



Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática - FACIN

LABORG

Prof. Dr. Rafael Garibotti

AULA SOBRE:

PROGRAMAÇÃO EM LINGUAGEM DE MONTAGEM DO MIPS

INTRODUÇÃO

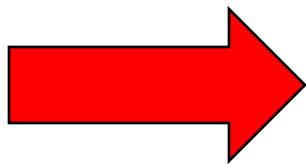
- Conjunto de instruções para estudo:
 - ✓ MIPS-I
- Arquitetura baseada no paradigma RISC.
- Ambiente de Simulação:
 - ✓ MIPS Assembler and Runtime Simulator. Fonte:
 - <http://courses.missouristate.edu/KenVollmar/MARS>
 - ✓ Comandos:
 - `java -jar Mars4_5.jar`
- Objetivo da aula:
 - ✓ Revisar os conceitos básicos de programação no MIPS.

COMANDO: IF THEN ELSE

IF THEN ELSE

➤ Mapeando linguagem de alto nível em Assembly

```
main() {  
    int i=4, j=6;  
  
    if(i==j)  
        i=i+2;  
    else  
        j=j-1;  
}
```



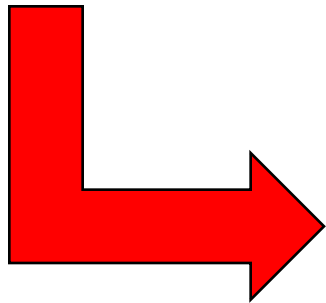
```
.text  
.globl main  
  
main:  
    li    $t0, 4           # i($t0) ← 4  
    li    $t1, 6           # j($t1) ← 6  
    beq   $t0, $t1, SeEntao # if (i==j)  
    subi  $t1, $t1, 1  
    j     fim  
  
SeEntao:  
    addi  $t0, $t0, 2  
  
fim:  
    jr    $ra
```

COMANDOS DE REPETIÇÃO

COMANDOS DE REPETIÇÃO

Exemplo 1 (Explicando o funcionamento):

```
main() {  
    int i;  
    int sum = 0;  
    for(i=0; i<=100; i=i+1)  
        sum += i * i;  
}
```

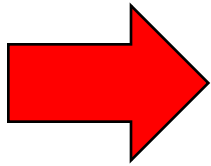


```
.text  
.globl main  
main:  
    move $t0, $zero        # sum ← 0;  
    move $t1, $zero        # i ← 0;  
    li    $t2, 100         # limite superior do for  
  
loop:  
    bgt   $t1, $t2, afterLoop # Verifica i<=100  
    mul   $t3, $t1, $t1      # i * i  
    add   $t0, $t0, $t3      # sum = sum + i * i;  
    add   $t1, $t1, 1        # i=i+1  
    j     loop              # volta p/ loop  
  
afterLoop:  
    jr    $ra
```

COMANDOS DE REPETIÇÃO

Exemplo 2 (Resolução interativa):

```
main() {  
    int index = 50;  
    int sumOfValues = 0;  
    do {  
        sumOfValues += index;  
        index = index - 1;  
    } while (index > 0);  
}
```



```
.text  
.globl main  
  
main:  
    li    $t0, 50          # index = 50;  
    move  $t1, $zero       # sumOfValues = 0;  
  
loop:  
    add   $t1, $t1, $t0     # sumOfValues+=index  
    addi  $t0, $t0, -1      # index--  
    bgtz  $t0, loop        # i > 0, volta p/ loop  
  
jr $ra
```


CHAMADA DE SISTEMA

CHAMADA DE SISTEMA

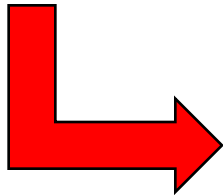
- Código da chamada de sistema fica no registrador **\$v0**.

Serviço	Código em \$v0	Argumentos	Resultados
print_int	1	\$a0 = o inteiro por imprimir	
print_float	2	\$f12 = o float por imprimir	
print_double	3	\$f12 = o double por imprimir	
print_string	4	\$a0 = endereço da string por imprimir	
read_int	5		\$v0 = o inteiro devolvido
read_float	6		\$f0 = o float devolvido
read_double	7		\$f0 = o double devolvido
read_string	8	\$a0 = endereço da string por ler \$a1 = comprimento da string	
sbrk/malloc	9	\$a0 = quantidade de memória por alocar	endereço em \$v0
exit	10	\$v0 = o código devolvido	

CHAMADA DE SISTEMA

Exemplo 1 (Associação com printf):

```
main() {  
    int x=5, y=3;  
  
    if((x+y)%2==0)  
        x=x+y;  
    else  
        x=y;  
  
    printf("O valor de  
        x é %d", x);  
}
```



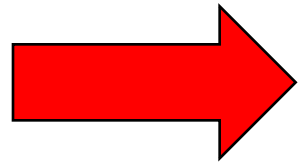
```
.data  
    #declaração das variáveis  
    x: .word 5  
    y: .word 3  
    #declaração do texto  
    texto: .asciiz "O valor de x é "  
.text  
    .globl main  
main:  
    lw    $t0, x($zero)    #x=5  
    lw    $t1, y($zero)    #y=3  
    add   $t2, $t0, $t1    #x+y  
    li    $t3, 2  
    ...
```

```
    rem   $t3, $t2, $t3    #(x+y)%2  
    beqz  $t3, IgualZero  #se == 0  
    move  $t0, $t1        #senao  
    j     printCoisa  
IgualZero:  
    add   $t0, $t0, $t1  
printCoisa:  
    li    $v0, 4  
    la    $a0, texto  
    syscall #imprime texto  
    li    $v0, 1  
    move  $a0, $t0  
    syscall #imprime inteiro  
  
    jr    $ra
```

CHAMADA DE SISTEMA

Exemplo 2 (Resolução interativa):

```
main() {  
    int x;  
    scanf("%d", &x);  
    printf("O valor lido  
    é %d", x);  
}
```



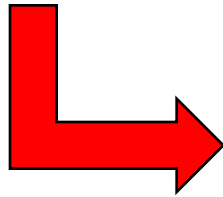
```
.data  
    txtLido: .asciiz "O valor lido é "  
.text  
    .globl main  
main:  
    li $v0, 5  
    syscall          #le inteiro  
    move $t0, $v0  
    li $v0, 4        #imprime texto  
    la $a0, txtLido  #endereço do texto  
    syscall  
    li $v0, 1  
    move $a0, $t0  
    syscall          #imprime valor lido  
    jr $ra
```

VETORES

VETORES

Exemplo 1 (Declarando e manipulando vetores):

```
main() {  
    int x[5]={0, 1, 2, 3, 4};  
    for(int i=0; i<5; i++)  
        printf("%d", x[i]);  
}
```

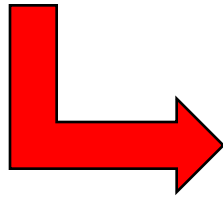


```
.data  
    x: .word 0 1 2 3 4 # Reserva 5 words de memória.  
                                # Adicionalmente inicializa os campos da memória.  
  
.text  
.globl main  
main:  
    li    $t0, 0           # i ← 0  
    li    $t1, 20          # Define última posicao do vetor  
  
loop:  
    li    $v0, 1           # Define o serviço  
    lw    $t2, x($t0)  
    move  $a0, $t2         # Define a posicao do vetor a ser impresso  
    syscall                # Imprime o valor de i  
    addi  $t0, $t0, 4       # i++, alinhado com a memória 4 bytes por palavra  
    bne   $t0, $t1, loop   # Se não é o fim do laço, volta para loop  
    jr    $ra
```

VETORES

Exemplo 2 (Declarando e manipulando vetores):

```
main() {  
    int x[10];  
    for(int i=0; i<10; i++)  
        scanf("%d", &x[i]);  
}
```



```
.data  
    x: .space 40    # Reserva 40 bytes na memória  
    ou  
    x: .word 0:10   # Reserva 10 words de memória e inicializa campos com 0  
.text  
.globl main  
main:  
    li    $t0, 0      # i ← 0  
    li    $t1, 40     # Define última posicao do vetor  
loop:  
    li    $v0, 5  
    syscall  
    sw    $v0, x($t0)  
    addi  $t0, $t0, 4  # i++, alinhado com a memória 4 bytes por palavra  
    bne   $t0, $t1, loop # Se não é o fim do laço, volta para loop  
    jr    $ra
```

SUPOORTE A SUBROTINAS

SUORTE A SUBROTINAS

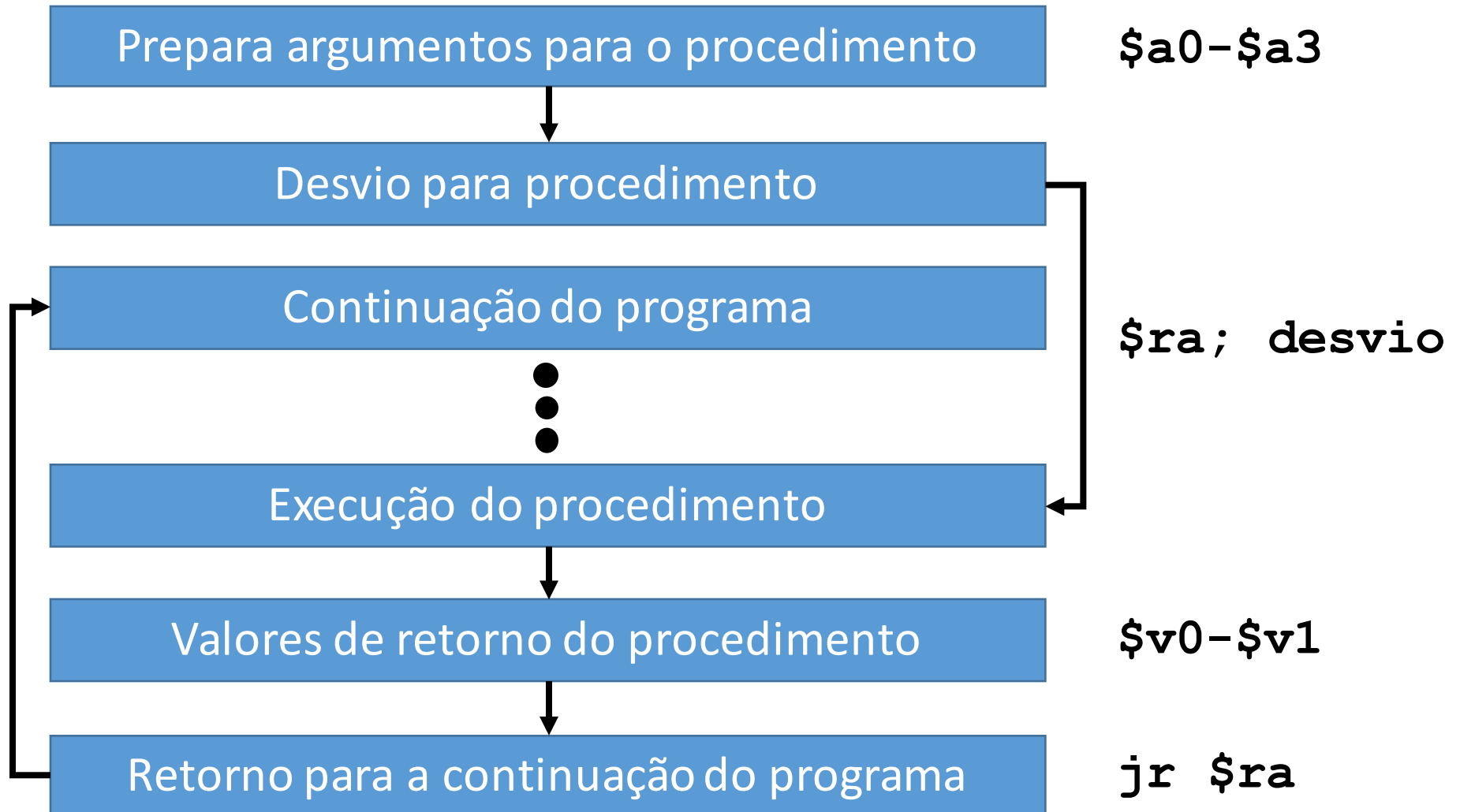
➤ Qual o lugar mais rápido que pode armazenar dados a serem capturados pela subrotina?

✓ **Registradores!**

Registradores MIPS:

- ✓ \$a0 - \$a3: parâmetros para a subrotina;
- ✓ \$v0 - \$v1: valores de retorno da subrotina;
- ✓ \$ra: registrador de endereço de retorno ao ponto de origem (*ra = return address*).

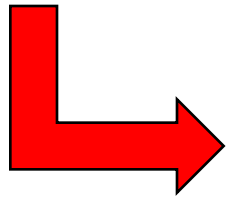
SUPOORTE A SUBROTINAS



CHAMADA DE SUBROTINAS

Exemplo 1 (Passando argumentos):

```
main() {  
    int index;  
    scanf("%d", &index);  
    ImprimeValor(index);  
}  
  
ImprimeValor(int _parametro){  
    printf("Imprimindo valor %d", _parametro);  
}
```

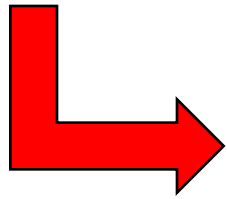


```
.data  
    txtImpressao: .asciiz "Imprimindo valor "  
.text  
    .globl main  
main:  
    move $s0, $ra    # salva o endereço de retorno  
    li    $v0, 5  
    syscall  
    move $a0, $v0  
    jal ImprimeValor # chama subrotina e salva  
                        #o ponto de retorno  
  
    move $ra, $s0  
    jr $ra  
    ...
```

CHAMADA DE SUBROTINAS

Exemplo 1 (Passando argumentos):

```
main() {  
    int index;  
    scanf("%d", &index);  
    ImprimeValor(index);  
}  
  
ImprimeValor(int _parametro){  
    printf("Imprimindo valor %d", _parametro);  
}
```

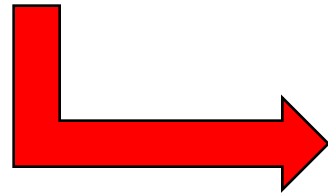


```
...  
ImprimeValor:  
    move $t0, $a0 # salva valor a ser impresso  
    li    $v0, 4  
    la    $a0, txtImpressao  
    syscall      # imprime texto  
    li    $v0, 1  
    move  $a0, $t0 # restaura valor a ser impresso  
    syscall  
    jr    $ra
```

CHAMADA DE SUBROTINAS

Exemplo 2 (Retorno de subrotinas):

```
main() {  
    int operadorA;  
    scanf("%d", &operadorA);  
    operadorA = incrementa(operadorA);  
}  
  
incrementa(int _opA){  
    return _opA + 1;  
}
```

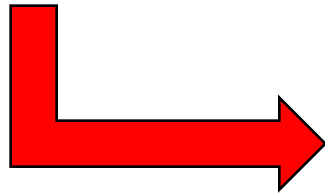


```
.data  
    operadorA: .space 4  
.text  
    .globl main  
main:  
    move $s0, $ra # salva o endereço de retorno  
    li    $v0, 5  
    syscall  
    move $a0, $v0 # define parametro  
    jal inc # chama subrotina e salva o ponto de retorno  
    sw    $v0, operadorA($zero)  
    move $ra, $s0 # restaura ponto de retorno anterior  
    jr    $ra  
inc:  
    addi $v0, $a0, 1 # retorno feito com o registrador $v0  
    jr    $ra
```

CHAMADA DE SUBROTINAS

Exemplo 2 (Retorno de subrotinas):

```
main() {  
    int operadorA;  
    scanf("%d", &operadorA);  
    operadorA = incrementa(operadorA);  
}  
  
incrementa(int _opA) {  
    return _opA + 1;  
}
```



```
.data  
    operadorA: .space 4  
.text  
    .globl main  
main:  
    move $s0, $ra # salva o endereço de retorno  
    li    $v0, 5  
    syscall  
    move $a0, $v0 # define parametro  
    jal inc # chama subrotina e salva o ponto de retorno  
    sw    $v0, operadorA($zero)  
    move $ra, $s0 # restaura ponto de retorno anterior  
    jr    $ra  
inc:  
    addi $v0, $a0, 1 # retorno feito com o registrador $v0  
    jr    $ra
```

TRABALHO

TRABALHO 2B

- Escreva um programa em linguagem de montagem do MIPS que leia uma matriz de anos bissextos (**BISSEXTO**), e informe quais deles são anos bissextos ou não. Você recebe a linha do usuário a qual deverá verificar os anos. Garanta que a linha informada pelo usuário exista, caso contrário, peça uma nova linha até que esta seja válida. Ao final da análise da linha, a quantidade de anos bissextos encontrados deve ser gravada no espaço de memória **CNT**, e cada ano bissexto encontrado no vetor **RESULTADO**. Lembre-se que um ano é bissexto **se $(ano \bmod 4 == 0$ e $(ano \bmod 100 != 0$ ou $ano \bmod 400 == 0)$**). Por fim, durante toda a execução do programa, mantenha o usuário informado.

TRABALHO 2B

➤ A área de dados que deverá ser utilizada é:

```
.data
BISSEXTO:    .word 2016 2012 2008 2004 2000
              1916 1911 1908 1904 1900
              1861 1857 1852 1846 1844
              1728 1727 1726 1725 1723

LINHA:       .word 4
COLUNA:      .word 5
CNT:         .word 0
RESULTADO:   .word 0 0 0 0 0
TEXTO_1:     .asciiz "Qual linha deseja verificar?"
TEXTO_2:     .asciiz "Total de anos bisextos: "
TEXTO_3:     .asciiz "\nOs anos bisextos são: "
TEXTO_4:     .asciiz ", "
```

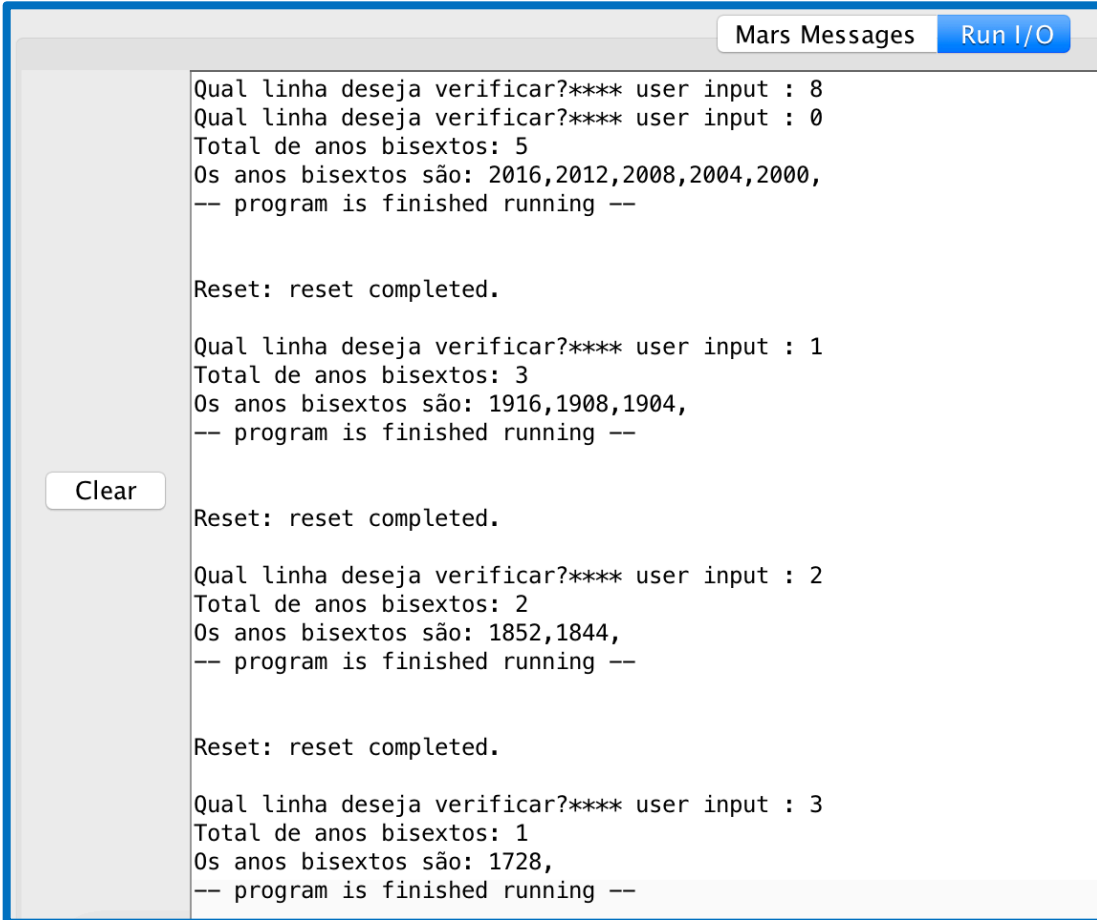
TRABALHO 2B

- Ao final da execução do assembly gerado, o seguinte resultado é esperado para as variáveis destacadas. Neste exemplo, o usuário indicou a linha 0!

```
CNT:          .word 5  
RESULTADO:    .word 2016 2012 2008 2004 2000
```

TRABALHO 2B

- A tela ao lado indica a execução de algumas vezes o programa esperado no Mars. Vale observar que a primeira linha indicada é 8, o qual é inválida por ser maior que o número de linhas da matriz que contém os anos. Desta forma, foi pedida uma nova vez que o usuário indicasse uma linha para verificação. Já nas outras simulações, foi informado uma linha válida, onde podemos observar qual é a resposta esperada!



The screenshot shows the 'Mars Messages' window with a 'Run I/O' button. The output text is as follows:

```
Qual linha deseja verificar?**** user input : 8
Qual linha deseja verificar?**** user input : 0
Total de anos bisextos: 5
Os anos bisextos são: 2016,2012,2008,2004,2000,
-- program is finished running --

Reset: reset completed.

Qual linha deseja verificar?**** user input : 1
Total de anos bisextos: 3
Os anos bisextos são: 1916,1908,1904,
-- program is finished running --

Reset: reset completed.

Qual linha deseja verificar?**** user input : 2
Total de anos bisextos: 2
Os anos bisextos são: 1852,1844,
-- program is finished running --

Reset: reset completed.

Qual linha deseja verificar?**** user input : 3
Total de anos bisextos: 1
Os anos bisextos são: 1728,
-- program is finished running --
```

A 'Clear' button is visible on the left side of the window.

RESUMO DO TRABALHO 2B

- O **Trabalho 2B (T2B)** consiste em um arquivo compactado (.zip) contendo:
 - ✓ Um relatório em PDF descrevendo a implementação do problema.
 - ✓ O código em linguagem de montagem do MIPS.
 - ✓ Dicas:
 - Você irá encontrar no material de apoio um código em C que resolve o problema proposto! Use ele como base para criar o seu programa em linguagem de montagem do MIPS.
 - Você irá encontrar no material de apoio um template do código assembly com a área de dados e as funções para preenchimento.