo descodificador de endereços. Geralmente, o **endereço inicial** da **primeira porta série** é **3F8H** e o da **segunda** é **2F8H**. Os nomes atribuídos pelo sistema de operação MS-DOS às portas são respectivamente COM1: e COM2:.

[1] note que a taxa de transferência de informação (útil) é sempre menor que a velocidade máxima, mesmo que não ocorram erros, visto que é necessário enviar os bits start, stop e pode ainda ter o de paridade

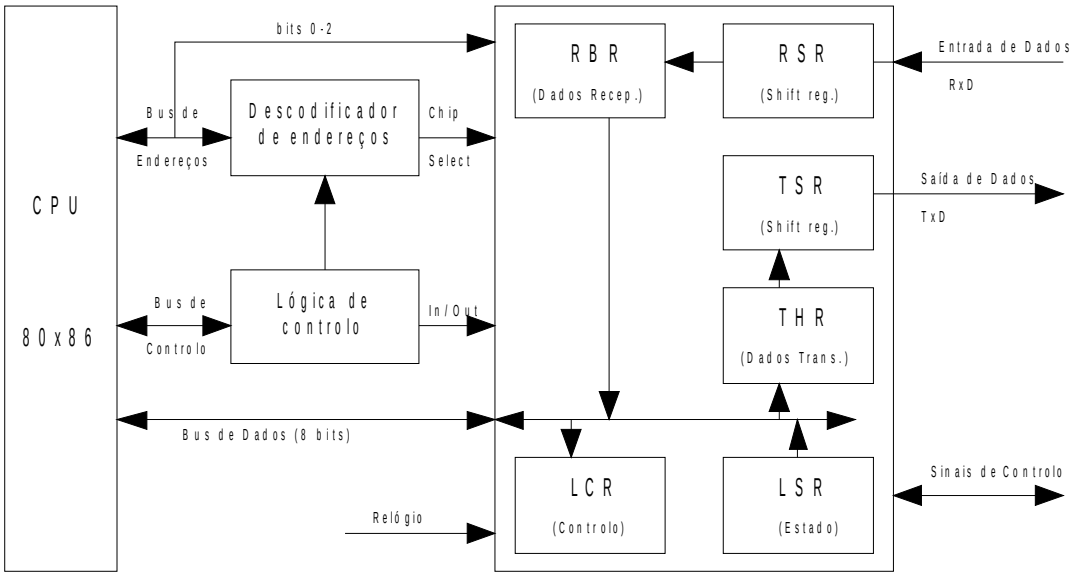

Figura 3: Interacção entre a UART, a linha série e o CPU

Acesso aos registos da COM*n*

♦ Na tabela seguinte são apresentados os registos e indicada a sua posição relativamente ao endereço inicial. Por exemplo: na ***primeira porta série*** (*COM1)*, o registo THR encontra-se no endereço **3F8H** (3F8+0), o registo LSR em 3FDH (3F8+5), etc; na ***segunda porta série*** (*COM2*), o respectivo registo LSR já se encontra no endereço **2FDH** (2F8+5).

Registo	Abrev.	Acesso	Posição
Dados - Transmissão	THR	Escrita	0
Dados - Recepção	RBR	Leitura	0
Interrupções	IER	Leit./Escr.	1
Causa da Interrupção	IIR	Leitura	2
Controlo da Linha	LCR	Leit./Esc.	3
Controlo do Modem	MCR	Leit./Esc.	4
Estado da Linha	LSR	Leitura	5
Estado do Modem	MSR	Leitura	6

De notar que os registos **RBR** e **THR** partilham o mesmo endereço de I/O — o primeiro é acedido através da instrução `in` e o segundo da instrução `out`.

- O registo **LCR** permite aceder aos parâmetros de funcionamento da porta série, como o tamanho dos "caracteres", paridade usada, velocidade de transmissão, etc. Esta facilidade não faz parte dos objectivos deste trabalho e, como tal, será aqui omitida.

O comando **MODE** do MS-DOS usa este registo para alterar os parâmetros de funcionamento da porta série pretendida. Resumidamente:

MODE COM*n*: *baud_rate*[[, *paridade*], *bits_dados*], *stop_bits*]

Em que:

n — Número da porta série (de 1 a 4)

baud_rate — velocidade de transmissão (ex: 150, 300, 600, 1200, 2400, 4800, 9600, 19200, ...)

paridade — n = sem paridade; o = paridade ímpar; e = paridade par

bits_dados — dimensão dos dados: de 5 a 8 bits

stop_bits — 1 ou 2 bits

- O registo de estado, **LSR**, dá informação sobre a recepção e transmissão dos dados. Cada um destes bits fica a 1 caso a condição enunciada seja verdadeira e a 0 se falsa:

bit 0 — Chegou um carácter que está disponível no registo RBR

bits 1-4 — Erros na recepção. Respectivamente: sobreposição, erro de paridade, erro de transferência, break.

bit 5 — O registo de transmissão (THR) está livre. (Podemos pedir o envio de um novo carácter)

bit 6 — Acabou de ser enviado o último bit pela porta série (TSR ficou vazio).

Quando é escrito um byte no registo de dados de transmissão (**THR**), este passa logo que possível para o *shift register* de envio (**TSR**) após o que começa a ser enviado pela linha série. Quando o bit 0 do registo de estado é colocado a 1, podemos ler um byte do registo de dados de recepção (**RBR**). O bit 0 do registo de estado passará a zero assim que essa leitura for efectuada.

- Os registos IER, IIR e MCR são apresentados noutro texto.

4 Biblioteca *standard* do C

A biblioteca *standard* da linguagem C inclui um grande conjunto de funções cujos protótipos (cabeçalhos das funções) se encontram declarados em vários ficheiros com a extensão .h. Estes são incluídos nos nossos programas fazendo uso da directiva `#include`. Chama-se a atenção que esta biblioteca é bastante diferente da *standard* do C++, como seria de esperar. Não existem classes!

Como ponto de partida para o seu trabalho, segue-se um pequeno conjunto de funções presentes no `stdio.h` que lhe são úteis. Para mais informação consulte qualquer livro da linguagem C, ou a documentação do próprio sistema de operação.

Referências a ficheiros (*streams*) prédefinidas (já abertos no início do programa):

```
FILE *stdin;
FILE *stdout;
```

Algumas funções para a manipulação de ficheiros:

```
FILE *fopen( char *filename, char *mode );
```

Abre o ficheiro de nome indicado devolvendo a referência para esse ficheiro. O `mode` indica para que queremos o ficheiro e deve incluir pelo menos um dos seguintes casos:

`"r"` – abrir para leitura

"w" – abrir para escrita (destrói qualquer conteúdo anterior).

```
fclose( FILE *stream );
```
Fecha o ficheiro indicado.

```
int fgetc( FILE *stream );
```
Lê um carácter do ficheiro indicado. Devolve o carácter lido ou a constante `EOF` se não há nada para ler.

```
fgets( char s[], int size, FILE *stream );
```
Lê para o *array* `s` um máximo de caracteres dados por `size` ou uma linha completa (até ao fim-de-linha '\n') do ficheiro indicado.

```
fputc( int char, FILE *stream );
```
Acrescenta o carácter dado ao ficheiro indicado.

```
fputs( char *string, FILE *stream );
```
Acrescenta a *string* dada mais um fim-de-linha ('\n') ao ficheiro indicado.

```
fprintf( FILE *stream, char *format, … );
```
Acrescenta uma *string* ao ficheiro indicado com base na formatação dada por `format` e os argumentos que se lhe seguem. Uma variante desta função para escrever no `stdout` é:

```
printf( char *format, ... );
```