AULA PRÁTICA N.º 8

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;
}
```

a) 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:     addiu   $sp,...         # reserva espaço na stack
          sw      $s0,...         # guarda registos $sx e $ra
          (...)
          move    $s0,...         # copia n, b e s para registos
          (...)                   # "callee-saved"
          move    $s3,$a2         # 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
          (...)
          addiu   $sp,...         # liberta espaço na stack
          jr      $ra             #
```

b) 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];
   int val;

   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:     addiu   $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
          addiu   $sp,...           # liberta espaço na stack
          jr      $ra               # termina programa
```

c) 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

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)
{
   static char str1[100];
   static 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
```

3. O programa seguinte preenche um *array* de inteiros com os valores introduzidos pelo utilizador, de seguida identifica os elementos repetidos desse *array* e marca-os num *array* auxiliar e, finalmente, imprime o *array* onde os elementos repetidos são impressos com o símbolo "*". Traduza esse programa para *assembly* e teste-o no MARS, aplicando a convenção de utilização e salvaguarda de registos.

```
// Marca elementos duplicados do array
void find_duplicates(int *array, int *dup_array, int size)
{
   int i, j, token;
   for(i=0; i < size; i++)
   {
      dup_array[i] = 0;
      for(j=0, token = 1; j < size; j++)
      {
         if(array[i] == array[j])
         {
            dup_array[j] = token;
            token++;
         }
      }
   }
}
```

```
#define SIZE 10

int main(void)
{
   static int array[SIZE];
   static int aux_array[SIZE];
   int i, dup_counter = 0;

   // Preenche array de inteiros
   for(i=0; i < SIZE; i++)
   {
      print_string("array[");
      print_int10(i);
      print_string("]=");
      array[i]=read_int();
   }

   // Identifica os elementos repetidos do array e
   // marca-os num array auxiliar (aux_array)
   find_duplicates(array, aux_array, SIZE);

   // Imprime array com * nos elementos repetidos
   for(i=0; i < SIZE; i++)
   {
      if(aux_array[i] >= 2)
      {
         print_string("*, ");
         dup_counter++;
      }
      else
      {
         print_int10(array[i]);
         print_string(", ");
      }
   }
   print_string("\n# repetidos: ");
   print_int10(dup_counter);
   return 0;
}
```