AULA PRÁTICA N.º 3

Objectivos:

- Implementação e codificação de estruturas de controlo de fluxo do tipo **`if..else`**, **`for(), while(), do..while()`**.

Conceitos básicos:

- Instruções de alteração de fluxo de execução no MIPS.
- Codificação de lógica relacional.

Guião:

1. O programa seguinte lê um número introduzido pelo utilizador e apresenta esse mesmo valor representado em binário.

```
void main(void)
{
    int value, bit, i;

    print_str("Introduza um numero: ");
    value = read_int();
    print_str("\nO valor em binario: ");
    for(i=0; i < 32; i++)
    {
        bit = value & 0x80000000;
        if(bit == 0)
            print_char('0');
        else
            print_char('1');
        value = value << 1;
    }
}
```

a) Codifique o programa em *assembly* do MIPS e teste o seu funcionamento no MARS. Para o armazenamento das variáveis do programa utilize registos **`$tn`**.

b) Altere o programa anterior de modo a imprimir um espaço entre cada grupo de 4 bits (por exemplo, **`1010 0110 0001`** ...).

c) Sabendo que o código ASCII do 0 é 0x30 e do 1 0x31, o programa anterior pode ser simplificado, eliminando a estrutura condicional. Proponha uma solução, em C, para essa alteração, implemente-a e teste-a no MARS.

d) Altere o programa em C que resultar da alínea anterior, de modo a que não sejam impressos os zeros à esquerda. Codifique as alterações em *assembly* e teste novamente o programa.

2. O programa seguinte lê um número introduzido pelo utilizador e apresenta-o em hexadecimal.

```
void main(void)
{
    int value, i, digito;

    print_str("Introduza um numero: ");
    value = read_int();
    print_str("\nO valor em hexadecimal: ");

    i=8;
    while( ((value & 0xF0000000) == 0 ) && (i > 0) )
    {
        value = value << 4;
        i--;
    }

    do
    {
        digito = (value & 0xF0000000) >> 28;
        if(digito <= 9)
            print_char(digito + '0');
        else
            print_char(digito + '0' + 7);
        value = value << 4;
        i--;
    } while(i > 0);
}
```

a) Codifique o programa em *assembly* do MIPS e teste o seu funcionamento no MARS. Para o armazenamento das variáveis do programa utilize registos **`$tn`**.

b) Altere o programa anterior de modo a imprimir o valor de entrada em octal. Codifique as alterações em *assembly* e teste o seu funcionamento.

3. A multiplicação de inteiros pode ser realizada recorrendo a sucessivas operações de deslocamentos e somas, tal como esquematizado na figura seguinte para operandos de 4 bits (o algoritmo usado é em tudo semelhante ao que todos aprendemos no ensino básico, aplicado agora a quantidades expressas em binário).

```
                0 1 0 1
              x 0 1 1 0
-------------------------

  0 0 0 0 0 0 0 0   Res. Inicial
+ 0 0 0 0 0 0 0 0   0.mdo.2^0
-------------------
  0 0 0 0 0 0 0 0
+ 0 0 0 0 1 0 1 0   1.mdo.2^1
-------------------
  0 0 0 0 1 0 1 0
+ 0 0 0 1 0 1 0 0   1.mdo.2^2
-------------------
  0 0 0 1 1 1 1 0
+ 0 0 0 0 0 0 0 0   0.mdo.2^3
-------------------
  0 0 0 1 1 1 1 0

  0 0 0 1 1 1 1 0   Res. FINAL
```

A implementação desse algoritmo, para quantidades de 4 bits, pode ser efectuada do modo que o programa seguinte apresenta.

```
void main(void)
{
    int res=0, i=0, mdor, mdo;

    print_str("Introduza dois numeros: ");
    mdor = read_int();
    mdo = read_int();

    while( (mdor != 0) && (i++ < 4) )
    {
        if( (mdor & 0x00000001) != 0 )
            res = res + mdo;
        mdo = mdo << 1;
        mdor = mdor >> 1;
    }
    print_str("Resultado: ");
    print_int10(res);
}
```

a) Codifique o programa em *assembly* do MIPS e teste o seu funcionamento no MARS. Para o armazenamento das variáveis do programa utilize registos **`$tn`**.

b) Execute o programa passo a passo e preencha a tabela seguinte, supondo que os valores iniciais do multiplicador e multiplicando são 11 e 6, respectivamente.

mdor	mdo	i	res	
0x0B	0x06			Valores iniciais
				Fim da 1ª iteração
				Fim da 2ª iteração
				Fim da 3ª iteração
				Fim da 4ª iteração

c) O programa apresentado apenas manipula valores de 4 bits. Altere o programa em C de modo a poder manipular valores de 16 bits. Codifique as alterações em *assembly* e teste o funcionamento com valores de entrada de 16 bits.