AULA PRÁTICA N.º 7

Objetivos:

- Implementação de sub-rotinas.
- Utilização da convenção do MIPS para passagem de parâmetros e uso dos registos.
- Implementação e utilização da *stack* no MIPS. Parte 2.

Guião:

1. A função seguinte converte para um inteiro de 32 bits a quantidade representada por uma *string* numérica em que cada carater representa o código ASCII de um dígito decimal (i.e., 0 - 9). A conversão termina quando é encontrado um carater não numérico.

```
unsigned int atoi(char *s)
{
   unsigned int digit, res = 0;

   while( (*s >= '0') && (*s <= '9') )
   {
       digit = *s++ - '0';
       res = 10 * res + digit;
   }
   return res;
}
```

a) Traduza para *assembly* a função **`atoi()`** (não se esqueça da aplicação das regras de utilização dos registos do MIPS).

Tradução parcial para *assembly* do código anterior:

```
# Mapa de registos
# res:      $v0
# s:        $a0
# *s:       $t0
# digit:    $t1
# Sub-rotina terminal: não devem ser usados registos $sx
atoi:     li      $v0,0           # res = 0;
while:    lb      $t0,...         # while(*s >= ...)
          b??     ...             #
          b??     ...             # {
          sub     $t1,...         #    digit = *s – '0'
          addiu   ...             #    s++;
          mul     $v0,$v0,10      #    res = 10 * res;
          add     ...             #    res = 10 * res + digit;
          (...)                   # }
          jr      $ra             # termina sub-rotina
```

b) O programa seguinte permite fazer o teste da função **`atoi()`**. Traduza para *assembly* e verifique o correto funcionamento da função com outras *strings*.

```
int main(void)
{
   static char str[]="2016 e 2020 sao anos bissextos";

   print_int10( atoi(str) );
   return 0;
}
```

c) Altere a função **`atoi()`** de modo a processar uma *string* binária (por exemplo **`atoi(`**"**`101101`**"**`)`** deverá produzir o resultado 45). Traduza as alterações para *assembly* e teste-as.

2. A função **`itoa()`**, que se apresenta de seguida, determina a representação do inteiro "n" na base "**`b`**" (**`b`** pode variar entre 2 e 16), colocando o resultado no *array* de carateres "**`s`**", em ASCII. Esta função utiliza o método das divisões sucessivas para efetuar a conversão entre a base original (hexadecimal) e a base destino "**`b`**": por cada nova divisão é encontrado um novo dígito da conversão (o resto da divisão inteira), esse dígito é convertido para ASCII e o resultado é colocado no *array* de carateres.
Como é conhecido, neste método de conversão o primeiro dígito a ser encontrado é o menos significativo do resultado. Assim, a última tarefa da função **`itoa()`** é a chamada à função **`strrev()`** (implementada na aula anterior) para efetuar a inversão da *string* resultado.

```
char toascii( char );
char *strrev( char *);

char *itoa(unsigned int n, unsigned int b, char *s)
{
   char *p = s;
   char digit;

   do
   {
       digit = n % b;
       n = n / b;
       *p++ = toascii( digit );
   } while( n > 0 );
   *p = '\0';
   strrev( s );
   return s;
}

// Converte o digito "v" para o respetivo código ASCII
char toascii(char v)
{
   v += '0';
   if( v > '9' )
       v += 7;  // 'A' - '9' - 1
   return v;
}
```

d) Traduza a função **`itoa()`** para *assembly*[1].

[1] A função **`strrev()`** foi já implementada no guião anterior. De modo a simplificar a gestão do código desenvolvido, pode usar várias janelas do editor do MARS (a que correspondem outros tantos ficheiros): por exemplo, uma janela para o código a escrever da função **`itoa()`** e respetivo **`main()`** e outra janela com a função **`strrev()`**. Nesse caso, deverá ter em atenção o seguinte:

- No menu *settings* a opção "*Assemble all files in directory*" tem que ser ativada.
- Os nomes das funções que sejam declaradas no(s) ficheiro(s) secundário(s) (o ficheiro principal é o que tem definido o label "**`main`**") têm que ser declarados como globais. Por exemplo, se o ficheiro que contém a declaração dos *labels* "**`strrev:`**" e "**`strcpy:`**" é um ficheiro secundário, no topo desse ficheiro deve aparecer a seguinte diretiva:

```
.globl strrev, strcpy
```

- Apenas um ficheiro pode conter a declaração do *label* "**`main:`**".

Tradução parcial para *assembly* do código anterior:

```
# Mapa de registos
# n:        $a0 -> $s0
# b:        $a1 -> $s1
# s:        $a2 -> $s2
# p:        $s3
# digit:    $t0
# Sub-rotina intermédia
itoa:     subu    $sp,...         # reserva espaço na stack
          sw      $s0,...         # guarda registos $sx e $ra
          (...)
          move    $s0,...         # copia n, b e s para registos
          (...)                   # "callee-saved"
          move    $s3,$a3         # p = s;
do:                               # do {
          (...)                   #
          b??     $s0,...         # } while(n > 0);
          sb      $0,0($s3)       # *p = 0;
          (...)                   #
          jal     strrev          # strrev( s );
          (...)                   # return s;
          lw      $s0,...         # repõe registos $sx e $ra
          (...)
          addu    $sp,...         # liberta espaço na stack
          jr      $ra             #
```

e) O programa seguinte permite testar a função **`itoa()`** fazendo a conversão de um valor lido do teclado para diferentes bases. Traduza-o para *assembly*, e teste o seu funcionamento no MARS.

```
#define MAX_STR_SIZE     33

int main(void)
{
   static char str[MAX_STR_SIZE]
   do {
       val = read_int();
       print_string( itoa(val, 2, str) );
       print_string( itoa(val, 8, str) );
       print_string( itoa(val, 16, str) );
   } while(val != 0);
   return 0;
}
```

Tradução parcial para *assembly* do código anterior:

```
# Mapa de registos
# str:      $s0
# val:      $s1
# O main é, neste caso, uma sub-rotina intermédia
          .data
str:      .space  ...
          .eqv    STR_MAX_SIZE,...
          .eqv    read_int,...
          .eqv    print_string,...
          .text
          .globl  main
main:     subu    $sp,...           # reserva espaço na stack
          (...)                     # guarda registos $sx na stack
          sw      $ra,...           # guarda $ra na stack
do:                                 # do {
          li      $v0,read_int
          syscall                   #
          move    $s1,$v0           #    val = read_int()
          (...)
          b??     $s1,...           # } while(val != 0)
          li      ...               # return 0;
          (...)                     # repoe registos $sx
          lw      $ra,...           # repõe registo $ra
          addu    $sp,...           # liberta espaço na stack
          jr      $ra               # termina programa
```

f) A função seguinte apresenta a implementação de uma função para impressão de um inteiro através da utilização da *system call* **`print_str()`** e da função **`itoa()`**. Traduza para *assembly* esta função e teste-a, escrevendo a respetiva função **`main()`**.

```
void print_int_ac1(unsigned int val, unsigned int base)
{
   static char buf[33];

   print_string( itoa(val, base, buf) );
}
```

3. A função seguinte implementa o algoritmo de divisão de inteiros apresentado nas aulas teóricas (versão otimizada), para operandos de 16 bits.

```
unsigned int div(unsigned int dividendo, unsigned int divisor)
{
   int i, bit, quociente, resto;

   divisor = divisor << 16;
   dividendo = (dividendo & 0xFFFF) << 1;

   for(i=0; i < 16; i++)
   {
        bit = 0;
        if(dividendo >= divisor)
        {
             dividendo = dividendo - divisor;
             bit = 1;
        }
        dividendo = (dividendo << 1) | bit;
   }
   resto = (dividendo >> 1) & 0xFFFF0000;
   quociente = dividendo & 0xFFFF;

   return (resto | quociente);
}
```

a) Traduza esta função para *assembly* e teste-a com diferentes valores de entrada, tendo em atenção que os operandos têm uma dimensão máxima de 16 bits.

b) O programa anterior apresenta uma deficiência de funcionamento em situações em que o dividendo é igual ou superior a **`0x8000`** e o divisor é superior ao dividendo. Verifique, com um exemplo, essa situação, identifique a origem do problema e proponha uma solução, em linguagem C, para o resolver.

Exercícios adicionais

Parte I

1. A função "**`insert()`**" permite inserir a *string* "**`src`**" na string "**`dst`**", a partir da posição "**`pos`**". A função "**`read_str()`**" usa a *system call* "**`read_string`**" para ler uma *string* do teclado e elimina o carater de mudança de linha (**`0x0A`**) introduzido quando se prime a tecla ENTER.

```
char *insert(char *dst, char *src, int pos)
{
   int len_dst, len_src;
   int i;
   char *p = dst;

   len_dst = strlen(dst);
   len_src = strlen(src);

   if(pos <= len_dst)
   {
       for(i = len_dst; i >= pos; i--)
           dst[i + len_src] = dst[i];
       for(i=0; i < len_src; i++)
           dst[i + pos] = src[i];
   }
   return p;
}

void read_str(char *s, int size)
{
   int len;
   read_string(s, size);
   len = strlen(s);
   if(s[len-1] == 0x0A)
       s[len-1] = '\0';
}
```

Traduza as duas funções anteriores para *assembly* (não se esqueça de aplicar a convenção de utilização de registos).

2. O programa seguinte permite o teste das funções desenvolvidas no exercício anterior. Traduza esse programa para *assembly* e teste-o no MARS. Relembre que o "main()" é tratado como qualquer outra sub-rotina, no que concerne à convenção de utilização e salvaguarda de registos.

```
// Protótipos das funções usadas

int strlen(char *s); // função desenvolvida no guião anterior
char *insert(char *dst, char *src, int pos);
void read_str(char *s, int size);
```

```
int main(void)
{
   char str1[100];
   char str2[50];
   int insert_pos;

   print_string("Enter a string: ");
   read_str(str1, 50);

   print_string("Enter a string to insert: ");
   read_str(str2, 50);

   print_string("Enter the position: ");
   insert_pos = read_int();

   print_string("Original string: ");
   print_string(str1);

   insert(str1, str2, insert_pos);

   print_string("\nModified string: ");
   print_string(str1);
   print_string("\n");
   return 0;
}
```

Exemplo de funcionamento:

```
Enter a string: Arquitadores
Enter a string to insert: tetura de Compu
Enter the position: 5
Original string: Arquitadores
Modified string: Arquitetura de Computadores
```

Parte II

Apresentam-se, nesta secção, vários exemplos de funções recursivas, tendo algumas delas sido já implementadas, na sua forma iterativa, em guiões anteriores. Note que, na maioria dessas funções, a solução recursiva não é a melhor em termos de eficiência. Essas funções são aqui usadas como exemplos por serem de fácil compreensão, o que permite praticar quer o conceito de recursividade quer a tradução de linguagem C para *assembly* desse tipo de funções.

1. A função seguinte apresenta a implementação, na forma recursiva, da função de contagem do número de carateres de uma *string* já implementada anteriormente na forma iterativa.

```
int strlen(char *s)
{
   if(*s != '\0')
       return(1 + strlen(s + 1));
   else
       return 0;
}
```

A tradução para *assembly* do MIPS da função **`strlen()`** pode ser feita do seguinte modo:

```
strlen: lb      $t0,0($a0)   # $t0 = *s
        beq     $t0,0,else   # if(*s != '\0') {
        subu    $sp,$sp,4    #   reserva espaço na stack
        sw      $ra,0($sp)   #   salvaguarda $ra
        addiu   $a0,$a0,1    #
        jal     strlen       #   strlen(s + 1)
        addi    $v0,$v0,1    #   return (1 + strlen(s+1))
        lw      $ra,0($sp)   #   repõe o valor de $ra
        addu    $sp,$sp,4    #   liberta espaço na stack
        jr      $ra          # }

else:   li      $v0,0        #   return 0
        jr      $ra          #
```

Escreva, em linguagem C, a função **`main()`** com código que permita efetuar o teste da função **`strlen()`**. e traduza-a para *assembly*. Teste o funcionamento do conjunto.

4. A função seguinte apresenta a implementação, na forma recursiva, da função de cópia de uma *string* (também já implementada anteriormente na forma iterativa).

```
char *strcpy(char *dst, char *src)
{
   if((*dst = *src) != '\0')
        strcpy(dst + 1, src + 1);
   return dst;
}
```

g) Escreva, em linguagem C, a função **`main()`** com código que permita efetuar o teste da função **`strcpy()`**.

h) Traduza as funções **`strcpy()`** e **`main()`** para *assembly*. Teste o funcionamento do conjunto.

5. A função seguinte obtém a soma dos elementos de um *array* de inteiros.

```
int soma(int *array, int nelem)
{
   if(nelem != 0)
        return *array + soma(array + 1, nelem - 1);
   else
        return 0;
}
```

i) Traduza a função **`soma()`** para *assembly*.

j) Escreva, em linguagem C, a função **`main()`** com código que permita efetuar o teste da função que escreveu na alínea a). Traduza esse programa para *assembly* e teste o funcionamento da função **`soma()`**.

6. A função seguinte imprime no ecrã o valor inteiro "**`num`**" em qualquer base entre 2 e 16 (note que essa verificação não é efetuada no código). Esta função é uma implementação recursiva mais simples e elegante que a solução iterativa já apresentada na primeira parte deste guião (que usava, recorde-se, a função **`itoa()`** para efetuar a conversão entre bases).

```
void print_int_ac1(unsigned int num, unsigned int base)
{
   if(num / base)
       print_int_ac1( num / base, base );
   print_char( toascii(num % base) );
}

char toascii(char v)
{
   v += '0';
   if( v > '9' )
       v += 7;  // 'A' - '9' - 1
   return v;
}
```

k) Escreva, em linguagem C, a função **`main()`** com código que permita efetuar o teste da função **`print_int_ac1()`**.

l) Traduza as funções **`print_int_ac1()`** e **`main()`** para *assembly*. Teste o funcionamento do conjunto.

7. A função seguinte apresenta uma implementação iterativa de uma função de cálculo do fatorial.

```
// Calculo do fatorial de n - algoritmo iterativo

unsigned int fact_i(unsigned int n)
{
   unsigned int i;
   unsigned int res;

   for(res = 1, i=2; i <= n; i++)
       res = res * i;
   return res;
}
```

A mesma função escrita na forma recursiva poderá ter a seguinte implementação:

```
// Calculo do fatorial de n - algoritmo recursivo

unsigned int fact(unsigned int n)
{
   if(n > 12)
       exit(1); // Overflow

   return (n > 1) ? n * fact(n-1) : 1;
}
```

m) Escreva, em linguagem C, a função **`main()`** com código que permita efetuar o teste da função **`fact()`** (implementação recursiva).

n) Traduza as funções **`fact()`** e **`main()`** para *assembly*. Teste o funcionamento do conjunto.

8. A função seguinte apresenta uma implementação iterativa de uma função de cálculo do valor de x^y.

```
// Cálculo de xʸ – algoritmo iterativo

int xtoy_i(int x, unsigned int y)
{
   int i, result = 1;
   for(i=0; i < y; i++)
       result *= x;
   return result;
}
```

A mesma função escrita na forma recursiva poderá ter a seguinte implementação:

```
// Cálculo de xʸ – algoritmo recursivo

int xtoy(int x, unsigned int y)
{
   return (y != 0) ? x * xtoy(x, y-1) : 1;
}
```

o) Escreva, em linguagem C, a função **`main()`** com código que permita efetuar o teste da função **`xtoy()`** (implementação recursiva).

p) Traduza as funções **`xtoy()`** e **`main()`** para *assembly*. Teste o funcionamento do conjunto.

9. A função seguinte apresenta a implementação recursiva da resolução do problema das torres de Hanoi. Apresenta-se também a implementação da função **`main()`**, bem como de uma função auxiliar para a impressão de uma mensagem.

```
void tohanoi(int n, int p1, int p2, int p3)
{
   static int count=0;

   if(n != 1)
   {
       tohanoi(n-1, p1, p3, p2);
       print_msg(p1, p2, ++count);
       tohanoi(n-1, p3, p2, p1);
   }
   else
       print_msg(p1, p2, ++count);
}


void print_msg(int t1, int t2, int cnt)
{
   print_str("\n");
   print_int_ac1(cnt, 10);
   print_str(" - Mover disco de topo de ");
   print_int_ac1(t1, 10);
   print_str(" para ");
   print_int_ac1(t2, 10);
}
```

```
int main(void)
{
   int ndiscs;

   print_str("\nIntroduza o numero de discos: ");
   ndiscs = read_int();
   if(ndiscs > 0)
       tohanoi(ndiscs, 1, 3, 2);

   return 0;
}
```

Traduza as funções anteriores para *assembly*. Teste o funcionamento do conjunto (pode encontrar um simulador deste problema em www.coolmath-games.com/0-tower-of-hanoi/).