

# Parte 1:

1. O que é um arquivo fonte?

- A. um arquivo de texto que contém instruções de linguagem de programação.
- B. um subdiretório que contém os programas.
- C. um arquivo que contém dados para um programa.
- D. um documento que contém os requisitos para um projeto.

R: A

2. O que é um registrador?

- A. parte do sistema de computador que mantém o controle dos parâmetros do sistema.
- B. uma parte do processador que possui um padrão de bits.
- C. parte do processador que contém o seu número de série único.
- D. parte do bus de sistema que contém dados.

R: B

3. Qual o caracter que, na linguagem assembly do SPIM, inicia um comentário?

- A. #
- B. \$
- C. //
- D. \*

R: A

4. Quantos bits há em cada instrução de máquina MIPS?

- A. 8
- B. 16
- C. 32
- D. instruções diferentes possuem diferentes comprimentos.

R: C

5. O que é o contador de programa?

- A. um registrador que mantém a conta do número de erros durante a execução de um programa.
- B. uma parte do processador que contém o endereço da primeira palavra de dados.
- C. uma variável na montadora que os números das linhas do arquivo de origem.
- D. parte do processador que contém o endereço da próxima instrução de máquina para ser obtida.

R: D

6. Ao executarmos uma instrução, quanto será adicionado ao contador de programa?

- A. 1
- B. 2
- C. 4
- D. 8

R: C

7. O que é uma diretiva, tal como a diretiva .text?

- A. uma instrução em linguagem assembly que resulta em uma instrução em linguagem de máquina.
- B. uma das opções de menu do sistema SPIM.
- C. uma instrução em linguagem de máquina que faz com que uma operação sobre os dados ocorra.
- D. uma declaração que diz o montador algo sobre o que o programador quer, mas não corresponde diretamente a uma instrução de máquina.

R: D

8. O que é um endereço simbólico?

- A. um local de memória que contém dados simbólicos.
- B. um byte na memória que contém o endereço de dados.
- C. símbolo dado como argumento para uma directiva.

D. um nome usado no código-fonte em linguagem assembly para um local na memória.

R: D

9. Em qual endereço o simulador SPIM coloca a primeira instrução de máquina quando ele está sendo executado?

A. 0x00000000

B. 0x00400000

C. 0x10000000

D. 0xFFFFFFFF

R: B

10. Algumas instruções de máquina possuem uma constante como um dos operandos. Como é chamado tal operando?

A. operando imediato

B. operando embutido

C. operando binário

D. operando de máquina

R: A

11. Como é chamada uma operação lógica executada entre bits de cada coluna dos operandos para produzir um bit de resultado para cada coluna?

A. operação lógica

B. operação bitwise

C. operação binária

D. operação coluna

R: B

12. Quando uma operação é de fato executada, como estão os operandos na ALU?

A. Pelo menos um operando deve ser de 32 bit.

B. Cada operando pode ser de qualquer tamanho.

- C. Ambos operandos devem que vir de registros.
- D. Cada um dos registradores deve possuir 32 bit.

R: D

13. Dezesesseis bits de dados de uma instrução de ori são usados como um operando imediato. Durante execução, o que deve ser feito primeiro?

- A. Os dados são estendidos em zero à direita por 16 bits.
- B. Os dados são estendidos em zero à esquerda por 16 bits.
- C. Nada precisa ser feito.
- D. Apenas 16 bits são usados pelo outro operando.

R: B

14. Qual das instruções seguintes armazenam no registrador \$5 um padrão de bits que representa positivo 48?

- A. ori \$5,\$0,0x48
- B. ori \$5,\$5,0x48
- C. ori \$5,\$0,48
- D. ori \$0,\$5,0x48

R: C

15. A instrução de ori pode armazenar o complemento de dois de um número em um registrador?

- A. Não.
- B. Sim.

R: A

16. Qual das instruções seguintes limpa todos os bits no registrador \$8 com exceção do byte de baixa ordem que fica inalterado?

- A. ori \$8,\$8,0xFF
- B. ori \$8,\$0,0x00FF

- C. xori \$8,\$8,0xFF
- D. andi \$8,\$8,0xFF

R: D

17. Qual é o resultado de um ou exclusivo de padrão sobre ele mesmo?

- A. Todos os bits em zero.
- B. Todos os bits em um.
- C. O padrão original utilizado.
- D. O resultado é o contrário do original.

R: A

18. Todas as instruções de máquina têm os mesmos campos?

- A. Não. Diferentes de instruções de máquina possuem campos diferentes.
- B. Não. Cada instrução de máquina é completamente diferente de qualquer outra.
- C. Sim. Todas as instruções de máquina têm os mesmos campos na mesma ordem.
- D. Sim. Todas as instruções de máquina têm os mesmos campos, mas eles podem estar em ordens diferentes.

R: A

## Parte 2:

```
//programa 1 (add, addi, sub, lógicas)
{
a =2;
```

```
b =3;  
c =4;  
d =5;  
x = (a+b) - (c+d);  
y = a - b + x;  
b = x - y;  
}
```

EditExecute

programa1.asm

```

1  # Programa 1
2  # a = 2;
3  # b = 3;
4  # c = 4;
5  # d = 5;
6  # x = (a+b) - (c+d);
7  # y = a - b + x;
8  # b = x - y;
9
10
11 #inicio:
12 .text
13 .globl main
14 main:
15 addi $s0, $zero, 2 # a = 2;
16 addi $s1, $zero, 3 # b = 3;
17 addi $s2, $zero, 4 # c = 4;
18 addi $s3, $zero, 5 # d = 5;
19
20 # x = (a+b) - (c+d);
21 add $t0, $s0, $s1 # t0 = a + b;
22 add $t1, $s2, $s3 # t1 = c + d;
23 sub $s4, $t0, $t1 # x = t0 - t1;
24
25 # y = a - b + x;
26 sub $t2, $s0, $s1 # t2 = a - b;
27 add $s5, $t2, $s4 # y = t2 + x;
28
29 # b = x - y;
30 sub $s1, $s4, $s5 # b = x - y;

```

Line: 30 Column: 33 Show Line Numbers

EditExecute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20100002	addi \$16,\$0,0x00000002	15: addi \$s0, \$zero, 2 # a = 2;
	0x00400004	0x20110003	addi \$17,\$0,0x00000003	16: addi \$s1, \$zero, 3 # b = 3;
	0x00400008	0x20120004	addi \$18,\$0,0x00000004	17: addi \$s2, \$zero, 4 # c = 4;
	0x0040000c	0x20130005	addi \$19,\$0,0x00000005	18: addi \$s3, \$zero, 5 # d = 5;
	0x00400010	0x02114020	add \$8,\$16,\$17	21: add \$t0, \$s0, \$s1 # t0 = a + b;
	0x00400014	0x02534820	add \$9,\$18,\$19	22: add \$t1, \$s2, \$s3 # t1 = c + d;
	0x00400018	0x0109a022	sub \$20,\$8,\$9	23: sub \$s4, \$t0, \$t1 # x = t0 - t1;
	0x0040001c	0x02115022	sub \$10,\$16,\$17	26: sub \$t2, \$s0, \$s1 # t2 = a - b;
	0x00400020	0x0154a820	add \$21,\$10,\$20	27: add \$s5, \$t2, \$s4 # y = t2 + x;
	0x00400024	0x02958822	sub \$17,\$20,\$21	30: sub \$s1, \$s4, \$s5 # b = x - y;

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data)

☒ Hexadecimal Addresses
 ☒ Hexadecimal Values
 ☐ ASCII

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000005
\$t1	9	0x00000009
\$t2	10	0xffffffff
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000002
\$s1	17	0x00000003
\$s2	18	0x00000004
\$s3	19	0x00000005
\$s4	20	0xffffffff
\$s5	21	0xffffffff
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400028
hi		0x00000000
lo		0x00000000

//programa 2 (add, addi, sub, lógicas)

```

{
x = 1;
y = 5*x + 15;
}

```

EditExecute

programa1.asm

programa2.asm

```

1  # Programa 2
2  # x = 1;
3  # y = 5 * x + 15;
4
5
6  #inicio:
7
8  .text
9  .globl main
10 main:
11 addi $s0, $zero, 1 # x = 1;
12
13 # y = 5 * x + 15;
14 add $t0, $t0, $s0 # t0 = X;
15 add $t0, $t0, $s0 # t0 = 2X;
16 add $t0, $t0, $s0 # t0 = 3X;
17 add $t0, $t0, $s0 # t0 = 4X;
18 add $t0, $t0, $s0 # t0 = 5X;
19
20 addi $s1, $t0, 15 # y = t0 + 15;

```

Line: 20 Column: 34 ☒ Show Line Numbers

EditExecute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x20100001	addi \$16,\$0,0x00000001	11: addi \$s0, \$zero, 1 # x = 1;
<input type="checkbox"/>	0x00400004	0x01104020	add \$8,\$8,\$16	14: add \$t0, \$t0, \$s0 # t0 = X;
<input type="checkbox"/>	0x00400008	0x01104020	add \$8,\$8,\$16	15: add \$t0, \$t0, \$s0 # t0 = 2X;
<input type="checkbox"/>	0x0040000c	0x01104020	add \$8,\$8,\$16	16: add \$t0, \$t0, \$s0 # t0 = 3X;
<input type="checkbox"/>	0x00400010	0x01104020	add \$8,\$8,\$16	17: add \$t0, \$t0, \$s0 # t0 = 4X;
<input type="checkbox"/>	0x00400014	0x01104020	add \$8,\$8,\$16	18: add \$t0, \$t0, \$s0 # t0 = 5X;
<input type="checkbox"/>	0x00400018	0x2111000f	addi \$17,\$8,0x0000000f	20: addi \$s1, \$t0, 15 # y = t0 + 15;

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data)

☒ Hexadecimal Addresses
 ☒ Hexadecimal Values
 ☐ ASCII

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000005
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
<b>\$s0</b>	<b>16</b>	<b>0x00000001</b>
\$s1	17	0x00000014
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffc00
\$fp	30	0x00000000
\$ra	31	0x00000000
\$pc		0x0040001c
\$hi		0x00000000
\$lo		0x00000000

```
// programa 3 (add, addi, sub, lógicas)
```

```
{
```

```
x = 3;
```

```
y = 4 ;
```

```
z = ( 15*x + 67*y)*4
```

```
}
```



EditExecute

programa3.asm

```

4  # z = ( 15 * x + 67 * y ) * 4;
5
6
7  #inicio:
8  .text
9  .globl main
10 main:
11 addi $s0, $zero, 3    # x = 3;
12 addi $s1, $zero, 4    # y = 4;
13
14 # z = ( 15 * x + 67 * y ) * 4;
15 add $t0, $t0, $s0    # t0 = x;
16 add $t0, $t0, $t0    # t0 = 2x;
17 add $t0, $t0, $t0    # t0 = 4x;
18 add $t0, $t0, $s0    # t0 = 5x;
19 add $t1, $t0, $t0    # t0 = 10x;
20 add $t0, $t1, $t0    # t0 = 15x;
21
22 add $t1, $zero, $s1   # t1 = y;
23 add $t1, $t1, $t1    # t1 = 2y;
24 add $t1, $t1, $t1    # t1 = 4y;
25 add $t1, $t1, $t1    # t1 = 8y;
26 add $t1, $t1, $t1    # t1 = 16y;
27 add $t1, $t1, $t1    # t1 = 32y;
28 add $t1, $t1, $t1    # t1 = 64y;
29 add $t1, $t1, $s1    # t1 = 65y;
30 add $t1, $t1, $s1    # t1 = 66y;
31 add $t1, $t1, $s1    # t1 = 67y;
32
33 add $t2, $t0, $t1    # t2 = 15x + 67y;
34 add $t2, $t2, $t2    # t2 = 2 * t2
35 add $s2, $t2, $t2    # z = 4 * t2

```

Line: 35 Column: 35 ☒ Show Line Numbers

EditExecute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x20100003	addi \$t6,\$0,3	11: addi \$s0, \$zero, 3 # x = 3;
<input type="checkbox"/>	0x00400004	0x20110004	addi \$t7,\$0,4	12: addi \$s1, \$zero, 4 # y = 4;
<input type="checkbox"/>	0x00400008	0x01104020	add \$t0,\$s0,\$t6	15: add \$t0, \$t0, \$s0 # t0 = x;
<input type="checkbox"/>	0x0040000c	0x01084020	add \$s,\$s,\$s	16: add \$t0, \$t0, \$t0 # t0 = 2x;
<input type="checkbox"/>	0x00400010	0x01084020	add \$s,\$s,\$s	17: add \$t0, \$t0, \$t0 # t0 = 4x;
<input type="checkbox"/>	0x00400014	0x01104020	add \$s,\$s,\$t6	18: add \$t0, \$t0, \$s0 # t0 = 5x;
<input type="checkbox"/>	0x00400018	0x01084820	add \$s,\$s,\$s	19: add \$t1, \$t0, \$t0 # t0 = 10x;
<input type="checkbox"/>	0x0040001c	0x01284020	add \$s,\$s,\$s	20: add \$t0, \$t1, \$t0 # t0 = 15x;
<input type="checkbox"/>	0x00400020	0x00114820	add \$s,\$0,\$t7	22: add \$t1, \$zero, \$s1 # t1 = y;
<input type="checkbox"/>	0x00400024	0x01294820	add \$s,\$s,\$s	23: add \$t1, \$t1, \$t1 # t1 = 2y;

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0	0	0	0	0	0	0	0
0x10010020	0	0	0	0	0	0	0	0
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
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0

Registers

Name	Number	Value
\$zero	0	0
\$at	1	0
\$v0	2	0
\$v1	3	0
\$a0	4	0
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	45
\$t1	9	268
\$t2	10	626
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	3
\$s1	17	4
\$s2	18	1252
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$s8	24	0
\$s9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	0
pc		4194388
hi		0
lo		0

// programa 4 (procure usar sll, srl e sra)

```

{
x = 3;
y = 4;
z = ( 15*x + 67*y)*4
}

```

EditExecute

programa3.asm

programa4.asm

```

1  # Programa 4
2  # x = 3;
3  # y = 4;
4  # z = ( 15 * x + 67 * y ) * 4;
5
6
7  #inicio:
8  .text
9  .globl main
10 main:
11 addi $s0, $zero, 3    # x = 3;
12 addi $s1, $zero, 4    # y = 4;
13
14 # z = ( 15 * x + 67 * y ) * 4;
15 sll $t0, $s0, 4      # t0 = x * 2^4
16 sub $t0, $t0, $s0    # t0 = 15x
17
18 sll $t1, $s1, 6      # t1 = y * 2^6
19 add $t1, $t1, $s1    # t1 = 65y
20 add $t1, $t1, $s1    # t1 = 66y
21 add $t1, $t1, $s1    # t1 = 67y
22
23 add $s2, $t0, $t1    # z = t0 + t1
24 sll $s2, $s2, 2      # z = z * 2^2

```

Line: 24 Column: 35

Show Line Numbers

EditExecute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20100003	addi \$s0, 3	11: addi \$s0, \$zero, 3 # x = 3;
	0x00400004	0x20110004	addi \$s1, 4	12: addi \$s1, \$zero, 4 # y = 4;
	0x00400008	0x00104100	sll \$t0, \$s0, 4	15: sll \$t0, \$s0, 4 # t0 = x * 2^4
	0x0040000c	0x01104022	sub \$t0, \$t0, \$s0	16: sub \$t0, \$t0, \$s0 # t0 = 15x
	0x00400010	0x00114900	sll \$t1, \$s1, 6	18: sll \$t1, \$s1, 6 # t1 = y * 2^6
	0x00400014	0x01314820	add \$t1, \$t1, \$s1	19: add \$t1, \$t1, \$s1 # t1 = 65y
	0x00400018	0x01314820	add \$t1, \$t1, \$s1	20: add \$t1, \$t1, \$s1 # t1 = 66y
	0x0040001c	0x01314820	add \$t1, \$t1, \$s1	21: add \$t1, \$t1, \$s1 # t1 = 67y
	0x00400020	0x01099020	add \$s2, \$t0, \$t1	23: add \$s2, \$t0, \$t1 # z = t0 + t1
	0x00400024	0x00129080	sll \$s2, \$s2, 2	24: sll \$s2, \$s2, 2 # z = z * 2^2

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0	0	0	0	0	0	0	0
0x10010020	0	0	0	0	0	0	0	0
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
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0

0x10010000 (.data)

☒ Hexadecimal Addresses
 ☐ Hexadecimal Values
 ☐ ASCII

Registers

Name	Number	Value
\$zero	0	0
\$at	1	0
\$v0	2	0
\$v1	3	0
\$a0	4	0
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	45
\$t1	9	268
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	3
\$s1	17	4
\$s2	18	1252
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$t8	24	0
\$t9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268469224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	0
pc		4194344
hi		0
lo		0

```
// programa 5
```

```
{
x = 100000;
y = 200000;
z = x + y;
}
```

EditExecute

programa3.asm\*

programa4.asm\*

programa5.asm\*

```

1  # Programa 5
2  # x = 100000;
3  # y = 200000;
4  # z = x + y;
5
6
7  #inicio:
8  .text
9  .globl main
10 main:
11 addi $t0, $zero, 2400 # t0 = 2400;
12 addi $s0, $zero, 100 # x = 100;
13 sll $s0, $s0, 10 # x = 100 * 2^10 = 102400;
14 sub $s0, $s0, $t0 # x = 102400 - 2400 = 100000;
15 add $s1, $s0, $s0 # y = 2x = 200000;
16
17 # z = x + y;
18 add $s2, $s0, $s1 # z = x + y;

```

Line: 18 Column: 36 ☒ Show Line Numbers

EditExecute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x20080960	addi \$t0,\$0,2400	11: addi \$t0, \$zero, 2400 # t0 = 2400;
<input type="checkbox"/>	0x00400004	0x20100064	addi \$s0,\$0,100	12: addi \$s0, \$zero, 100 # x = 100;
<input type="checkbox"/>	0x00400008	0x00108280	sll \$s0,\$s0,10	13: sll \$s0, \$s0, 10 # x = 100 * 2^10 = 102400;
<input type="checkbox"/>	0x0040000c	0x02088022	sub \$s0,\$s0,\$t0	14: sub \$s0, \$s0, \$t0 # x = 102400 - 2400 = 100000;
<input type="checkbox"/>	0x00400010	0x02108820	add \$s1,\$s0,\$s0	15: add \$s1, \$s0, \$s0 # y = 2x = 200000;
<input type="checkbox"/>	0x00400014	0x02119020	add \$s2,\$s0,\$s1	18: add \$s2, \$s0, \$s1 # z = x + y;

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0	0	0	0	0	0	0	0
0x10010020	0	0	0	0	0	0	0	0
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
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0

Registers

Name	Number	Value
\$zero	0	0
\$at	1	0
\$v0	2	0
\$v1	3	0
\$a0	4	0
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	2400
\$t1	9	0
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	100000
\$s1	17	200000
\$s2	18	300000
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$t8	24	0
\$t9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	0
pc		4194328
hi		0
lo		0

Mars Messages

Run I/O

Clear

```

-- program is finished running (dropped off bottom) --
-- program is finished running (dropped off bottom) --

```

/ programa 6

```

{
x = o maior inteiro possível;
y = 300000;
z = x - 4y
}

```

EditExecute

programa3.asm\*

programa4.asm\*

programa5.asm

programa6.asm

```

1  # Programa 6
2  # x = o maior inteiro possivel;
3  # y = 300000;
4  # z = x - ( 4 * y );
5
6
7  #inicio:
8  .text
9  .globl main
10 main:
11 addi $s0, $zero, -1    # x = -1;
12 srl  $s0, $s0, 1      # x = o maior inteiro possivel;
13 addi $t0, $zero, 7200 # t0 = 7200;
14 addi $s1, $zero, 300  # y = 300;
15 sll  $s1, $s1, 10     # y = 300 * 2^10;
16 sub  $s1, $s1, $t0    # y = 307200 - 7200;
17
18 # z = x - ( 4 * y );
19 sll  $t0, $s1, 2      # t0 = y * 2^2;
20 sub  $s2, $s0, $t0    # z = x - ( 4 * y );

```

Line: 20 Column: 45 ☒ Show Line Numbers

EditExecute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x2010ffff	addi \$16,\$0,-1	11: addi \$s0, \$zero, -1 # x = -1;
	0x00400004	0x00108042	srl \$16,\$16,1	12: srl \$s0, \$s0, 1 # x = o maior inteiro possivel;
	0x00400008	0x20081c20	addi \$8,\$0,7200	13: addi \$t0, \$zero, 7200 # t0 = 7200;
	0x0040000c	0x2011012c	addi \$17,\$0,300	14: addi \$s1, \$zero, 300 # y = 300;
	0x00400010	0x00118a80	sll \$17,\$17,10	15: sll \$s1, \$s1, 10 # y = 300 * 2^10;
	0x00400014	0x02289822	sub \$17,\$17,\$8	16: sub \$s1, \$s1, \$t0 # y = 307200 - 7200;
	0x00400018	0x00114080	sll \$8,\$17,2	19: sll \$t0, \$s1, 2 # t0 = y * 2^2;
	0x0040001c	0x02089022	sub \$18,\$16,\$8	20: sub \$s2, \$s0, \$t0 # z = x - ( 4 * y );

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0	0	0	0	0	0	0	0
0x10010020	0	0	0	0	0	0	0	0
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
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0

0x10010000 (.data)

☒ Hexadecimal Addresses ☐ Hexadecimal Values ☐ ASCII

Registers Coproc 1 Coproc 0

Name	Number	Value
\$zero	0	0
\$at	1	0
\$v0	2	0
\$v1	3	0
\$a0	4	0
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	1200000
\$t1	9	0
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	2147483647
\$s1	17	300000
<b>\$s2</b>	<b>18</b>	<b>2146283647</b>
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$t8	24	0
\$t9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	0
pc		4194336
hi		0
lo		0

// programa 7

Considere a seguinte instruao iniciando um programa:

ori \$8, \$0, 0x01

Usando apenas instruoes reg-reg lgicas e/ou instruoes de deslocamento (sll, srl e

sra), continuar o programa de forma que ao final, tenhamos o seguinte conteudo no

registrador \$8:

\$8 = 0xFFFFFFFF



Edit

Execute

programa3.asm\*

programa4.asm\*

programa5.asm

programa6.asm

programa7.asm

programa8.asm

```

1  # Programa 8
2  # Inicialmente escreva um programa que faça:
3  # $8 = 0x12345678.
4  # A partir do registrador $8 acima, usando apenas instruções lógicas (or, ori, and, andi,
5  # xor, xori) e instruções de deslocamento (sll, srl e sra), você deverá obter os seguintes
6  # valores nos respectivos registradores:
7  # $9 = 0x12
8  # $10 = 0x34
9  # $11 = 0x56
10 # $12 = 0x78
11
12
13 #inicio:
14 .text
15 .globl main
16 main:
17 ori $8, $0, 0x1234 # $8 = 0x00001234
18 sll $8, $8, 16 # $8 = 0x12340000
19 ori $8, $8, 0x5678 # $8 = 0x12345678
20
21 srl $9, $8, 24 # $9 = 0x00000012
22
23 srl $10, $8, 16 # $10 = 0x00001234
24 andi $10, $10, 0xff # $10 = 0x00000034
25
26 srl $11, $8, 8 # $11 = 0x00123456
27 andi $11, $11, 0xff # $11 = 0x00000056
28
29 andi $12, $8, 0xff # $12 = 0x00000056

```

Edit

Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	ori \$8,\$0,0x00001234	17: ori \$8, \$0, 0x1234	# \$8 = 0x00001234
	0x00400004	sll \$8,\$8,0x00000010	18: sll \$8, \$8, 16	# \$8 = 0x12340000
	0x00400008	ori \$8,\$8,0x00005678	19: ori \$8, \$8, 0x5678	# \$8 = 0x12345678
	0x0040000c	srl \$9,\$8,0x00000018	21: srl \$9, \$8, 24	# \$9 = 0x00000012
	0x00400010	srl \$10,\$8,0x00000010	23: srl \$10, \$8, 16	# \$10 = 0x00001234
	0x00400014	andi \$10,\$10,0x000000ff	24: andi \$10, \$10, 0xff	# \$10 = 0x00000034
	0x00400018	srl \$11,\$8,0x00000008	26: srl \$11, \$8, 8	# \$11 = 0x00123456
	0x0040001c	andi \$11,\$11,0x000000ff	27: andi \$11, \$11, 0xff	# \$11 = 0x00000056
	0x00400020	andi \$12,\$8,0x000000ff	29: andi \$12, \$8, 0xff	# \$12 = 0x00000056

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Registers

Coproc 1

Coproc 0

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x12345678
\$t1	9	0x00000012
\$t2	10	0x00000034
\$t3	11	0x00000056
\$t4	12	0x00000078
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7fffffc0
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400024
hi		0x00000000
lo		0x00000000

// programa 9

Considere a memória inicial da seguinte forma:

.text

.data

x1: .word 15

x2: .word 25

x3: .word 13

x4: .word 17

soma: .word -1

Escrever um programa que leia todos os números, calcule e substitua o valor da variável soma por este valor.

The screenshot shows a MIPS assembler interface with the following components:

- Assembly Code:**

```

5  # x1: .word 15
6  # x2: .word 25
7  # x3: .word 13
8  # x4: .word 17
9  # soma: .word -1
10 # Escrever um programa que leia todos os números, calcule e substitua o valor da variável soma por este valor.
11
12
13 #inicio:
14 .text
15 .globl main
16 main:
17 addi $t0, $zero, 0x1001 # t0 = 0x00001001;
18 sll $t0, $t0, 16 # t0 = 0x10010000;
19
20 lw $s0, 0($t0) # x1 = 15;
21 lw $s1, 4($t0) # x2 = 25;
22 lw $s2, 8($t0) # x3 = 13;
23 lw $s3, 12($t0) # x4 = 17;
24
25 add $t1, $s0, $s1 # t1 = x1 + x2;
26 add $t1, $t1, $s2 # t1 = x1 + x2 + x3;
27 add $s4, $t1, $s3 # s4 = x1 + x2 + x3 + x4;
28
29 sw $s4, 16($t0) # soma = s4;
30
31 .data
32 x1: .word 15
33 x2: .word 25
34 x3: .word 13
35 x4: .word 17
36 soma: .word -1

```
- Text Segment Table:**

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20081001	addi \$s, \$0, 0x00001001	17: addi \$t0, \$zero, 0x1001 # t0 = 0x00001001;
	0x00400004	0x00084400	sll \$s, \$s, 0x00000010	18: sll \$t0, \$t0, 16 # t0 = 0x10010000;
	0x00400008	0x8d100000	lw \$s0, 0(\$t0)	20: lw \$s0, 0(\$t0) # x1 = 15;
	0x0040000c	0x8d110004	lw \$s1, 4(\$t0)	21: lw \$s1, 4(\$t0) # x2 = 25;
	0x00400010	0x8d120008	lw \$s2, 8(\$t0)	22: lw \$s2, 8(\$t0) # x3 = 13;
	0x00400014	0x8d13000c	lw \$s3, 12(\$t0)	23: lw \$s3, 12(\$t0) # x4 = 17;
	0x00400018	0x02114520	add \$s, \$s, \$s1	25: add \$t1, \$s0, \$s1 # t1 = x1 + x2;
	0x0040001c	0x01324820	add \$s, \$s, \$s2	26: add \$t1, \$t1, \$s2 # t1 = x1 + x2 + x3;
	0x00400020	0x0133a020	add \$s, \$s, \$s3	27: add \$s4, \$t1, \$s3 # s4 = x1 + x2 + x3 + x4;
	0x00400024	0xad140010	sw \$s, 16(\$t0)	29: sw \$s4, 16(\$t0) # soma = s4;
- Data Segment Table:**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x0000000f	0x00000019	0x0000000d	0x00000011	0x0000004e	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
- Registers Table:**

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x10010000
\$t1	9	0x00000035
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000019
\$s2	18	0x00000000
\$s3	19	0x00000011
\$s4	20	0x0000004e
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffcfc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400028
hi		0x00000000
lo		0x00000000

// programa 10

Considere o seguinte programa:  $y = 127x - 65z + 1$

Faça um programa que calcule o valor de y conhecendo os valores de x e z. Os valores de x e z estão armazenados na memória e, na posição imediatamente a seguir, o valor de y deverá ser escrito, ou seja:

.data

x: .word 5

z: .word 7

y: .word 0 # esse valor deverá ser sobrescrito após a execução do programa.

The screenshot displays a MIPS assembler interface with two main panels. The top panel shows the assembly code for a program, and the bottom panel shows the execution results, including registers and memory segments.

**Assembly Code:**

```
4  # estão armazenados na memória e, na posição imediatamente a seguir, o valor de y deverá ser
5  # escrito, ou seja:
6  # .data
7  # x: .word 5
8  # z: .word 7
9  # y: .word 0 # esse valor deverá ser sobrescrito após a execução do programa.
10
11
12 #inicio:
13 .text
14 .globl main
15 main:
16 addi $t0, $zero, 0x1001 # t0 = 0x00001001;
17 sll $t0, $t0, 16 # t0 = 0x10010000;
18
19 lw $s0, 0($t0) # x = 5;
20 lw $s2, 4($t0) # z = 7;
21
22 # y = 127x - 65z + 1;
23 sll $t1, $s0, 7 # t1 = x * 2^7;
24 sub $t1, $t1, $s0 # t1 = 128x - x;
25 sll $t2, $s2, 6 # t2 = z * 2^6;
26 add $t2, $t2, $s2 # t2 = 64z + z;
27 sub $t3, $t2, $t1 # t3 = 127x - 65z;
28 addi $s1, $t3, 1 # y = 127x - 65z + 1;
29
30 sw $s1, 8($t0) # y = 127x - 65z + 1;
31
32 .data
33 x: .word 5
34 z: .word 7
35 y: .word 0
```

**Execution Results:**

**Registers:**

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x10010000
\$t1	9	0x0000002b
\$t2	10	0x000001c7
\$t3	11	0xffffffffc
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000005
\$s1	17	0xffffffffc
\$s2	18	0x00000007
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x0040002c
hi		0x00000000
lo		0x00000000

**Text Segment:**

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20081001	addi \$t0, \$zero, 0x1001	16: addi \$t0, \$zero, 0x1001 # t0 = 0x00001001;
	0x00400004	0x00084400	sll \$t0, \$t0, 16	17: sll \$t0, \$t0, 16 # t0 = 0x10010000;
	0x00400008	0x8d100000	lw \$s0, 0(\$t0)	19: lw \$s0, 0(\$t0) # x = 5;
	0x0040000c	0x8d120004	lw \$s2, 4(\$t0)	20: lw \$s2, 4(\$t0) # z = 7;
	0x00400010	0x001048c0	sll \$t1, \$s0, 7	23: sll \$t1, \$s0, 7 # t1 = x * 2^7;
	0x00400014	0x01304822	sub \$t1, \$t1, \$s0	24: sub \$t1, \$t1, \$s0 # t1 = 128x - x;
	0x00400018	0x00125180	sll \$t2, \$s2, 6	25: sll \$t2, \$s2, 6 # t2 = z * 2^6;
	0x0040001c	0x01525020	add \$t2, \$t2, \$s2	26: add \$t2, \$t2, \$s2 # t2 = 64z + z;
	0x00400020	0x01495822	sub \$t3, \$t2, \$t1	27: sub \$t3, \$t2, \$t1 # t3 = 127x - 65z;
	0x00400024	0x21710001	addi \$s1, \$t3, 1	28: addi \$s1, \$t3, 1 # y = 127x - 65z + 1;
	0x00400028	0xad110008	sw \$s1, 8(\$t0)	30: sw \$s1, 8(\$t0) # y = 127x - 65z + 1;

**Data Segment:**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000005	0x00000007	0xffffffffc	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

// programa 11

Considere o seguinte programa:  $y = x - z + 300000$

Faça um programa que calcule o valor de y conhecendo os valores de x e z. Os valores de x e z estão armazenados na memória e, na posição imediatamente a seguir, o valor de y deverá ser escrito, ou seja:

.data

x: .word 100000



z: .word 200000

y: .word 0 # esse valor deverá ser sobrescrito após a execução do programa.

**programa11.asm**

```
1 # Programa 11
2 # Considere o seguinte programa: y = x - z + 300000
3 # Faça um programa que calcule o valor de y conhecendo os valores de x e z. Os valores de x e z
4 # estão armazenados na memória e, na posição imediatamente a seguir, o valor de y deverá ser escrito, ou seja:
5 # .data
6 # x: .word 100000
7 # z: .word 200000
8 # y: .word 0 # esse valor deverá ser sobrescrito após a execução do programa.
9
10
11 #inicio:
12 .text
13 .globl main
14 main:
15 addi $t0, $zero, 0x1001 # t0 = 0x00001001;
16 sll $t0, $t0, 16 # t0 = 0x10010000;
17
18 lw $s0, 0($t0) # x = 100000;
19 lw $s2, 4($t0) # z = 200000;
20
21 ori $t1, $zero, 37500 # t1 = 37500;
22 sll $t1, $t1, 3 # t1 = 37500 * 2^3 = 300000;
23
24 # y = x - z + 300000;
25 sub $t2, $s0, $s2 # t2 = x - z;
26 add $s1, $t2, $t1 # y = t2 + t1;
27
28 sw $s1, 8($t0) # y = x - z + 300000;
29
30 .data
31 x: .word 100000
32 z: .word 200000
33 y: .word 0
```

**Registers**

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x10010000
\$t1	9	0x000493e0
\$t2	10	0xffff79e0
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x000186a0
\$s1	17	0x00030d40
\$s2	18	0x00030d40
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc0
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400024
hi		0x00000000
lo		0x00000000

**Data Segment**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x000186a0	0x00030d40	0x00030d40	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

// programa 12

Considere a seguinte situação:

int \*\*\*x;

considere que a posição inicial de memória contenha o inteiro em questão.

k = MEM [ MEM [ MEM [ x ] ] ].

Crie um programa que implemente a estrutura de dados acima, leia o valor de K, o multiplique por 2 e o reescreva no local correto conhecendo-se apenas o valor de x.

**programa11.asm**   **programa10.asm**   **programa12.asm\***   **mips2.asm**

```

10 # Crie um programa que implemente a estrutura de dados acima, leia o valor de K, o multiplique por
11 # 2 e o reescreva no local correto conhecendo-se apenas o valor de x.
12
13
14 #inicio:
15 .text
16 .globl main
17 main:
18 addi $t0, $zero, 0x1001 # t0 = 0x00001001;
19 sll $t0, $t0, 16 # t0 = 0x10010000;
20 addi $t1, $t0, 4 # t1 = 0x10010004;
21 addi $t2, $t0, 8 # t2 = 0x10010008;
22 addi $t3, $t0, 12 # t3 = 0x10010012;
23
24 sw $t0, 4($t0) # px = 0x00001001;
25 sw $t1, 8($t0) # ppx = 0x00001001;
26 sw $t2, 12($t0) # pppx = 0x00001001;
27
28 lw $s0, 0($t3) # s0 = t3 = k;
29 lw $s1, 0($t2) # s1 = t2 = k;
30 lw $s2, 0($t1) # s2 = t1 = k;
31 lw $s3, 0($t0) # s4 = t0 = k;
32
33 add $t4, $s3, $s3 # t4 = 2 * k;
34
35 sw $t4, 0($t1) # x = 2 * k;
36
37 .data
38 x: .word 10
39 px: .word 0
40 ppx: .word 0
41 pppx: .word 0

```

**Text Segment**

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20081001	addi \$t0,\$0,0x00001001	18: addi \$t0, \$zero, 0x1001 # t0 = 0x00001001;
	0x00400004	0x00084400	sll \$t0,\$t0,16	19: sll \$t0, \$t0, 16 # t0 = 0x10010000;
	0x00400008	0x21090004	addi \$t1,\$t0,4	20: addi \$t1, \$t0, 4 # t1 = 0x10010004;
	0x0040000c	0x210a0008	addi \$t2,\$t0,8	21: addi \$t2, \$t0, 8 # t2 = 0x10010008;
	0x00400010	0x210b000c	addi \$t3,\$t0,12	22: addi \$t3, \$t0, 12 # t3 = 0x10010012;
	0x00400014	0xad080004	sw \$t0,4(\$t0)	24: sw \$t0, 4(\$t0) # px = 0x00001001;
	0x00400018	0xad090008	sw \$t1,8(\$t0)	25: sw \$t1, 8(\$t0) # ppx = 0x00001001;
	0x0040001c	0xad0a000c	sw \$t2,12(\$t0)	26: sw \$t2, 12(\$t0) # pppx = 0x00001001;
	0x00400020	0xad700000	lw \$s0,0(\$t3)	28: lw \$s0, 0(\$t3) # s0 = t3 = k;
	0x00400024	0xad510000	lw \$s1,0(\$t2)	29: lw \$s1, 0(\$t2) # s1 = t2 = k;
	0x00400028	0xad320000	lw \$s2,0(\$t1)	30: lw \$s2, 0(\$t1) # s2 = t1 = k;

**Data Segment**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x0000000a	0x00000014	0x10010004	0x10010008	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

**Registers**

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x10010000
\$t1	9	0x10010004
\$t2	10	0x10010008
\$t3	11	0x1001000c
\$t4	12	0x00000014
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x10010008
\$s1	17	0x10010004
\$s2	18	0x10010000
\$s3	19	0x0000000a
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc0
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400038
io		0x00000000

// programa 13:

Escreva um programa que leia um valor A da memória, identifique se o número é negativo ou não e encontre o seu módulo. O valor deverá ser reescrito sobre A.

EditExecute

programa11.asm

programa10.asm

programa12.asm

programa13.asm

```

1  # Programa 13:
2  # Escreva um programa que leia um valor A da memória, identifique se o número é negativo ou
3  # não e encontre o seu módulo. O valor deverá ser reescrito sobre A.
4
5
6  #inicio:
7  .text
8  .globl main
9  main:
10 addi $t0, $zero, 0x1001 # t0 = 0x00001001;
11 sll $t0, $t0, 16 # t0 = 0x10010000;
12 lw $s0, 0($t0) # s0 = A;
13
14 slt $t1, $zero, $s0 # if ( A > 0 ) { fim }
15 bne $t1, $zero, fim # if ( A > 0 ) { fim }
16 sub $s0, $zero, $s0 # else { A = |A| }
17 sw $s0, 0($t0) # else { A = |A| }
18 fim:
19
20 .data
21 A: .word -1

```

EditExecute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x20081001	addi \$s, \$0, 0x00001001	10: addi \$t0, \$zero, 0x1001 # t0 = 0x00001001;
<input type="checkbox"/>	0x00400004	0x00084400	sll \$s, \$s, 0x00000010	11: sll \$t0, \$t0, 16 # t0 = 0x10010000;
<input type="checkbox"/>	0x00400008	0x8d100000	lw \$s0, 0x00000000(\$s)	12: lw \$s0, 0(\$t0) # s0 = A;
<input type="checkbox"/>	0x0040000c	0x0010482a	slt \$s0, \$s0, \$s0	14: slt \$t1, \$zero, \$s0 # if ( A > 0 ) { fim }
<input type="checkbox"/>	0x00400010	0x15200002	bne \$s0, \$s0, 0x00000002	15: bne \$t1, \$zero, fim # if ( A > 0 ) { fim }
<input type="checkbox"/>	0x00400014	0x00108022	sub \$s0, \$s0, \$s0	16: sub \$s0, \$zero, \$s0 # else { A =  A  }
<input type="checkbox"/>	0x00400018	0xad100000	sw \$s0, 0x00000000(\$s)	17: sw \$s0, 0(\$t0) # else { A =  A  }

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000001	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x10010000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000001
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc0
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x0040001e
hi		0x00000000
lo		0x00000000

// programa 14:

Escreva um programa que leia um valor A da memória, identifique se o número é par ou não.

Um valor deverá ser escrito na segunda posição livre da memória (0 para par e 1 para ímpar)

EditExecute

programa11.asm

programa10.asm

programa12.asm

programa13.asm

programa14.asm

```

1  # Programa 14:
2  # Escreva um programa que leia um valor A da memória, identifique se o número é par ou não.
3  # Um valor deverá ser escrito na segunda posição livre da memória (0 para par e 1 para ímpar)
4
5
6  #inicio:
7  .text
8  .globl main
9  main:
10 addi $t0, $zero, 0x1001 # t0 = 0x00001001;
11 sll $t0, $t0, 16 # t0 = 0x10010000;
12 lw $s0, 0($t0) # s0 = A;
13
14 andi $t1, $s0, 1 # AND 1
15 beq $t1, $zero, par # if ( A % 2 = 0 ) { par }
16 sw $t1, 4($t0) # else { 0x10010004 = 1 }
17 j fim # fim
18
19 par: # par
20 sw $t1, 4($t0) # 0x10010004 = 0
21
22 fim:
23
24 .data
25 A: .word 15

```

EditExecute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20081001	addi \$s, \$0, 0x00001001	10: addi \$t0, \$zero, 0x1001 # t0 = 0x00001001;
	0x00400004	0x00084400	sll \$s, \$s, 0x00000010	11: sll \$t0, \$t0, 16 # t0 = 0x10010000;
	0x00400008	0x8d100000	lw \$s0, 0(\$s0)	12: lw \$s0, 0(\$t0) # s0 = A;
	0x0040000c	0x32090001	andi \$s, \$s, 0x00000001	14: andi \$t1, \$s0, 1 # AND 1
	0x00400010	0x11200002	beq \$s, \$0, 0x00000002	15: beq \$t1, \$zero, par # if ( A % 2 = 0 ) { par }
	0x00400014	0xad090004	sw \$s, 0x00000004(\$s)	16: sw \$t1, 4(\$t0) # else { 0x10010004 = 1 }
	0x00400018	0x08100008	j 0x00400020	17: j fim # fim
	0x0040001c	0xad090004	sw \$s, 0x00000004(\$s)	20: sw \$t1, 4(\$t0) # 0x10010004 = 0

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x0000000f	0x00000001	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x10010000
\$t1	9	0x00000001
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x0000000f
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400020
hi		0x00000000
lo		0x00000000

// programa 15:

Escrever um programa que crie um vetor de 100 elementos na memória onde  $\text{vetor}[i] = 2*i + 1$ .

Após a última posição do vetor criado, escrever a soma de todos os valores armazenados do vetor.

Use o MARS para verificar a quantidade de instruções conforme o tipo (ULA, Desvios, Mem ou Outras)

EditExecute

programa11.asm

programa10.asm

programa12.asm

programa13.asm

programa14.asm

programa15.asm

```

4
5
6 #inicio:
7 .text
8 .globl main
9 main:
10 addi $t0, $zero, 0x1001 # t0 = 0x00001001;
11 sll $t0, $t0, 16 # t0 = 0x10010000;
12 addi $t1, $zero, 100 # t1 = 100;
13 addi $s0, $zero, 0 # i = 0;
14
15 loop:
16 # 2 * i + 1;
17 add $t2, $s0, $s0 # t2 = 2 * i;
18 addi $t2, $t2, 1 # t2 = 2 * i + 1;
19 sw $t2, 0($t0) # vetor[i] = t2;
20
21 addi $s0, $s0, 1 # i = i + 1;
22 addi $t0, $t0, 4 # t0 = t0 + 4;
23 bne $s0, $t1, loop # for i = 0; i < 100
24
25 addi $t0, $zero, 0x1001 # t0 = 0x00001001;
26 sll $t0, $t0, 16 # t0 = 0x10010000;
27 addi $s0, $zero, 0 # i = 0;
28 soma:
29 lw $t2, 0($t0) # t2 = vetor[i];
30 add $t3, $t3, $t2 # t3 = t3 + t2;
31
32 addi $s0, $s0, 1 # i = i + 1;
33 addi $t0, $t0, 4 # t0 = t0 + 4;
34 bne $s0, $t1, soma # for i = 0; i < 100
35 sw $t3, 0($t0) # vetor[i] = t2;
36

```

Line: 36 Column: 42

Show Line Numbers

EditExecute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20081001	addi \$s0, 0x00000001	11: addi \$t0, \$zero, 0x1001 # t0 = 0x00001001;
	0x00400004	0x00084400	sll \$s0, 0x00000010	12: sll \$t0, \$t0, 16 # t0 = 0x10010000;
	0x00400008	0x20090064	addi \$s0, 0x00000064	13: addi \$t1, \$zero, 100 # t1 = 100;
	0x0040000c	0x20100000	addi \$s0, 0x00000000	14: addi \$s0, \$zero, 0 # i = 0;
	0x00400010	0x22105020	addi \$s0, \$s0, \$s0	17: add \$t2, \$s0, \$s0 # t2 = 2 * i; ...
	0x00400014	0x22105020	addi \$s0, \$s0, \$s0	18: addi \$t2, \$s0, \$s0 # t2 = 2 * i; ...
	0x00400018	0x22105020	addi \$s0, \$s0, \$s0	19: addi \$t2, \$t2, 1 # t2 = 2 * i + 1;
	0x0040001c	0x22105020	addi \$s0, \$s0, \$s0	20: sw \$t2, 0(\$t0) # vetor[i] = t2;
	0x00400020	0x22105020	addi \$s0, \$s0, \$s0	21: addi \$s0, \$s0, 1 # i = i + 1;
	0x00400024	0x22105020	addi \$s0, \$s0, \$s0	22: addi \$t0, \$t0, 4 # t0 = t0 + 4;
	0x00400028	0x22105020	addi \$s0, \$s0, \$s0	23: bne \$s0, \$t1, loop # for i = 0; i < 100
	0x0040002c	0x22105020	addi \$s0, \$s0, \$s0	24: bne \$s0, \$t1, loop # for i = 0; i < 100
	0x00400030	0x22105020	addi \$s0, \$s0, \$s0	25: addi \$t0, \$zero, 0x1001 # t0 = 0x00001001;
	0x00400034	0x22105020	addi \$s0, \$s0, \$s0	26: sll \$t0, \$t0, 16 # t0 = 0x10010000;
	0x00400038	0x22105020	addi \$s0, \$s0, \$s0	27: addi \$s0, \$zero, 0 # i = 0;
	0x0040003c	0x22105020	addi \$s0, \$s0, \$s0	28: addi \$s0, \$zero, 0 # i = 0;
	0x00400040	0x22105020	addi \$s0, \$s0, \$s0	29: lw \$t2, 0(\$t0) # t2 = vetor[i];
	0x00400044	0x22105020	addi \$s0, \$s0, \$s0	30: add \$t3, \$t3, \$t2 # t3 = t3 + t2;
	0x00400048	0x22105020	addi \$s0, \$s0, \$s0	31: add \$t3, \$t3, \$t2 # t3 = t3 + t2;
	0x0040004c	0x22105020	addi \$s0, \$s0, \$s0	32: addi \$s0, \$s0, 1 # i = i + 1;
	0x00400050	0x22105020	addi \$s0, \$s0, \$s0	33: addi \$t0, \$t0, 4 # t0 = t0 + 4;
	0x00400054	0x22105020	addi \$s0, \$s0, \$s0	34: bne \$s0, \$t1, soma # for i = 0; i < 100
	0x00400058	0x22105020	addi \$s0, \$s0, \$s0	35: sw \$t3, 0(\$t0) # vetor[i] = t2;
	0x0040005c	0x22105020	addi \$s0, \$s0, \$s0	36: sw \$t3, 0(\$t0) # vetor[i] = t2;

Labels

(global)

main

loop

soma

Registers

Coproc 1

Coproc 0

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x10010190
\$t1	9	0x00000064
\$t2	10	0x000000c7
\$t3	11	0x00002710
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000064
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc0
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x0040004c
hi		0x00000000
lo		0x00000000

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000001	0x00000003	0x00000005	0x00000007	0x00000009	0x0000000b	0x0000000d	0x0000000f
0x10010020	0x00000011	0x00000013	0x00000015	0x00000017	0x00000019	0x0000001b	0x0000001d	0x0000001f
0x10010040	0x00000021	0x00000023	0x00000025	0x00000027	0x00000029	0x0000002b	0x0000002d	0x0000002f
0x10010060	0x00000031	0x00000033	0x00000035	0x00000037	0x00000039	0x0000003b	0x0000003d	0x0000003f
0x10010080	0x00000041	0x00000043	0x00000045	0x00000047	0x00000049	0x0000004b	0x0000004d	0x0000004f
0x100100a0	0x00000051	0x00000053	0x00000055	0x00000057	0x00000059	0x0000005b	0x0000005d	0x0000005f
0x100100c0	0x00000061	0x00000063	0x00000065	0x00000067	0x00000069	0x0000006b	0x0000006d	0x0000006f
0x100100e0	0x00000071	0x00000073	0x00000075	0x00000077	0x00000079	0x0000007b	0x0000007d	0x0000007f
0x10010100	0x00000081	0x00000083	0x00000085	0x00000087	0x00000089	0x0000008b	0x0000008d	0x0000008f
0x10010120	0x00000091	0x00000093	0x00000095	0x00000097	0x00000099	0x0000009b	0x0000009d	0x0000009f
0x10010140	0x000000a1	0x000000a3	0x000000a5	0x000000a7	0x000000a9	0x000000ab	0x000000ad	0x000000af
0x10010160	0x000000b1	0x000000b3	0x000000b5	0x000000b7	0x000000b9	0x000000bb	0x000000bd	0x000000bf
0x10010180	0x000000c1	0x000000c3	0x000000c5	0x000000c7	0x000000c9	0x000000cb	0x000000cd	0x000000cf

// programa 16

Escreva um programa que avalie a expressão: (x\*y)/z.

Use x = 1600000 (=0x186A00), y = 80000 (=0x13880), e z = 400000 (=0x61A80).

Inicializar os

registradores com os valores acima.

EditExecute

programa11.asm

programa10.asm

programa12.asm

programa13.asm

programa14.asm

programa15.asm

programa16.asm\*

```

1  # Programa 16
2  # Escreva um programa que avalie a expressão: (x*y)/z.
3  # Use x = 1600000 (=0x186A00), y = 80000 (=0x13880), e z = 400000 (=0x61A80). Inicializar os
4  # registradores com os valores acima.
5
6
7  #inicio:
8  .text
9  .globl main
10 main:
11 addi $s0, $zero, 0x186A # x = 0x0000186A;
12 sll $s0, $s0, 8 # x = 0x00186A00;
13 addi $s1, $zero, 0x1388 # y = 0x00001388;
14 sll $s1, $s1, 4 # y = 0x00013880;
15 addi $s2, $zero, 0x61A8 # z = 0x000061A8;
16 sll $s2, $s2, 4 # z = 0x00061A80;
17
18 # ( x * y ) / z;
19 mul $t0, $s0, $s1 # t0 = x * y
20
21 mfhi $t0 # t0 = hi
22 sll $t0, $t0, 24 # t0 = hi
23 mflo $t1 # ti = lo
24 srl $t1, $t1, 8 # ti = lo
25 add $t0, $t0, $t1 # t0 = hi + lo
26
27 srl $t2, $s2, 4 # t2 = 0x000061A8;
28
29 div $t0, $t2 # t0 = t0 / z
30
31 mflo $s3 # s3 = lo
32 sll $s3, $s3, 4 # s3 = resultado

```

Line: 30 Column: 1 ☒ Show Line Numbers

EditExecute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x2010186a	addi \$16,\$0,0x0000186a	11: addi \$s0, \$zero, 0x186A # x = 0x0000186A;
<input type="checkbox"/>	0x00400004	0x00108200	sll \$16,\$16,0x00000008	12: sll \$s0, \$s0, 8 # x = 0x00186A00;
<input type="checkbox"/>	0x00400008	0x20111388	addi \$17,\$0,0x00001388	13: addi \$s1, \$zero, 0x1388 # y = 0x00001388;
<input type="checkbox"/>	0x0040000c	0x00118900	sll \$17,\$17,0x00000004	14: sll \$s1, \$s1, 4 # y = 0x00013880;
<input type="checkbox"/>	0x00400010	0x201261a8	addi \$18,\$0,0x000061a8	15: addi \$s2, \$zero, 0x61A8 # z = 0x000061A8;
<input type="checkbox"/>	0x00400014	0x00129100	sll \$18,\$18,0x00000004	16: sll \$s2, \$s2, 4 # z = 0x00061A80;
<input type="checkbox"/>	0x00400018	0x72114002	mul \$8,\$16,\$17	19: mul \$t0, \$s0, \$s1 # t0 = x * y
<input type="checkbox"/>	0x0040001c	0x00004010	mfhi \$8	21: mfhi \$t0 # t0 = hi
<input type="checkbox"/>	0x00400020	0x00084600	sll \$8,\$8,0x00000018	22: sll \$t0, \$t0, 24 # t0 = hi
<input type="checkbox"/>	0x00400024	0x00004912	mflo \$9	23: mflo \$t1 # ti = lo
<input type="checkbox"/>	0x00400028	0x00094a02	srl \$9,\$9,0x00000008	24: srl \$t1, \$t1, 8 # ti = lo
<input type="checkbox"/>	0x0040002c	0x01004030	add \$8,\$8,\$9	25: add \$t0,\$t0,\$t1 # t0 = hi + lo

☒ Data ☒ Text

Labels

Label	Address
(global)	
main	0x00400000

☒ Data ☒ Text

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x1dcde500
\$t1	9	0x00cd6500
\$t2	10	0x000061a8
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$t8	16	0x00186a00
\$s1	17	0x00013880
\$s2	18	0x00061a80
\$s3	19	0x0004e200
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffcfc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400040
hi		0x00000000
lo		0x00004e20

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (data)

☒ Hexadecimal Addresses ☒ Hexadecimal Values ☐ ASCII

// programa 17

Para a expressão a seguir, escreva um programa que calcule o valor de k:

$k = x * y$  (Você deverá realizar a multiplicação através de somas!)

O valor de x deve ser lido da primeira posição livre da memória e o valor de y deverá lido da segunda posição livre. O valor de k, após calculado, deverá ainda ser escrito na terceira posição livre da memória.



EditExecute

programa11.asm

programa10.asm

programa12.asm

programa13.asm

programa14.asm

programa15.asm

programa16.asm

programa17.asm

```

1 # Programa 17
2 # Para a expressão a seguir, escreva um programa que calcule o valor de k:
3 # k = x * y (Você deverá realizar a multiplicação através de somas!)
4 # O valor de x deve ser lido da primeira posição livre da memória e o valor de y deverá lido da
5 # segunda posição livre. O valor de k, após calculado, deverá ainda ser escrito na terceira posição
6 # livre da memória.
7
8
9 #inicio:
10 .text
11 .globl main
12 main:
13 addi $t0, $zero, 0x1001 # t0 = 0x00001001;
14 sll $t0, $t0, 16 # t0 = 0x10010000;
15 lw $s0, 0($t0) # x = 10;
16 lw $s1, 4($t0) # x = 5;
17 addi $t1, $zero, 0 # i = 0;
18
19 loop:
20 add $s2, $s2, $s0 # k = k + x;
21 addi $t1, $t1, 1 # i++;
22 bne $t1, $s1, loop # for i = 0; i < y;
23
24 sw $s2, 8($t0) # k = x * y
25
26 .data
27 x: .word 10
28 y: .word 5

```

EditExecute

Text Segment

Labels

Data Segment

Registers

Coproc 1

Coproc 0

// programa 18

Para a expressão a seguir, escreva um programa que calcule o valor de k:

$$k = x^y$$

Obs: Você poderá utilizar o exercício anterior.

O valor de x deve ser lido da primeira posição livre da memória e o valor de y deverá lido da segunda posição livre. O valor de k, após calculado, deverá ainda ser escrito na terceira posição livre da memória.

Dê um valor para x e y (dê valores pequenos !!) e use o MARS para verificar a quantidade de instruções conforme o tipo (ULA, Desvios, Mem ou Outras)





1. Se tivermos 2 inteiros, cada um com 32 bits, quantos bits podemos esperar para o produto?

- A. 16
- B. 32
- C. 64
- D. 128

R: C

2. Quais os registradores que armazenam os resultados na multiplicação?

- A. high e low
- B. hi e lo
- C. R0 e R1
- D. \$0 e \$1

R: B

3. Qual a operação usada para multiplicar inteiros em comp. de dois?

- A. mult
- B. multu
- C. multi
- D. Mutt

R: A

4. Qual instrução move os bits menos significativos da multiplicação para o reg. 8?

- A. move \$8,lo
- B. mvlo \$8,lo
- C. mflo \$8
- D. addu \$8,\$0,lo

R: C

5. Se tivermos dois inteiros, cada um com 32 bits, quantos bits deveremos estar preparados para receber no quociente?

- A. 16
- B. 32
- C. 64
- D. 128

R: B

6. Após a instrução div, qual registrador possui o quociente?

- A. lo
- B. hi
- C. high
- D. \$2

R: A

7. Qual a inst. Usada para dividir dois inteiros em comp. de dois?

- A. dv
- B. divide
- C. divu
- D. Div

R: D

8. Faça um arithmetic shift right de dois no seguinte padrão de bits: 1001 1011

- A. 1110 0110
- B. 0010 0110
- C. 1100 1101
- D. 0011 0111

R: A

9. Qual o efeito de um arithmetic shift right de uma posição?

- A. Se o inteiro for unsigned, o shift o divide por 2. Se o inteiro for signed, o shift o divide por 2.
- B. Se o inteiro for unsigned, o shift o divide por 2. Se o inteiro for signed, o shift pode resultar em um valor errado.
- C. Se o inteiro for unsigned, o shift pode ocasionar um valor errado. Se o inteiro for signed, o shift o divide por 2.
- D. O shift multiplica o número por dois.

R: A

10. Qual sequencia de instruções avalia  $3x+7$ , onde x é iniciado no reg. \$8 e o resultado armazenado em \$9?

- A. ori \$3,\$0,3  
mult \$8,\$3  
mflo \$9  
addi \$9,\$9,7
- B. ori \$3,\$0,3  
mult \$8,\$3  
addi \$9,\$8,7
- C. ori \$3,\$0,3  
mult \$8,\$3  
mfhi \$9  
addi \$9,\$9,7
- D. mult \$8,3  
mflo \$9  
addi \$9,\$9,7

R: A

// programa 19

Escrever um programa que leia dois números da memória, a primeira e segunda posições respectivamente (os coloque em \$s0 e \$s1) e determine a quantidade de bits significantes de cada um. Coloque as respostas em \$t0 e \$t1, a partir desse resultado faça a multiplicação. Caso o número de bits significantes de ambos seja menor do que 32 a resposta deverá estar apenas em \$s2, caso contrário a resposta estará em \$s2 e \$s3 (LO e HI respectivamente).

```

9  #inicio:
10 .text
11 .globl main
12 main:
13 addi $t0, $zero, 0x1001 # t0 = 0x00001001;
14 sll $t0, $t0, 16 # t0 = 0x10010000;
15 lw $s0, 0($t0) # s0 = x;
16 lw $s1, 4($t0) # s1 = y;
17 lw $t4, 0($t0) # t4 = x;
18 lw $t5, 4($t0) # t5 = y;
19 addi $t0, $zero, 0 # i = 0;
20
21 significativox:
22 beq $t4, $zero, endx # while x != 0;
23 srl $t4, $t4, 1 # conta os bits significativos;
24 addi $t0, $t0, 1 # i++;
25 j significativox
26
27 endx:
28 addi $t1, $zero, 0 # j = 0;
29 significativoy:
30 beq $t5, $zero, endy # while y != 0;
31 srl $t5, $t5, 1 # conta os bits significativos;
32 addi $t1, $t1, 1 # j++;
33 j significativoy
34
35 endy:
36 mult $s0, $s1 # s0 * s1;
37
38 addi $t2, $zero, 32 # t2 = 32
39 slt $t3, $t2, $t0
40 beq $t2, $zero, menor # Se $t3 == 0, x <= 32
41 slt $t3, $t2, $t1
42 beq $t2, $zero, menor # Se $t3 == 0, x <= 32
43 mflo $s2 # s2 = lo
44 mfhi $s3 # s3 = hi
45 menor:
46 mflo $s2 # s2 = lo
47
48 .data
49 x: .word 100000
50 y: .word 100000

```

The screenshot shows the Mars MIPS simulator interface. The main window is divided into several panes:

- Text Segment:** Displays the assembly code with addresses, instructions, and comments. The code is the same as the one provided in the previous block.
- Data Segment:** Shows memory addresses and their corresponding values. The values are mostly 0, except for the addresses 0x100000 (100000) and 0x100004 (100000).
- Registers:** A table showing the current values of the MIPS registers. The registers are organized into columns: Name, Number, and Value. The values are mostly 0, except for \$s0 (100000), \$s1 (100000), and \$s2 (1410065408).

// programa 20

$y = x^4 + x^3 - 2x^2$  se x for par

$x^5 - x^3 + 1$  se x for impar

Os valores de x devem ser lidos da primeira posição livre da memória e o valor de y deverá ser escrito na segunda posição livre.

```

8  #inicio:
9  .text
10 .globl main
11 main:
12 addi $t0, $zero, 0x1001 # t0 = 0x00001001;
13 sll $t0, $t0, 16 # t0 = 0x10010000;
14 lw $s0, 0($t0) # x = 2;
15 addi $t3, $zero, 1 # t3 = 1;
16 addi $t4, $zero, 1 # t4 = 1;
17
18 andi $t2, $s0, 1 # AND 1 ;
19 beq $t2, $zero, par # if ( A % 2 = 0 ) { par }
20 impar:
21 addi $t1, $zero, 0 # i = 0;
22 addi $t5, $zero, 5 # t5 = 5;
23 #  $x^5 - x^3 + 1$ ;
24 loop_i1:
25 mul $t3, $t3, $s0 # t3 =  $x^5$ ;
26 addi $t1, $t1, 1 # i++;
27 bne $t1, $t5, loop_i1 # for 1 = 0; i < t5
28
29 addi $t5, $zero, 3 # t5 = 3;
30 addi $t1, $zero, 0 # i = 0;
31 loop_i2:
32 mul $t4, $t4, $s0 # t4 =  $x^3$ ;
33 addi $t1, $t1, 1 # i++;
34 bne $t1, $t5, loop_i2 # for 1 = 0; i < t5
35
36 sub $s1, $t3, $t4 # y =  $x^5 - x^3$ ;
37 addi $s1, $s1, 1 # y =  $x^5 - x^3 + 1$ ;
38 j fim # fim
39
40 par:
41 addi $t1, $zero, 0 # i = 0;
42 addi $t5, $zero, 4 # t5 = 4;
43
44 #  $x^4 + x^3 - 2x^2$ ;
45 loop_p1:
46 mul $t3, $t3, $s0 # t3 =  $x^4$ ;
47 addi $t1, $t1, 1 # i++;
48 bne $t1, $t5, loop_p1 # for 1 = 0; i < t5
49
50 addi $t5, $zero, 3 # t5 = 3;
51 addi $t1, $zero, 0 # i = 0;
52 loop_p2:
53 mul $t4, $t4, $s0 # t4 =  $x^3$ ;
54 addi $t1, $t1, 1 # i++;
55 bne $t1, $t5, loop_p2 # for 1 = 0; i < t5
56
57 add $s1, $t3, $t4 # y =  $x^4 + x^3$ ;
58 mul $t3, $s0, $s0 #  $x^2$ ;
59 add $t3, $t3, $t3 #  $2x^2$ ;
60 sub $s1, $s1, $t3 # y =  $x^4 + x^3 - 2x^2$ ;
61 j fim # fim
62
63 fim:
64 sw $s1, 4($t0) # memoria
65
66 .data
67 x: .word 2

```

The screenshot shows a debugger interface with two main panels. The left panel displays assembly code for a MIPS program. The right panel shows the state of the processor registers.

**Assembly Code (Text Segment):**

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20081001	addi \$t0, \$zero, 0x1001	# t0 = 0x00001001;
	0x00400004	0x00084400	sll \$t0, \$t0, 16	# t0 = 0x10010000;
	0x00400008	0x8d100000	lw \$s0, 0(\$t0)	# x = 2;
	0x0040000c	0x200b0001	addi \$t1, \$zero, 0	# i = 0;
	0x00400010	0x200c0001	addi \$t2, \$zero, 1	# t2 = 1;
	0x00400014	0x320a0001	andi \$t2, \$s0, 1	# AND 1;
	0x00400018	0x1140000d	beq \$t0, \$zero, par	# if (A % 2 = 0) { par }
	0x0040001c	0x20090000	addi \$s, \$s0, 0x00000000	# i = 0;
	0x00400020	0x200d0005	addi \$t3, \$zero, 5	# t3 = 5;
	0x00400024	0x71705802	mul \$t3, \$t3, \$s0	# t3 = x^5;

**Registers (Registers window):**

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x10010000
\$t1	9	0x00000003
\$t2	10	0x00000000
\$t3	11	0x00000008
\$t4	12	0x00000000
\$t5	13	0x00000003
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000002
\$s1	17	0x00000010
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffcfc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400090
hi		0x00000000
lo		0x00000004

// programa 21

$y = x^3 + 1$       se  $x > 0$   
 $x^4 - 1$       se  $x \leq 0$

Os valores de x devem ser lidos da primeira posição livre da memória e o valor de y deverá ser escrito na segunda posição livre.

```

1 #
2 #inicio:
3 .text
4 .globl main
5 main:
6 addi $t0, $zero, 0x1001 # t0 = 0x00001001;
7 sll $t0, $t0, 16 # t0 = 0x10010000;
8 lw $s0, 0($t0) # x = 2;
9 addi $t1, $zero, 0 # i = 0;
10 addi $t2, $zero, 1 # t2 = 1;
11 addi $s2, $zero, 1 # s2 = 1;
12
13 slt $t4, $zero, $s0 # if ( x > 0 ) { maior }
14 bne $t4, $zero, maior # if ( x > 0 ) { maior }
15 maior:
16 addi $t3, $zero, 4 # t3 = 4;
17 loop1:
18 # x^4 - 1;
19 mul $t2, $t2, $s0 # t2 = x^4;
20 addi $t1, $t1, 1 # i++;
21 bne $t1, $t3, loop1 # for 1 = 0; i < t3
22
23 sub $s1, $t2, $s2 # y = x^4 - 1;
24 j fim
25
26 fim:
27 addi $t3, $zero, 3 # t3 = 3;
28 loop2:
29 # x^3 + 1;
30 mul $t2, $t2, $s0 # t2 = x^3;
31 addi $t1, $t1, 1 # i++;
32 bne $t1, $t3, loop2 # for 1 = 0; i < t3
33
34 addi $s1, $t2, 1 # y = x^3 + 1;
35 j fim
36
37 fim:
38 sw $s1, 4($t0) # memoria
39
40 .data
41 x: .word -2

```

Execute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x20081001	addi \$t0,\$zero,0x00001001	l2: addi \$t0, \$zero, 0x1001 # t0 = 0x00001001;
<input type="checkbox"/>	0x00400004	0x00084400	sll \$t0,\$t0,0x00000010	l3: sll \$t0, \$t0, 16 # t0 = 0x10010000;
<input type="checkbox"/>	0x00400008	0x8d100000	lw \$f6,0x00000000(\$t0)	l4: lw \$s0, 0(\$t0) # x = 2;
<input type="checkbox"/>	0x0040000c	0x20090000	addi \$t0,\$zero,0x00000000	l5: addi \$t1, \$zero, 0 # i = 0;
<input type="checkbox"/>	0x00400010	0x200a0001	addi \$t0,\$zero,0x00000001	l6: addi \$t2, \$zero, 1 # t2 = 1;
<input type="checkbox"/>	0x00400014	0x20120001	addi \$t0,\$zero,0x00000001	l7: addi \$s2, \$zero, 1 # s2 = 1;
<input type="checkbox"/>	0x00100018	0x00106020	slt \$t0,\$t0,\$f6	l8: slt \$t2, \$zero, \$s0 # if ( x > 0 ) { maior }
<input type="checkbox"/>	0x0040001c	0x1580000e	bne \$t2,\$zero,major	l9: bne \$t4, \$zero, maior # if ( x > 0 ) { maior }
<input type="checkbox"/>	0x00400020	0x200b0004	addi \$t1,\$zero,0x00000004	l20: addi \$t3, \$zero, 4 # t3 = 4;
<input type="checkbox"/>	0x00400024	0x71505002	muli \$t0,\$t0,\$f6	l21: mul \$t2, \$t2, \$s0 # t2 = x*4;
<input type="checkbox"/>	0x00400028	0x12390001	addi \$t0,\$zero,0x00000001	l22: addi \$t1, \$t1, 1 # i++;
<input type="checkbox"/>	0x0040002c	0x12390001	bne \$t1,\$t2,loop1	l23: bne \$t1, \$t2, loop1 # \$s0-1 = 0, i-1 = 2

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0xffffffff	0x0000000f	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data)

Hexadecimal Addresses

Hexadecimal Values

ASCII

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x10010000
\$t1	9	0x00000004
\$t2	10	0x00000000
\$t3	11	0x00000004
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0xffffffff
\$s1	17	0x0000000f
\$s2	18	0x00000001
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$s8	28	0x10010000
\$s9	29	0x7fffffff
\$sp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400054
hi		0x00000000
lo		0x00000010

Todos os programas em:

<https://github.com/N4lberth/Ac2/tree/main/EP06>