Escola Politécnica da Universidade de Pernambuco
Curso de Engenharia de Computação
Disciplina: Arquitetura de Computadores
Local e Data: Recife, 07 de outubro de 2016

1) Um procedimento lê palavras de dois bancos de dados, A[ ] e B[ ], cujos endereços-base devem ser armazenados em 800030F0 e 800040C4, respectivamente. O procedimento lê as palavras do banco A[ ] e as palavras do banco B[ ] uma a uma e as compara, a partir do endereço final. Os elementos de A[ ] e B[ ] são comparados e copiados para um terceiro banco C[ ], cujo endereço-base 80002080_{16} (deverá ser armazenado em $a3), de acordo com as seguintes regras: a) Caso o elemento do banco A[ ] seja menor ou igual ao elemento do banco B[ ], o elemento de A[ ] deve ser copiado em C[ ]; b) Caso o elemento de A[ ] seja maior do que o elemento de B[ ], o elemento de B[ ] deve ser copiado em C[ ]. Quantos elementos de A[ ] e B[ ]são copiados para o banco C[ ]? **(2,5 pontos)**

Parada A[i] = B[j] = 0 Tamanho = 100

2) Dado o procedimento abaixo em linguagem algorítmica, desenvolva o código em Mips. O aluno poderá escolher qualquer registrador, exceto os registradores S_i, para alocar os endereços-base e as variáveis. O código em Mips deverá estar de acordo com as regras aplicadas pelo Mips aos procedimentos, aos registradores e a pilha: **(2,75 pontos)**
Obs.: NÃO USAR OS REGISTRADORES $Si para i = 0,1,.....7.

1) Dado o procedimento abaixo em linguagem algorítmica, desenvolva o código em Mips. O aluno poderá escolher qualquer registrador, exceto os $Si e as regras aplicadas pelo Mips aos procedimentos, aos registradores e a pilha (a qual deverá ser a mais eficiente possível): (2,5 pontos)

```
int  Proc1 (int Matriz[ ], Vetor[ ], int i, j )
  {
   While (Vetor[8i] > Matriz[5j])
        {
          Matriz[5j]=  Proc2(Vetor[8i])
                   j = i + Proc3(j)
        }
   return  j, Matriz[5j]
  }
```

```
Proc2:
    input:  $a2
    output: $v0
Proc 3:
  input: $a0
   output: $v0
```

4/25.

1.EE Arquitetura de Computadores 2016.2
Parte 1 07/10/16

Alexandre Diego Santos Silva

(2) matriz → $a0 vetor → $a1 i → $a2 j → $a3

```
Proc1:
        addi  $sp, $sp, -4
        sw    $ra, 0($sp)
while:
        add   $t0, $a2, $a2     # 2i
        add   $t0, $t0, $t0     # 4i
        add   $t0, $t0, $t0     # 8i
        add   $t0, $t0, $a1     # vetor[8i]
        lw    $t0, 0($t0)
        add   $t1, $a3, $a3     # 2j
        add   $t1, $t1, $t1     # 4j
        add   $t1, $t1, $a3     # 5j
        add   $t1, $t1, $a0     # matriz[5j]
        lw    $t2, 0($t1)       # matriz[5j] < vetor[8i]
        slt   $t3, $t2, $t0
        beq   $t3, $zero, exit_while
        addi  $sp, $sp, -20
        sw    $a0, 0($sp)       # matriz
        sw    $a1, 4($sp)       # vetor
        sw    $a2, 8($sp)       # i
        sw    $a3, 12($sp)      # j
        sw    $t1, 16($sp)      # 5j + base(matriz)
        add   $a2, $zero, $t0
        jal   Proc2             # Proc2(vetor[8i])
        lw    $t1, 16($sp)
        sw    $v0, 0($t1)       # matriz[5j] = $v0   (retorno de Proc2)
```

(×4 bytes) falta

(×4 bytes) falta

ok.

```
    lw $a0, 12($sp)
    jal Proc3
    lw $a0, 0($sp)      #matriz
    lw $a1, 4($sp)      #vetor
    lw $a2, 8($sp)      #i
    lw $t1, 16($sp)     # $s1 - base (matriz)
    addi $sp, $sp, 20                  # retorno de Proc3
    add $a3, $a2, $v0   # j = i + $v0
    j while
exit_while:
    add $v0, $zero, $a3   #$v0 = j
    lw $v1, 0($t1)        #$v1 = matriz[5,]
    lw $ra, 0($sp)
    addi $sp, $sp, 4
    jr $ra
```

$v0 retorna a qtd de elementos armazenados em C

① Procedimento:

```
                lui $a1, 0x8000
                ori $a1, 0x30F0
addi $a0, $zero, 100
lui $a1, 0x800030F0
                lui $a2, 0x8000
lui $a2, 0x800040C4
                ori $a2, 0x40C4
lui $a3, 0x80002080
                lui $a3, 0x8000
                ori $a3, 0x2080
addi $t0, $a0, -1       # i = 99
add $v0, $zero, $zero   # j = 0
while: slt $t3, $t0, $zero      # i < 0
bne $t3, $zero, exit_while
add $t4, $t0, $t0       # 2i
add $t4, $t4, $t4       # 4i
add $t1, $t4, $a1       # 4i + base (A)
lw $t1, 0($t1)          # A[i]
add $t2, $t4, $a2       # 4i + base (B)
lw $t2, 0($t2)          # B[i]
bne $t1, $zero, entra   # A[i] != 0 entra se chegar aqui então A[i] == 0
beq $t2, $zero, exit_while      # B[i] == 0 sai
entra:
slt $t3, $t2, $t1       # A[i] <= B[i]
add $t4, $v0, $v0       # 2j
add $t4, $t4, $t4       # 4j
add $t4, $t4, $a3       # 4j + base (C)
bne $t3, $zero, saveB   # (B[i] < A[i]) != 0
sw $t1, 0($t4)          # C[j] = A[i]
j pos_save
saveB:
sw $t2, 0($t4)          # C[j] = B[i]
pos_save: addi $t0, $t0, -1     # i--
addi $v0, $v0, 1        # j++
j while
exit_while: jr $ra
```