AULA PRÁTICA N.º 5

Objectivos:

- Manipulação de *arrays* em linguagem C, usando índices e ponteiros.
- Tradução para *assembly* de código de acesso sequencial a *arrays* usando índices e ponteiros.

Guião:

1. O programa seguinte lê da consola 5 valores inteiros e armazena-os no *array* "lista".

```
#define SIZE 5

void main(void)
{
    static int lista[SIZE];   // declara um array de inteiros
                              //  residente no segmento de dados
    int    i;

    for(i=0; i < SIZE; i++)
    {
        print_str("\nIntroduza um numero:");
        lista[i] = read_int();
    }
}
```

a) Traduza o programa anterior para *assembly* do MIPS. Utilize, para armazenar as variáveis, os seguintes registos: **`i`** (**`$t0`**), endereço inicial do *array*, **`lista`** (**`$t1`**), endereço do elemento "**`i`**" do *array*, **`lista[i]`** (**`$t2`**) e valor lido do teclado (**`$v0`**). **Nota**: não se esqueça que, caso não declare o espaço para o *array* no início do segmento de dados (antes da declaração da *string*, neste caso), deverá obrigatoriamente incluir a directiva **`.align 2`** antes da declaração do *array*, de modo a garantir que o seu endereço inicial é múltiplo de 4.

b) Verifique o correcto funcionamento do programa. Para isso, execute-o, introduza a sequência de números 14, 4660, 11211350, -1, -1412589450 e observe o conteúdo da memória na zona de endereços reservada para o *array*.

c) Execute o programa passo a passo e preencha a tabela abaixo com os valores que as diferentes variáveis vão tomando, introduzindo a sequência de números da alínea anterior.

i ($t0)	lista ($t1)	&(lista[i])($t2)	($v0)	
				Fim 1ª iteração
				Fim 2ª iteração
				Fim 3ª iteração
				Fim 4ª iteração
				Fim 5ª iteração

d) Observe o conteúdo da memória na zona de endereços respeitante ao *array* "lista" (janela *Data Segment* do MARS).

2. O programa seguinte envia para o ecrã o conteúdo de um *array* de 10 inteiros, previamente inicializado (a declaração **`static int lista[]={8,-4,3,5,124,-15,87,9,27, 15};`** declara espaço para um *array* de inteiros de 10 posições e inicializa-o com os valores especificados).

```
#define SIZE 10

void main(void)
{
    static int lista[]={8, -4, 3, 5, 124, -15, 87, 9, 27, 15};
    int *p;      // Declara um ponteiro para inteiro (reserva
                 //  espaço para o ponteiro, mas não o inicializa)
    int i;

    p = lista;   // Inicializa o ponteiro: "lista" representa o
                 //  endereço do 1º elemento do array
    print_str("\nConteudo do array:\n");

    for(i=0; i < SIZE; i++)
    {
        print_int10( *p );    // Imprime o conteúdo da posição do
                              //  array cujo endereço é "p"
        print_str("-");
        p++;     // Incrementa o ponteiro (p = p + 1), isto é,
                 //  o ponteiro passa a apontar para a posição
                 //  seguinte do array
    }
}
```

a) Codifique o programa em *assembly* do MIPS e teste o seu funcionamento no MARS. Note que, nesta implementação, usou-se acesso ao *array* por ponteiro, enquanto que no exercício anterior usou-se acesso indexado.

b) Execute o programa e observe o conteúdo da memória na zona de endereços respeitante ao *array* "**`lista`**".

3. Considere agora o seguinte programa que ordena por ordem crescente os elementos de um *array* de números inteiros (algoritmo *bubble-sort* não optimizado).

```
#define SIZE 10
void main(void)
{
    static int lista[SIZE];
    int houveTroca, i, aux;

    // inserir aqui o código para leitura de valores e
    //  preenchimento do array
    do
    {
        houveTroca = FALSE;
        for (i=0; i < SIZE-1; i++)
        {
            if (lista[i] > lista[i+1])
            {
                aux = lista[i];
                lista[i] = lista[i+1];
                lista[i+1] = aux;
                houveTroca = TRUE;
            }
        }
    } while (houveTroca==TRUE);

    // inserir aqui o código de impressão do conteúdo do array
}
```

a) Reescreva o programa anterior, acrescentando-lhe o código para preenchimento do *array* usando acesso por ponteiro, e para a impressão do seu conteúdo usando acesso indexado.

b) Traduza para *assembly* o programa que resultou do ponto anterior. Verifique o funcionamento do seu programa inserindo diferentes sequências de valores inteiros, positivos e/ou negativos.

c) Pretende-se agora que o programa de ordenação trate os conteúdos do *array* como quantidades sem sinal (i.e., interpretadas em binário natural). Para isso, no programa anterior é apenas necessário alterar a declaração do *array*, passando a ser:

```
static unsigned int  lista[SIZE];
```

Na tradução para *assembly* esta alteração implica que em todas as instruções de decisão que envolvam elementos do *array* se trate os respectivos operandos como quantidades sem sinal (i.e. em binário natural). No *assembly* do MIPS isso é feito acrescentando o sufixo "u" à mnemónica da instrução. Por exemplo, para verificar a condição "menor ou igual" de duas quantidades com sinal, residentes nos registos **`$t0`** e **`$t1`**, a instrução *assembly* é:

```
ble   $t0,$t1,target
```

A mesma condição, tratando as quantidades em binário natural, é feita pela instrução:

```
bleu  $t0,$t1,target
```

.

Altere o programa *assembly* que escreveu em b) e teste-o inserindo diferentes sequências de valores inteiros, positivos e/ou negativos. Interprete os resultados obtidos.

4. Um programa de ordenação equivalente ao anterior que usa ponteiros em vez de índices é apresentado a seguir.

```
#define SIZE 10

void main(void)
{
    static int lista[SIZE];
    int houveTroca;
    int aux;
    int *p, *pUltimo;

    // inserir aqui o código para leitura de valores e
    //  preenchimento do array

    pUltimo = lista + (SIZE - 1);

    do
    {
        houveTroca = FALSE;
        for (p = lista; p < pUltimo; p++)
        {
            if (*p > *(p+1))
            {
                aux = *p;
                *p = *(p+1);
                *(p+1) = aux;
                houveTroca = TRUE;
            }
        }
    } while (houveTroca==TRUE);

    // inserir aqui o código de impressão do conteúdo do array
}
```

a) Traduza o programa anterior para *assembly* (incluindo igualmente o código para entrada e saída de valores) e teste-o inserindo diferentes sequências de valores inteiros, positivos e/ou negativos.

b) O programa de ordenação apresentado pode ainda ser optimizado, tornando-o mais eficiente. Sugira as alterações necessárias para essa optimização, altere correspondentemente o programa em C e reflicta essas alterações no código *assembly*.

*Opção de "congelar" o valor mais elevado numa posição mais elevada

5. Um outro programa de ordenação, baseado no algoritmo conhecido como *sequential-sort*, é apresentado de seguida.

```
#define SIZE 10

void main(void)
{
   static int lista[SIZE];
   int i, j, aux;

   // inserir aqui o código para leitura de valores e
   //  preenchimento do array

   for(i=0; i < SIZE-1; i++)
   {
        for(j = i+1; j < SIZE; j++)
        {
             if(lista[i] > lista[j])
             {
                  aux = lista[i];
                  lista[i] = lista[j];
                  lista[j] = aux;
             }
        }
   }

   // inserir aqui o código de impressão do conteúdo do array

}
```

a) Traduza o programa anterior para *assembly* (incluindo igualmente o código para entrada e saída de valores) e teste-o inserindo diferentes sequências de valores inteiros, positivos e/ou negativos