

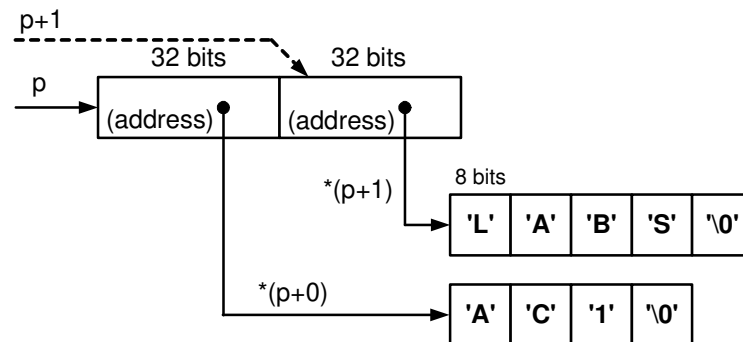
AULA PRÁTICA N.º 6

Objetivos:

- Utilização de ponteiros em linguagem C – *array* de ponteiros para caracter. Tradução para *assembly*.

Guião:

Para além do acesso simples a *arrays* visto nas aulas anteriores, em linguagem C os ponteiros podem ser usados em estruturas de dados mais elaboradas como a que se apresenta neste guião. A figura seguinte apresenta esquematicamente o conceito que pode ser descrito sucintamente como um *array* de ponteiros (dois, no exemplo), cada um deles apontando para um caracter (no exemplo, esse caracter é o primeiro de um *array* de caracteres). Se "**p**" for o ponteiro para o início do *array* de ponteiros (i.e. "**p**" é o endereço da posição zero do *array* de ponteiros), então "**p+1**" é um ponteiro para o segundo elemento desse *array*. O endereço do caracter apontado pela primeira posição do *array* "**p**" será então "***p**" e, de modo idêntico, o endereço do caracter apontado pela segunda posição do *array* "**p**" será "***(p+1)**".



Em linguagem C esta estrutura de dados é definida do seguinte modo:

```
char *p[]={ "AC1", "LABS" };    // Array de ponteiros para caracter
ou,
char *p[2]={ "AC1", "LABS" };    // Array de ponteiros para caracter
```

- O programa seguinte define 3 strings, organizadas na estrutura de dados descrita anteriormente, e imprime-as.

```
#define SIZE 3

void main(void)
{
    static char *array[SIZE]={ "Array", "de", "ponteiros" };
    int i;

    for(i=0; i < SIZE; i++)
    {
        print_str(array[i]);
        print_char('\n');
    }
}
```

- Traduza o programa para *assembly* do MIPS e teste o seu funcionamento no MARS.

Tradução parcial do código anterior para *assembly*:

```
# i:    $t0
#
    .eqv SIZE,...
    .data
array: .word    str1,...
str1:  .asciiz "Array"
str2:  .asciiz "de"
str3:  ...
    .text
    .globl main
main:
    ...
for:   ...
    la      $t1,array    # $t1 = &array[0]
    sll     $t2,$t0,2     #
    addu    $t2,...      # $t2 = &array[i]
    lw      $a0,...      # $a0 = array[i]
    ...
    jr      $ra
```

2. No programa apresentado no exercício anterior utilizou-se o modo indexado para aceder ao *array* de ponteiros. Uma implementação alternativa é apresentada de seguida onde o acesso sequencial ao *array* é efectuado por ponteiro.

```
#define SIZE 3

void main(void)
{
    static char *array[SIZE]={"Array", "de", "ponteiros"};
    char **p;
    char **pultimo;

    p = array;
    pultimo = array + SIZE;

    for(; p < pultimo; p++)
    {
        print_str(*p);
        print_char('\n');
    }
}
```

- a) Traduza o programa para *assembly* do MIPS e teste o seu funcionamento no MARS.

Tradução parcial do código anterior para *assembly*:

```
# p      :    $t1
# pultimo:    $t2
    ...
    la      $t1,array    # $t1 = p = &array[0] = array
    li      $t0,SIZE     #
    sll     $t0,$t0,2     #
    addu    $t2,...      # $t2 = pultimo = array + SIZE
for:   ...
```

3. O programa seguinte imprime as 3 strings, caracter a caracter (separados pelo caracter '-'), usando acesso indexado.

```
#define SIZE 3

void main(void)
{
    static char *array[SIZE]={"Array", "de", "ponteiros"};
    int i, j;

    for(i=0; i < SIZE; i++)
    {
        print_str( "\nString #" );
        print_int10( i );
        print_str( ": " );
        j = 0;
        while(array[i][j] != '\0')
        {
            print_char(array[i][j]);
            print_char('-');
            j++;
        }
    }
}
```

- a) Traduza o programa para *assembly* do MIPS e teste o seu funcionamento no MARS.

Tradução parcial do código anterior para *assembly*:

```
# i      :      $t0
# j:      $t1
# array[i][j]: $t3
...
while:
    la      $t3,array      # $t3 = &array[0]
    sll     $t2,$t0,2      #
    addu    $t3,$t3,$t2    # $t3 = &array[i]
    lw      $t3,0($t3)     # $t3 = array[i] = &array[i][0]
    addu    $t3,$t3,$t1    # $t3 = &array[i][j]
    lb      $t3,...        # $t3 = array[i][j]
    ...
```

4. Em alguns sistemas operativos é possível executar programas em linha de comando (exemplo no S.O. linux: **evince AC1-P-Aula6.pdf**). Nesses casos há necessidade de passar ao programa argumentos de entrada (por exemplo o nome de um ficheiro) ou parâmetros que condicionam o modo como o programa deve executar. Para isso o conjunto de argumentos introduzidos na linha de comando (sequências de caracteres separadas por um ou mais espaços) é passado para a função **main()** através de uma estrutura de dados idêntica à apresentada nos exercícios anteriores. Nesse caso, o protótipo da função **main()** passa a ser:

```
int main(int argc, char *argv[]);
```

em que **argc** representa o número de argumentos de entrada introduzidos na linha de comando e **argv[]** é um *array* de ponteiros (com dimensão **argc**) para as strings que representam esses argumentos.

```
int main(int argc, char *argv[])
{
    int i;
    print_str("Nr. de parametros: ");
    print_int10(argc);

    for(i=0; i < argc; i++)
    {
        print_str("\nP");
        print_int(i);
        print_str(": ");
        print_str(argv[i]);
    }
    return 0;
}
```

- a) Traduza o programa anterior para *assembly* do MIPS, considerando que os argumentos para a função `main()` são passados nos registos `$a0` e `$a1` pela ordem com que aparecem no protótipo (`argc` em `$a0` e `argv` em `$a1`).
- b) Teste o programa no MARS. No MARS pode passar os argumentos para o programa através da linha "Program Arguments" disponível na janela de texto ("Text Segment"). Para isso terá que ativar a opção "Program arguments provided to MIPS program", disponível no menu "Settings".

Exercícios adicionais

1. Escreva um programa em linguagem C que determine e imprima no ecrã a seguinte informação relativa aos argumentos passados através da linha de comando:
 - o número de caracteres de cada um dos argumentos;
 - o número de letras (maiúsculas e minúsculas) de cada um dos argumentos;
 - a string com o maior número de caracteres.
2. Traduza o código que escreveu no exercício anterior para *assembly* do MIPS e teste-o no MARS.
3. Reescreva o código C apresentado no exercício 3 do guião, de modo a efetuar o acesso sequencial aos dois *arrays* através de ponteiros (use como base o código C apresentado no exercício 2). Traduza o código resultante para *assembly* do MIPS e teste-o no MARS.