

## ATIVIDADE AVALIATIVA

### Instruções - 2

Estudante: Wagner Clemente Coelho Batalha 20/0044486

Estudante: Eder de Amaral Amorim 17/0140636

Obs.: a) Cole no texto de solução de cada exercício *print* de tela do simulador MARS (ou DrMIPS) com o código em assembly, tabela de registradores (abra Registers) e tabela de memória RAM (Data Segment). Use-o para buscar erros de sintaxe.

b) Introduza valores nos registradores e memória (manualmente, diretamente nas tabelas) para verificar o funcionamento do código assembly. Por padrão, há somente zero em todas as posições no início da execução.

c) Para carregar (“atribuir”) valor a um registrador via código assembly, é possível usar a instrução `addi`. Ex.: `addi $t0, $zero, 2` (o efeito é `$t0=2`). Isso ajuda a verificar o funcionamento do código.

d) Caso não for possível visualizar o valor na RAM (Data Segment), altere o offset (`load`, `store`) para um número negativo (-10, -100, -1000, -10000 etc) até ele constar na tela. Escreva um comentário (#) no código assembly para avisar que foi necessário ajustar o offset. Essa problemática de *range* de endereços de memória depende da versão do MARS; basta avisar esse problema em comentário no código assembly que o/a estudante não perderá pontos na correção.

**Escreva em *assembly* do MIPS o seguinte código em C.**

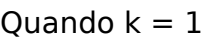
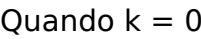
Sugestões: comece reescrevendo esse código como uma sequência de quatro *if-then*; no assembly, o *break* pode ser implementado como um *jump* (j) para a última linha do código (sai dessa estrutura); atenção para pular para o final do código após executar o equivalente de um *case*.

```
switch(k)
{
    case 0:f=i+j; break;
    case 1:f=g+h; break;
    case 2:f=g-h; break;
    case 3:f=i-j; break;
}
```

```

1  # if(k==0){
2  #     f=i+j;
3  #     goto FINAL;
4  # } else if (k==1){
5  #     f=g+h;
6  #     goto FINAL;
7  # } else if (k==2){
8  #     f=g-h;
9  #     goto FINAL;
10 # } else if (k==3){
11 #     f=i-j;
12 #     goto FINAL;
13 # }
14 #
15 # FINAL:
16
17 # k = $s0
18 # g = $s1
19 # h = $s2
20 # i = $s3
21 # j = $s4
22 # f = $s5
23
24 #####
25 # Valores de debug
26 li $s3, 2                # i = 2
27 li $s4, 3                # j = 3
28 li $s1, 4                # g = 4
29 li $s2, 5                # h = 5
30 #####
31
32
33 beq $s0, $zero, L0        # k = 0
34 li $s5, 1
35 beq $s0, $s5, L1         # k = 1
36 li $s5, 2
37 beq $s0, $s5, L2         # k = 2
38 li $s5, 3
39 beq $s0, $s5, L3         # k = 3
40
41 L0: add $s5, $s3, $s4     # f = i + j
42 j FINAL                  # break
43 L1: add $s5, $s1, $s2     # f = g + h
44 j FINAL                  # break
45 L2: sub $s5, $s1, $s2     # f = g - h
46 j FINAL                  # break
47 L3: sub $s5, $s3, $s4     # f = i - j
48 j FINAL
49 FINAL:

```



Interface of the Mars 4.5.2 (Community) emulator. The main window displays assembly code in the "Text Segment" pane, with columns for instruction number, address, code, basic instruction, and source code. The code includes instructions like `addiu $10, $0, 0x0000...` and `beq $10, $zero, L0`. The "Data Segment" pane shows memory addresses and their corresponding values. The "Registers" pane on the right lists registers \$zero through \$31, their numbers, and their current values. The "Mars Messages" pane at the bottom shows the execution progress, indicating that the program is finished running.

Quando K = 2

Interface of the Mars 4.5.2 (Community) emulator, showing the same assembly code and registers as the previous image. The "Mars Messages" pane at the bottom shows the execution progress, indicating that the program is finished running.

Quando K = 3