Arquitetura de Computadores
Segunda Lista de Exercícios
Assembly MIPS
(Gabarito)

Prof. Norton Trevisan Roman

17 de outubro de 2019

1. Uma possível solução é:

```
addi $t0, $s2, -5
add  $s0, $s1, $t0
```

2.
```
lw   $s0, 0($t0)   # f' = A[f]
addi $t2, $t0, 4   # $t2 = &A[f+1]
lw   $t0, 0($t2)   # $t0 = A[f+1]
add  $t0, $t0, $s0 # $t0 = A[f+1] + A[f]
sw   $t0, 0($t1)   # B[g] = A[f+1] + A[f]
```

   (como `f` é modificado, mudando seu significado, chamei essa nova versão de `f'`, diferenciando assim de seu significado original, mantido como `f`)

   O comando é `B[g] = A[f+1] + A[f]`.

3. Uma possível solução é:

```
sll  $t1, $s4, 2    # $t1 = j*4
add  $t2, $s6, $t1  # $t2 = &A[j]
lw   $t3, 0($t2)    # $t3 = A[j]
sll  $t1, $s3, 2    # $t1 = i*4
add  $t2, $s6, $t1  # $t2 = &A[i]
lw   $t4, 0($t2)    # $t4 = A[i]
add  $t4, $t4, $t3  # $t4 = A[i] + A[j]
addi $t1, $zero, 32 # $t1 = 32 (8*4)
add  $t2, $s7, $t1  # $t2 = &B[8]
sw   $t4, 0($t2)    # B[8] = A[i] + A[j]
```

4.
```
addi $t0, $s6, 4   # $t0 = &A[1]
add  $t1, $s6, $0  # $t1 = &A[0] ($t1 = A)
sw   $t1, 0($t0)   # A[1] = &A[0] (A[1] = A)
lw   $t0, 0($t0)   # $t0 = A[1]
add  $s0, $t1, $t0 # f = &A[0] + &A[0]
```

   A tradução é `f = &A[0] + &A[0]` (ou `f = 2 * &A[0]`, ou `f = 2 * A` etc.)

5. Uma possível solução é:

```
srl $t0, $t0, 11 # joguei os bits 11-16 para 0-5
sll $t0, $t0, 26 # joguei os bits 0-5 para 26-31
sll $t1, $t1, 6 # derrubo os bits 26-31
srl $t1, $t1, 6 # devolvo os demais para seu lugar
or $t1, $t1, $t0 # uno $t0 e $t1
```

6.
```
       slt  $t2, $0, $t0  # $t2 = 1
       bne  $t2, $0, ELSE # vai ao ELSE
       j    DONE
ELSE:  addi $t2, $t2, 2   # $t2 = 3
DONE:
```
Resposta: `$t2 = 3`

7. (a) `$s2 = 20`

   (b) Uma possível solução é (note que não usamos todas as variáveis):

```
B = 0;
for (i=10; i>0; i--)
  B += 2;
```

8. Uma possível solução é:

```
        addi $t0, $zero, 0      # i = 0
FOR1:   slt  $t2, $t0, $s0      # $t2 = 1 se i < a
        beq  $t2, $zero, SAIDAe # se i >= a sai do for externo
        addi $t1, $zero, 0      # j = 0
FOR2:   slt  $t3, $t1, $s1      # $t3 = 1 se j < b
        beq  $t3, $zero, SAIDAi # se j >= b sai do for interno
        sll  $t4, $t1, 4        # $t4 = 4*j (como cada palavra tem 4B, pula 16B)
        add  $t5, $s2, $t4      # $t5 = &D[4*j]
        add  $t6, $t0, $t1      # $t6 = i + j
        sw   $t6, 0($t5)        # D[4*j] = i + j
        addi $t1, $t1, 1        # j++
        j    FOR2               # de volta ao laço interno
SAIDAi: addi $t0, $t0, 1        # i++
        j    FOR1               # de volta ao laço externo
SAIDAe:
```

9. Traduzindo o código:

```
      addi $t1, $0, 0     # i = 0
LOOP: lw   $s1, 0($s0)    # $s1 = *arranjo
      add  $s2, $s2, $s1  # resultado += *arranjo
      addi $s0, $s0, 4    # arranjo++ (segundo elemento de arranjo)
      addi $t1, $t1, 1    # i++
      slti $t2, $t1, 100  # $t2 = 1 se i < 100, senão $t2 = 0
      bne  $t2, $0, LOOP  # vai a LOOP se i < 100
```

Uma possível solução é:

```
int *p = arranjo;

int i=0;
do {
  resultado += *p;
  p++;
  i++;
} while (i<5);
```

10. Uma possível solução é:

```
and $s1, $s0, 1
```