

**Pseudocódigo em C++:**

```

int A[8] = {120, 50, -80, -100, 180, -130, 40, -20};
int B[8] = {80, -600, 100, -70, -40, 20, -90, 60};
int C[8] = {0, 0, 0, 0, 0, 0, 0, 0};
int D[8] = {0, 0, 0, 0, 0, 0, 0, 0};
int somaC = 0;
int somaD = 0;
int maior = INT_MIN;

for (int i = 0; i < 8; ++i) {
    C[i] = A[i] + B[i];
    D[i] = A[i] - B[i];

    somaC += C[i];
    somaD += D[i];

    if (C[i] > D[i]) {
        if (C[i] > maior) {
            maior = C[i];
        }
    } else {
        if (D[i] > maior) {
            maior = D[i];
        }
    }
}

int SM = 0;
for (int i = 0; i < maior; ++i) {
    SM += somaC + somaD;
}

```

O pseudocódigo realiza operações em dois arrays A e B. Para cada posição i:

- Calcula  $C[i] = A[i] + B[i]$  e  $D[i] = A[i] - B[i]$ .
- Ao mesmo tempo, acumula o somatório de todos os valores em C e D.
- Durante essas operações, determina o maior valor entre  $C[i]$  e  $D[i]$ . Inicialmente, compara  $C[i]$  com  $D[i]$ . Se  $C[i]$  é maior que  $D[i]$ , verifica se  $C[i]$  é maior que um valor "maior" atual. Se sim, atualiza "maior" para  $C[i]$ . Caso contrário, verifica se  $D[i]$  é maior que o valor atual de "maior" e atualiza "maior" para  $D[i]$  se for o caso. Se nenhum dos valores for maior que o atual "maior", o valor de "maior" permanece inalterado.
- No fim, soma o valor de "soma" a si mesmo "maior" vezes, realizando então uma multiplicação.

**Tabela que relaciona as principais variáveis com registradores:**

Registrador	Variável
\$t0	A[ ]
\$t1	B[ ]
\$t2	C[ ]
\$t3	D[ ]
\$t4	somaC
\$t5	somaD
\$t6	maior
\$t7	soma
\$t8 e \$t9	index

### Exemplo da Área de dados:

```
.data

A:      .word 15 12 -17 25 -30 -18 22 11
B:      .word 20 -14 19 -16 -28 13 -10 27
C:      .word 0 0 0 0 0 0 0 0
D:      .word 0 0 0 0 0 0 0 0
somaC:  .word 0
somaD:  .word 0
maior:  .word 0
soma:   .word 0
```

Primeiramente, são realizadas a soma e a subtração nos arrays C e D, respectivamente:

$C = \{ 35, -2, 2, 9, -58, -5, 12, 38 \}$

$D = \{ -5, 26, -36, 41, -2, -31, 32, -16 \}$

Conforme há a adição de valores nos arrays, é verificado qual o maior valor e também realizado o somatório de C e D:

- Para  $C[1] = 35$  e  $D[1] = -5$ , o maior valor inicial é 0. A verificação entre todos os valores atualiza "maior" para 35.
- $somaC = (35 + (-2) + 2 + 9 + (-58) + (-5) + 12 + 38) \rightarrow somaC = 31$ .
- $somaD = (-5 + 26 + (-36) + 41 + (-2) + (-31) + 32 + (-16)) \rightarrow somaD = 9$ .

O processo é realizado até o fim dos arrays A e B (index = 8).

Após a execução especificada geral, é realizada a especificação de matrícula (Especificação 3):

- $soma = maior * (somaC + somaD)$ .
- $soma = 41 * (31 + 9) = 1640$ .

Todos os processos envolvendo os registradores são salvos na memória após a execução do programa.

## Capturas do Mars:

Text Segment: Processos a executar(dump .text).

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui \$1,4097	5: la \$s0, A # carrega endereço de A
	0x00400004	0x34300000	ori \$16,\$1,0	
	0x00400008	0x3c011001	lui \$1,4097	6: la \$s1, B # carrega endereço de B
	0x0040000c	0x34310020	ori \$17,\$1,32	
	0x00400010	0x3c011001	lui \$1,4097	7: la \$s2, C # carrega endereço de C
	0x00400014	0x34320040	ori \$18,\$1,64	
	0x00400018	0x3c011001	lui \$1,4097	8: la \$s3, D # carrega endereço de D
	0x0040001c	0x34330060	ori \$19,\$1,96	
	0x00400020	0x3c011001	lui \$1,4097	9: la \$s4, somaC # carrega endereço de somaC
	0x00400024	0x34340080	ori \$20,\$1,128	
	0x00400028	0x3c011001	lui \$1,4097	10: la \$s5, somaD # carrega endereço de somaD
	0x0040002c	0x34350084	ori \$21,\$1,132	
	0x00400030	0x3c011001	lui \$1,4097	11: la \$s6, maior # carrega endereço de maior
	0x00400034	0x34360088	ori \$22,\$1,136	
	0x00400038	0x3c011001	lui \$1,4097	12: la \$s7, soma # carrega endereço de soma
	0x0040003c	0x3437008c	ori \$23,\$1,140	
	0x00400040	0x0000c021	addu \$24,\$0,\$0	13: move \$t8, \$zero # t9 -> i = 0
	0x00400044	0x8e080000	lw \$8,0(\$16)	16: lw \$t0, 0(\$s0) # carrega A[i] em \$t0
	0x00400048	0x8e290000	lw \$9,0(\$17)	17: lw \$t1, 0(\$s1) # carrega B[i] em \$t1
	0x0040004c	0x01095020	add \$10,\$8,\$9	19: add \$t2, \$t0, \$t1 # C[i](\$t2) = A[i] + B[i]
	0x00400050	0x018a6020	add \$12,\$12,\$10	20: add \$t4, \$t4, \$t2 # t4 += C[i]
	0x00400054	0x01095022	sub \$11,\$8,\$9	22: sub \$t3, \$t0, \$t1 # D[i](\$t3) = A[i] - B[i]
	0x00400058	0x01ab6020	add \$13,\$13,\$11	23: add \$t5, \$t5, \$t3 # t5 += D[i]
	0x0040005c	0x016a002a	slt \$1,\$11,\$10	25: bgt \$t2, \$t3, maiorC # se C[i] > D[i], vai para maiorC
	0x00400060	0x14200003	bne \$1,\$0,3	
	0x00400064	0x01cb082a	slt \$1,\$14,\$11	26: bgt \$t3, \$t6, atualiza_maior # se D[i] > maior, atualiza maior
	0x00400068	0x14200004	bne \$1,\$0,4	
	0x0040006c	0x08100029	j 0x004000a4	28: j incrementa # se nenhum dos casos, vai para incrementa
	0x00400070	0x01ca082a	slt \$1,\$14,\$10	31: bgt \$t2, \$t6, atualiza_maior_t2 # se C[i] > maior, vai para atualiza_ma...
	0x00400074	0x14200003	bne \$1,\$0,3	
	0x00400078	0x08100029	j 0x004000a4	32: j incrementa
	0x0040007c	0x000b7021	addu \$14,\$0,\$11	35: move \$t6, \$t3 # atualiza maior com \$t3(D[i])
	0x00400080	0x08100029	j 0x004000a4	36: j incrementa
	0x00400084	0x000a7021	addu \$14,\$0,\$10	39: move \$t6, \$t2 # atualiza maior com \$t2(C[i])
	0x00400088	0x08100029	j 0x004000a4	40: j incrementa
	0x0040008c	0xae4a0000	sw \$10,0(\$18)	43: sw \$t2, 0(\$s2) # salva (A[i] + B[i]) em C[i]
	0x00400090	0xae6b0000	sw \$11,0(\$19)	44: sw \$t3, 0(\$s3) # salva (A[i] - B[i]) em D[i]
	0x00400094	0xae8c0000	sw \$12,0(\$20)	45: sw \$t4, 0(\$s4) # salva atual valor de somaC
	0x00400098	0xaeaa0000	sw \$13,0(\$21)	46: sw \$t5, 0(\$s5) # salva atual valor de somaD
	0x0040009c	0xaeec0000	sw \$14,0(\$22)	47: sw \$t6, 0(\$s6) # salva atual maior valor
	0x004000a0	0x03e00008	jr \$31	49: jr \$ra
	0x004000a4	0x0c100023	jal 0x0040008c	53: jal write_mem
	0x004000a8	0x22100004	addi \$16,\$16,4	55: addi \$s0, \$s0, 4 # incrementa posicao de A[]
	0x004000ac	0x22310004	addi \$17,\$17,4	56: addi \$s1, \$s1, 4 # incrementa posicao de B[]
	0x004000b0	0x22520004	addi \$18,\$18,4	57: addi \$s2, \$s2, 4 # incrementa posicao de C[]
	0x004000b4	0x22730004	addi \$19,\$19,4	58: addi \$s3, \$s3, 4 # incrementa posicao de D[]
	0x004000b8	0x23180001	addi \$24,\$24,1	59: addi \$t8, \$t8, 1 # i++
	0x004000bc	0x2b010008	slt \$1,\$24,8	61: blt \$t8, 8, loop # se i < 8, entao vai para loop
	0x004000c0	0x1420fff0	bne \$1,\$0,-32	
	0x004000c4	0x8e940000	lw \$20,\$0(\$20)	65: lw \$s4, 0(\$s4) # carrega valor de somaC
	0x004000c8	0x8eb50000	lw \$21,\$0(\$21)	66: lw \$s5, 0(\$s5) # carrega valor de somaD
	0x004000cc	0x8ed60000	lw \$22,\$0(\$22)	67: lw \$s6, 0(\$s6) # carrega valor de maior
	0x004000d0	0x02958020	add \$16,\$20,\$21	69: add \$s0, \$s4, \$s5 # s0 = somaC + somaD
	0x004000d4	0x0000c021	addu \$25,\$0,\$0	70: move \$t9, \$zero # i = 0
	0x004000d8	0x23390001	addi \$25,\$25,1	74: addi \$t9, \$t9, 1 # i++
	0x004000dc	0x01f07820	add \$15,\$15,\$16	75: add \$t7, \$t7, \$s0 # SM += (somaC + somaD)
	0x004000e0	0x0336082a	slt \$1,\$25,\$22	76: blt \$t9, \$s6, multi # adiciona "soma" a si mesmo, "maior" vezes
	0x004000e4	0x1420fff0	bne \$1,\$0,-4	
	0x004000e8	0xaeef0000	sw \$15,0(\$23)	79: sw \$t7, 0(\$s7)
	0x004000ec	0x0810003b	j 0x004000ec	80: fim: j fim

Data Segment (antes da execução):

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	15	12	-17	25	-30	-18	22	11
0x10010020	20	-14	19	-16	-28	13	-10	27
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0

A B C D SomaC SomaD Maior Soma

Data Segment (após a execução):

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	15	12	-17	25	-30	-18	22	11
0x10010020	20	-14	19	-16	-28	13	-10	27
0x10010040	35	-2	2	9	-58	-5	12	38
0x10010060	-5	26	-36	41	-2	-31	32	-16
0x10010080	31	5	41	1640	0	0	0	0

The screenshot shows a debugger window with a list of memory addresses (mem0 to mem35) on the left. The main window displays the values stored at these addresses. Annotations include:

- Criação de C =**: Points to the memory address 0x00000000, which contains the value 0.
- Criação de D = {**: Points to the memory address 0x00000000, which contains the value 0.
- Valor do somatório**: Points to the memory address 0x00000000, which contains the value 0.
- Valor do somatório**: Points to the memory address 0x00000000, which contains the value 0.
- Maior valor de**: Points to the memory address 0x00000000, which contains the value 0.

**Soma = 41 \* ( 31 + 9 )**

**Soma = 41 \* ( 31 + 9 )**

Capturas do ModelSim(Registradores):

