Aula prática N.º 2

Objetivos

- Utilizar o *core timer* do MIPS para gerar atrasos programáveis.

Introdução

O *core* MIPS disponível no microcontrolador PIC32 implementa, no coprocessador 0, um contador crescente de 32 bits, designado por *core timer*, atualizado a cada dois ciclos de relógio do CPU. Na placa DETPIC32 o relógio do CPU está configurado a 40 MHz, pelo que o contador é incrementado a uma frequência de 20 MHz. Isto significa que o tempo necessário para incrementar o contador desde o valor 0 até 20.000.000 é 1 segundo.

O *core timer* é frequentemente utilizado para medições de tempo, implementação de atrasos temporais (*delays*) e temporizações simples em software.

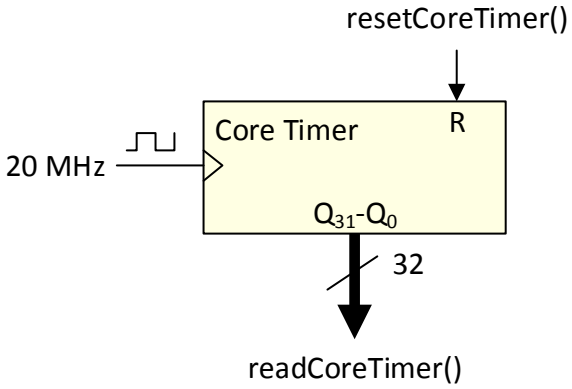

Figura 1. Modelo de interação das *system calls* **`resetCoreTimer()`** e **`readCoreTimer()`** com o *core timer*.

A placa DETPIC32 disponibiliza duas *system calls* para interagir com esse contador: ler o valor atual do contador (**`readCoreTimer()`**) e reposição do seu valor a zero (**`resetCoreTimer()`**).

Trabalho a realizar

Parte I

1. O programa seguinte incrementa o valor de uma variável e, de cada vez que a variável é atualizada, o seu valor é apresentado no ecrã do PC.

```
int main(void)
{
   int counter = 0;
   while(1)
   {
      putChar('\r');    // cursor regressa ao inicio da linha no ecrã
      printInt(counter, 10 | 4 << 16);
      resetCoreTimer();
      while(readCoreTimer() < 200000);
      counter++;
   }
   return 0;
}
```

   a) Qual a frequência de incremento da variável **`counter`**?

   b) Traduza o código C fornecido para *assembly* do MIPS, e teste-o na placa.

```
        .equ    READ_CORE_TIMER,11
        .equ    RESET_CORE_TIMER,?
        .equ    PUT_CHAR,?
        .equ    PRINT_INT,?
        .data
        .text
        .globl  main
main:   li      $t0,0                   # counter=0
while:                                  # while (1) {
        ...
        li      $v0,RESET_CORE_TIMER    #
        syscall                         #   resetCoreTimer()
        ...
        j       while                   # }
        jr      $ra                     #
```

   c) Altere sucessivamente o código que escreveu de forma a que a variável seja incrementada com uma frequência de 10 Hz, 5 HZ e de 1Hz, e teste o resultado.

2. O objetivo da função **`delay()`**, apresentada a seguir, é gerar um atraso temporal programável múltiplo de 1ms.

```
void delay(unsigned int ms)
{
   resetCoreTimer();
   while(readCoreTimer() < K * ms);
}
```

   a) Determine o valor da constante "**`K`**", de modo a que para "**`ms`**" igual a 1 o atraso gerado seja de 1ms (note que **`K=20*10⁶*t`**, em que "**`t`**" é o valor do atraso, em segundos).

   b) Com o valor de "**`K`**" que obteve na alínea anterior, calcule o valor máximo de atraso que é possível gerar com a função **`delay()`**.

   Traduza para *assembly* do MIPS a função **`delay()`** e teste-a com diferentes valores de entrada (para o teste utilize como base o código C fornecido no ponto 1).

Notas:

1. Para as operações de multiplicação e divisão de inteiros devem, obrigatoriamente, ser usadas as instruções virtuais: "**`mul $Rdst,$Rsrc1,$Rsrc2`**", "**`div $Rdst,$Rsrc1,$Rsrc2`**", "**`rem $Rdst,$Rsrc1,$Rsrc2`**" (ou **`mulou`**, **`divu`**, **`remu`**).
2. As funções devem obrigatoriamente ser colocadas (no segmento de código), após a função **`main()`**.
3. Para valores de atraso que não sejam múltiplos de 1ms, a função **`delay()`**, tal como implementada anteriormente, não pode ser usada. Para esses casos terá que utilizar diretamente as *system calls* **`resetCoreTimer()`** e **`readCoreTimer()`** comparando o valor do *core timer* com a constante correspondente ao atraso pretendido (por exemplo, **`26000`**, para um atraso de **`1.3ms`**).

Parte II

1. Usando a função **`delay()`** como base para a temporização, escreva um programa em linguagem C que incremente, em ciclo infinito, 3 variáveis inteiras: a variável **`cnt1`** deve ser incrementada a uma frequência de **1 Hz**, a variável **`cnt5`** deve ser incrementada a uma frequência de **5 Hz**, e a variável **`cnt10`** deve ser incrementada a uma frequência de **10 Hz**.

   O valor das 3 variáveis deve ser mostrado no ecrã, sempre na mesma linha, formatado em base 10 com 5 dígitos. Exemplo de visualização:

   ```
   00020   00100   00200
   ```

   Nota: reflita sobre a necessidade de utilizar uma única base de tempo para gerir as 3 frequências de incremento, bem como o valor que essa base de tempo deverá ter.

   Traduza o programa, que escreveu no ponto anterior, para *assembly* do MIPS e teste-o na placa.

2. Altere o programa de modo a que as frequências de incremento dos contadores sejam 1 Hz 3.3333 Hz e 5 Hz. O valor da frequência deve ser obtido com o menor erro possível.

3. Altere o programa que fez em 1) de modo a que quando for premida a tecla **`'A'`**, a frequência de incremento dos contadores passe para o dobro, i.e., 2Hz, 10Hz e 20Hz. Premindo a tecla **`'N'`**, a frequência de incremento dos contadores deve voltar ao valor normal. Para a leitura do caracter utilize o *system call* **`inkey()`**.

4. Altere o programa que resultou do ponto anterior de modo a que quando for premida a tecla **`'S'`**, a contagem dos contadores seja suspensa e quando for premida a tecla **`'R'`** a contagem seja retomada. Para a leitura do caracter utilize o *system call* **`inkey()`**.

Exercícios adicionais

1. Considere agora a função **`timeDone()`** que se apresenta a seguir. Esta função permite verificar se já decorreu um determinado tempo (múltiplo de 1ms) desde a última vez que foi efetuado o seu *reset*. Caso o tempo não se tenha esgotado a função devolve o valor **`0`** (zero). Caso contrário devolve o número de milissegundos que decorreram desde o último *reset* ao *core timer*.

   Nota: Este código não pode ser usado juntamente com a função **`delay()`**, uma vez que ambas as funções efetuam, ou podem efetuar, *reset* ao *core timer*.

```
unsigned int timeDone(int ms, unsigned char reset)
{
   unsigned int curCount;
   unsigned int retValue = 0;

   if (reset > 0)
   {
      resetCoreTimer();
   }
   else
   {
      curCount = readCoreTimer();
      if (curCount > (K * ms))
         retValue = curCount / K;
   }
   return retValue;
}
```

NOTA: use o valor de "**K**" que determinou no exercício 2 da parte 1.

a) Traduza para *assembly* do MIPS a função **timeDone()** e teste-a com diferentes valores de entrada (para o teste adapte o código C fornecido no ponto 1).

2. Retome o exercício 1 desta secção e reescreva o programa de modo a utilizar a função **timeDone()**. Adicionalmente, por cada segundo decorrido o código deve enviar para o terminal o caracter '**\n**'.

PDF criado em 29/01/2026