

# TP4

Assembly

## Integrantes

Guilherme Kollet  
Bruno Thofehrn

## Especificação

03

# Pseudocódigo do Programa

Essa seção comprehende a execução do algoritmo.

## Variáveis

**A[]** <-- 810 ,100, 560, 380, 600, 87

**B[]** <-- 800, 555, 817, 124, 890, 456

**C[]** <-- 345, 200, 700, 180, 600, 490

**n** <-- 6

**k** <-- 0

**MIN** <-- 1000

**sumA** <-- 0

**sumB** <-- 0

**sumC** <-- 0

**D[]** <-- 0

## Somatório

**enquanto( i = 0; i < n; i++)**

**sumA** +<-- A[i]

**sumB** +<-- B[i]

**sumC** +<-- C[i]

## Calcula Média

//divide o somatorio pelo n

**sumA** <-- sumA / n;

**sumB** <-- sumB / n;

**sumC** <-- sumC / n;

## Acha valor “mínimo”

//encontra o valor min entre sumA, sumB, sumC

**se**(sumA < MIN)

**MIN** <-- sumA

**se**(sumB < MIN)

**MIN** <-- sumB

**se**(sumC < MIN)

**MIN** <-- sumC

## Percorre vetores e incrementa o vetor D

**enquanto( i = 0; i < n; i++)**

**se(A[i] < MIN)**

**D[k] +<-- A[i]**

**k++**

**enquanto( i = 0; i < n; i++)**

**se(B[i] < MIN)**

**D[k] +<-- B[i]**

**k++**

**enquanto( i = 0; i < n; i++)**

**se(B[i] < MIN)**

**D[k] +<-- C[i]**

**k++**

O pseudocódigo foi implementado em C++ e está disponível no seguinte repositório:

[https://github.com/guilhermekollet/Assembly\\_Somatorio\\_Vetores.git](https://github.com/guilhermekollet/Assembly_Somatorio_Vetores.git)

Afim de complementar essa seção, o código estará contido nas próximas páginas.

## Algoritmo em C++

Include da library de output.

```
1 #include <iostream>
```

Declaração de variáveis.

```
3 int main()
4 {
5     int A [] = {810,100, 560, 380, 600, 87};
6     int B [] = {800, 555, 817, 124, 890, 456};
7     int C [] = {345, 200, 700, 180, 600, 490};
8
9     int n = 6;
10    int k = 0;
11
12    int MIN = 1000;
13
14    int sumA = 0;
15    int sumB = 0;
16    int sumC = 0;
17
18    int D[50];
```

Somatório.

```
22 for(int i = 0; i < n; i++)
23 {
24
25     sumA += A[i];
26     sumB += B[i];
27     sumC += C[i];
28
29 }
```

## Calcula média.

```
31 | sumA = sumA / n;
32 | sumB = sumB / n;
33 | sumC = sumC / n;
```

## Achando o valor Mínimo.

```
35 | if(sumA < MIN)
36 | MIN = sumA;
37 |
38 | if(sumB < MIN)
39 | MIN = sumB;
40 |
41 | if(sumC < MIN)
42 | MIN = sumC;
```

## Percorre o Vetor A e incrementa o Vetor D.

```
44 | for(int i = 0; i < n; i++)
45 | {
46 |   if(A[i] < MIN)
47 |   {
48 |     D[k] = A[i];
49 |     std::cout << D[k] << " ";
50 |     k++;
51 |   }
52 | }
```

## Percorre o Vetor B e incrementa o Vetor D.

```
54 | for(int i = 0; i < n; i++)
55 | {
56 |   if(B[i] < MIN)
57 |   {
58 |     D[k] = B[i];
59 |     std::cout << D[k] << " ";
60 |     k++;
61 |   }
62 | }
```

Percorre o Vetor C e incrementa o Vetor D.

```
64 | for(int i = 0; i < n; i++)
65 | {
66 |     if(C[i] < MIN)
67 |     {
68 |         D[k] = C[i];
69 |         std::cout << D[k] << " ";
70 |         k++;
71 |     }
72 | }
```

As linhas **49, 59, 69**, contemplam o fluxo de stream cout que exibe o conteúdo do vetor D na execução do arquivo **app.exe**.

```
73 | std::cout << "- k: " << k;
74 | std::cin >> n; //apenas para não encerrar a execução
75 |
76 | return 0;
77 }
```

# Tabela de Registradores

Essa seção estabelece as principais variáveis do programa.

Registrador	Variável ou Valor	Comentário
\$t0	0	int pos = 0
\$t1	0	int i = 0
\$t2	n	int n = 6
\$t3	MIN	int MIN = 1000
\$t4	sumA	int sumA = 0
\$t5	sumB	int sumB = 0
\$t6	sumC	int sumC = 0
\$t7	*Temporário*	uso geral
\$t9	k	int k = 0
\$s0	A	A [ ... ]
\$s1	B	B [ ... ]
\$s2	C	C [ ... ]
\$s3	D	D [ ... ]

# Área de Dados Solução

Essa seção demonstra a execução do algoritmo desenvolvido.

## Antes

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	A	810	100	560	380	600	87	B
0x10010020		817	124	890	456	C	345	200
0x10010040		600	490	n	6	k	0	MIN
0x10010060		0	0	0	0	D	0	0

## Depois

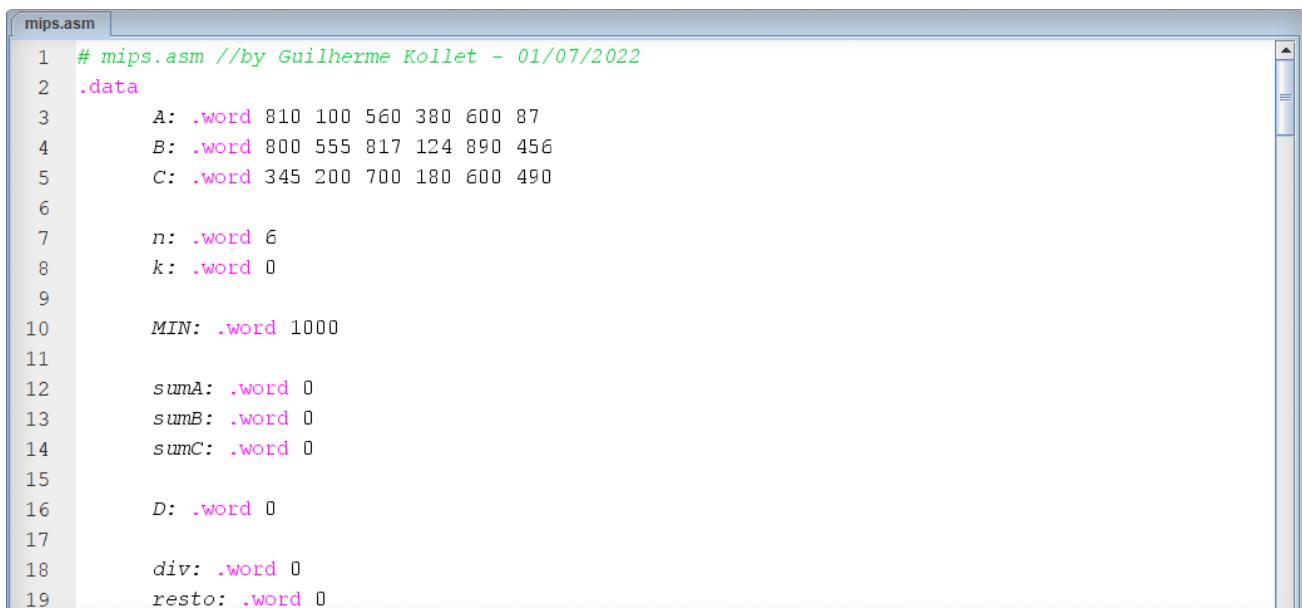
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	A	810	100	560	380	600	87	B
0x10010020		817	124	890	456	C	345	200
0x10010040		600	490	n	6	k	7	MIN
0x10010060	D	100	380	87	124	345	419	médiaA

Resultado esperado de acordo com o assembly.cpp

```
C:\Users\Guisk\Documents\GitHub\Assembly_Sol> 100 380 87 124 345 200 180 - k: 7
```

# Capturas do Mars

Essa seção exibe algumas capturas de telas realizadas no simulador Mars.



```
mips.asm
1 # mips.asm //by Guilherme Kollet - 01/07/2022
2 .data
3     A: .word 810 100 560 380 600 87
4     B: .word 800 555 817 124 890 456
5     C: .word 345 200 700 180 600 490
6
7     n: .word 6
8     k: .word 0
9
10    MIN: .word 1000
11
12    sumA: .word 0
13    sumB: .word 0
14    sumC: .word 0
15
16    D: .word 0
17
18    div: .word 0
19    resto: .word 0
```

A figura acima exibe a área .data que declara algumas variáveis que são utilizadas na execução. Além da solução exibida na página anterior, a execução teve ao total 234 linhas, considerando o algoritmo de divisão disponibilizado pelo professor.

# Nova área de dados

Essa seção demonstra uma nova execução, com vetores A, B e C modificados.

**A:** .word 345 100 560 943

**B:** .word 800 255 817 765

**C:** .word 740 200 700 180

**n:** .word 4

**k:** .word 0

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	A 345	100	560	943	B 800	255	817	765
0x10010020	C 740	200	700	180	n 4	k 5	MIN 455	médiaA 487
0x10010040	médiaB 659	médiaC 455	D 345	100	255	200	180	0

Resultado esperado de acordo com o assembly.cpp modificado.

```
C:\Users\Guisk\Documents\GitHub\  
345 100 255 200 180 - k: 5
```

Podemos observar que, o valor MIN encontrado nos vetores foi 455. O mesmo serviu de base para compor o nosso Vetor D.

# Capturas do Modelsim

Essa seção exibe as capturas de telas realizadas no simulador Modelsim.

Para a solução proposta na 3º especificação será simulado no Modelsim.

## Carregamento dos dados

DATA MEMORY	
mem0	32'dX 810
mem1	32'dX 100
mem2	32'dX 560
mem3	32'dX 380
mem4	32'dX 600
mem5	32'dX 87
mem6	32'dX 800
mem7	32'dX 555
mem8	32'dX 817
mem9	32'dX 124
mem10	32'dX 890
mem11	32'dX 456
mem12	32'dX 345
mem13	32'dX 200
mem14	32'dX 700
mem15	32'dX 180
mem16	32'dX 600
mem17	32'dX 490
mem18	32'dX 6

A: .word 810 100 560 380 600 87  
B: .word 800 555 817 124 890 456  
C: .word 345 200 700 180 600 490  
n: .word 6  
k: .word 0

The screenshot shows the Modelsim Data Memory viewer with 19 memory locations (mem0 to mem18). Red arrows point from specific memory cells to the corresponding assembly code values:

- mem0: 32'dX 810 → A: .word 810
- mem1: 32'dX 100 → B: .word 100
- mem2: 32'dX 560 → C: .word 560
- mem3: 32'dX 380 → A: .word 380
- mem4: 32'dX 600 → B: .word 600
- mem5: 32'dX 87 → C: .word 87
- mem6: 32'dX 800 → A: .word 800
- mem7: 32'dX 555 → B: .word 555
- mem8: 32'dX 817 → C: .word 817
- mem9: 32'dX 124 → A: .word 124
- mem10: 32'dX 890 → B: .word 890
- mem11: 32'dX 456 → C: .word 456
- mem12: 32'dX 345 → A: .word 345
- mem13: 32'dX 200 → B: .word 200
- mem14: 32'dX 700 → C: .word 700
- mem15: 32'dX 180 → A: .word 180
- mem16: 32'dX 600 → B: .word 600
- mem17: 32'dX 490 → C: .word 490
- mem18: 32'dX 6 → n: .word 6
- mem18: 32'dX 6 → k: .word 0

## Cálculo das Médias

Considerando que o resultado do cálculo das médias ficariam armazenados nos registradores temporários: \$t4, \$t5, \$t6. Temos:

+◆ r10 (t2)	32'd6	6
+◆ r11 (t3)	32'd419	419
+◆ r12 (t4)	32'd422	0
+◆ r13 (t5)	32'd607	422
+◆ r14 (t6)	32'd419	0
+◆ r15 (t7)	32'd810	607
		419

Importante salientar, o \$t3 é o registrador responsável por guardar o valor mínimo entre os somatórios A, B e C.

Na próxima página é possível vermos o carregamento dos Vetores.

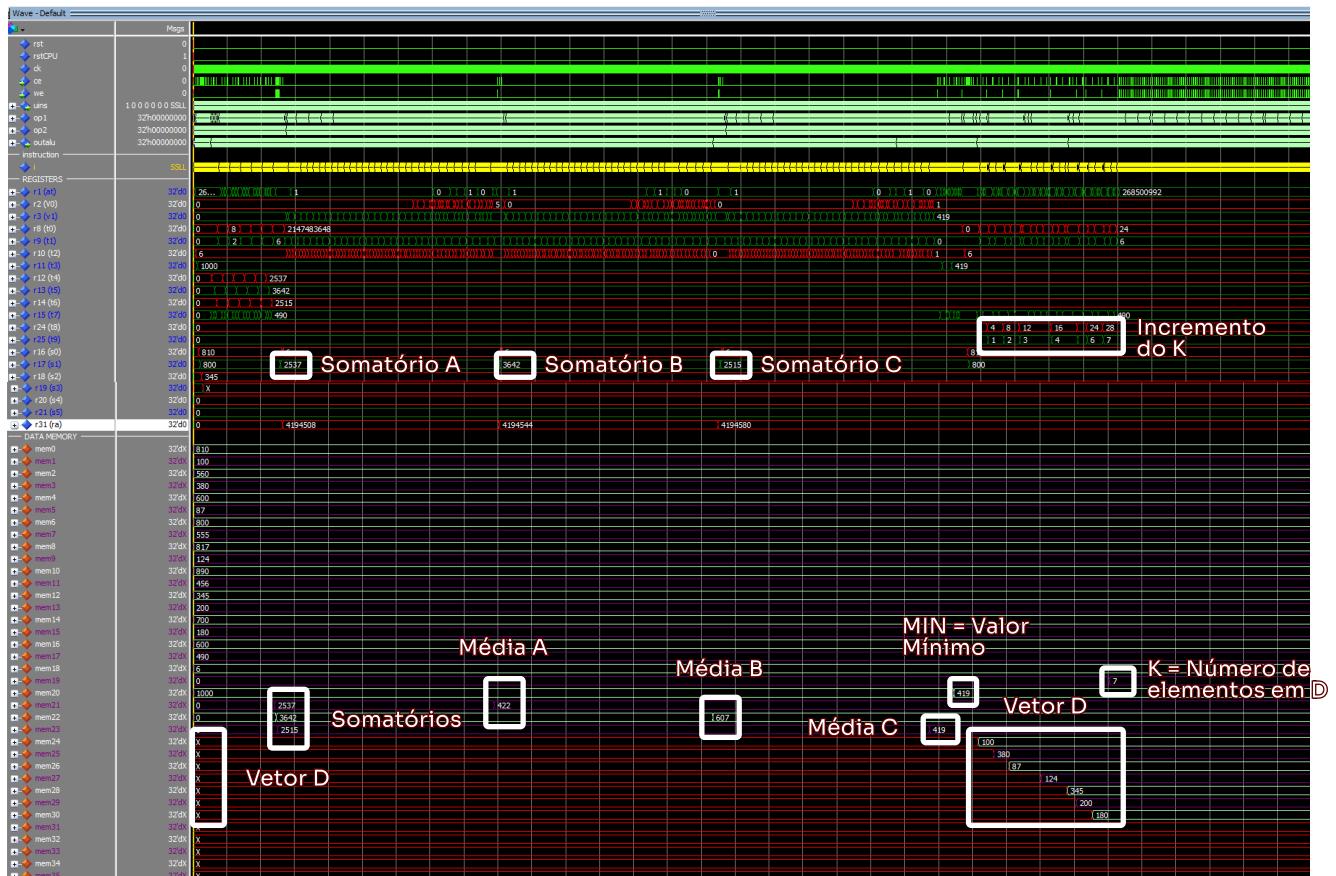
## Carregamento dos Vetores

Os vetores A, B e C, estão organizados respectivamente pelos registradores: \$s0, \$s1, \$s2.

r16 (\$s0)	32'd0	0   810
r17 (\$s1)	32'd0	0   800
r18 (\$s2)	32'd0	0   345
r19 (\$s3)	32'd0	0   100
r20 (\$s4)	32'd0	0

O Vetor D, é registrado no \$s3, podemos observar na imagem acima o carregamento linear dos vetores e o incremento do Vetor D, considerando o valor mínimo de 419. A segunda posição do Vetor A, corresponde ao 100, que foi registrado no novo ciclo do loop.

## Carregamento dos Vetores e Visão Geral



Podemos observar com ajuda deste diagrama o carregamento do vetor D na memória, realizado através de uma série de registradores e circuitos combinacionais da ULA.