

Trabalho prático N.º 3

Objetivos

- Conhecer a estrutura básica e o modo de configuração de um porto de I/O no microcontrolador PIC32.
- Configurar em *assembly* os portos de I/O do PIC32 e aceder para escrever/ler informação do exterior.

Introdução

O microcontrolador PIC32 disponibiliza vários portos de I/O, com várias dimensões (número de bits), identificados com as siglas **RB**, **RC**, **RD**, **RE**, **RF** e **RG**. Cada um dos bits de cada um destes portos pode ser configurado, por programação, como entrada ou saída. Um porto de I/O de **n** bits do PIC32 é então um conjunto de **n** portos de I/O de 1 bit, independentes. Por exemplo, o bit 0 do porto E (designado por **RE0**) pode ser configurado como entrada e o bit 1 do mesmo porto (**RE1**) ser configurado como saída.

A Figura 1 apresenta o diagrama de blocos simplificado de um porto de I/O de 1 bit, no PIC32.

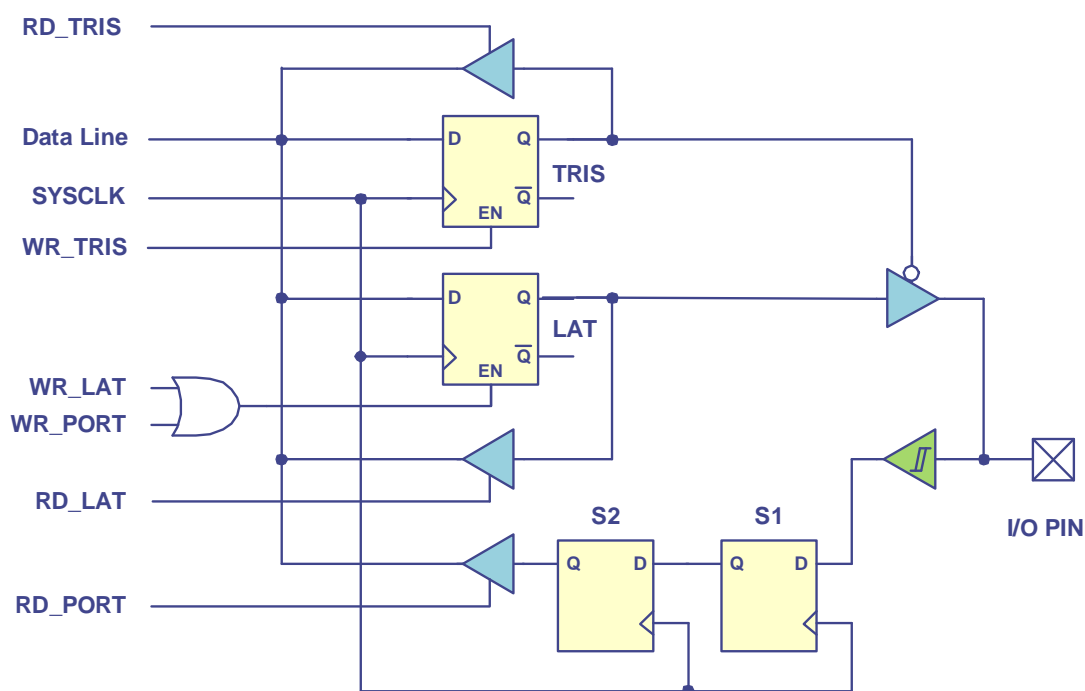


Figura 1. Diagrama de blocos simplificado de um porto de I/O de 1 bit, no PIC32.

A configuração de cada um dos bits de um porto como entrada ou saída é efetuada através dos flip-flops **TRIS_{xn}**, em que **x** é a letra identificativa do porto e **n** o bit desse porto que se pretende configurar. Por exemplo, para configurar o bit 0 do porto E (**RE0**) como entrada, o bit 0 do registo **TRISE** deve ser colocado a 1 (i.e. **TRISE0=1**); para configurar o bit 1 do porto E (**RE1**) como saída, o bit 1 do registo **TRISE** deve ser colocado a 0 (**TRISE1=0**).

Em termos de modelo de programação, cada porto tem associados 12 registos (4 de controlo e 8 de dados) de 32 bits em que apenas os 16 menos significativos (ou um subconjunto destes, dependendo do porto) têm informação útil. Desse conjunto de registos apenas usaremos 3: **TRIS_x**, **PORT_x** e **LAT_x**. O registo **TRIS_x** é usado para configurar os portos como entrada ou saída, o registo **LAT_x** é usado para escrever um valor num porto configurado como saída e o **PORT_x** para ler o valor de um porto configurado como entrada.

Os registos **TRISx**, **PORTx** e **LATx** estão mapeados no espaço de endereçamento de memória (área designada por **SFRs**), em endereços pré-definidos (disponíveis nos manuais do fabricante). O acesso para leitura e escrita desses registos é efetuado através das instruções **LW** e **SW** da arquitetura MIPS. Por exemplo, o endereço atribuído ao registo **TRISE** é **0xBF886100**, ao registo **PORTE** é **0xBF886110** e ao registo **LATE** é **0xBF886120**.

```
.equ SFR_BASE_HI, 0xBF88      # 16 MSbits of SFR area
.equ TRISE, 0x6100           # TRISE address is 0xBF886100
.equ PORTE, 0x6110           # PORTE address is 0xBF886110
.equ LATE, 0x6120            # LATE address is 0xBF886120
```

Uma vez que um registo incorpora a informação individual de todos portos de 1 bit a que esse registo diz respeito, a modificação do valor de 1 bit (ou conjuntos de bits) tem que ser efetuada sem alterar os restantes. Ou seja, para se alterar um conjunto restrito de bits nestes registos é obrigatório usar uma sequência de instruções do tipo "read-modify-write". Por exemplo, se se pretender configurar os bits 0 e 3 do porto E (**RE0** e **RE3**) como saídas teremos que colocar a 0 apenas os bits 0 e 3 do registo **TRISE**, mantendo os restantes inalterados. Isso traduz-se na seguinte sequência de instruções (em conjunto com as definições anteriores):

```
lui    $t1, SFR_BASE_HI      #
lw     $t2, TRISE($t1)        # READ (Mem_addr = 0xBF880000 + 0x6100)
andi   $t2, $t2, 0xFFFF6     # MODIFY (bit0=bit3=0 (0 means OUTPUT))
sw     $t2, TRISE($t1)        # WRITE (Write TRISE register)
```

Para colocar a saída do porto **RE0** a 0 e do **RE3** a 1 (sem alterar os restantes) pode fazer-se:

```
lui    $t1, SFR_BASE_HI      #
lw     $t2, LATE($t1)        # READ (Read LATE register)
andi   $t2, $t2, 0xFFFFE     # MODIFY (bit0 = 0)
ori    $t2, $t2, 8            # MODIFY (bit3 = 1)
sw     $t2, LATE($t1)        # WRITE (Write LATE register)
```

Como auxiliar de memória, note que:

- **TRIS** é relativo a *tri-state* ('0' => *tri-state off*, i.e., o porto não está no estado de alta impedância, ou seja, é um porto de saída; '1' => *tri-state on*, i.e., o porto está no estado de alta impedância, ou seja, é uma entrada);
- **PORT** diz respeito ao valor do porto de entrada;
- **LAT** refere-se a *latch*, i.e., ao registo que armazena o valor a enviar para as saídas.

Trabalho a realizar**Parte I**

A Figura 2 mostra parte do esquema elétrico de ligação dos LEDs e do *switch* que já montou na aula2.

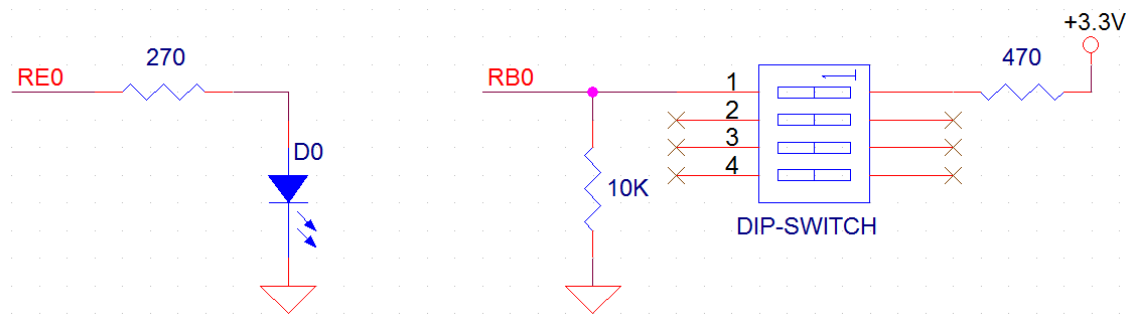


Figura 2. Ligação de um LED e um *switch* a portos do PIC32.

- Escreva e teste um programa em *assembly* que:
 - configure o porto **RE0** como saída e o porto **RB0** como entrada;
 - em ciclo infinito, leia o valor do porto de entrada e escreva esse valor no porto de saída (i.e., **RE0 = RB0**).
- Altere o programa anterior de modo a escrever no porto de saída o valor lido do porto de entrada, negado (i.e., **RE0 = RB0**).
- Configure agora os portos **RB0** a **RB3** como entradas e os portos **RE0** a **RE3** como saídas e, em ciclo infinito, faça as seguintes atribuições nas saídas:

$$\text{RE0} = \text{RB0}\backslash, \text{RE1} = \text{RB1}, \text{RE2} = \text{RB2} \text{ e } \text{RE3} = \text{RB3}\backslash$$
- Traduza para *assembly* o trecho de código seguinte¹, em que **delay()** é a função que implementou na aula anterior. Compile, transfira para a placa DETPIC32 e execute esse código.

```
void main(void)
{
    int v = 0;

    TRISE0 = 0;    // Configura o porto RE0 como saída
    while(1)
    {
        LATE0 = v; // Escreve v no bit 0 do porto E
        delay(500); // Atraso de 500ms
        v ^= 1;    // complementa o bit 0 de v (v = v xor 1)
    }
}
```

- Observe que o LED0 está a piscar com uma frequência de 1Hz.
- Altere o valor do atraso para 20 ms. Com a ponta do osciloscópio no porto **RE0** meça o tempo a 1 e o tempo a 0 do sinal de saída gerado pelo programa.
- Altere o valor do atraso para 10ms e repita a medição com o osciloscópio.

¹ No que respeita à configuração e ao acesso aos portos de I/O, o programa anterior não está escrito de forma compatível com o compilador de C que será utilizado nas aulas práticas de AC2. A forma correta de o fazer será descrita no trabalho prático n.º 4.

Parte II

1. Escreva um programa *assembly* que comece por configurar os portos **RE0**, **RE1**, **RE2** e **RE3** como saídas e os portos **RB0**, **RB1**, **RB2** e **RB3** como entradas (esquema representado na Figura 3).

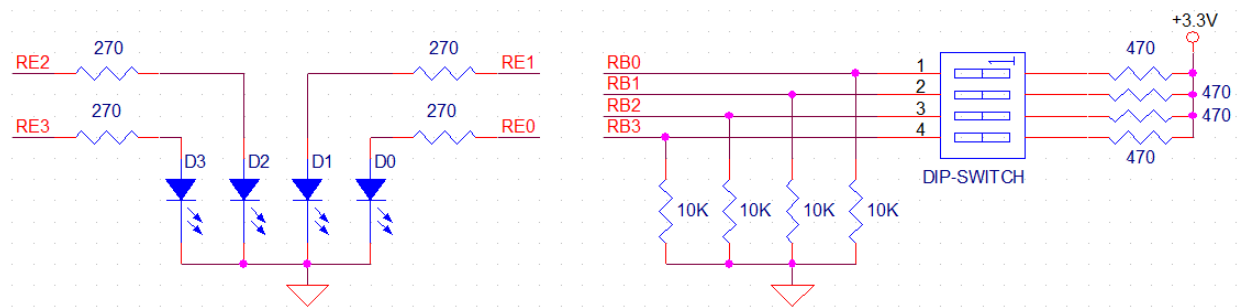


Figura 3. Ligação de 4 leds e um *dip-switch* de 4 posições a portos do PIC32

Escreva o código que realize as tarefas abaixo descritas (uma de cada vez):

- a) Contador binário crescente de 4 bits, atualizado com uma frequência de 1Hz (para controlar o tempo, utilize a subrotina `delay()` implementada na aula anterior).
- b) Contador binário decrescente de 4 bits, atualizado com uma frequência de 4Hz.
- c) Contador binário crescente/decrescente cujo comportamento depende do valor lido do porto de entrada **RB3**: se **RB3=1**, contador crescente; caso contrário contador decrescente; frequência de atualização de 2 Hz.
- d) Contador Johnson de 4 bits com deslocamento à esquerda (sequência: 0000, 0001, 0011, 0111, 1111, 1110, 1100, 1000, 0000, 0001, ...), com uma frequência de atualização de 1.5 Hz; para implementar este contador observe que o bit a introduzir na posição menos significativa quando se faz o deslocamento à esquerda corresponde ao valor negado que o bit mais significativo tinha na iteração anterior.
- e) Contador Johnson de 4 bits com deslocamento à direita (sequência: 0000, 1000, 1100, 1110, 1111, 0111, 0011, 0001, 0000, 1000, ...); frequência de atualização de 1.5 Hz (poderá usar um raciocínio análogo ao descrito na alínea anterior para a implementação deste contador).
- f) Contador Johnson de 4 bits com deslocamento à esquerda ou à direita, dependendo do valor lido do porto de entrada **RB2**: se **RB2=1**, deslocamento à esquerda; frequência de atualização de 1.5 Hz.
- g) Contador em anel de 4 bits (*ring counter*) com deslocamento à esquerda ou à direita, dependendo do valor lido do porto **RB1**: se **RB1=1**, deslocamento à esquerda. Frequência de atualização de 3 Hz (deslocamento à esquerda: 0001, 0010, 0100, 1000, 0001, ...).

Elementos de apoio

- PIC32 Family Reference Manual, Section 12 – I/O Ports.
- PIC32MX5XX/6XX/7XX, Family Data Sheet, Pág. 159 a 164.
- *Slides* das aulas teóricas.