Trabalho prático N.º 3

Objetivos

- Conhecer a estrutura básica e o modo de configuração de um porto de I/O no microcontrolador PIC32.
- Interligar dispositivos simples de interação com o utilizador a portos de I/O do PIC32.
- Configurar em *assembly* os portos de I/O do PIC32 e aceder para enviar / receber informação do exterior.

Nota: montar placa antes da aula.

Introdução

O microcontrolador PIC32 disponibiliza vários portos de I/O, com várias dimensões (número de bits), identificados com as siglas PORTB, PORTC, PORTD, PORTE, PORTF e PORTG. Cada um dos bits de cada um destes portos pode ser configurado, por programação, como entrada ou saída. Poderemos então considerar um porto de I/O de **n** bits do PIC32 como um conjunto de **n** portos de I/O de 1 bit. 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 configuração de cada um dos bits de um porto como entrada ou saída pode ser efetuada através dos registos 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 de 32 bits (numa visão simplificada), dos quais, nesta fase, apenas usaremos 3: os registos TRISx, PORTx e LATx. Esses registos estão mapeados no espaço de endereçamento de memória (área designada por SFRs), em endereços pré-definidos disponíveis nos manuais do fabricante, pelo que o acesso para leitura e escrita é efetuado através das instruções LW e SW da arquitetura MIPS.

```
.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
```

A título de exemplo, se se pretender configurar, em *assembly*, os bits 0 e 3 do porto E (RE0 e RE3) como saída podemos utilizar a seguinte sequência de código:

```
lui     $t1, SFR_BASE_HI      #
lw      $t2, TRISE($t1)       # Mem_addr = 0xBF880000 + 0x6100
andi    $t2, $t2, 0xFFF6      # bit0 = bit3 = 0 (0 means OUTPUT)
sw      $t2, TRISE($t1)       # Write TRISE register
```

Para colocar as saída dos portos RE0 e RE3 a 1 pode fazer-se:

```
lui     $t1, SFR_BASE_HI      #
lw      $t2, LATE($t1)        # Read LATE register
ori     $t2, $t2, 9           # Set bit0 and bit3
sw      $t2, LATE($t1)        # Write LATE register
```

Como auxiliar de memória, note que o registo TRIS controla o estado *tri-state* do porto (0 = *tri-state off* → o porto não está no estado de alta impedância → porto de saída), PORT diz respeito ao valor no pino (como veremos mais adiante, o valor lido corresponde ao valor no pino dois ciclos de relógio atrás), e LAT diz respeito ao valor na *latch*.

Trabalho a realizar[2]

Parte I

Ligue, de acordo com o esquema seguinte, um *led* e um contacto do *dip-switch* aos bits 0 e 6 do porto E, pinos RE0 e RE6, respetivamente, da placa DETPIC32.[3]

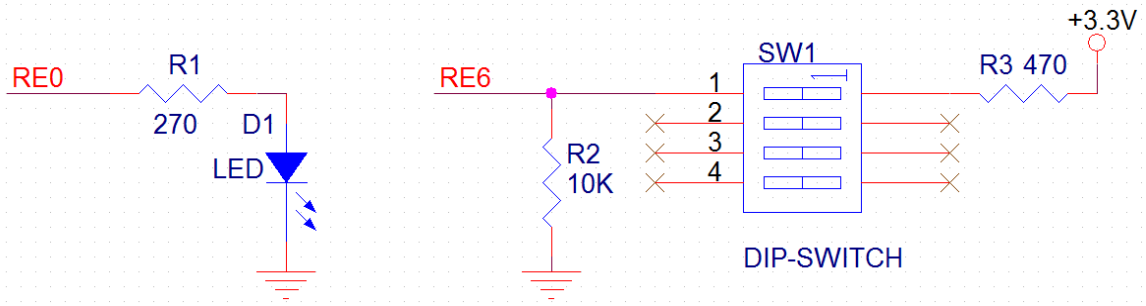

Figura 1. Ligação de um *led* e um *dip-switch* a portos do PIC32.

1. Escreva e teste um programa em *assembly* que:
   - configure os portos RE0 e RE6 como saída e entrada, respetivamente;
   - em ciclo infinito, leia o valor do porto de entrada e escreva esse valor no porto de saída (i.e RE0 = RE6).
2. Altere o programa anterior de modo a escrever no porto de saída o valor lido do porto de entrada, negado (i.e. RE0 = RE6\).
3. Ligue um segundo contacto do *dip-switch* ao porto RE7 e mais três *leds* (e as respetivas resistências), aos portos RE1, RE2 e RE3.

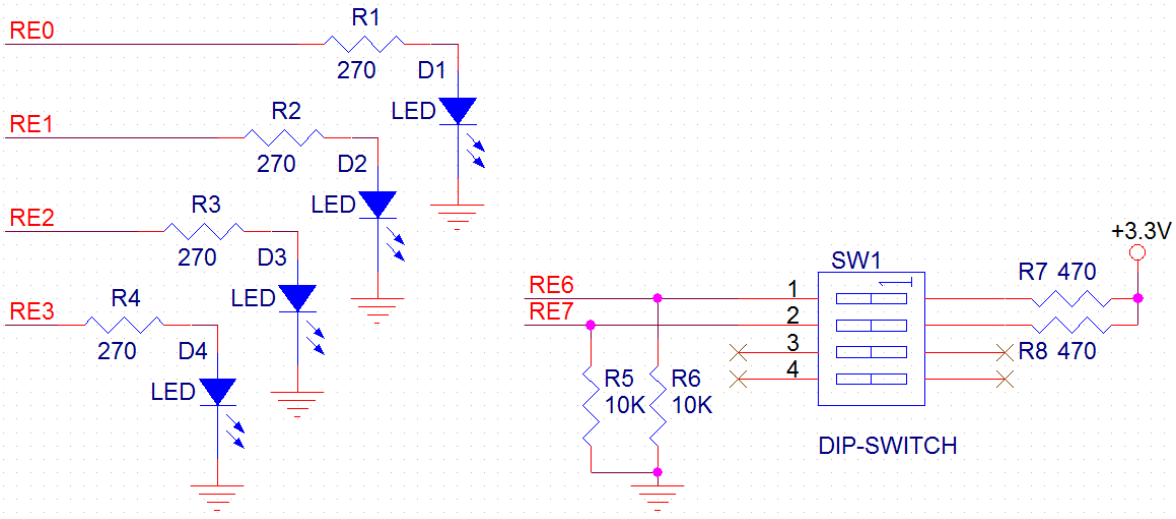

Figura 2. Ligação de 4 *leds* e um *dip-switch* de 4 posições a portos do PIC32.

4. Escreva e teste um programa em *assembly* que:
   - configure os portos RE0, RE1, RE2 e RE3 como saídas e os portos RE6 e RE7 como entradas;

[2] **Note bem**: a organização da componente prática de AC2 pressupõe que todo o trabalho de montagem dos circuitos seja efetuado fora da aula prática. (Isto aplica-se a este e aos guiões seguintes.) Este aspeto será tido em consideração para efeitos de cálculo da nota respeitante à avaliação do docente.

[3] A resistência de 10kΩ destina-se a forçar a tensão em RE6 a zero quando o *switch* está aberto. A de 470Ω foi incluida apenas por segurança, para evitar uma possível corrente muito elevada caso o porto RE6 seja configurado por engano como uma saída (quando o *switch* está fechado com o porto configurado como uma entrada, a tensão em RE6 estará perto de 3.3V). Em ambos os casos, é possível utilizar resistências com outros valores (digamos, entre 10kΩ e 33kΩ, e entre 470Ω e 1kΩ).

- em ciclo infinito, leia os valores dos portos de entrada e escreva nos portos de saída os resultados das seguintes expressões lógicas:

```
RE0 =   RE6 & RE7   (AND)
RE1 =   RE6 | RE7   (OR)
RE2 =   RE6 ^ RE7   (XOR)
RE3 = ~(RE6 | RE7)  (NOR)
```

Parte II

1. Escreva e teste um programa, em *assembly*, que, para além das necessárias inicializações, implemente os seguintes contadores (atualizados com uma frequência de 4 Hz, i.e. a cada 0.25 s):
   - contador binário crescente de 4 bits;
   - contador Johnson de 4 bits (sequência: 0000, 0001, 0011, 0111, 1111, 1110, 1100, 1000, 0000, 0001, ...)

   (utilize os *system calls* de manipulação do *Core Timer* para gerar a frequência de 4Hz)
2. Faça as alterações ao programa que achar convenientes de modo a que seja possível escolher, através do bit 6 do porto E, qual dos dois contadores deverá estar em execução. O bit 7 do porto E deverá controlar a direção da contagem (0 ascendente, 1 descendente). Sempre que haja alteração no valor lido do bit 6 do porto E, para além da comutação de comportamento, o contador deve ser inicializado a 0000.

Parte III

A função seguinte gera um atraso programável, em múltiplos de 1ms, cujo valor é passado como argumento:

```
// Geracao de um atraso programável:
//  - valor mínimo: 1 ms (para n_intervals = 1)
//  - valor máximo: (2^32-1) * 1ms (para n_intervals = 0xFFFFFFFF)
//    (aproximadamente 4294967 s, i.e., 49.7 dias :) )
//
void delay(unsigned int n_intervals)
{
   volatile unsigned int i;

   for(; n_intervals != 0; n_intervals--)
      for(i = CALIBRATION_VALUE; i != 0; i--)
         ;
}
```

A função deverá portanto demorar a executar um tempo de **`n_intervals * 1ms`**, ou seja, o ciclo interno deverá demorar a executar **`1ms`**. Para que o atraso gerado seja correto há pois necessidade de determinar o valor da constante **`CALIBRATION_VALUE`**. Para isso, poderá ser obtida uma estimativa bastante aproximada usando os *system calls* de manipulação do *Core Timer*. Repare que o *Core Timer* é incrementado com uma frequência de relógio de 20 MHz (i.e., metade da frequência de relógio do CPU) pelo que 1ms corresponde a 20000 ciclos desse sinal.

Nota: A palavra-chave **`volatile`** dá a indicação ao compilador que a variável pode ser alterada de forma não explicitada na zona de código onde está a ser usada (i.e., noutra zona de código, como por exemplo numa rotina de serviço à interrupção). Com esta palavra-chave força-se o compilador a, sempre que o valor da variável seja necessário, efetuar o acesso à posição de memória onde essa variável reside, em vez de usar uma eventual cópia residente num registo interno do CPU. Na função anterior, sem o **`volatile`** um compilador inteligente eliminaria os dois ciclos **`for`** visto que não têm efeitos secundários (apenas gastam tempo, que é coisa que os compiladores tentam minimizar).

1. Traduza para *assembly* do MIPS a função **`delay()`**, utilizando para a constante **`CALIBRATION_VALUE`** o valor 6000. Escreva também o programa principal que invoque, em ciclo infinito, a função **`delay()`**:

```
void main(void)
{
   while(1)
   {
      resetCoreTimer();
      delay(1);
      printInt(readCoreTimer(), 10 + (10 << 16));
      printStr("\r\n");
   }
}
```

2. Execute o programa e tome nota do valor impresso.

3. Repita a execução do programa com um novo valor para a constante, por exemplo 12000. Tome nota do valor produzido pelo programa e, com os dois pares de valores (constante, valor lido do *CoreTimer*) determine a equação da recta que une os dois pontos (declive e ordenada na origem). A partir dessa equação extrapole o valor correto da constante **`CALIBRATION_VALUE`**, para obter um valor lido do *CoreTimer* de 20000. Com essa constante, o atraso gerado pela função **`delay()`** será muito próximo de 1 ms.

4. Outra possibilidade para calibrar a função de geração do atraso consiste em usar um porto de saída de 1 bit (por exemplo o RE4) para gerar uma onda quadrada, e o osciloscópio do laboratório para medir o tempo a 1 e o tempo a 0 desse sinal.

   Traduza para *assembly* o trecho de código seguinte, usando a função **`delay()`** com a constante que determinou no ponto anterior. Compile, transfira para a placa DETPIC32 e execute esse código. Com o osciloscópio no porto RE4 meça o tempo a 1 e o tempo a 0 do sinal. Proceda de forma semelhante à descrita no ponto 3 e extrapole o valor da constante de maneira a obter exactamente 1ms em cada um desses tempos.

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

   TRISE4 = 0;    // Configura o porto RE4 como saída
   while(1)
   {
      LATE4 = v;  // Escreve v no bit 4 do porto E
      delay(1);   // Atraso de 1ms
      v ^= 1;     // complementa o bit 0 de v
   }
}
```

Nota. 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 correcta de o fazer será descrita no trabalho prático n.º 4.

Elementos de apoio

- PIC32 Family Reference Manual, Section 12 – I/O Ports.
- PIC32MX5XX/6XX/7XX, Family Data Sheet, Pág. 96 a 101.
- *Slides* das aulas teóricas.

Adenda ao trabalho prático N.º 3

1. Cada *led* do *display* de 7 segmentos tem de ter em série uma resistência; a não colocação desta resistência tem como consequência a destruição dos *displays* e/ou da saída do microcontrolador à qual o led estiver ligado. A corrente consumida por cada led pode ser calculada como: I = (3.3 – 1.5) / R; para uma resistência de 270 ohm, a corrente no *led* é, aproximadamente, 6.7 mA.
2. Não deve utilizar qualquer fonte de tensão externa. Deve-se ter, no entanto, em atenção que o consumo máximo admissível do conjunto não deverá exceder 100 mA.
3. A inserção/remoção da placa DETPIC32 na placa branca deve ser realizada com muito cuidado de forma a evitar o empeno e consequente quebra dos seus pinos. A inserção deve ser feita pressionando lentamente a placa em ambas as extremidades A remoção deve ser evitada, mas sempre que houver necessidade de o fazer, deverá ser efetuada de forma a puxar simultaneamente as duas extremidades da placa DETPIC32.
4. Sugestão de organização do espaço nas placas brancas:

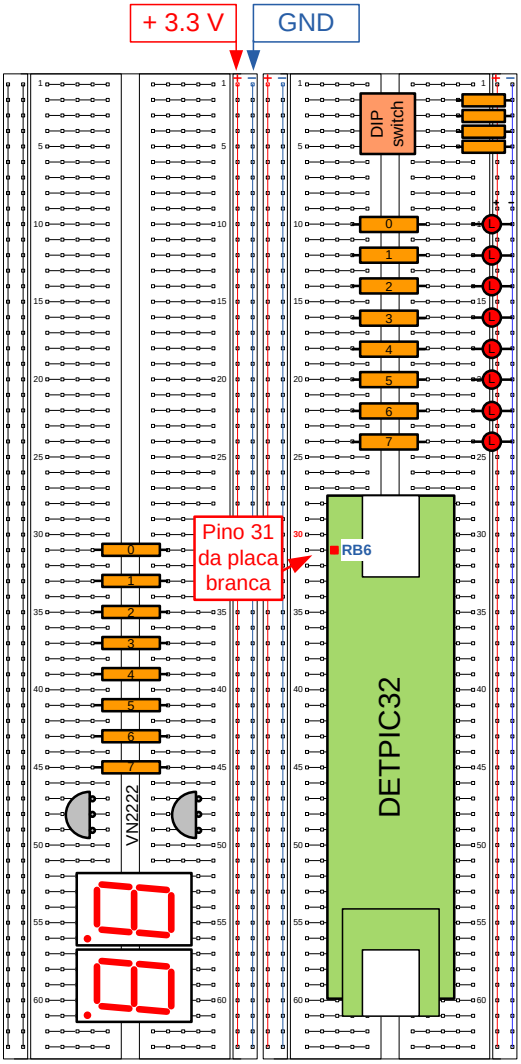

Figura 3. A placa DETPIC32 deverá ser montada de tal forma que a ficha de ligação USB fique posicionada na extremidade da placa branca. Por outro lado, a ligação à placa DETPIC32 estará menos sujeita a erros se o pino 1 (correspondente ao porto RB6) ficar posicionado no pino 31 da placa branca. Deste modo, para se obter o número do pino da placa DETPIC32 basta subtrair 30 ao número da pista correspondente da placa branca.

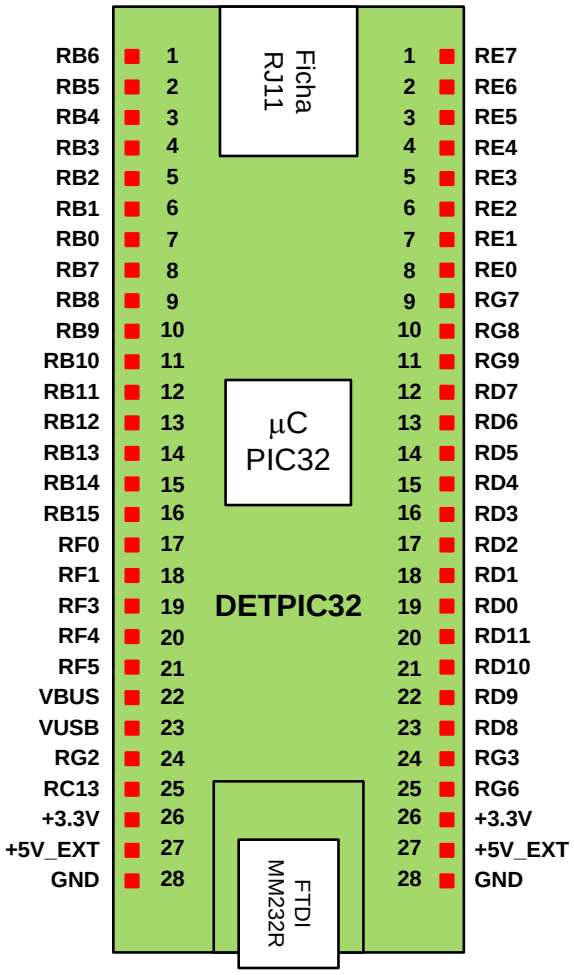

Figura 4. Disposição dos pinos de ligação à placa branca no DETPIC32.