Aula prática 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`**[1]. 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.

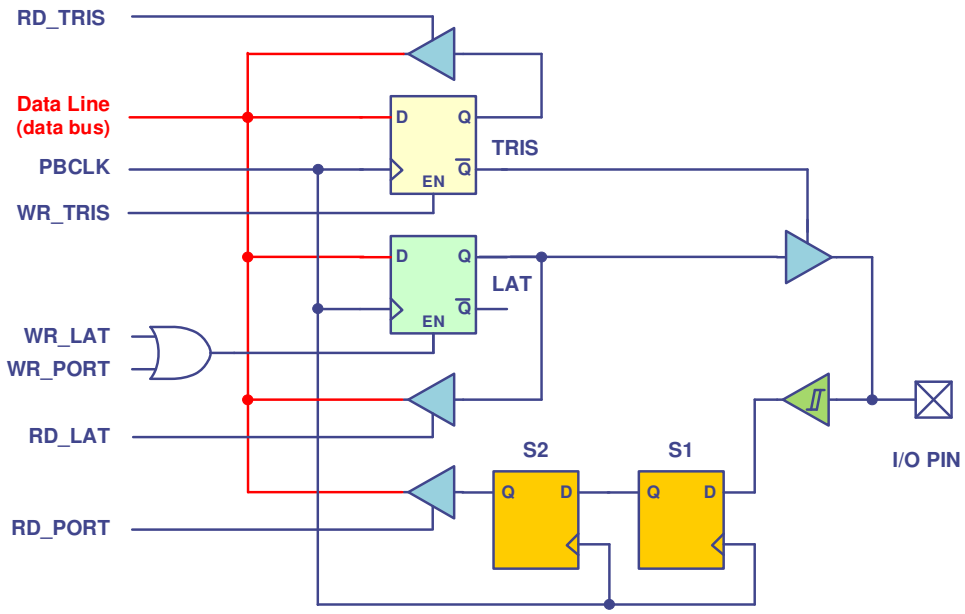

Figura 1. Diagrama de blocos simplificado de um porto de I/O de 1 bit, no PIC32.

A configuração de cada um dos bits de um porto como entrada ou saída é feita através dos *flip-flops* **`TRISxn`**, 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 bits menos significativos (ou um subconjunto destes, dependendo do porto) têm informação útil. Desse conjunto de registos apenas usaremos 3: **`TRISx`**, **`PORTx`** e **`LATx`**.

O registo **`TRISx`** é usado para configurar os portos como entrada ou saída, o registo **`LATx`** é usado para escrever um valor num porto configurado como saída e o **`PORTx`** para ler o valor de um porto configurado como entrada.

[1] Estes nomes são também usados pelo fabricante para designar cada um dos pinos físicos do microcontrolador: **`RB0`** a **`RB15`**, **`RD0`** a **`RD11`**, … (ver esquema elétrico no final deste guião).

Os registos **TRISx**, **PORTx** e **LATx** estão mapeados no espaço de endereçamento de memória, em endereços pré-definidos (disponíveis nos manuais do fabricante). O acesso para leitura e escrita desses registos é feito 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 ADDR_BASE_HI,0xBF88     # Base address: 16 MSbits
.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 os portos de 1 bit a que esse registo diz respeito, a alteração do valor de 1 bit (ou conjuntos de bits) tem que ser feita sem alterar o valor dos 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,ADDR_BASE_HI     # $t1=0xBF880000
lw      $t2,TRISE($t1)       # READ  (Mem_addr = 0xBF880000 + 0x6100)
andi    $t2,$t2,0xFFF6       # 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,ADDR_BASE_HI     # $t1=0xBF880000
lw      $t2,LATE($t1)        # READ (Read LATE register)
andi    $t2,$t2,0xFFFE       # MODIFY (bit0 = 0)
ori     $t2,$t2,0x0008       # 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 o porto de saída.

Notas importantes:

- A escrita num porto configurado como entrada não tem qualquer consequência: o valor é escrito no *flip-flop* LAT associado ao porto mas não fica disponível no exterior uma vez que é barrado pela porta *tri-state* que se encontra na saída e que está em alta impedância (ver Figura 1).
- A configuração como saída de um porto que deveria estar configurado como entrada (e que tem um dispositivo de entrada associado) pode, em algumas circunstâncias, destruir esse porto. É, assim, muito importante que a configuração dos portos seja feita com grande cuidado.
- É obrigatório configurar, como entrada ou como saída, todos os portos do PIC32 que forem usados na aplicação.
- A escrita nos registos associados aos portos do PIC32 (configuração e dados) é obrigatoriamente feita com uma sequência do tipo "Read-Modify-Write". Nesse acesso, apenas os bits usados podem ser alterados.

Trabalho a realizar

Parte I

A Figura 2 mostra parte do esquema elétrico de ligação dos LED e de um *dip-switch* na placa DETPIC32 (consulte o esquema elétrico na parte final do guião).

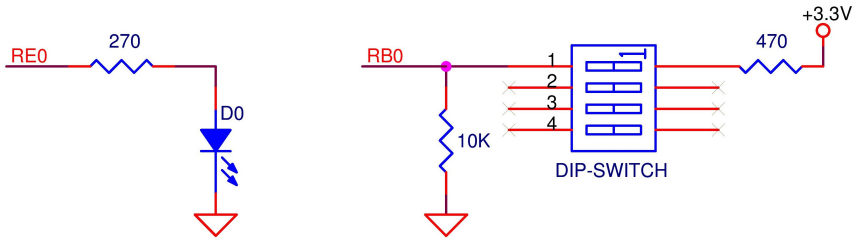

Figura 2. Ligação de um LED e um *switch* a portos do PIC32.

1. 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`**).
2. Altere o programa anterior de modo a escrever no porto de saída o valor lido do porto de entrada, negado (i.e., **`RE0 = RB0\`**).
3. Repita os exercícios anteriores usando como porto de entrada o porto **`RD8`**, que permite ler o estado do pulsador "**`INT1`**" da placa DETPIC32 (i.e., **`RE0 = RD8`** e **`RE0 = RD8\`**).

Parte II

Nesta parte pretende-se a implementação, em *assembly*, de vários tipos de contadores. Para todos os exercícios o valor do respetivo contador deve ser observado nos 4 LED ligados aos portos **`RE4`** a **`RE1`** (esquema representado na Figura 3). Para o controlo da frequência deve utilizar o *core timer*.

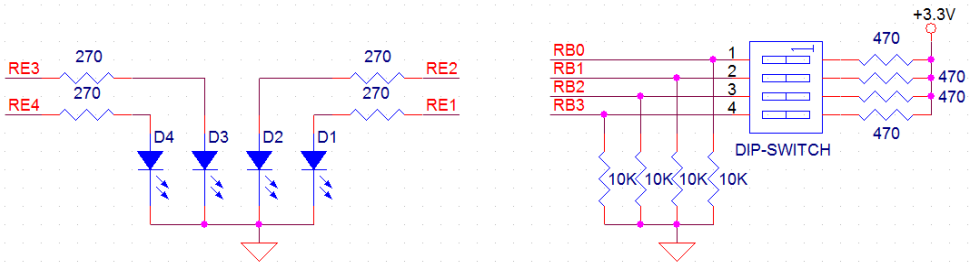

Figura 3. Ligação de 4 LED e um *dip-switch* de 4 posições a portos do PIC32

Para este tipo de exercício deve estruturar o seu código do seguinte modo:

1. Configurar os portos; neste caso: **`RE4-RE1`** como saídas (registo **`TRISE`**), e, quando se aplique, **`RB3`**, **`RB2`** ou **`RB1`** como entradas (registo **`TRISB`**).

```
lw      $t1,TRISE($t0)   # $t0 must be previously initialized
andi    $t1,$t1,0xFFE1   # Reset bits 4-1
sw      $t1,TRISE($t0)   # Update TRISE register
lw      $t1,TRISB($t0)   # Read TRISB register
ori     $t1,$t1,0x????   # Set bits ???
sw      $t1,TRISB($t0)   # Update TRISB register
```

2. Inicializar a variável de contagem.

```
li      $t2,0            #e.g. up counter (initial value is 0)
```

3. Atualizar os portos de saída (**RE4-RE1**) com o valor da variável de contagem (registo **LATE**).

```
        lw      $t1,LATE($t0)    # Read LATE register
        andi    $t1,$t1,0xFFE1   # Reset bits 4-1
        sll     $t3,$t2,1        # Shift counter value to "position" 1
        or      $t1,$t1,$t3      # Merge counter w/ LATE value
        sw      $t1,LATE($t0)    # Update LATE register
```

4. Esperar **t** segundos, em que **t = 1/f** (se **t** for múltiplo de 1ms pode usar a função **delay()**[2]; caso contrário, deve usar diretamente os *system calls* de interação com o *core timer*, ajustando a constante de comparação em função do tempo pretendido).

```
        li      $v0,RESET_CORE_TIMER
        syscall
wait:   li      $v0,READ_CORE_TIMER
        syscall
        blt     $v0,4347826,wait # e.g. f=4.6Hz
```

5. Atualizar a variável de contagem (em função do valor lido de um porto de entrada, quando isso for pedido).

```
        addi    $t2,$t2,1
        andi    $t2,$t2,0x000F   # e.g. up counter MOD 16
```

6. Repetir desde 3.

Escreva o código *assembly* para implementar cada um dos contadores seguintes (um programa para cada contador).

1. Contador binário crescente de 4 bits (módulo 16), atualizado com uma frequência de 1Hz.
2. Contador binário decrescente de 4 bits (módulo 16), atualizado com uma frequência de 4Hz.
3. Contador binário módulo 16 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.
4. 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**, …).
5. Contador Johnson de 4 bits (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.
6. Contador de 4 bits com um comportamento idêntico ao contador de Johnson, mas 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).
7. 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.

[2] Se utilizar a função delay(), ou outras funções, deve respeitar as convenções de utilização e salvaguarda de registo do MIPS.

Exercícios adicionais

1. Configure os portos **RB0** a **RB3** como entradas e os portos **RE2** a **RE5** como saídas e, em ciclo infinito, faça as seguintes atribuições nas saídas:

   RE2 = RB0, **RE3 = RB1**, **RE4 = RB2** e **RE5 = RB3**

2. Traduza para *assembly* o trecho de código seguinte[3], 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)
   }
}
```

   a) Qual a frequência a que o LED0 deve piscar?

   b) Acrescente ao programa a configuração do porto **RD0** como saída, e altere o programa de modo a enviar o mesmo valor para os portos **RE0** e **RD0**.

   c) Ligue a ponta do osciloscópio no porto **RD0** (ponto de teste **OC1**), e meça o tempo a 1 e o tempo a 0 do sinal de saída gerado pelo programa.

   d) Altere o programa de modo a que o atraso gerado seja 10.7ms e repita a medição dos tempos com o osciloscópio.

Mapa de Registos dos portos de I/O do PIC32

Registo	Endereço
TRISB	0xBF886040
PORTB	0xBF886050
LATB	0xBF886060
TRISC	0xBF886080
PORTC	0xBF886090
LATC	0xBF8860A0
TRISD	0xBF8860C0
PORTD	0xBF8860D0
LATD	0xBF8860E0
TRISE	0xBF886100
PORTE	0xBF886110
LATE	0xBF886120

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 (aulas 2 a 4).

PDF criado em 24/02/2025

[3] No que respeita à configuração e ao acesso aos portos de I/O, o programa apresentado 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.

Esquema elétrico da placa DETPIC32