universidade de aveiro deti departamento de electrónica, telecomunicações e informática

Trabalho Prático 8

Funções e Convenções de Uso de Registos

Objetivos

- Implementação de funções ou subrotinas.
- Utilização da convenção do MIPS para uso de registos e passagem de argumentos.
- A *stack* do MIPS.

Introdução

As linguagens de Alto-Nível usam funções para estruturar os programas em módulos reutilizáveis e para aumentar a clareza do código. As funções podem ter argumentos como entradas e um valor de retorno como saída.

A convenção de uso de registos no MIPS pode resumir-se da seguinte forma:

Passagem de argumentos	$a0..$a3
Retorno de valores	$v0, $v1
Uso Geral (não preservados pelas funções)	$t0..$t9
Uso Geral (preservados pelas funções)	$s0..$s7
Stack Pointer	$sp
Endereço de Retorno	$ra

Durante a invocação a função chamadora (*caller*) passa os argumentos nos registos **$a's** à função chamada (*callee*) e 'salta' para o código desta. A função chamada executa o seu código e devolve o resultado à função chamadora nos registos **$v's**, regressando ao código de onde foi chamada. Durante este processo o valor dos registos **($s's e $ra**) necessários à *caller* não pode ser alterado.

Guião

Parte I

1. A função seguinte, escrita em C, devolve o factorial de um número passado como argumento (compare com o código do exercício 1.3 do guião 6).

```
int factorial(int num)
{
    int res,i;
    res=1;
    for ( i = num; i>0; i--)
    {
      res = res*i;
    }
    return res;
}
```

a) Traduza função `factorial` para *Assembly*.

b) A função *main*, apresentada de seguinda permite testar a função anterior. Traduza-a para assembly e teste o seu funcionamento.

```
void main(void)
{
     int n, f;
     char prompt1[] = "Introduza um numero\n";
     char result[] = "O fatorial do número inserido é: ";

     print_str( prompt1 );
     n = read_int();

     f = factorial(n);

     print_str( result );
     print_int( f );

     exit();
}
```

c) Que alterações teria de fazer na função *main* caso a mensagem final a imprimir fosse da forma: "O factorial do número [imprimir o número] é: [imprimir o factorial]". Faça as alterações e teste o programa.

Parte II

O conjunto de exercícios abaixo propostos constituem um conjunto de funções úteis para trabalhar com *strings*, tais como a função *strlen* que devolve o número de caracteres, a função *strcopy* e *strcat* que fazem respetivamente a cópia e a concatenação de *strings*.

2. A função seguinte, escrita em C, devolve o número de caracteres de uma *string*.

```
int strlen( char * str ){
  int n=0, i=0;

        while ( str[i++] != '\0' ) // i é pós-incrementada
        {
              n++;
        }
        return n;   // o valor deve ser retornado em $v0
}
```

d) Traduza o programa para *Assembly*.

e) Escreva uma função *main* que permita testar a função anterior.

3. O código seguinte é uma possível implementação das funções *strcopy* e *strcat* .

```
char * strcopy(char *dst, char *src)
{
int i=0;

 while ( src[i] != '\0' )      {
```

```
        dst[i] = src[i];
        i++;
    }
    dst[i] = '\0' ;
    return dst;
}
char * strcat(char *dst, char *src)
{
    char * aux = dst;
    while (*dst != '\0' )
        dst++;      // o ponteiro dst é incrementado ficando a apontar
                    // para a posição de memória seguinte
    strcopy( dst, src );
    return aux;
}
```

a) Traduza a função `strcopy` para *Assembly* e verifique o seu funcionamento.

b) Implemente em Assembly a função `strcat`.

4. Considere o seguinte programa que testa as funções anteriores.

```
void main( void )
{
  static char frase1[20];
  static char frase2[20];
  static char frase3[40];
  int n;

  print_string( "\nInsira a frase1: " );
  read_string(frase1,20);

  print_string( "\nInsira a frase2: " );
  read_string(frase2,20);

  print_string( "\n O numero de caracteres da frase1 é: " );
  n = strlen( frase1 );
  print_int10(n);

  str_copy( frase3,frase1 );
  str_cat( frase3, frase2 );

  print_string( "\nA frase concatenada é: " );
  print_string( frase3 );

  exit();
}
```

Exercícios Adicionais

1. A função seguinte calcula recursivamente o valor de uma potência de um número.

```
int  x_to_y( int base, int exp )
{
      if (exp == 0) return 1;
      return base * x_to_y(base, exp - 1);
}
```

a) Traduza a função `x_to_y` para assembly e escreva uma função *main* que permita testá-la.

b) Ensaie a chamada à função `x_to_y` com diferentes pares de valores (por exemplo: 10^1, 2^3, 4^2) e preencha a tabela seguinte com o valor das variáveis e o conteúdo da *stack* a cada chamada da função. (coloque um *breakpoint* no inicio da função).

Stack Pointer	Return Address	Conteúdo da Stack		base	Exp

2. A função seguinte implementa o algoritmo *bubble_sort* usando ponteiros.

a) Traduza-a para Assembly respeitando a convenção de uso de registos do MIPS.

```
void bubble_sort( int *array, int n )
{
  int i, aux;
  int * p_aux;
  char houveTroca;

  do {
      houveTroca = FALSE;

      for (p_aux = array; p_aux < array+n-1; p_aux++ )
      {
            if ( *p_aux > *(p_aux+1) )
            {
                  aux = *p_aux;
                  *p_aux = *(p_aux+1);
                  *(p_aux+1) = aux;
                  houveTroca = TRUE;
            }
      }
  } while (houveTroca == TRUE );
}
```

b) Escreva uma função `main` para realizar o teste a função `bubble_sort`. O teste pode ser feito considerando um *array* de inteiros previamente inicializado na memória (usando a directiva .word) ou lendo um *array* de inteiros do teclado, preenchendo-o na memória e invocando a função `bubble_sort` de seguida.