Arquitetura de Computadores I 2ª série de problemas

 $9.11.\overline{2015}$

1. Converta o seguinte código assembly do MIPS em código máquina

```
      lw
      $t7, 20($t0)
      100011 01000 01111 000000000000010100
      0x8D0F0014

      sub
      $t1, $t7, $s0
      000000 01111 10000 01001 00000 100010
      0x01F048022

      addi
      $t0, $t0, 4
      001000 01000 01000 00000000000000100
      0x21080004
```

2. Traduza os blocos de código C seguintes para assembly do MIPS utilizando as instruções *beq*, *bne* e *slt*. Assuma que g e h estão, respetivamente, nos registos \$s0 e \$s1

```
a. if (g > h)
          g = g+h;
    else
          g = g-h;
      slt $t0, $s1, $s0
                            # if h < g, t0 = 1
      beq $t0, $0, else
                            # if t0 == 0, do else
      add $s0, $s0, $s1
                            \# g = g + h
      j done
 else: sub $s0, $s0, $s1
                            \# g = g - h
 done:
 b. if (g >= h)
          g = g+1;
    else
          g = g-1;
          slt $t0, $s0, $s1 # if g < h, $t0 = 1
          bne $t0, $0, else # if $t0 != 0, do else
          addi $s0, $s0, 1 \# g = g + 1
          i done
 else:
          addi \$s1, \$s1, -1 # h = h - 1
 done:
 c. if (g \le h)
          g = 0;
    else
          h = 0;
          slt $t0, $s1, $s0 # if h < g, $t0 = 1
          bne $t0, $0, else # if $t0 != 0, do else
          add $s0, $0, $0 # g = 0
          j done
 else:
          sub \$s1, \$0, \$0 # h = 0
 done:
```

3. A instrução *li* (Load Immediate) é uma instrução virtual. Qual a tradução em instruções nativas de li \$t0, 0x1002002C

```
lui $t0, 0x1002
ori $t0, 0x002C
```

- 4. No MIPS as únicas instruções nativas de **branch** são branch on equal e branch on not equal. Indique como são traduzidas para instruções nativas as seguintes instruções:
 - a) bge \$t0, \$t1, Label

```
slt $t2, $t0, $t1
beq $t2, $0, Label
```

b) blt \$t0, \$t1, Label

```
slt $t2, $t0, $t1
bne $t2, $0, Label
```

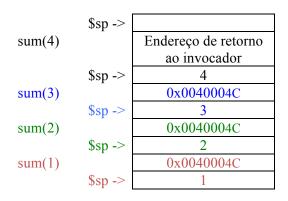
5. unsigned int sum(unsigned int n)
 {
 if (n == 0) return 0;
 else return n + sum(n-1);
 }

 a) Converta a função C para assembly do MIPS. Siga as convenções de invocação de funções do MIPS.

```
0x00400028 sum:
                       addi
                              $sp, $sp, -8
                                             # reservar espaço no stack
                                             # guardar endereço de retorno
0x0040002C
                              $ra, 4($sp)
                       sw
0x00400030
                              $a0, 0($sp)
                                             # guardar argumento
                       sw
                              $a0, $0, cont
                                             # se n \neq 0, do cont
0x00400034
                       bne
0x00400038
                       add
                              $v0, $0, $0
                                             \# sum = 0
0x0040003C
                       addi
                              $sp, $sp, 8
                                             # limpar stack
                                                    # retornar
0x00400040
                       ir
                                      $ra
0x00400044 cont:
                       addi
                              $a0, $a0, -1
                                             # decrementar n
                                             # invocar sum(n-1)
0x00400048
                       jal
                              sum
0x0040004C
                       lw
                              $a0, 0($sp)
                                             # restaurar argumento
                              $ra, 4($sp)
                                             # restaurar endereço de retorno
0x00400050
                       lw
                              $sp, $sp, 8
0x00400054
                                             # limpar stack
                       addi
0x00400058
                       add
                              $v0, $a0, $v0
                                             # somar
0x0040005C
                       jr
```

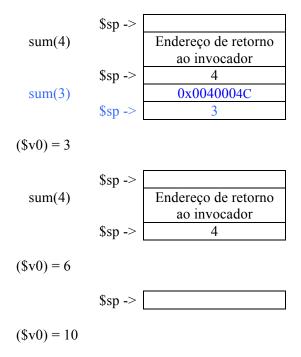
b) Suponha que o código da função está carregado em memória a partir do endereço 0x00400028 e que é invocada sum(4). Represente os estados do stack ao longo da execução da função.

Representando com cores diferentes a evolução do stack ao longo da cadeia de invocações:



$$(\$v0) = 0$$

$$(\$v0) = 1$$



sum = 10; retomada a execução do programa que invocou a função sum

6. Nas aulas foi traduzida para assembly a seguinte versão recursiva do cálculo de factorial(n):

```
int fact (int n)
{
   if (n < 1) return (1);
   else return n * fact(n - 1);
}</pre>
```

Faça a tradução para uma versão iterativa de factorial(n). Indique o conteúdo do stack durante a execução da versão recursiva e da versão iterativa quando é invocado fact(3).

```
int fact (int n)
{
    int produto, i;

    produto = 1;
    for (i = 1; i <= n; i++) {
        produto = produto * i;
    }
    return (produto);
}</pre>
```

Fatorial recursivo:

```
fact:
       addi
                 $sp, $sp, -8
                                    # ajusta o stack para 2 items
                 $ra, 4($sp)
                                    # save return address
       \mathbf{s}\mathbf{w}
       sw
                 $a0, 0($sp)
                                   # save argument
                 $t0, $a0, 1
       slti
                                   # teste se n < 1
       beq
                 $t0, $zero, L1
       addi
                 $v0, $zero, 1
                                   # if n < 1, resultado = 1
       addi
                 $sp, $sp, 8
                                   # limpar stack
                 $ra
                                   # return
       jr
L1:
       addi
                 $a0, $a0, -1
                                   # else decrementar n
        jal
                 fact
                                    # call recursiva
```

```
$a0, 0($sp)
lw
                         # restaurar valor original de n
lw
        $ra, 4($sp)
                         # e return address
addi
        $sp, $sp, 8
                         # pop 2 items do stack
        $v0, $a0, $v0
                         # multiplicar para obter resultado
mul
                         # return
jr
        $ra
             $sp ->
 fact(3)
                       Endereço de retorno
                           ao invocador
                                 3
             $sp ->
                               L1+8
 fact(2)
                                 2
             $sp ->
                               L1+8
 fact(1)
             $sp ->
                                 1
(\$v0) = 1
             $sp ->
 fact(3)
                       Endereço de retorno
                           ao invocador
             $sp ->
                                 3
 fact(2)
                               L1+8
                                 2
             $sp ->
(\$v0) = 1
             $sp ->
                       Endereço de retorno
 fact(3)
                           ao invocador
             $sp ->
(\$v0) = 2
             $sp ->
```

fact = 6; retomada a execução do programa que invocou a função fact.

Fatorial iterativo:

(\$v0) = 6

A função não invoca nenhuma outra (*leaf function*). Se forem alocados registos \$t para armazenar o valor das variáveis **produto** e **i** não é necessário usar o stack. Se se alocarem registos \$s para armazenar o valor das variáveis a função terá previamente de salvaguardar no stack os valores desses registos e restaurá-los antes de retornar.