Exercício Prático 06 - AC-II

Parte I

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.: Letra 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.: Letra B

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

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

R.: Letra A

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

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

R.: Letra 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.: Letra D

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

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

R.: Letra 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.: Letra 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.: Letra 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.: Letra 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.: Letra 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.: Letra B

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

A. Pelo menos um operando deve ser de 32 bits.
B. Cada operando pode ser de qualquer tamanho.
C. Ambos os operandos devem vir de registros.
D. Cada um dos registradores deve possuir 32 bit.

R.: Letra D

13. Dezesseis bits de dados de uma instrução de ori são usados como um operando imediato. Durante a 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.: Letra 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.: Letra C

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

A. Não.
B. Sim.

R.: Letra 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.: Letra 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.: Letra A

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

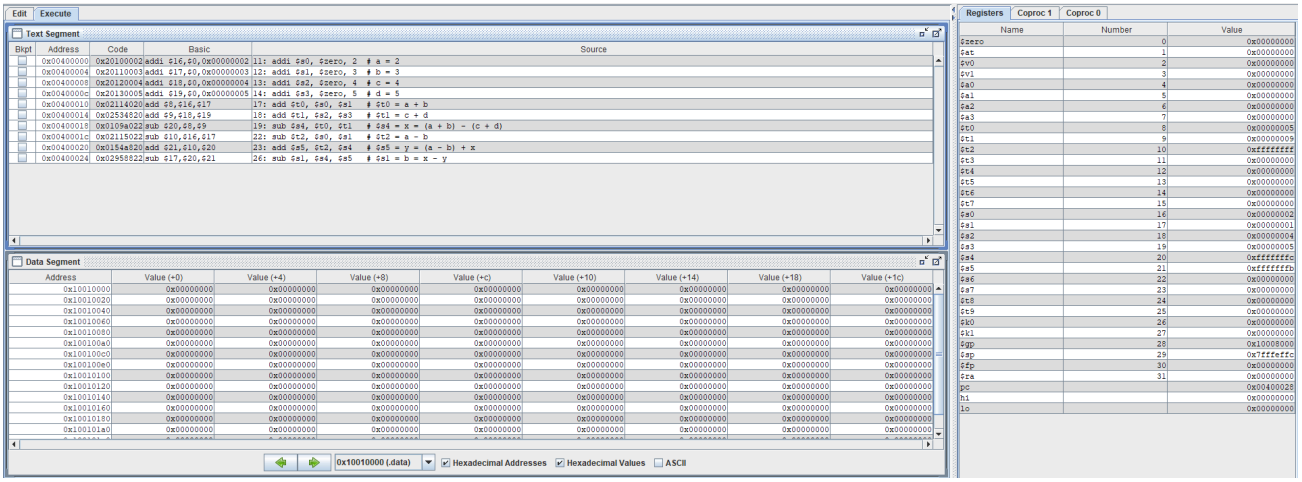
A. Não. Diferentes de instruções de máquina possuem campos diferentes.
B. Não. Cada instrução de máquina é completamente diferente de qualquer outra.
C. Sim. Todas as instruções de máquina têm os mesmos campos na mesma ordem.
D. Sim. Todas as instruções de máquina têm os mesmos campos, mas eles podem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

R.: Letra A

Parte II

```
//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;
}
```

```assembly
# Exercício 1

#inicio

.text
.globl main

main:

# Atribuições
addi $s0, $zero, 2  # a = 2
addi $s1, $zero, 3  # b = 3
addi $s2, $zero, 4  # c = 4
addi $s3, $zero, 5  # d = 5

# x = (a + b) - (c + d)
add $t0, $s0, $s1   # $t0 = a + b
add $t1, $s2, $s3   # $t1 = c + d
sub $s4, $t0, $t1   # $s4 = x = (a + b) - (c + d)

# y = a - b + x
sub $t2, $s0, $s1   # $t2 = a - b
add $s5, $t2, $s4   # $s5 = y = (a - b) + x

# b = x - y
sub $s1, $s4, $s5   # $s1 = b = x - y

#fim
```



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

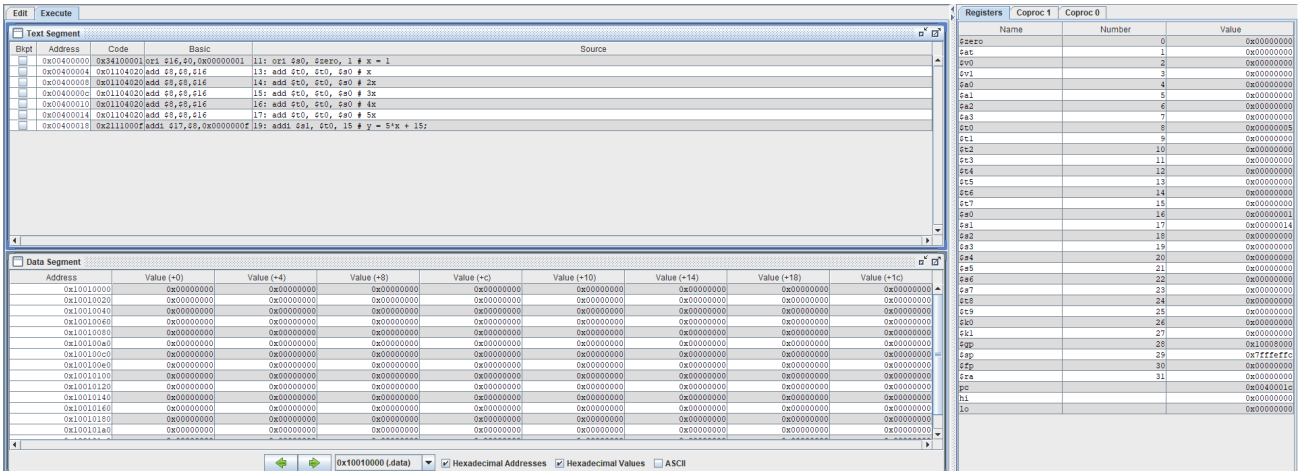
{

x = 1;

y = 5*x + 15;

}

```assembly
# Exercício 2

#inicio

.text
.globl main

main:

# Atribuições
ori $s0, $zero, 1 # x = 1

add $t0, $t0, $s0 # x
add $t0, $t0, $s0 # 2x
add $t0, $t0, $s0 # 3x
add $t0, $t0, $s0 # 4x
add $t0, $t0, $s0 # 5x

addi $s1, $t0, 15 # y = 5*x + 15;


#fim
```



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

{

x = 3;

y = 4 ;

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

}

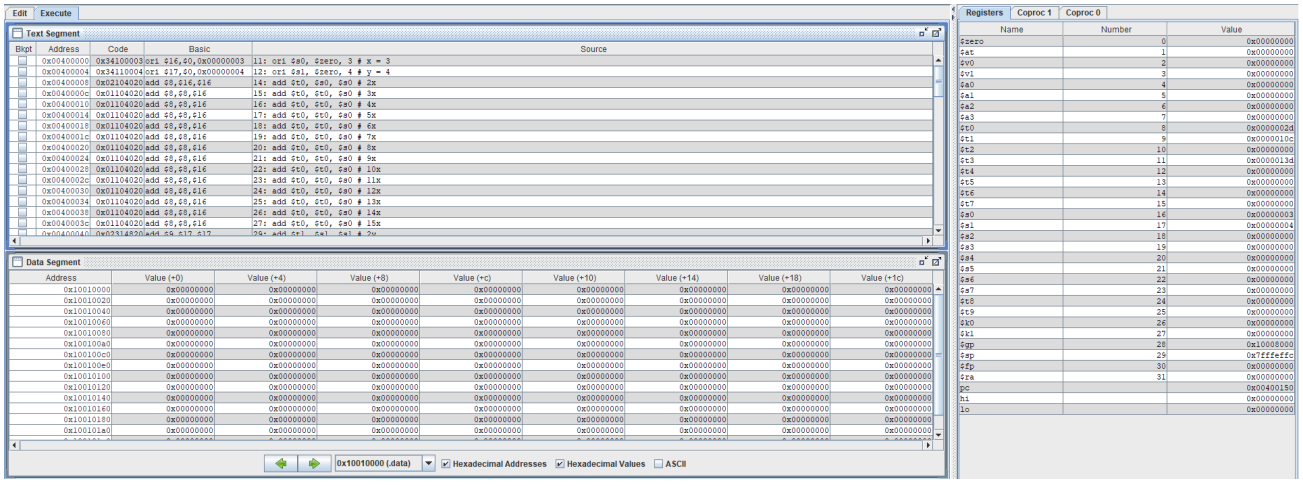
ex03.asm

```
# Exercício 3

#inicio

.text
.globl main

main:

# Atribuições
ori $s0, $zero, 3 # x = 3
ori $s1, $zero, 4 # y = 4

add $t0, $s0, $s0 # 2x
add $t0, $t0, $s0 # 3x
add $t0, $t0, $s0 # 4x
add $t0, $t0, $s0 # 5x
add $t0, $t0, $s0 # 6x
add $t0, $t0, $s0 # 7x
add $t0, $t0, $s0 # 8x
add $t0, $t0, $s0 # 9x
add $t0, $t0, $s0 # 10x
add $t0, $t0, $s0 # 11x
add $t0, $t0, $s0 # 12x
add $t0, $t0, $s0 # 13x
add $t0, $t0, $s0 # 14x
add $t0, $t0, $s0 # 15x

add $t1, $s1, $s1 # 2y
add $t1, $t1, $s1 # 3y
add $t1, $t1, $s1 # 4y
add $t1, $t1, $s1 # 5y
add $t1, $t1, $s1 # 6y
add $t1, $t1, $s1 # 7y
add $t1, $t1, $s1 # 8y
```

ex03.asm

```
35  add $t1, $t1, $s1 # 8y
36  add $t1, $t1, $s1 # 9y
37  add $t1, $t1, $s1 # 10y
38  add $t1, $t1, $s1 # 11y
39  add $t1, $t1, $s1 # 12y
40  add $t1, $t1, $s1 # 13y
41  add $t1, $t1, $s1 # 14y
42  add $t1, $t1, $s1 # 15y
43  add $t1, $t1, $s1 # 16y
44  add $t1, $t1, $s1 # 17y
45  add $t1, $t1, $s1 # 18y
46  add $t1, $t1, $s1 # 19y
47  add $t1, $t1, $s1 # 20y
48  add $t1, $t1, $s1 # 21y
49  add $t1, $t1, $s1 # 22y
50  add $t1, $t1, $s1 # 23y
51  add $t1, $t1, $s1 # 24y
52  add $t1, $t1, $s1 # 25y
53  add $t1, $t1, $s1 # 26y
54  add $t1, $t1, $s1 # 27y
55  add $t1, $t1, $s1 # 28y
56  add $t1, $t1, $s1 # 29y
57  add $t1, $t1, $s1 # 30y
58  add $t1, $t1, $s1 # 31y
59  add $t1, $t1, $s1 # 32y
60  add $t1, $t1, $s1 # 33y
61  add $t1, $t1, $s1 # 34y
62  add $t1, $t1, $s1 # 35y
63  add $t1, $t1, $s1 # 36y
64  add $t1, $t1, $s1 # 37y
65  add $t1, $t1, $s1 # 38y
66  add $t1, $t1, $s1 # 39y
67  add $t1, $t1, $s1 # 40y
68  add $t1, $t1, $s1 # 41y
69  add $t1, $t1, $s1 # 42y
```

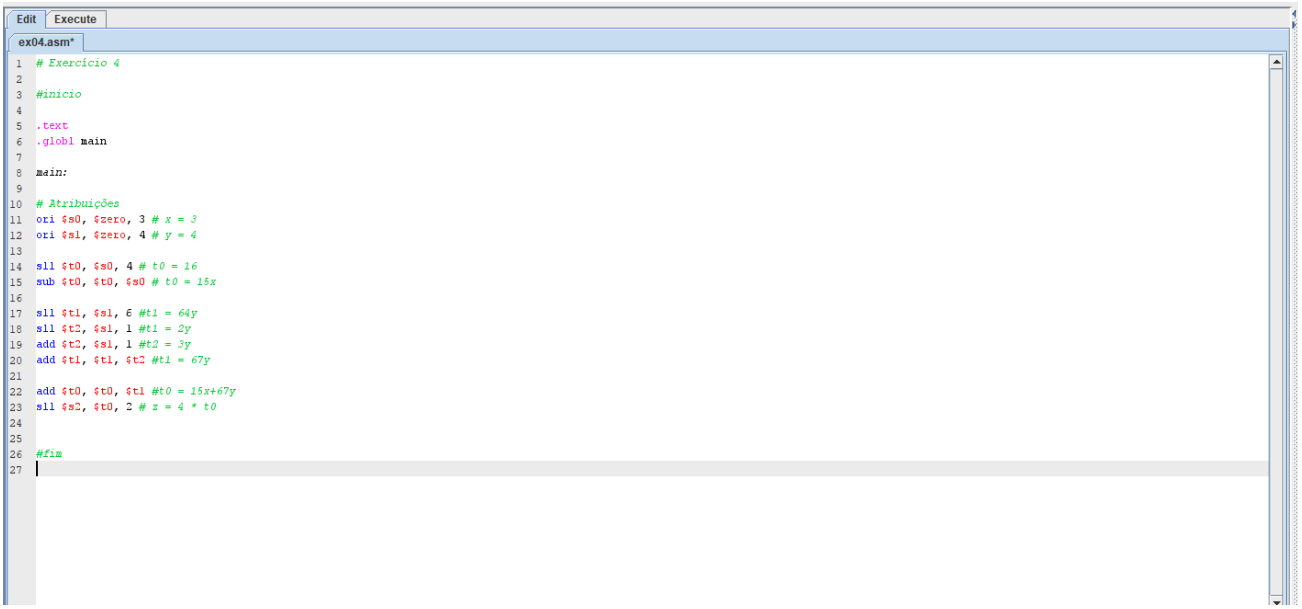
```
69  add $t1, $t1, $s1 # 42y
70  add $t1, $t1, $s1 # 43y
71  add $t1, $t1, $s1 # 44y
72  add $t1, $t1, $s1 # 45y
73  add $t1, $t1, $s1 # 46y
74  add $t1, $t1, $s1 # 47y
75  add $t1, $t1, $s1 # 48y
76  add $t1, $t1, $s1 # 49y
77  add $t1, $t1, $s1 # 50y
78  add $t1, $t1, $s1 # 51y
79  add $t1, $t1, $s1 # 52y
80  add $t1, $t1, $s1 # 53y
81  add $t1, $t1, $s1 # 54y
82  add $t1, $t1, $s1 # 55y
83  add $t1, $t1, $s1 # 56y
84  add $t1, $t1, $s1 # 57y
85  add $t1, $t1, $s1 # 58y
86  add $t1, $t1, $s1 # 59y
87  add $t1, $t1, $s1 # 60y
88  add $t1, $t1, $s1 # 61y
89  add $t1, $t1, $s1 # 62y
90  add $t1, $t1, $s1 # 63y
91  add $t1, $t1, $s1 # 64y
92  add $t1, $t1, $s1 # 65y
93  add $t1, $t1, $s1 # 66y
94  add $t1, $t1, $s1 # 67y
95
96  add $t3, $t0, $t1 # j = 15*x + 67*y
97  addi $t3, $t3, 4 # 4*j
98
99  #fim
```

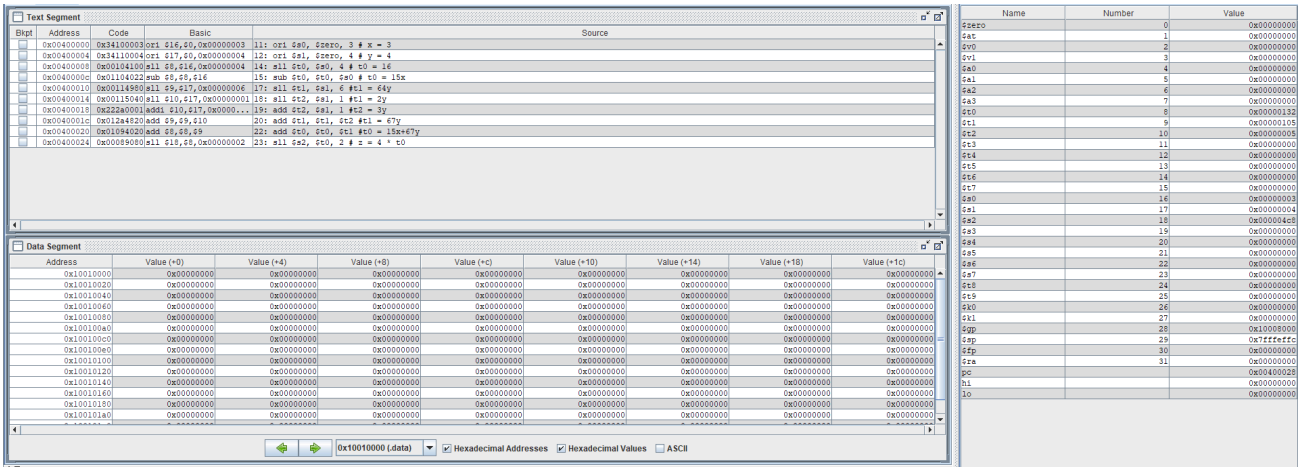


Nos exercícios a seguir procure usar as inst. sll, srl e sra:

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

}





// programa 5

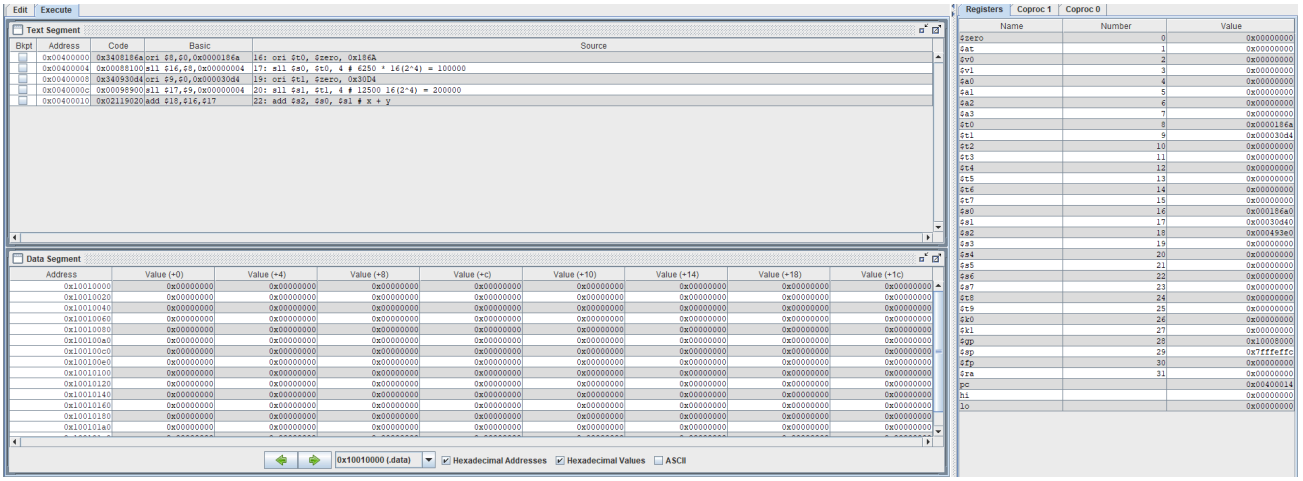
{

x = 100000;

y = 200000;

z = x + y;

}



// programa 6

{

x = o maior inteiro possível;

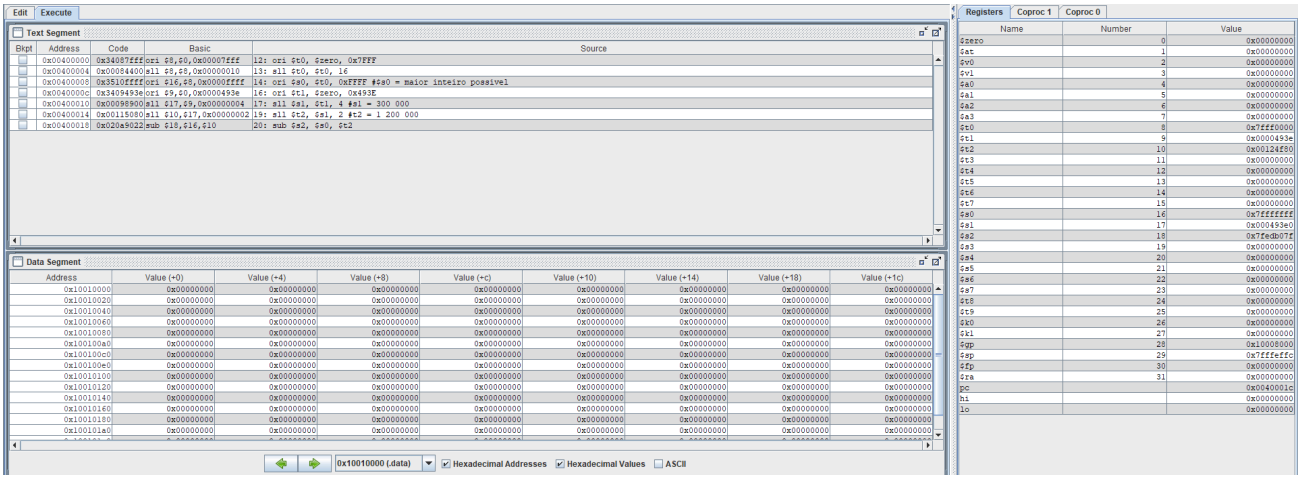
y = 300000;

z = x - 4y

}

ex06.asm

```
1  # Exercício 6
2
3  #inicio
4
5  .text
6  .globl main
7
8  main:
9
10 # Atribuições
11
12 ori $t0, $zero, 0x7FFF
13 sll $t0, $t0, 16
14 ori $s0, $t0, 0xFFFF #$s0 = maior inteiro possivel
15
16 ori $t1, $zero, 0x493E
17 sll $s1, $t1, 4 #s1 = 300 000
18
19 sll $t2, $s1, 2 #t2 = 1 200 000
20 sub $s2, $s0, $t2
21
22 #fim
23
```



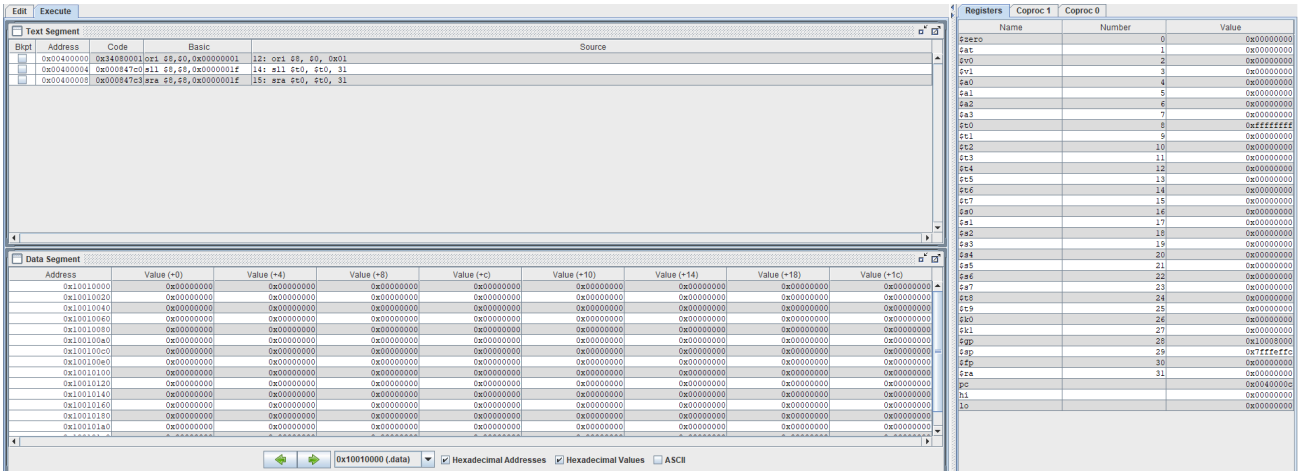
// programa 7

Considere a seguinte instrução iniciando um programa:

ori $8, $0, 0x01

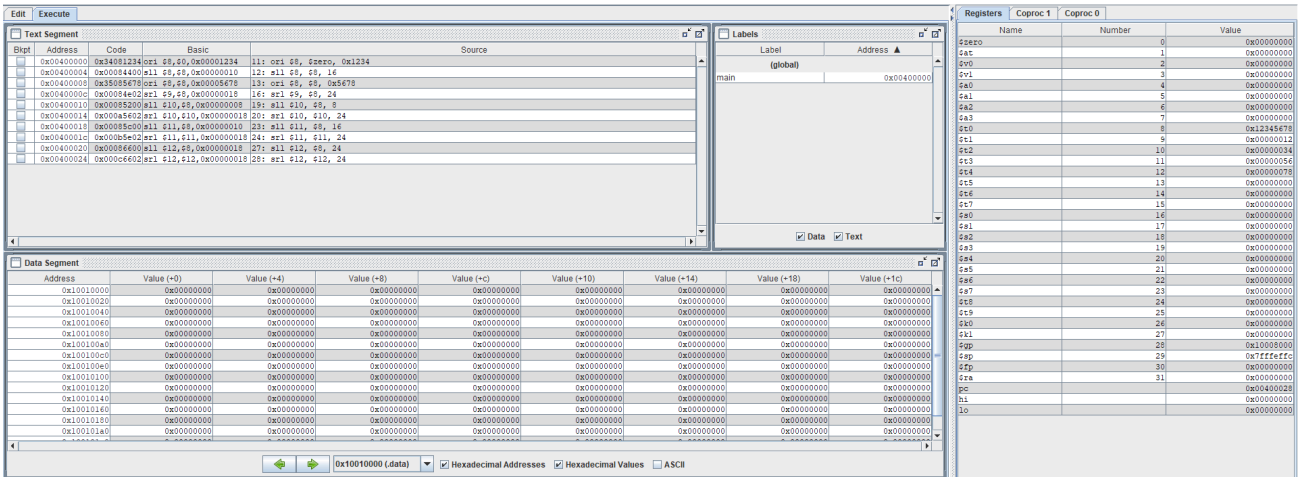
Usando apenas instruções reg-reg lógicas e/ou instruções de deslocamento (sll, srl e sra), continuar o programa de forma que ao final, tenhamos o seguinte conteúdo no registrador $8:
$8 = 0xFFFFFFFF

```
# Exercício 7

#inicio

.text
.globl main

main:

# Atribuições

ori $8, $0, 0x01

sll $t0, $t0, 31
sra $t0, $t0, 31


#fim
```



// programa 8
Inicialmente escreva um programa que faça:
$8 = 0x12345678.
A partir do registrador $8 acima, usando apenas instruções lógicas (or, ori, and, andi, xor, xori) e instruções de deslocamento (sll, srl e sra), você deverá obter os seguintes valores nos respectivos registradores:
$9 = 0x12
$10 = 0x34
$11 = 0x56
$12 = 0x78
Para os programas a seguir use instruções de Memória (lw e sw)

```
1  # Exercício 8
2
3  #inicio
4
5  .text
6  .globl main
7
8  main:
9
10 # Atribuições
11 ori $8, $zero, 0x1234
12 sll $8, $8, 16
13 ori $8, $8, 0x5678
14 # t0 = 0x12345678
15
16 srl $9, $8, 24
17 # t1 = 0x00000012
18
19 sll $10, $8, 8
20 srl $10, $10, 24
21 # t2 = 0x00000034
22
23 sll $11, $8, 16
24 srl $11, $11, 24
25 # t3 = 0x00000034
26
27 sll $12, $8, 24
28 srl $12, $12, 24
29 # t4 = 0x00000078
30
31 #fim
```



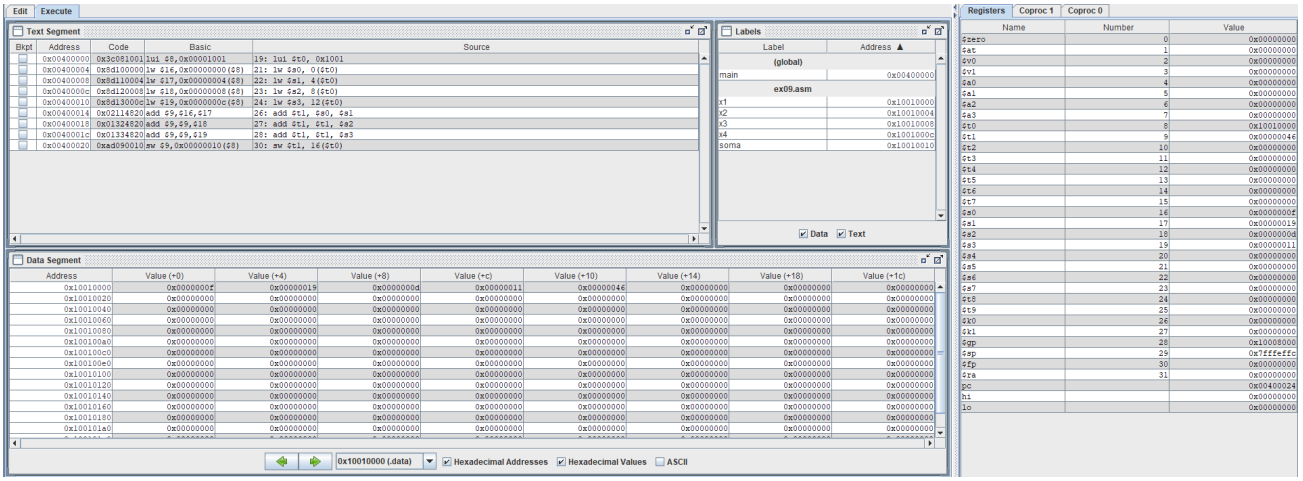
// programa 9
Considere a memória inicial da seguinte forma:
.text

.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.

ex09.asm

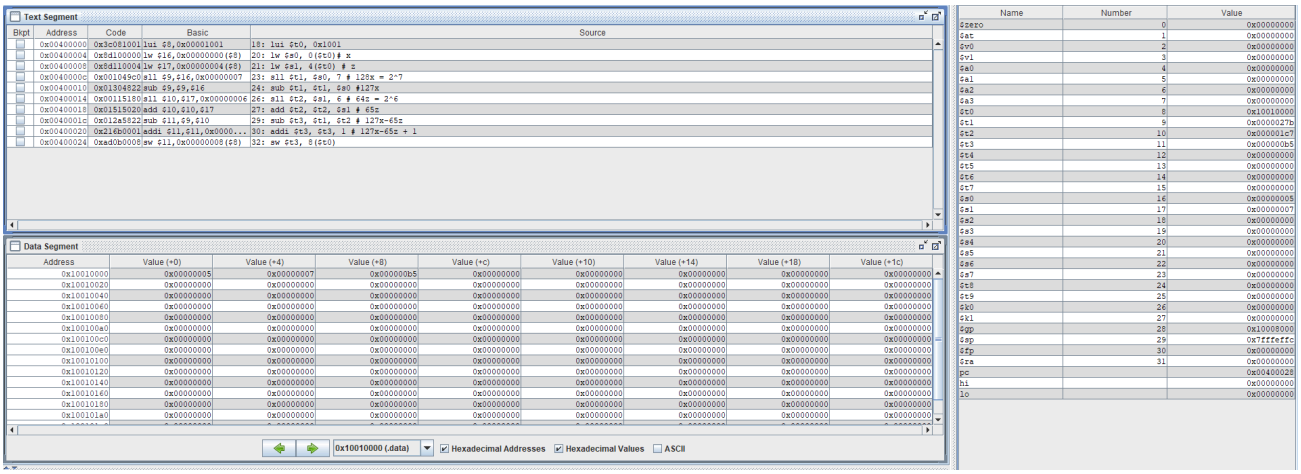
```
1  # Exercício 9
2
3  #inicio
4
5  .text
6  .globl main
7
8  main:
9
10 # Atribuições
11 .data
12 x1: .word 15
13 x2: .word 25
14 x3: .word 13
15 x4: .word 17
16 soma: .word -1
17
18 .text
19 lui $t0, 0x1001
20
21 lw $s0, 0($t0)
22 lw $s1, 4($t0)
23 lw $s2, 8($t0)
24 lw $s3, 12($t0)
25
26 add $t1, $s0, $s1
27 add $t1, $t1, $s2
28 add $t1, $t1, $s3
29
30 sw $t1, 16($t0)
31
32 #fim
```



// 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.

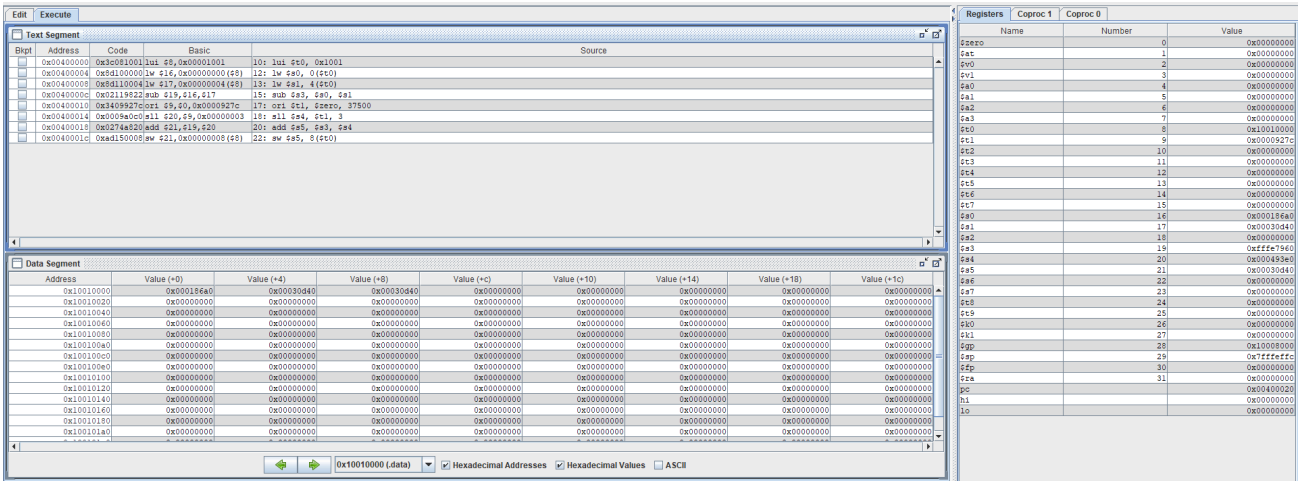
```assembly
# Exercício 10

#inicio

.text
.globl main

main:

# Atribuições
.data
x: .word 5
z: .word 7
y: .word 0

.text

lui $t0, 0x1001

lw $s0, 0($t0)# x
lw $s1, 4($t0) # z

sll $t1, $s0, 7 # 128x = 2^7
sub $t1, $t1, $s0 #127x

sll $t2, $s1, 6 # 64z = 2^6
add $t2, $t2, $s1 # 65z

sub $t3, $t1, $t2 # 127x-65z
addi $t3, $t3, 1 # 127x-65z + 1

sw $t3, 8($t0)

#fim
```



// 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.

```assembly
# Exercício 11

.data
x: .word 100000
z: .word 200000
y: .word 0

.text

lui $t0, 0x1001

lw $s0, 0($t0)
lw $s1, 4($t0)

sub $s3, $s0, $s1

ori $t1, $zero, 37500
sll $s4, $t1, 3

add $s5, $s3, $s4

sw $s5, 8($t0)
```



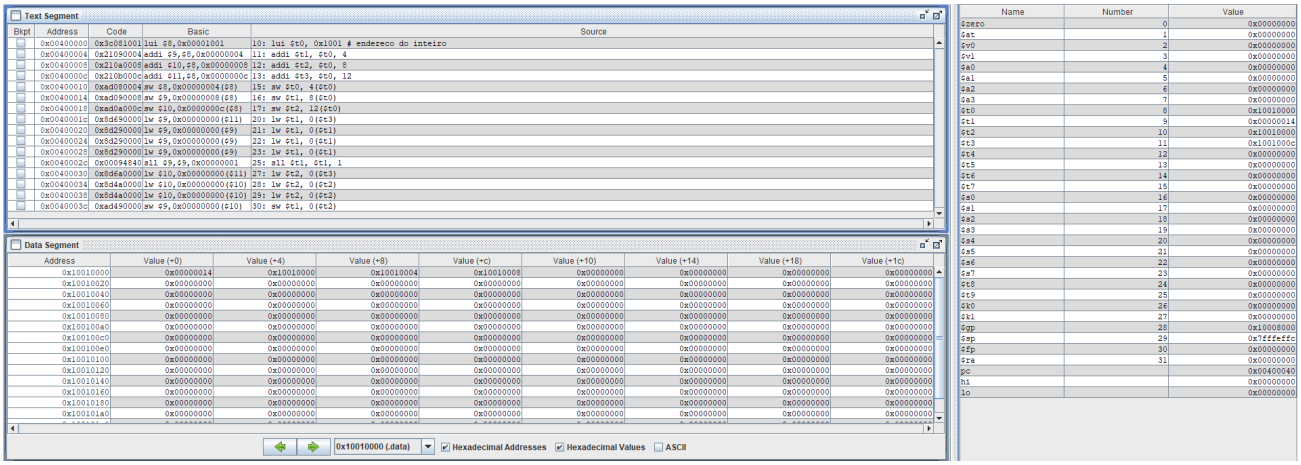
// programa 12
Considere a seguinte situação:
int ***x;

onde x contem um ponteiro para um ponteiro para um ponteiro para um inteiro. Nessa situação, considere que a posição inicial de memória contenha o inteiro em questão. Coloque todos os outros valores em registradores, use os endereços de memória que quiser dentro do espaço de endereçamento do Mips.

Resumo do problema:
k = MEM [MEM [MEM [x]]].

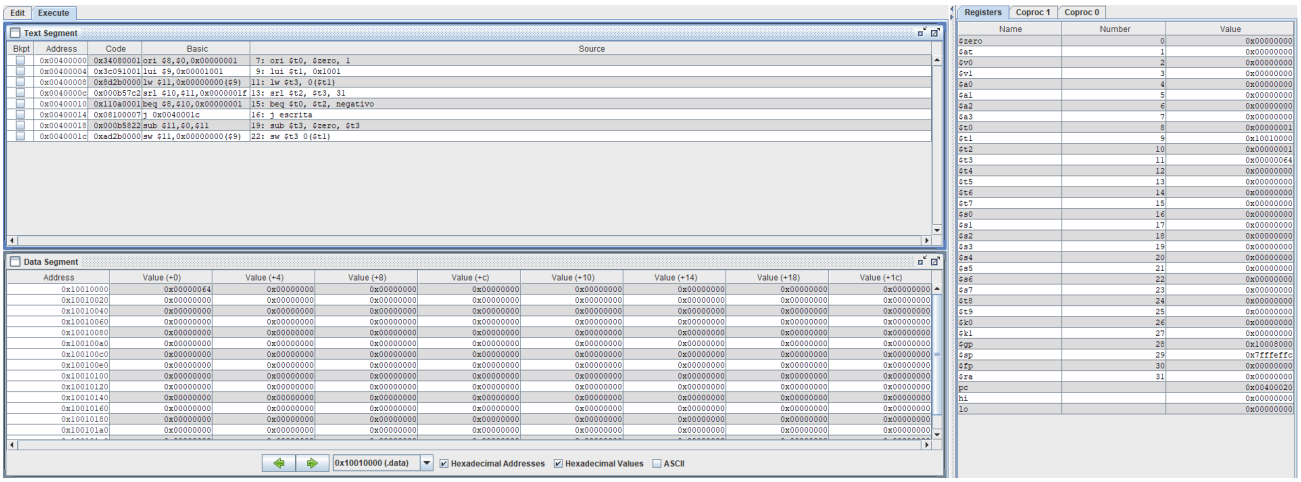
Crie um programa que implemente a estrutura de dados acima, leia o valor de K, o multiplique por 2 e o reescreva no local correto conhecendo-se apenas o valor de x.
Para os programas a seguir use instruções de desvio (beq, bne, j)

```
1  # Exercicio 12
2
3  .data
4  x: .word 10
5  p: .word 0
6  pp: .word 0
7  ppp: .word 0
8
9  .text
10 lui $t0, 0x1001 # endereco do inteiro
11 addi $t1, $t0, 4
12 addi $t2, $t0, 8
13 addi $t3, $t0, 12
14
15 sw $t0, 4($t0)
16 sw $t1, 8($t0)
17 sw $t2, 12($t0)
18
19
20 lw $t1, 0($t3)
21 lw $t1, 0($t1)
22 lw $t1, 0($t1)
23 lw $t1, 0($t1)
24
25 sll $t1, $t1, 1
26
27 lw $t2, 0($t3)
28 lw $t2, 0($t2)
29 lw $t2, 0($t2)
30 sw $t1, 0($t2)
```



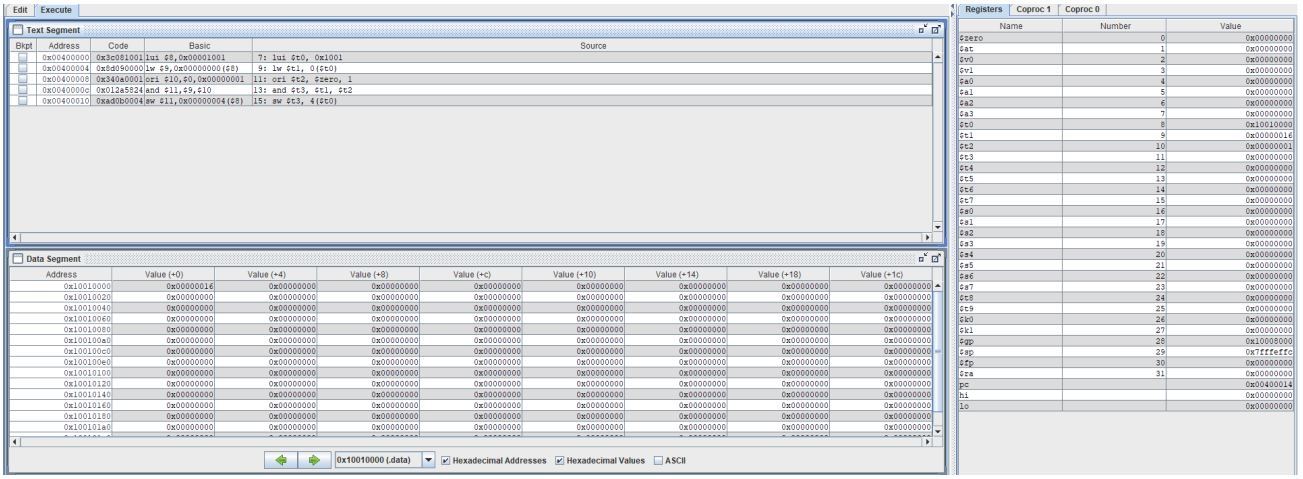
// programa 13:
Escreva um programa que leia um valor A da memória, identifique se o número é negativo ou não e encontre o seu módulo. O valor deverá ser reescrito sobre A.

```
1  # Exercicio 13
2
3  .data
4  A: .word -100
5
6  .text
7  ori $t0, $zero, 1
8
9  lui $t1, 0x1001
10
11 lw $t3, 0($t1)
12
13 srl $t2, $t3, 31
14
15 beq $t0, $t2, negativo
16 j escrita
17
18 negativo:
19 sub $t3, $zero, $t3
20
21 escrita:
22 sw $t3 0($t1)
23
```



// 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).

```
# Exercicio 14

.data
A: .word 22

.text
lui $t0, 0x1001

lw $t1, 0($t0)

ori $t2, $zero, 1

and $t3, $t1, $t2

sw $t3, 4($t0)
```

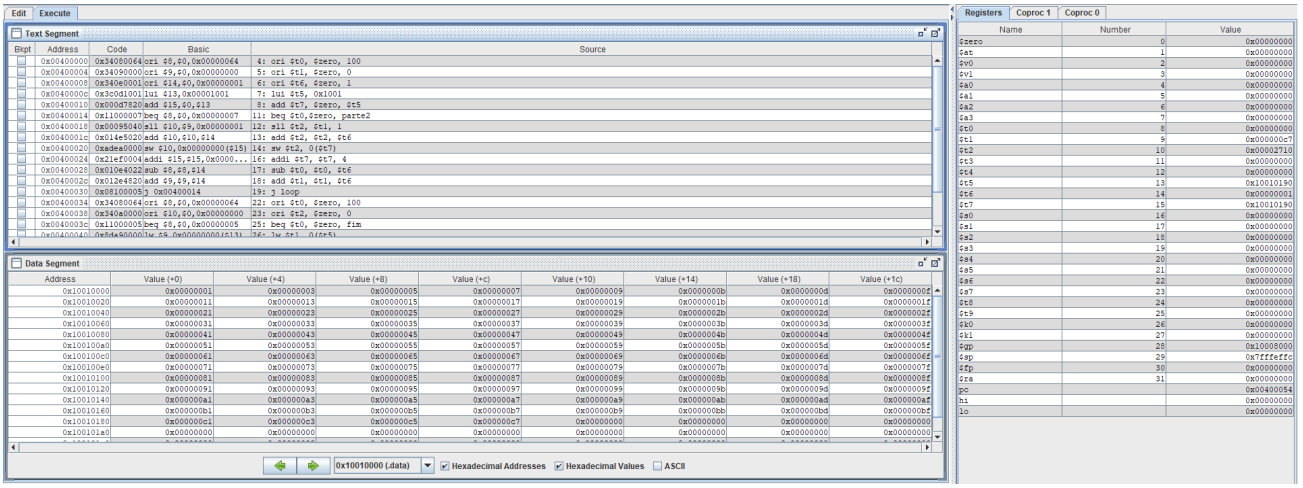


// programa 15:

Escrever um programa que crie um vetor de 100 elementos na memória onde 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)

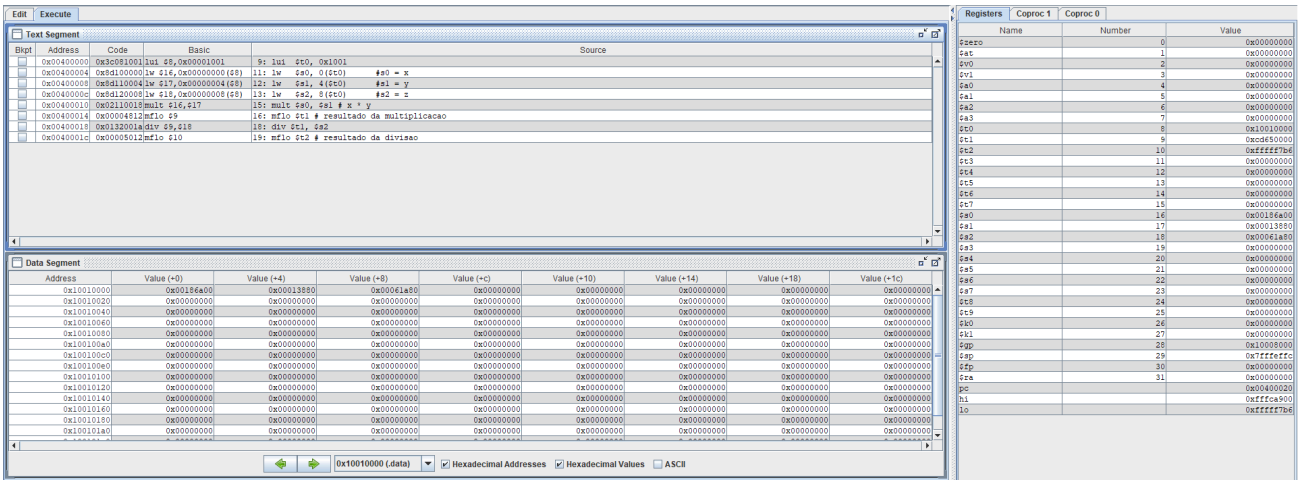
```
# Exercicio 15

.text
ori $t0, $zero, 100
ori $t1, $zero, 0
ori $t6, $zero, 1
lui $t5, 0x1001
add $t7, $zero, $t5

loop:
beq $t0,$zero, parte2
sll $t2, $t1, 1
add $t2, $t2, $t6
sw $t2, 0($t7)

addi $t7, $t7, 4
sub $t0, $t0, $t6
add $t1, $t1, $t6
j loop

parte2:
ori $t0, $zero, 100
ori $t2, $zero, 0
soma:
beq $t0, $zero, fim
lw $t1, 0($t5)
add $t2, $t1, $t2

addi $t5, $t5, 4
sub $t0, $t0, $t6
j soma
```



// 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.

```
1  # Exercicio 16
2
3  .data
4  x:.word 0x186A00
5  y:.word 0x13880
6  z:.word 0x61A80
7
8  .text
9  lui  $t0, 0x1001
10
11 lw   $s0, 0($t0)      #s0 = x
12 lw   $s1, 4($t0)      #s1 = y
13 lw   $s2, 8($t0)      #s2 = z
14
15 mult $s0, $s1 # x * y
16 mflo $t1 # resultado da multiplicacao
17
18 div $t1, $s2
19 mflo $t2 # resultado da divisao
```

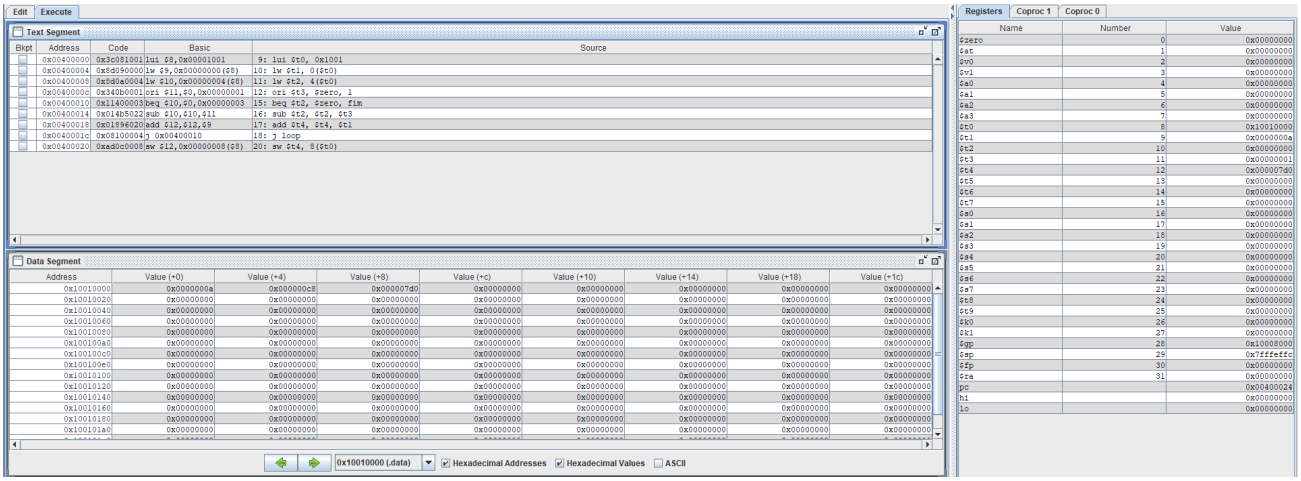


// programa 17

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.

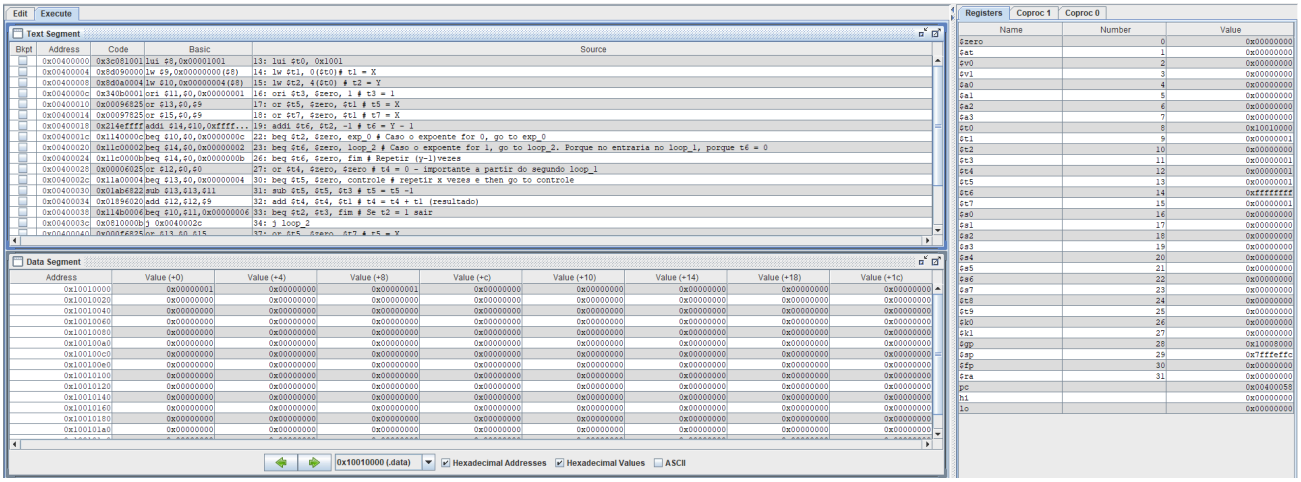
```
# Exercicio 17

.data
x: .word 10
y: .word 200

.text

lui $t0, 0x1001
lw $t1, 0($t0)
lw $t2, 4($t0)
ori $t3, $zero, 1

loop:
beq $t2, $zero, fim
sub $t2, $t2, $t3
add $t4, $t4, $t1
j loop
fim:
sw $t4, 8($t0)
```



// 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)

```
.data
x: .word 1
y: .word 0

.text

# 5^2 = 5 * 5 = (5+5+5+5+5)
# 5^3 = 5 * 5 * 5 = (5+5+5+5+5) + (5+5+5+5+5) + (5+5+5+5+5) + (5+5+5+5+5) + (5+5+5+5+5)
# 5^4 = 5*5*5*5 = 5^3 + 5^3 + 5^3 + 5^3 + 5^3

lui $t0, 0x1001
lw $t1, 0($t0)# t1 = X
lw $t2, 4($t0) # t2 = Y
ori $t3, $zero, 1 # t3 = 1
or $t5, $zero, $t1 # t5 = X
or $t7, $zero, $t1 # t7 = X
addi $t6, $t2, -1 # t6 = Y - 1


beq $t2, $zero, exp_0 # Caso o expoente for 0, go to exp_0
beq $t6, $zero, loop_2 # Caso o expoente for 1, go to loop_2. Porque no entraria no loop_1, porque t6 = 0

loop_1:
beq $t6, $zero, fim # Repetir (y-1)vezes
or $t4, $zero, $zero # t4 = 0 - importante a partir do segundo loop_1

loop_2:
beq $t5, $zero, controle # repetir x vezes e then go to controle
sub $t5, $t5, $t3 # t5 = t5 -1
add $t4, $t4, $t1 # t4 = t4 + t1 (resultado)
beq $t2, $t3, fim # Se t2 = 1 sair
j loop_2

controle:
or $t5, $zero, $t7 # t5 = X
or $t1, $zero, $t4 # t1 = t4
sub $t6, $t6, $t3 # t6 = t6 - 1
j loop_1


exp_0:
ori $t4, $zero, 1 # resultado = 1

fim:
sw $t4, 8($t0) # Resultado
```

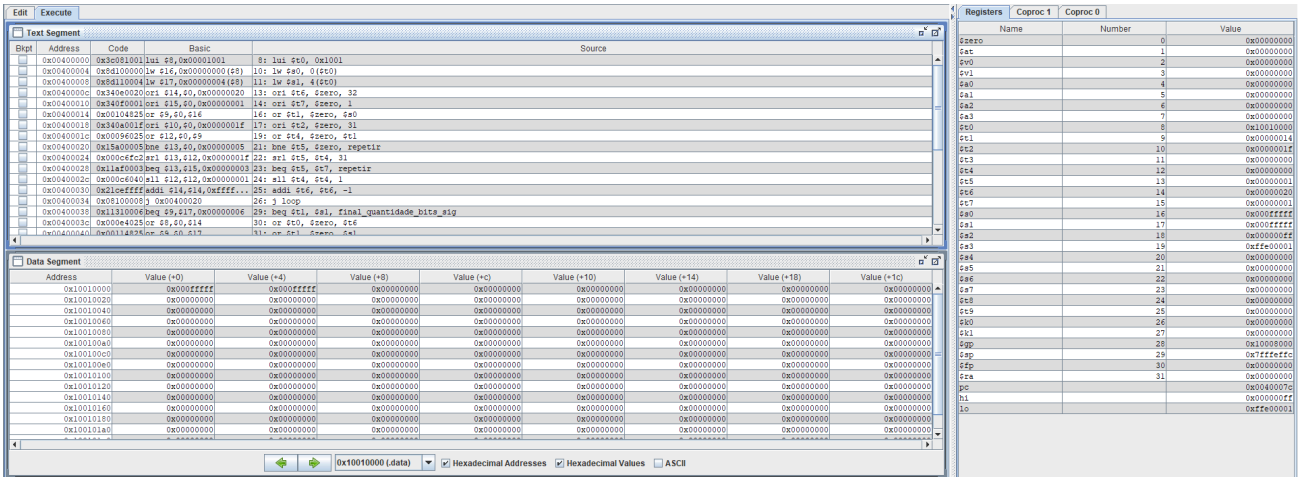


// programa 19

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).

ex19.asm | ex20.asm | ex21.asm

```
.data
x: .word 0xFFFFF
y: .word 0xFFFFF

.text
lui $t0, 0x1001

lw $s0, 0($t0)
lw $s1, 4($t0)

ori $t6, $zero, 32
ori $t7, $zero, 1

or $t1, $zero, $s0
ori $t2, $zero, 31

or $t4, $zero, $t1
loop:
bne $t5, $zero, repetir
srl $t5, $t4, 31
beq $t5, $t7, repetir
sll $t4, $t4, 1
addi $t6, $t6, -1
j loop

repetir:
beq $t1, $s1, final_quantidade_bits_sig
or $t0, $zero, $t6
or $t1, $zero, $s1
or $t4, $zero, $t1
or $t5, $zero, $zero
ori $t6, $zero, 32

j loop

final_quantidade_bits_sig:
or $t1, $zero, $t6
ori $t6, $zero, 32


mult $s0, $s1

add $t4, $t0, $t1

slt $t4, $t4, $t6

beq $t4, $zero, maior_32
mflo $s2
```



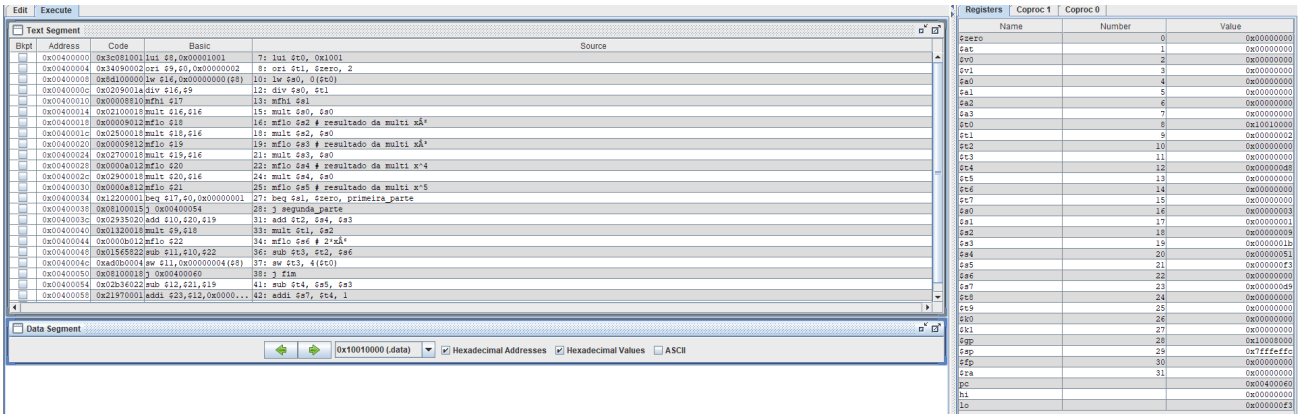
Para os exercícios a seguir, considere as variáveis com números abaixo de 16 bits, salvo se mencionado ao contrário.

// programa 20

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

$x^5 – x^3 + 1$ se x for impar

```
# Exercicio20

.data
x: .word 3

.text
lui $t0, 0x1001
ori $t1, $zero, 2

lw $s0, 0($t0)

div $s0, $t1
mfhi $s1

mult $s0, $s0
mflo $s2 # resultado da multi xÃ²

mult $s2, $s0
mflo $s3 # resultado da multi xÃ³

mult $s3, $s0
mflo $s4 # resultado da multi x^4

mult $s4, $s0
mflo $s5 # resultado da multi x^5

beq $s1, $zero, primeira_parte
j segunda_parte

primeira_parte:
add $t2, $s4, $s3

mult $t1, $s2
mflo $s6 # 2*xÃ²

sub $t3, $t2, $s6
sw $t3, 4($t0)
j fim

segunda_parte:
sub $t4, $s5, $s3
addi $s7, $t4, 1
sw $s7, 4($t0)

fim:
```



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.
// programa 21

$y = x^3 + 1$ se $x > 0$

$x^4 - 1$ se $x <= 0$

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.

Desafio:
Todos viram durante a parte aritmética que podemos utilizar a ULA e registradores para multiplicar dois números através de 3 algoritmos.
Você deverá escrever um programa que leia dois números da memória (primeira posição e segunda posição) os multiplique e coloque o resultado na terceira

posição a memória. Procure usar a versão 3 do algoritmo de multiplicação, pode ser mais simples !!

Atenção que, ao multiplicarmos dois números de 32 bits a reposta poderá ser um número de 64 bits, assim a resposta deverá estar contida em dois registradores temporários, um armazenará a parte superior do número e outro a parte inferior, portanto duas posições de memória serão escritas (a terceira e a quarta).
Para os programas a seguir use instruções mult, div, mflo e mfhi.

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.: Letra 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.: Letra B

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

A. mult
B. multu
C. multi
D. mutt

R.: Letra 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.: Letra 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.: Letra B

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

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

R.: Letra A

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

A. dv
B. divide
C. divu
D. div

R.: Letra

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.: Letra 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.: Letra A

10. Qual sequência 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.: Letra A