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. Dezesseis 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

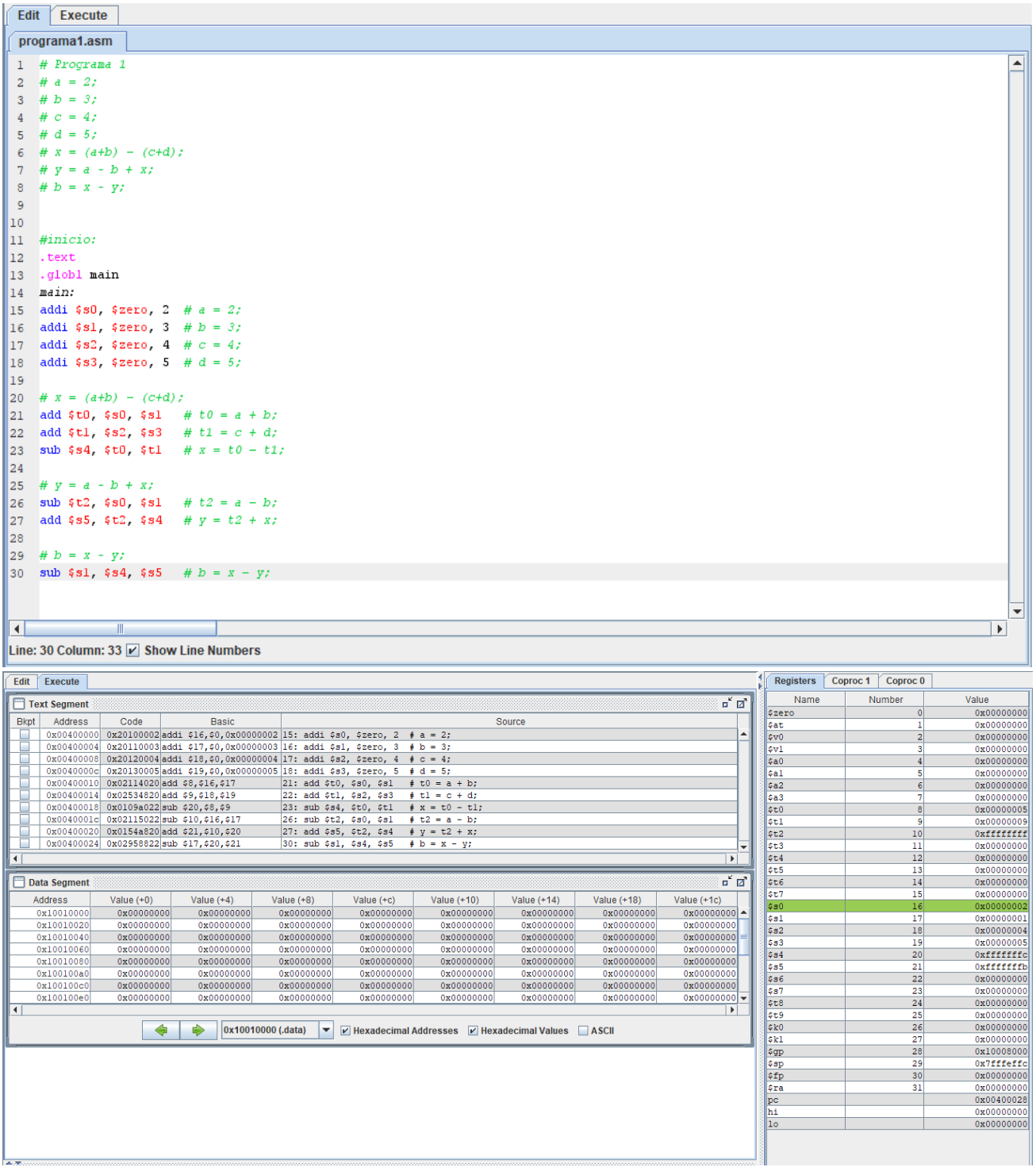
18. Todas as instruções de máquina têm os mesmos campos?
A. Não. Diferentes de instruções de máquina possuem campos diferentes.
B. Não. Cada instrução de máquina é completamente diferente de qualquer outra.
C. Sim. Todas as instruções de máquina têm os mesmos campos na mesma ordem.
D. Sim. Todas as instruções 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;
}
```



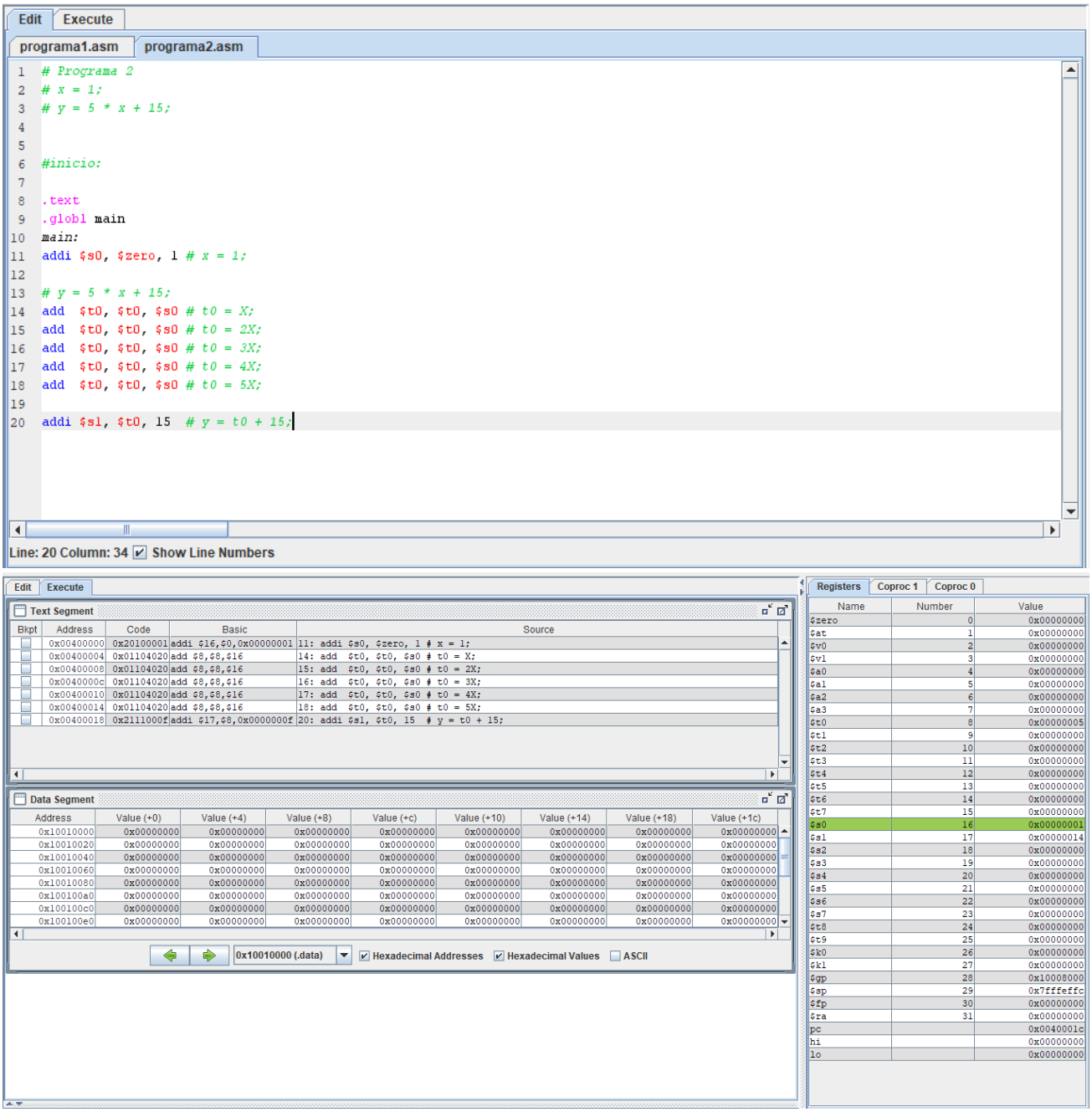
//programa 2 (add, addi, sub, lógicas)

{

x = 1;

y = 5*x + 15;

}



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

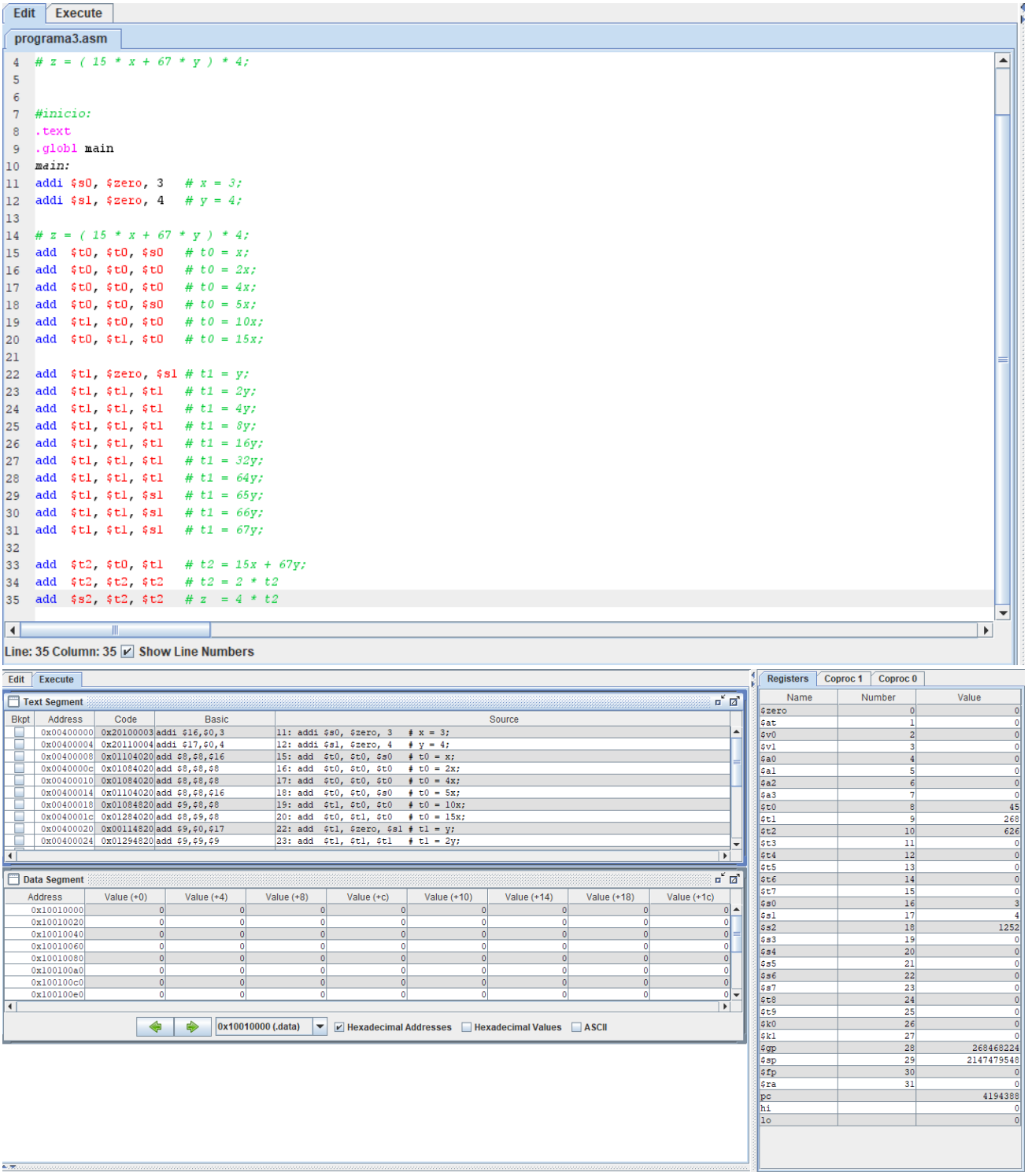
{

x = 3;

y = 4 ;

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

}



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

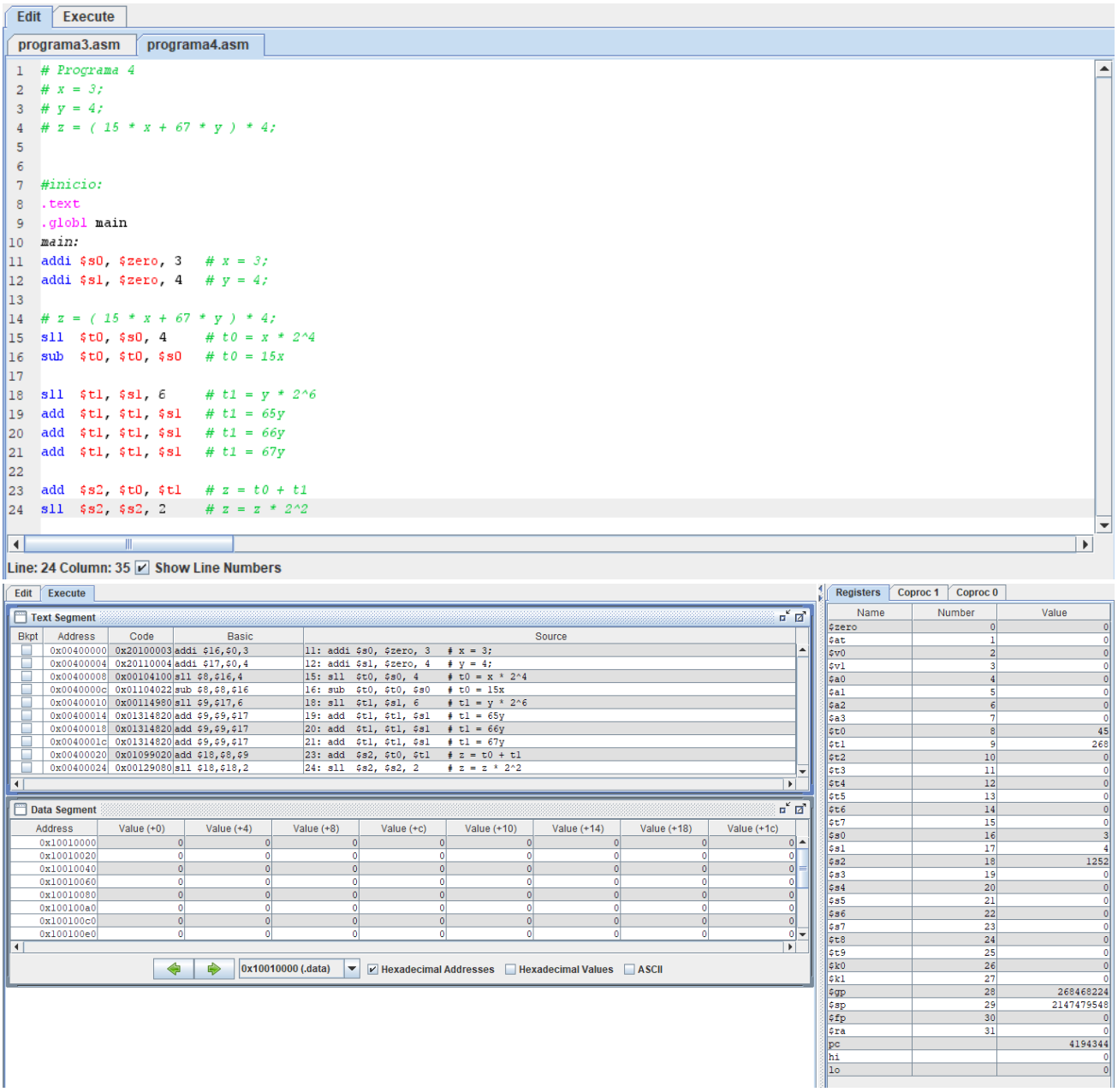
{

x = 3;

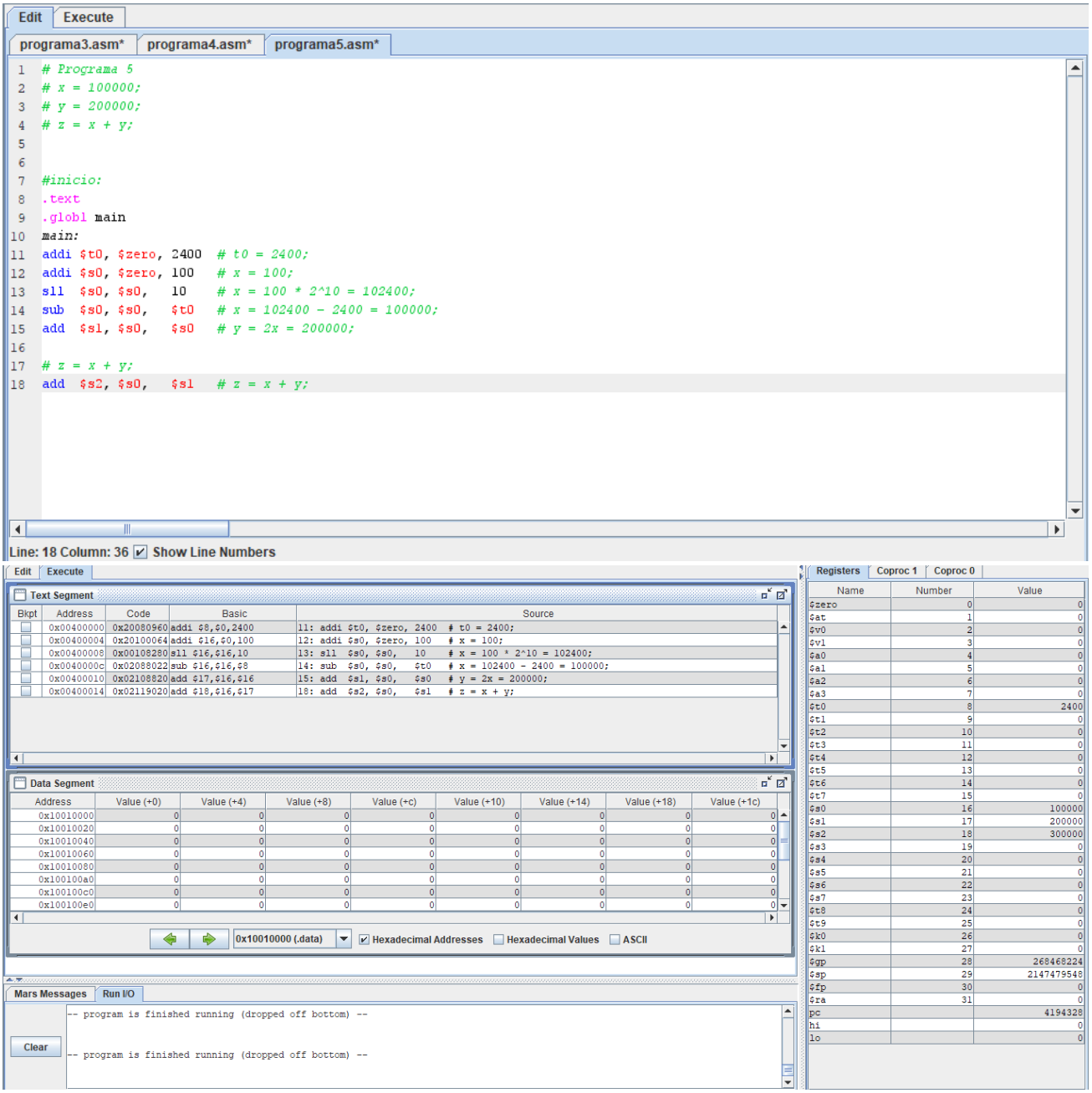
y = 4 ;

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

}




```
// programa 5
{
x = 100000;
y = 200000;
z = x + y;
}
```



/ programa 6

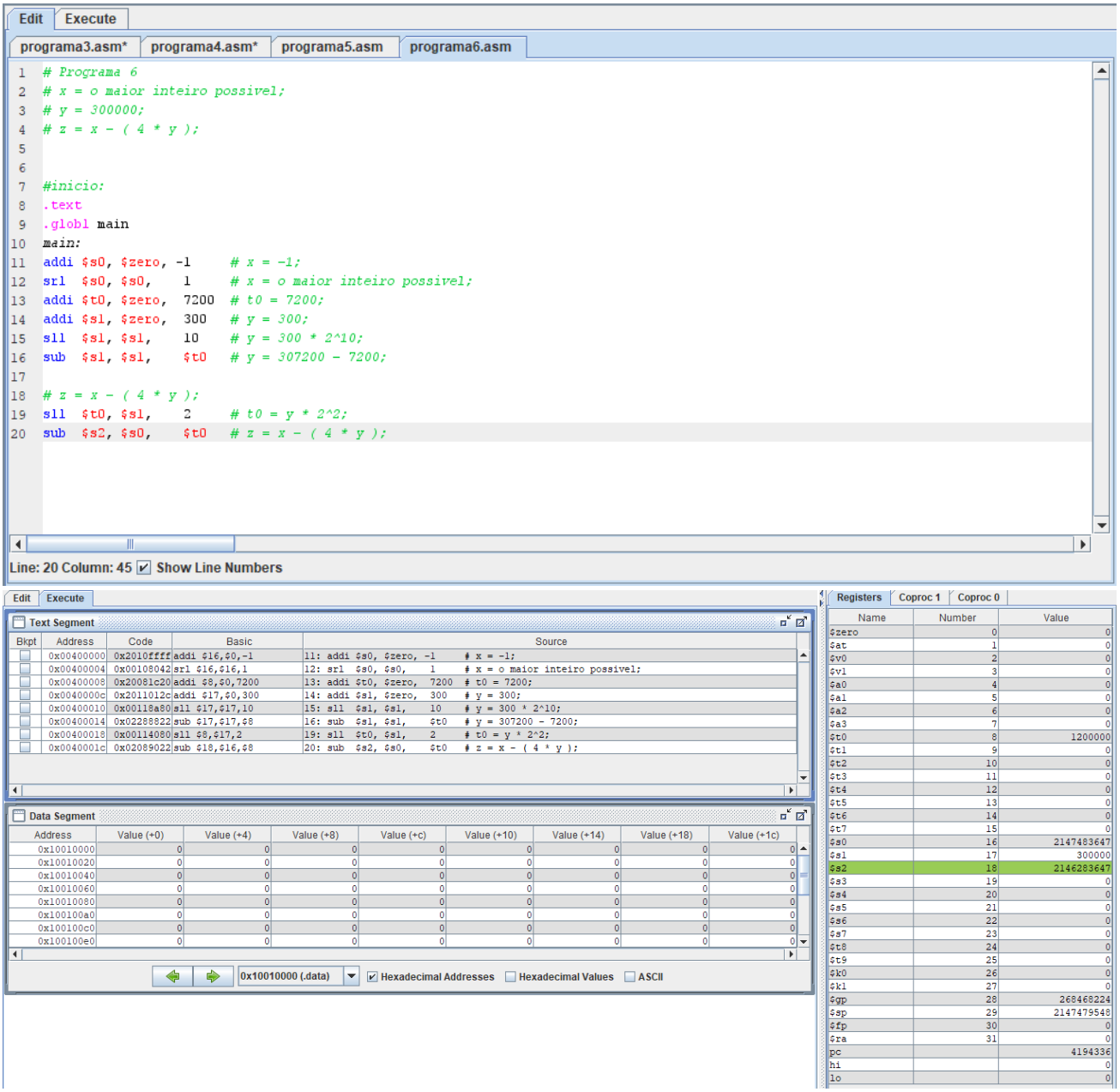
{

x = o maior inteiro possível;

y = 300000;

z = x - 4y

}



// programa 7

Considere a seguinte instrução iniciando um programa:

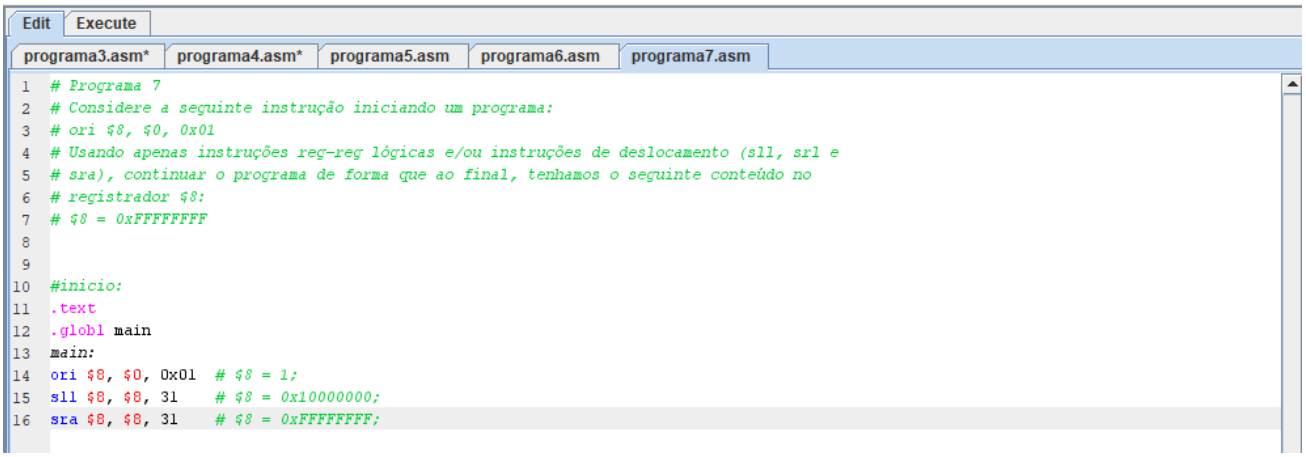
ori $8, $0, 0x01

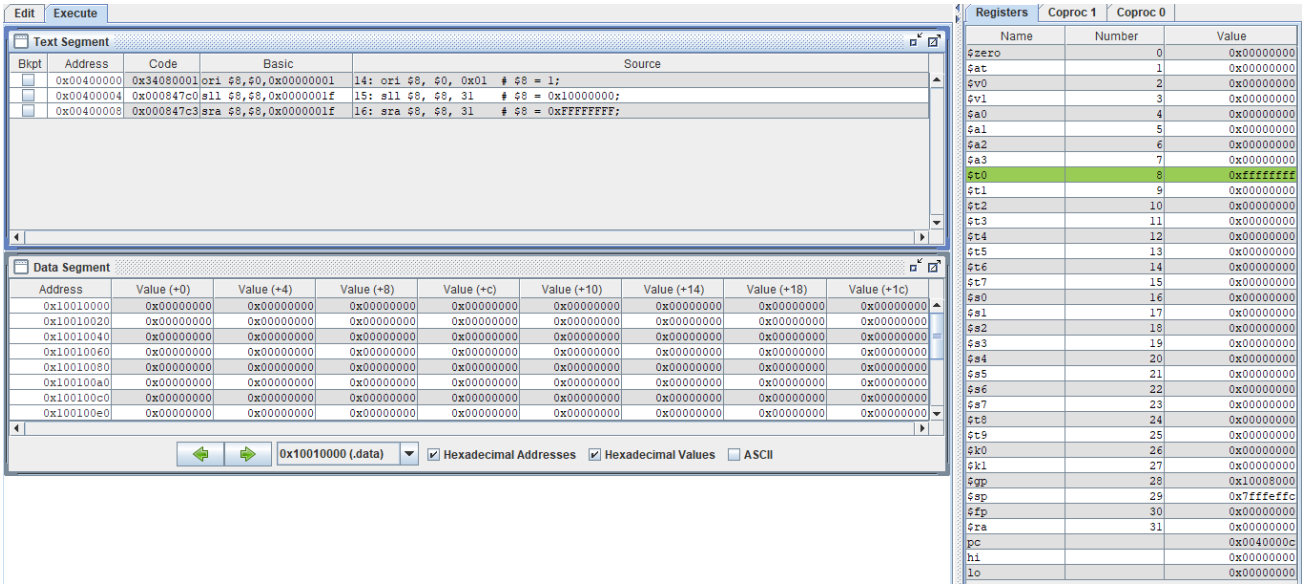
Usando apenas instruções reg-reg lógicas e/ou instruções de deslocamento (sll, srl e

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

registrador $8:

$8 = 0xFFFFFFFF





// 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

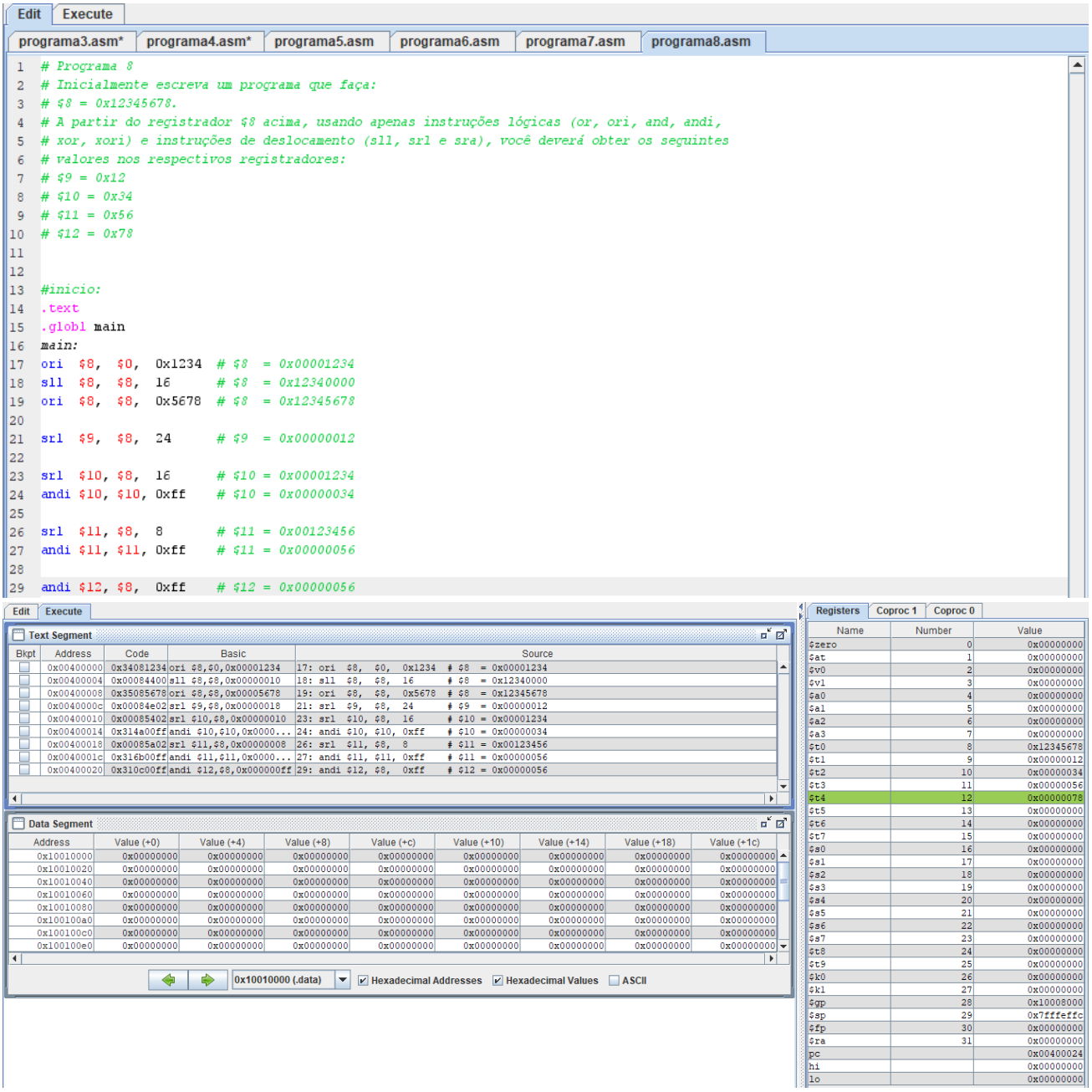
valores nos respectivos registradores:

$9 = 0x12

$10 = 0x34

$11 = 0x56

$12 = 0x78



// programa 9

Considere a memória inicial da seguinte forma:

.text

.data

x1: .word 15

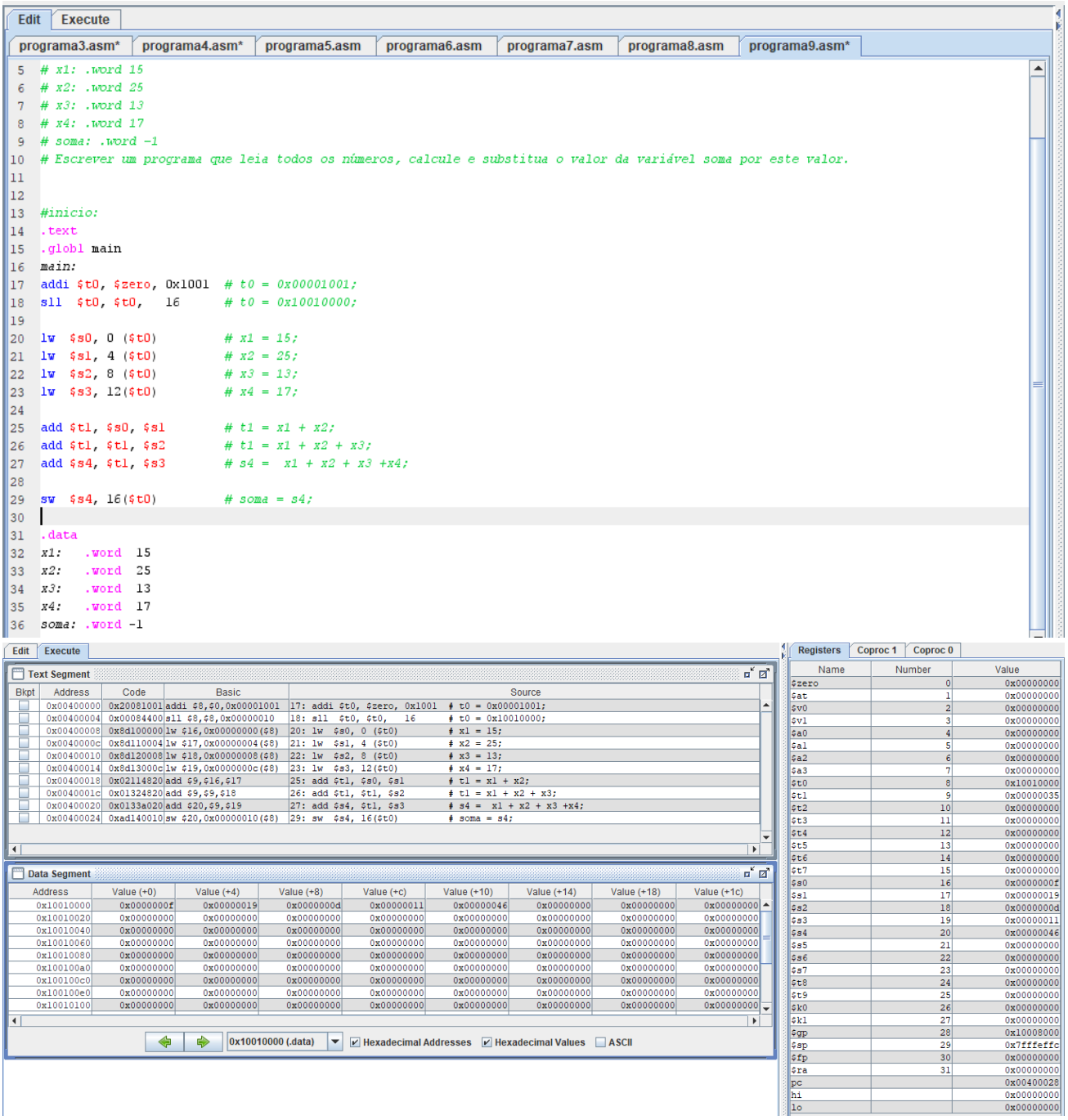
x2: .word 25

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.



```
# 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.


#inicio:
.text
.globl main
main:
addi $t0, $zero, 0x1001  # t0 = 0x00001001;
sll  $t0, $t0,   16      # t0 = 0x10010000;

lw  $s0, 0 ($t0)         # x1 = 15;
lw  $s1, 4 ($t0)         # x2 = 25;
lw  $s2, 8 ($t0)         # x3 = 13;
lw  $s3, 12($t0)         # x4 = 17;

add $t1, $s0, $s1        # t1 = x1 + x2;
add $t1, $t1, $s2        # t1 = x1 + x2 + x3;
add $s4, $t1, $s3        # s4 =  x1 + x2 + x3 +x4;

sw  $s4, 16($t0)         # soma = s4;

.data
x1:   .word  15
x2:   .word  25
x3:   .word  13
x4:   .word  17
soma: .word -1
```

// programa 10

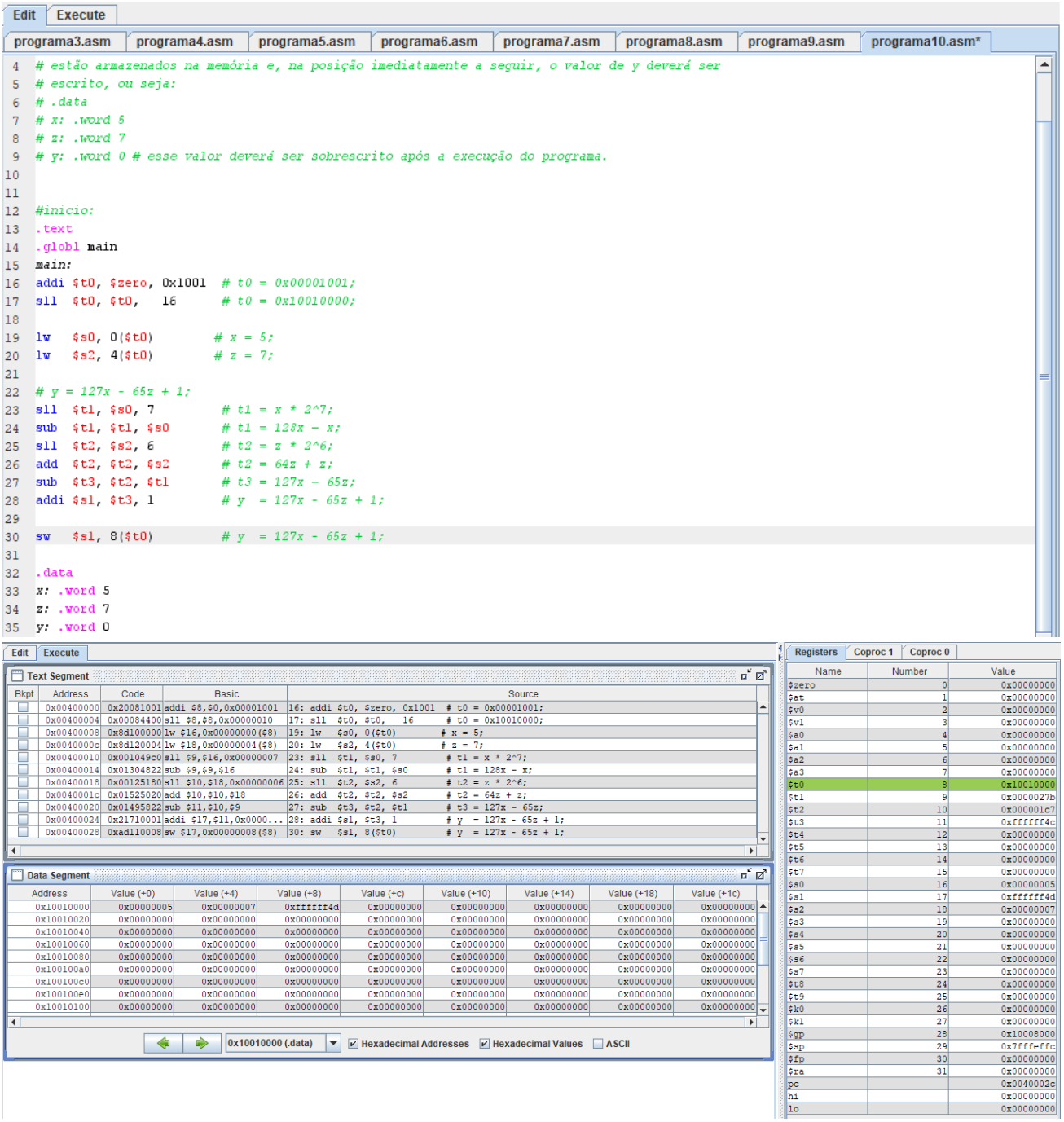
Considere o seguinte programa: y = 127x – 65z + 1

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.



// programa 11

Considere o seguinte programa: y = x – z + 300000

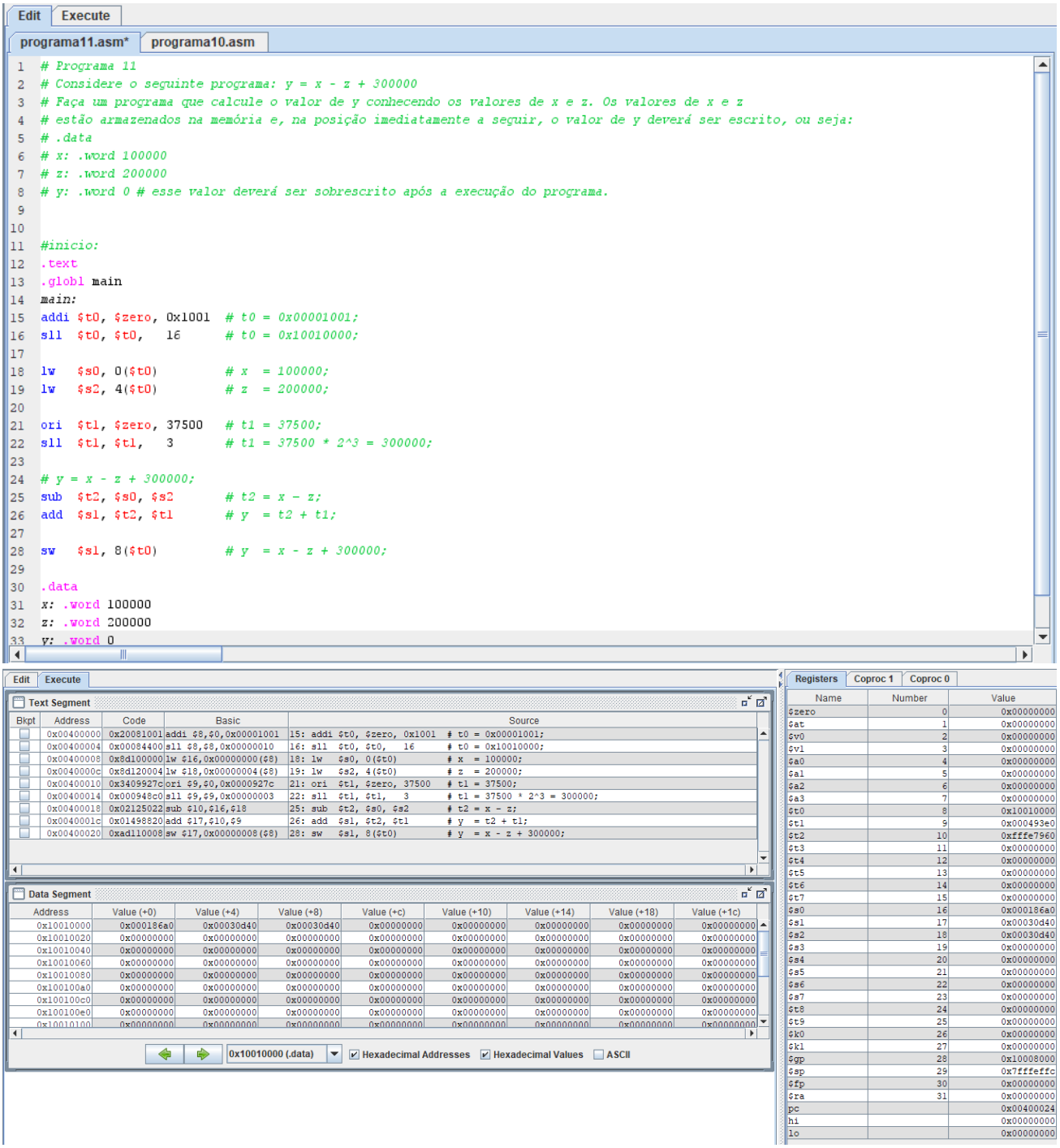
Faça um programa que calcule o valor de y conhecendo os valores de x e z. Os valores de x e z estão armazenados na memória e, na posição imediatamente a seguir, o valor de y deverá ser escrito, ou seja:

.data

x: .word 100000

z: .word 200000

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



```
# 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.


#inicio:
.text
.globl main
main:
addi $t0, $zero, 0x1001  # t0 = 0x00001001;
sll  $t0, $t0,   16      # t0 = 0x10010000;

lw   $s0, 0($t0)         # x  = 100000;
lw   $s2, 4($t0)         # z  = 200000;

ori  $t1, $zero, 37500   # t1 = 37500;
sll  $t1, $t1,   3       # t1 = 37500 * 2^3 = 300000;

# y = x - z + 300000;
sub  $t2, $s0, $s2       # t2 = x - z;
add  $s1, $t2, $t1       # y  = t2 + t1;

sw   $s1, 8($t0)         # y  = x - z + 300000;

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

Edit | Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20081001	addi $8,$0,0x00001001	15: addi $t0, $zero, 0x1001 # t0 = 0x00001001;
	0x00400004	0x00084400	sll $8,$8,0x00000010	16: sll $t0, $t0, 16 # t0 = 0x10010000;
	0x00400008	0x8d100000	lw $16,0x00000000($8)	18: lw $s0, 0($t0) # x = 100000;
	0x0040000c	0x8d120004	lw $18,0x00000004($8)	19: lw $s2, 4($t0) # z = 200000;
	0x00400010	0x3409927c	ori $9,$0,0x0000927c	21: ori $t1, $zero, 37500 # t1 = 37500;
	0x00400014	0x000948c0	sll $9,$9,0x00000003	22: sll $t1, $t1, 3 # t1 = 37500 * 2^3 = 300000;
	0x00400018	0x02125022	sub $10,$16,$18	25: sub $t2, $s0, $s2 # t2 = x - z;
	0x0040001c	0x01498820	add $17,$10,$9	26: add $s1, $t2, $t1 # y = t2 + t1;
	0x00400020	0xad110008	sw $17,0x00000008($8)	28: sw $s1, 8($t0) # y = x - z + 300000;

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

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values ☐ ASCII

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	0x10010000
$t1	9	0x000493e0
$t2	10	0xfffe7960
$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
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400024
hi		0x00000000
lo		0x00000000

// programa 12

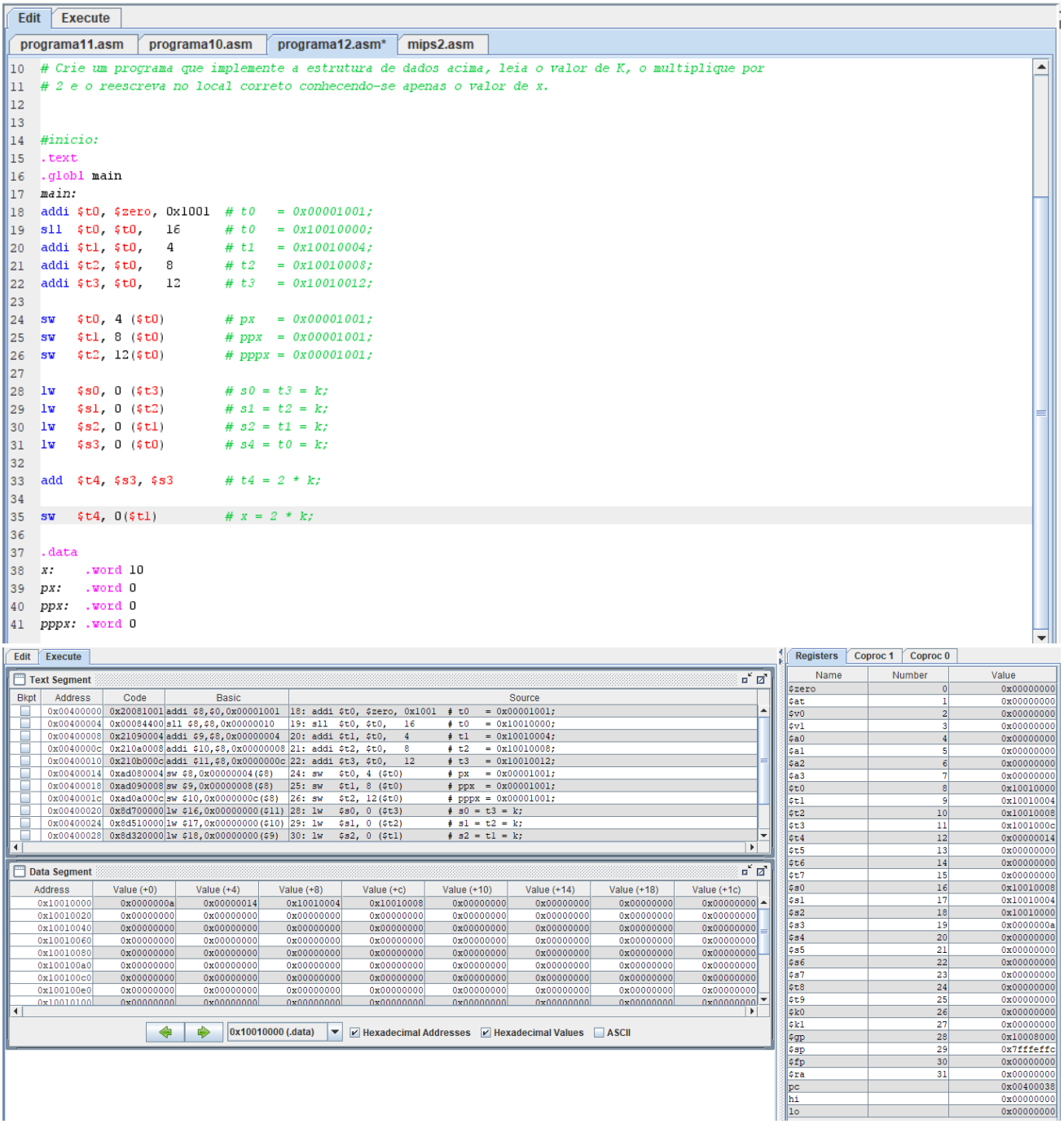
Considere a seguinte situação:

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.

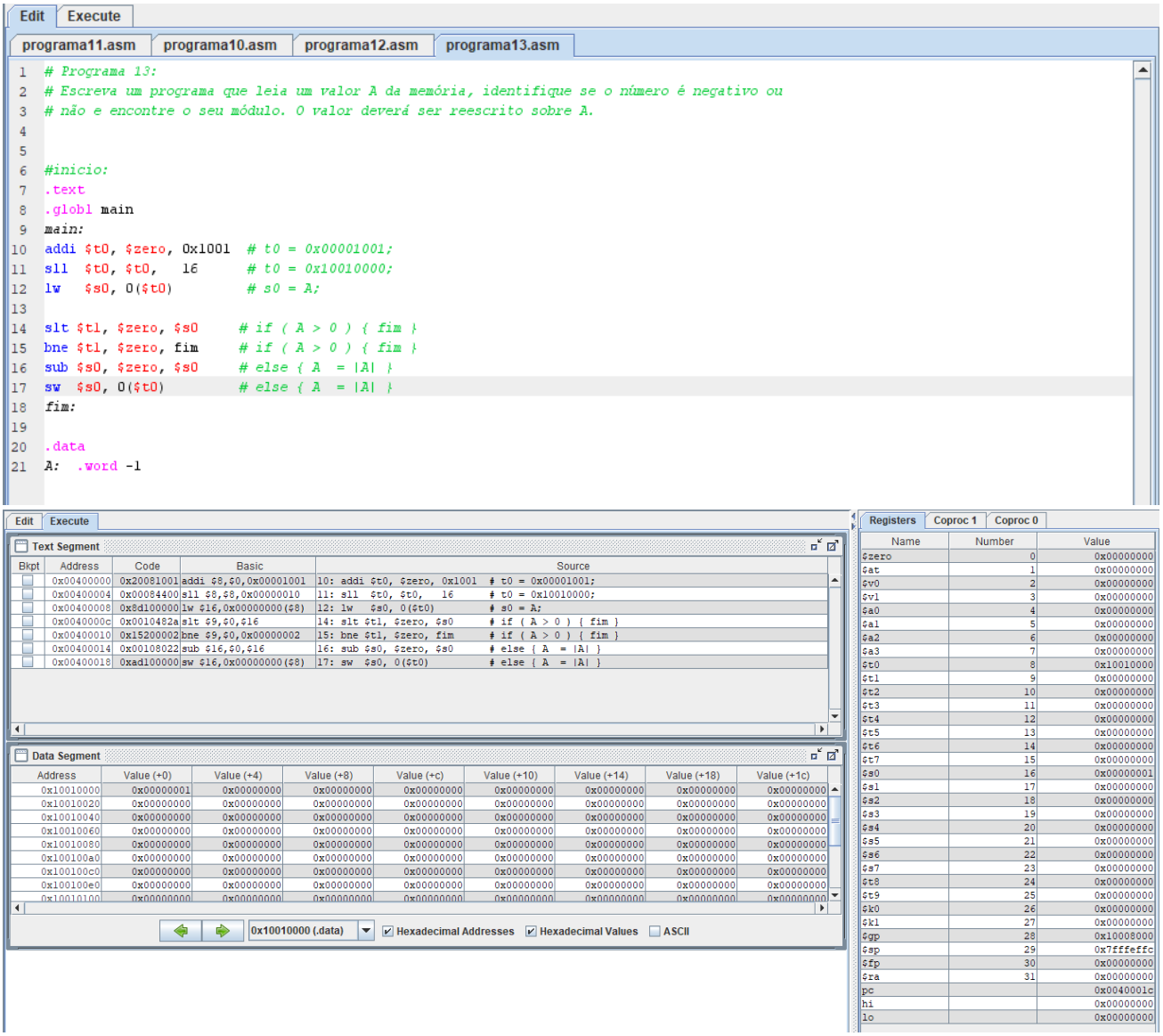


```
# Crie um programa que implemente a estrutura de dados acima, leia o valor de K, o multiplique por
# 2 e o reescreva no local correto conhecendo-se apenas o valor de x.


#inicio:
.text
.globl main
main:
addi $t0, $zero, 0x1001  # t0   = 0x00001001;
sll  $t0, $t0,   16      # t0   = 0x10010000;
addi $t1, $t0,   4       # t1   = 0x10010004;
addi $t2, $t0,   8       # t2   = 0x10010008;
addi $t3, $t0,   12      # t3   = 0x10010012;

sw   $t0, 4 ($t0)        # px   = 0x00001001;
sw   $t1, 8 ($t0)        # ppx  = 0x00001001;
sw   $t2, 12($t0)        # pppx = 0x00001001;

lw   $s0, 0 ($t3)        # s0 = t3 = k;
lw   $s1, 0 ($t2)        # s1 = t2 = k;
lw   $s2, 0 ($t1)        # s2 = t1 = k;
lw   $s3, 0 ($t0)        # s4 = t0 = k;

add  $t4, $s3, $s3       # t4 = 2 * k;

sw   $t4, 0($t1)         # x = 2 * k;

.data
x:    .word 10
px:   .word 0
ppx:  .word 0
pppx: .word 0
```

```
Text Segment
Bkpt  Address     Code        Basic                    Source
0x00400000  0x20081001  addi $8,$0,0x00001001   18: addi $t0, $zero, 0x1001  # t0   = 0x00001001;
0x00400004  0x00084400  sll $8,$8,0x00000010    19: sll  $t0, $t0,   16      # t0   = 0x10010000;
0x00400008  0x21090004  addi $9,$8,0x00000004   20: addi $t1, $t0,   4       # t1   = 0x10010004;
0x0040000c  0x210a0008  addi $10,$8,0x00000008  21: addi $t2, $t0,   8       # t2   = 0x10010008;
0x00400010  0x210b000c  addi $11,$8,0x0000000c  22: addi $t3, $t0,   12      # t3   = 0x10010012;
0x00400014  0xad080004  sw $8,0x00000004($8)    24: sw   $t0, 4 ($t0)        # px   = 0x00001001;
0x00400018  0xad090008  sw $9,0x00000008($8)    25: sw   $t1, 8 ($t0)        # ppx  = 0x00001001;
0x0040001c  0xad0a000c  sw $10,0x0000000c($8)   26: sw   $t2, 12($t0)        # pppx = 0x00001001;
0x00400020  0x8d700000  lw $16,0x00000000($11)  28: lw   $s0, 0 ($t3)        # s0 = t3 = k;
0x00400024  0x8d510000  lw $17,0x00000000($10)  29: lw   $s1, 0 ($t2)        # s1 = t2 = k;
0x00400028  0x8d320000  lw $18,0x00000000($9)   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
0x10010000 (.data)  Hexadecimal Addresses  Hexadecimal Values  ASCII

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   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  0x7fffeffc
$fp     30  0x00000000
$ra     31  0x00000000
pc          0x00400038
hi          0x00000000
lo          0x00000000
```

// programa 13:

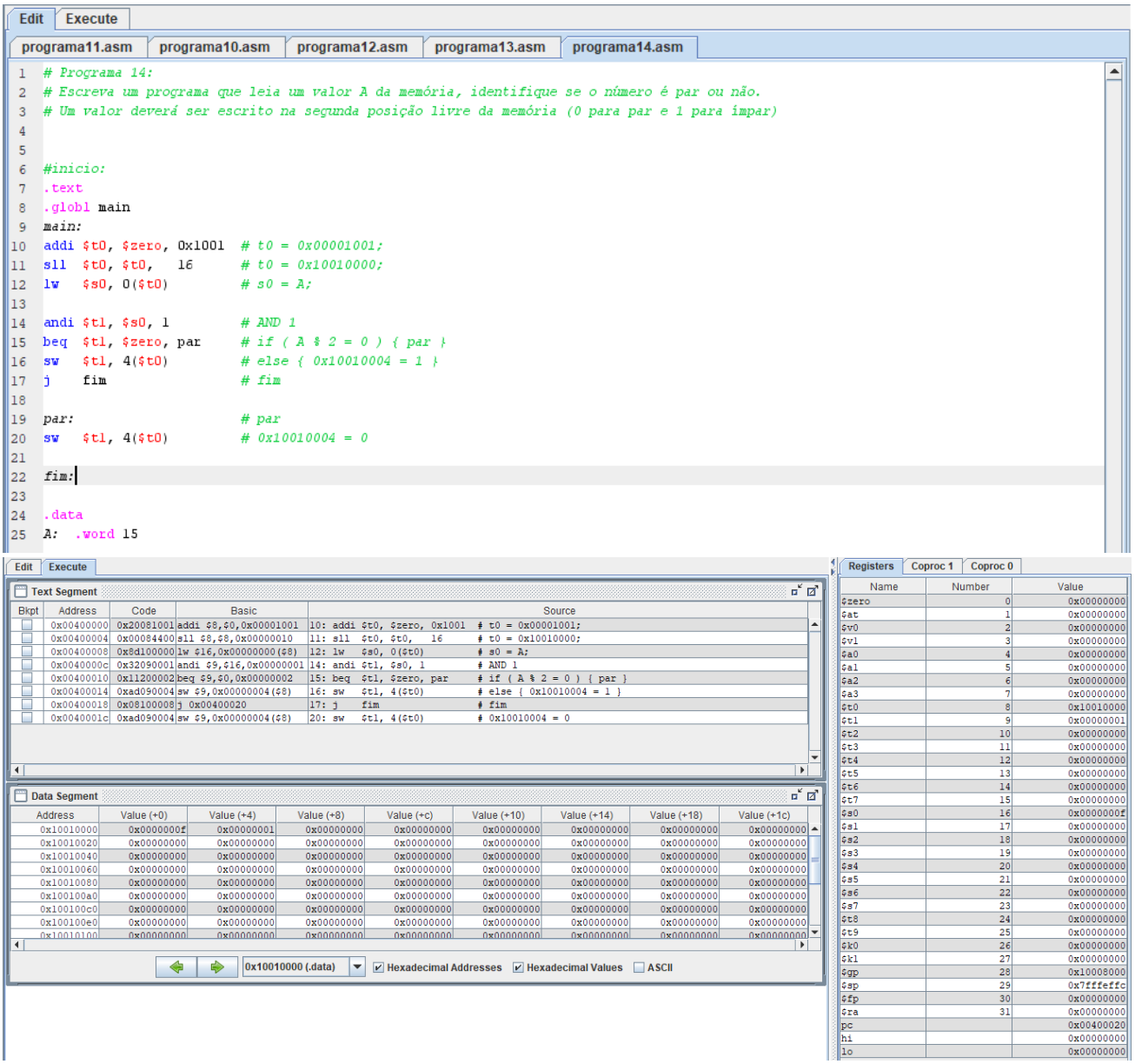
Escreva um programa que leia um valor A da memória, identifique se o número é negativo ou não e encontre o seu módulo. O valor deverá ser reescrito sobre A.



// 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)

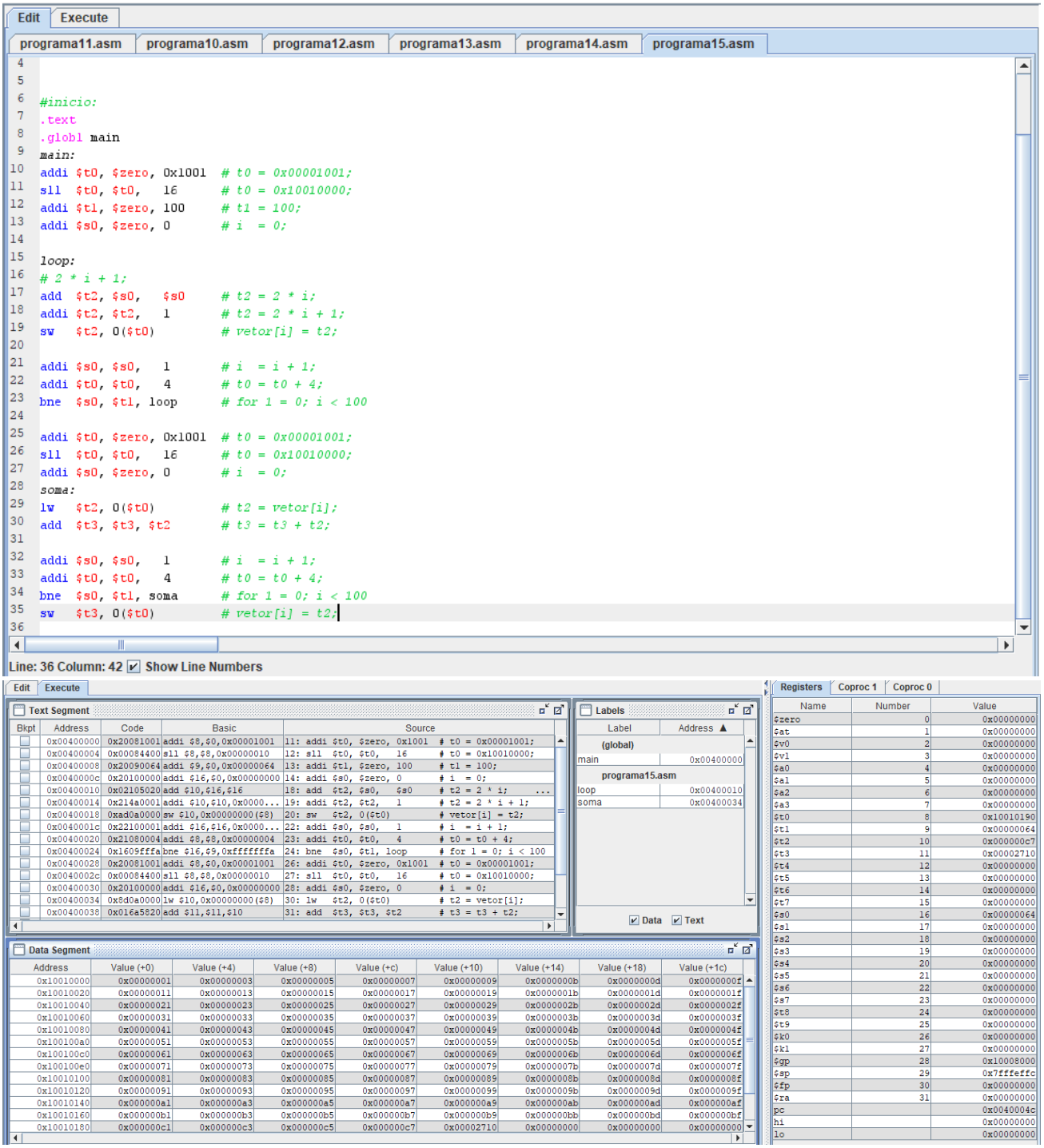


// programa 15:

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)



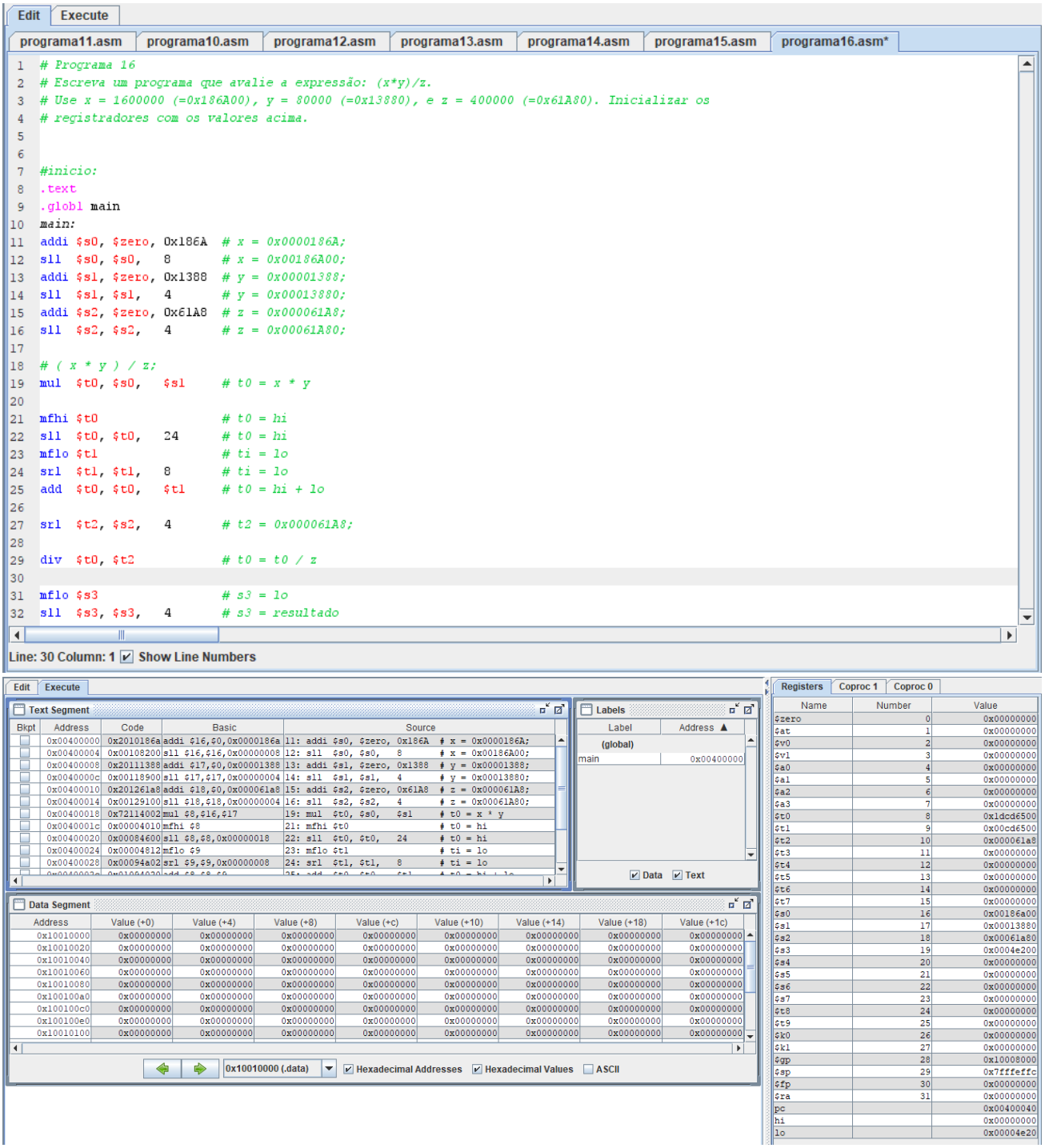
// programa 16

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.

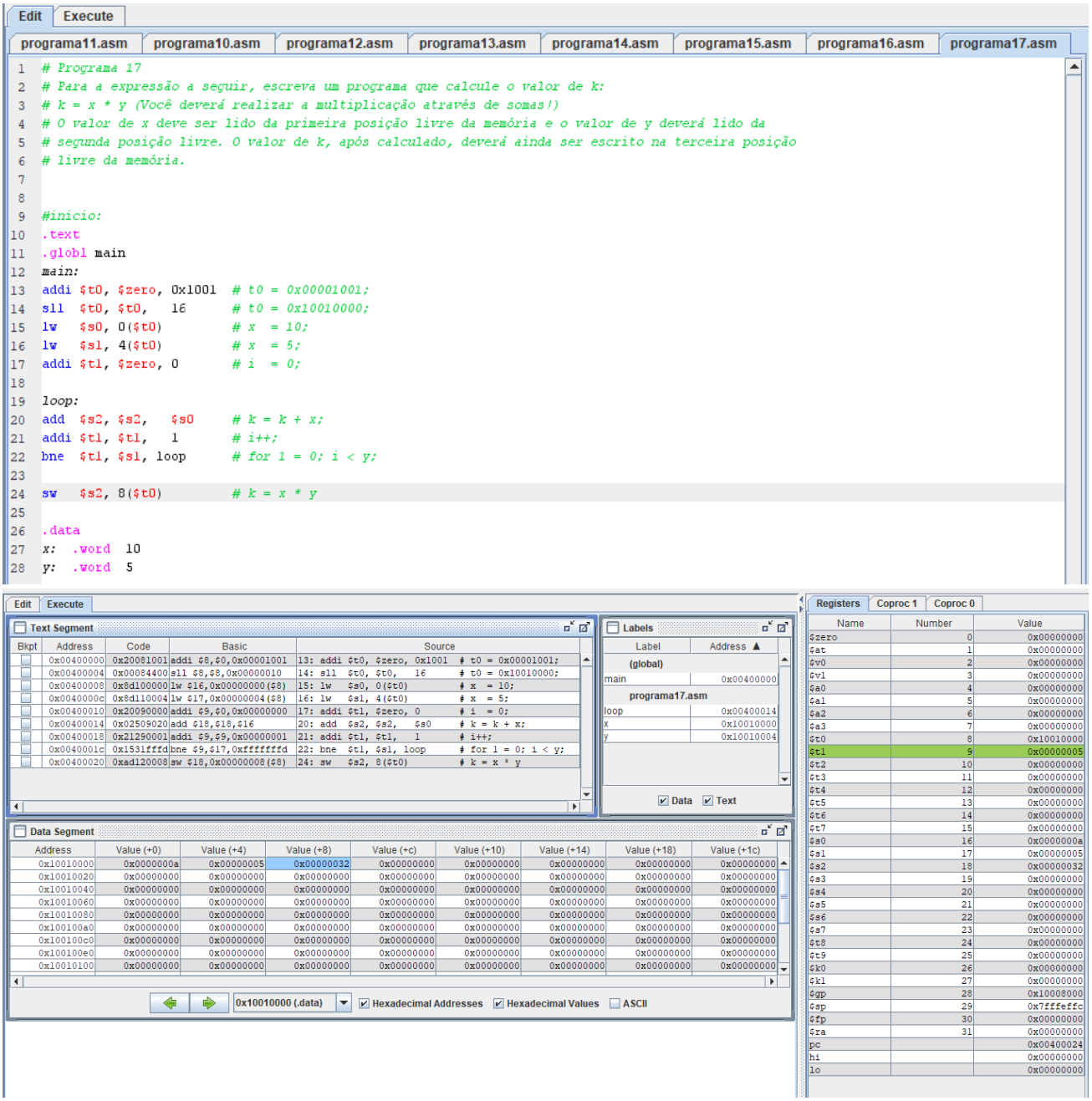


// programa 17

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.



// 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)

```
11  #inicio:
12  .text
13  .globl main
14  main:
15  addi $t0, $zero, 0x1001  # t0 = 0x00001001;
16  sll  $t0, $t0,   16      # t0 = 0x10010000;
17  lw   $s0, 0($t0)         # x  = 2;
18  lw   $s1, 4($t0)         # y  = 5;
19  addi $t1, $zero, 0       # i  = 0;
20  addi $s2, $zero, 1       # k  = 1;
21
22  loopi:
23  beq  $s1, $zero, fim     # if (y == 0) {x^0 = 1}
24
25
26  addi $t2, $zero, 0       # t2 = 0;
27  addi $t3, $zero, 0       # t3 = 0;
28  loopj:
29      beq  $t3, $s0, multf # if ( j == x ) { multf }
30      add  $t2, $t2, $s2   # t2 = k + k;
31      addi $t3, $t3, 1     # j ++;
32      j    loopj           # loopi;
33  multf:
34      add  $s2, $zero, $t2 # k = x^y;
35      addi $s1, $s1, -1    # y --;
36      j    loopi           # loopi;
37
38  fim:
39      sw   $s2, 8($t0)     # memoria;
40
41  .data
42  x:  .word  2
43  y:  .word  5
```

Edit | Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20081001	addi $8,$0,0x00001001	15: addi $t0, $zero, 0x1001 # t0 = 0x00001001;
	0x00400004	0x00084400	sll $8,$8,0x00000010	16: sll $t0, $t0, 16 # t0 = 0x10010000;
	0x00400008	0x8d100000	lw $16,0x00000000($8)	17: lw $s0, 0($t0) # x = 2;
	0x0040000c	0x8d110004	lw $17,0x00000004($8)	18: lw $s1, 4($t0) # y = 5;
	0x00400010	0x20090000	addi $9,$0,0x00000000	19: addi $t1, $zero, 0 # i = 0;
	0x00400014	0x20120001	addi $18,$0,0x00000001	20: addi $s2, $zero, 1 # k = 1;
	0x00400018	0x12200009	beq $17,$0,0x00000009	23: beq $s1, $zero, fim # if (y == 0) {x^0 = 1}
	0x0040001c	0x200a0000	addi $10,$0,0x00000000	26: addi $t2, $zero, 0 # t2 = 0;
	0x00400020	0x200b0000	addi $11,$0,0x00000000	27: addi $t3, $zero, 0 # t3 = 0;
	0x00400024	0x11700003	beq $11,$16,0x00000003	29: beq $t3, $s0, multf # if (j == x) { multf }
	0x00400028	0x01525020	add $10,$10,$18	30: add $t2, $t2, $s2 # t2 = k + k;

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000002	0x00000005	0x00000020	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 | 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	0x10010000
$t1	9	0x00000000
$t2	10	0x00000020
$t3	11	0x00000002
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000002
$s1	17	0x00000000
$s2	18	0x00000020
$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	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400044
hi		0x00000000
lo		0x00000000

Responda:

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).

```
#inicio:
.text
.globl main
main:
addi $t0, $zero, 0x1001  # t0 = 0x00001001;
sll  $t0, $t0,   16      # t0 = 0x10010000;
lw   $s0, 0($t0)         # s0  = x;
lw   $s1, 4($t0)         # s1  = y;
lw   $t4, 0($t0)         # t4  = x;
lw   $t5, 4($t0)         # t5  = y;
addi $t0, $zero, 0       # i = 0;

significativox:
beq  $t4, $zero, endx    # while x != 0;
srl  $t4, $t4, 1         # conta os bits significativos;
addi $t0, $t0, 1         # i++;
j significativox

endx:
addi $t1, $zero, 0       # j = 0;
significativoy:
beq  $t5, $zero, endy    # while y != 0;
srl  $t5, $t5, 1         # conta os bits significativos;
addi $t1, $t1, 1         # j++;
j significativoy

endy:
mult  $s0, $s1           # s0 * s1;

addi $t2, $zero, 32      # t2 = 32
slt  $t3, $t2, $t0
beq  $t2, $zero, menor   # Se $t3 == 0, x <= 32
slt  $t3, $t2, $t1
beq  $t2, $zero, menor   # Se $t3 == 0, x <= 32
mflo $s2                 # s2 = lo
mfhi $s3                 # s3 = hi
menor:
mflo $s2                 # s2 = lo

.data
x:  .word  100000
y:  .word  100000
```

Edit | Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20081001	addi $8,$0,4097	13: addi $t0, $zero, 0x1001 # t0 = 0x00001001;
	0x00400004	0x00084400	sll $8,$8,16	14: sll $t0, $t0, 16 # t0 = 0x10010000;
	0x00400008	0x8d190000	lw $16,0($8)	15: lw $s0, 0($t0) # s0 = x;
	0x0040000c	0x8d110004	lw $17,4($8)	16: lw $s1, 4($t0) # s1 = y;
	0x00400010	0x8d0c0000	lw $12,0($8)	17: lw $t4, 0($t0) # t4 = x;
	0x00400014	0x8d0d0004	lw $13,4($8)	18: lw $t5, 4($t0) # t5 = y;
	0x00400018	0x20080000	addi $8,$0,0	19: addi $t0, $zero, 0 # i = 0;
	0x0040001c	0x11800003	beq $12,$0,3	22: beq $t4, $zero, endx # while x != 0;
	0x00400020	0x000c6042	srl $12,$12,1	23: srl $t4, $t4, 1 # conta os bits significativos;
	0x00400024	0x21080001	addi $8,$8,1	24: addi $t0, $t0, 1 # i++;
	0x00400028	0x08100007	j 0x0040001c	25: j significativox
	0x0040002c	0x20090000	addi $9,$0,0	28: addi $t1, $zero, 0 # j = 0;
	0x00400030	0x11a00003	beq $13,$0,3	30: beq $t5, $zero, endy # while y != 0;
	0x00400034	0x000d6842	srl $13,$13,1	31: srl $t5, $t5, 1 # conta os bits significativos;
	0x00400038	0x21290001	addi $9,$9,1	32: addi $t1, $t1, 1 # j++;

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	100000	100000	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
0x10010100	0	0	0	0	0	0	0	0
0x10010120	0	0	0	0	0	0	0	0
0x10010140	0	0	0	0	0	0	0	0

Mars Messages | Run I/O

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	17
$t1	9	17
$t2	10	32
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	100000
$s1	17	100000
$s2	18	1410065408
$s3	19	2
$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		4194404
hi		2
lo		1410065408

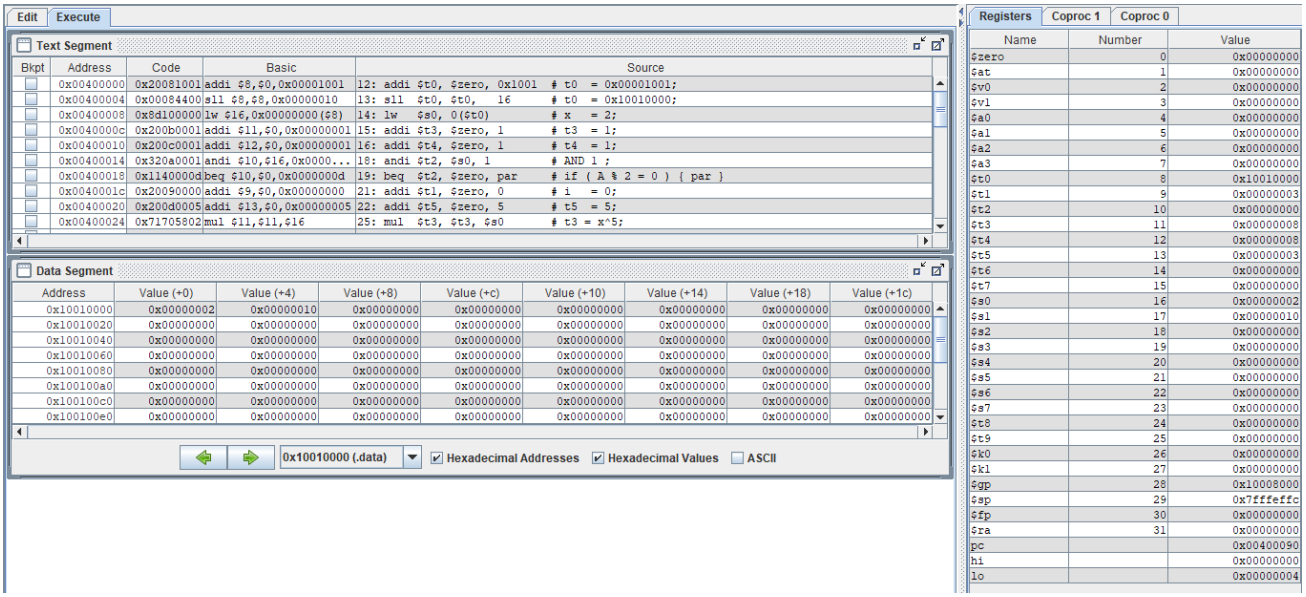
// programa 20

$y = x^4 + x^3 - 2x^2$ 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.

```
#inicio:
.text
.globl main
main:
addi $t0, $zero, 0x1001  # t0  = 0x00001001;
sll  $t0, $t0,   16      # t0  = 0x10010000;
lw   $s0, 0($t0)         # x   = 2;
addi $t3, $zero, 1       # t3  = 1;
addi $t4, $zero, 1       # t4  = 1;

andi $t2, $s0, 1         # AND 1 ;
beq  $t2, $zero, par     # if ( A % 2 = 0 ) { par }
impar:
addi $t1, $zero, 0       # i   = 0;
addi $t5, $zero, 5       # t5  = 5;
# x^5 - x^3 + 1;
loop_i1:
mul  $t3, $t3, $s0       # t3 = x^5;
addi $t1, $t1,   1       # i++;
bne  $t1, $t5,   loop_i1 # for 1 = 0; i < t5

addi $t5, $zero, 3       # t5  = 3;
addi $t1, $zero, 0       # i   = 0;
loop_i2:
mul  $t4, $t4, $s0       # t4 = x^3;
addi $t1, $t1,   1       # i++;
bne  $t1, $t5,   loop_i2 # for 1 = 0; i < t5

sub  $s1, $t3, $t4       # y = x^5 - x^3;
addi $s1, $s1, 1        # y = x^5 - x^3 + 1;
j fim                    # fim

par:
addi $t1, $zero, 0       # i   = 0;
addi $t5, $zero, 4       # t5  = 4;

# x^4 + x^3 - 2x^2;
loop_p1:
mul  $t3, $t3, $s0       # t3 = x^4;
addi $t1, $t1,   1       # i++;
bne  $t1, $t5,   loop_p1 # for 1 = 0; i < t5

addi $t5, $zero, 3       # t5  = 3;
addi $t1, $zero, 0       # i   = 0;
loop_p2:
mul  $t4, $t4, $s0       # t4 = x^3;
addi $t1, $t1,   1       # i++;
bne  $t1, $t5,   loop_p2 # for 1 = 0; i < t5

add  $s1, $t3, $t4       # y = x^4 + x^3;
mul  $t3, $s0, $s0       # x^2;
add  $t3, $t3, $t3       # 2x^2;
sub  $s1, $s1, $t3       # y = x^4 + x^3 - 2x^2;
j fim                    # fim

fim:
sw   $s1, 4($t0)         # memoria

.data
x:  .word  2
```



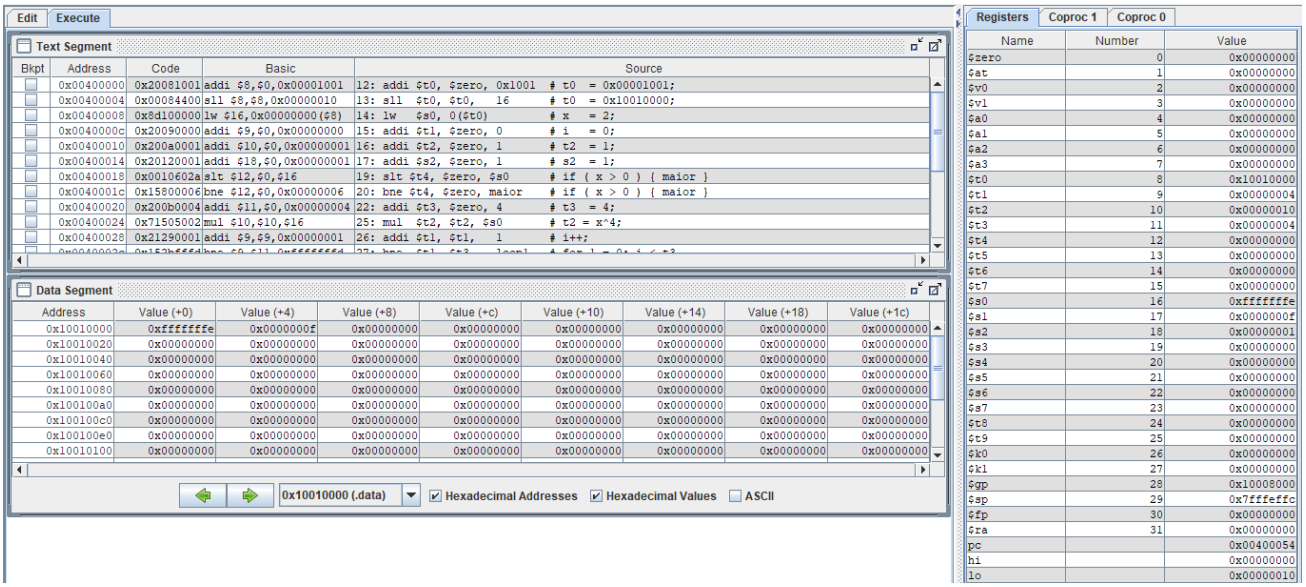
// programa 21

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.

```
#inicio:
.text
.globl main
main:
addi $t0, $zero, 0x1001  # t0  = 0x00001001;
sll  $t0, $t0,   16      # t0  = 0x10010000;
lw   $s0, 0($t0)         # x   = 2;
addi $t1, $zero, 0       # i   = 0;
addi $t2, $zero, 1       # t2  = 1;
addi $s2, $zero, 1       # s2  = 1;

slt $t4, $zero, $s0      # if ( x > 0 ) { maior }
bne $t4, $zero, maior    # if ( x > 0 ) { maior }
menor:
addi $t3, $zero, 4       # t3  = 4;
loop1:
# x^4 - 1;
mul  $t2, $t2, $s0       # t2 = x^4;
addi $t1, $t1,   1       # i++;
bne  $t1, $t3,   loop1   # for 1 = 0; i < t3

sub  $s1, $t2, $s2       # y = x^4 - 1;
j fim

maior:
addi $t3, $zero, 3       # t3  = 3;
loop2:
# x^3 + 1;
mul  $t2, $t2, $s0       # t2 = x^3;
addi $t1, $t1,   1       # i++;
bne  $t1, $t3,   loop2   # for 1 = 0; i < t3

addi $s1, $t2, 1         # y = x^3 + 1;
j fim

fim:
sw   $s1, 4($t0)         # memoria

.data
x:  .word  -2
```



Todos os programas em:

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