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.

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.

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

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

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

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

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.

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

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

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.

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.

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

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

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

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.

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.

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

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

A. Não.
B. Sim.

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

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.

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

A. Não. Diferentes de instruções de máquina possuem campos diferentes.

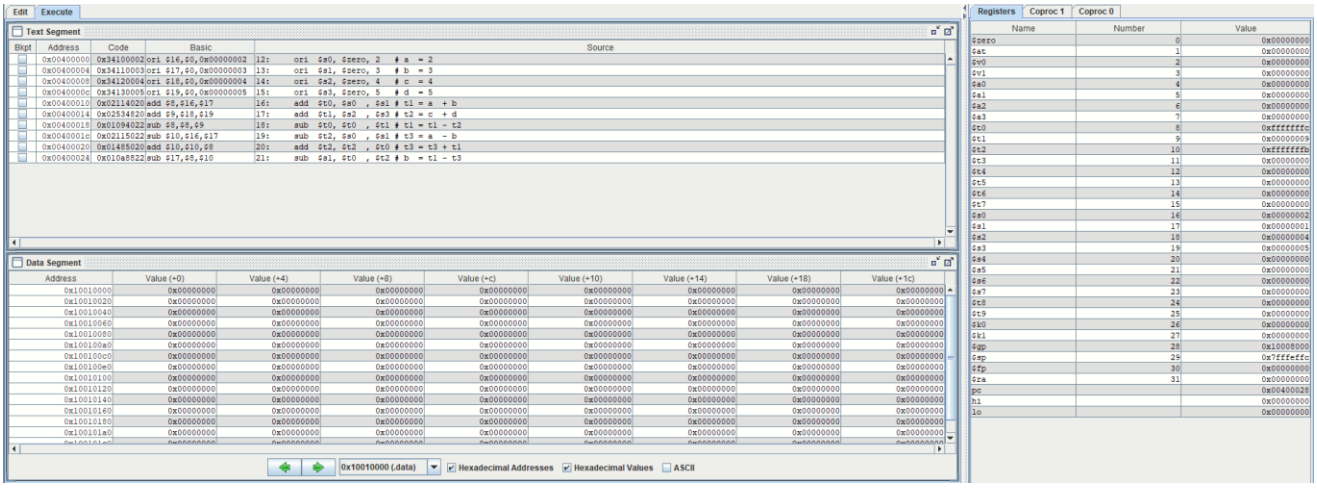
B. Não. Cada instrução de máquina é completamente diferente de qualquer outra.

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.

Programa 01:

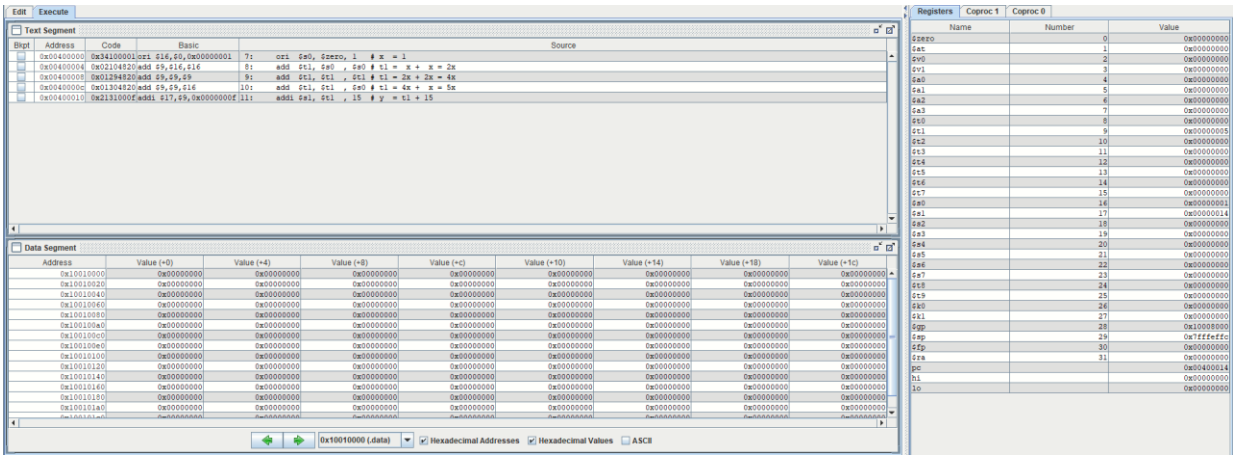
```
1  # a =2;
2  # b =3;
3  # c =4;
4  # d =5;
5  # x = (a+b) - (c+d);
6  # y = a - b + x;
7  # b = x - y;
8  # add, addi, sub, lógicas
9
10 .text
11 main:
12     ori  $s0, $zero, 2   # a  = 2
13     ori  $s1, $zero, 3   # b  = 3
14     ori  $s2, $zero, 4   # c  = 4
15     ori  $s3, $zero, 5   # d  = 5
16     add  $t0, $s0  , $s1 # t1 = a  + b
17     add  $t1, $s2  , $s3 # t2 = c  + d
18     sub  $t0, $t0  , $t1 # t1 = t1 - t2
19     sub  $t2, $s0  , $s1 # t3 = a  - b
20     add  $t2, $t2  , $t0 # t3 = t3 + t1
21     sub  $s1, $t0  , $t2 # b  = t1 - t3
22
23 # EOF
```



Programa 02:

P02.asm

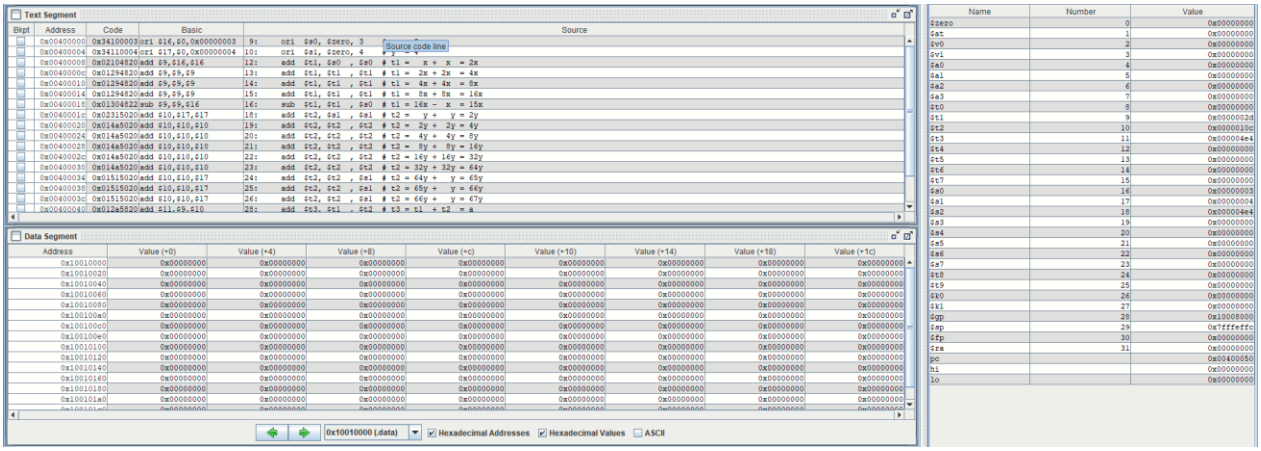
```
1   # x = 1;
2   # y = 5*x + 15;
3   # add, addi, sub, lógicas
4
5   .text
6   main:
7       ori  $s0, $zero, 1    # x  = 1
8       add  $t1, $s0  , $s0 # t1 =  x +  x = 2x
9       add  $t1, $t1  , $t1 # t1 = 2x + 2x = 4x
10      add  $t1, $t1  , $s0 # t1 = 4x +  x = 5x
11      addi $s1, $t1  , 15   # y  = t1 + 15
12
13  # EOF
```



Programa 03:

P03.asm

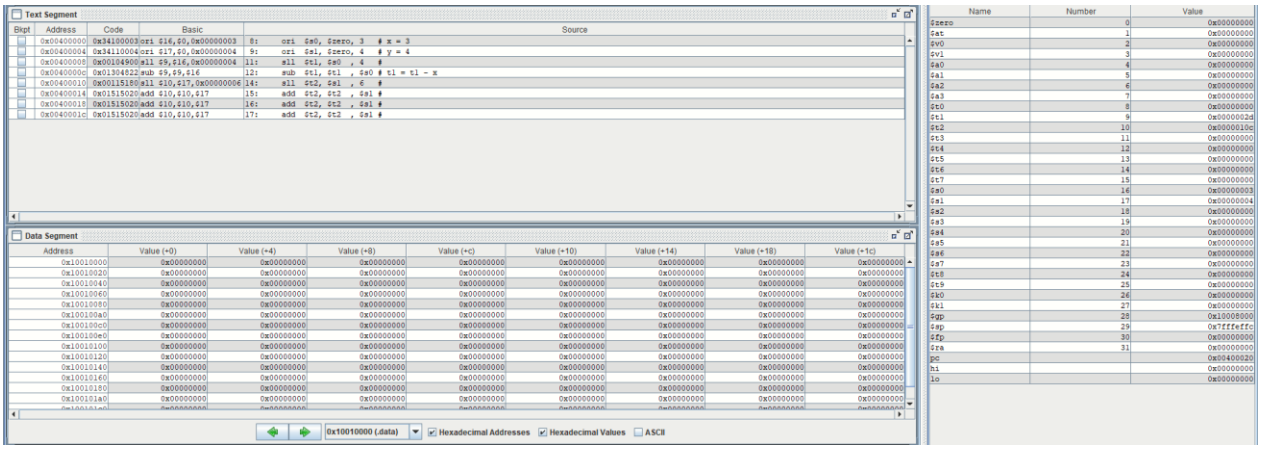
```
# x = 3;
# y = 4 ;
# z = ( 15*x + 67*y)*4
# add, addi, sub, lógicas

.text
main:

    ori  $s0, $zero, 3    # x  = 3
    ori  $s1, $zero, 4    # y  = 4
    # 15 * x
    add  $t1, $s0  , $s0  # t1 =   x +  x  = 2x
    add  $t1, $t1  , $t1  # t1 =  2x + 2x  = 4x
    add  $t1, $t1  , $t1  # t1 =  4x + 4x  = 8x
    add  $t1, $t1  , $t1  # t1 =  8x + 8x  = 16x
    sub  $t1, $t1  , $s0  # t1 = 16x -  x  = 15x
    # 67 * y
    add  $t2, $s1  , $s1  # t2 =   y +   y = 2y
    add  $t2, $t2  , $t2  # t2 =  2y +  2y = 4y
    add  $t2, $t2  , $t2  # t2 =  4y +  4y = 8y
    add  $t2, $t2  , $t2  # t2 =  8y +  8y = 16y
    add  $t2, $t2  , $t2  # t2 = 16y + 16y = 32y
    add  $t2, $t2  , $t2  # t2 = 32y + 32y = 64y
    add  $t2, $t2  , $s1  # t2 = 64y +   y = 65y
    add  $t2, $t2  , $s1  # t2 = 65y +   y = 66y
    add  $t2, $t2  , $s1  # t2 = 66y +   y = 67y
    # 15x + 67y
    add  $t3, $t1  , $t2  # t3 = t1  + t2  = a
    # ( 15x + 67y ) * 4
    add  $t3, $t3  , $t3  # t3 = s   + a   = 2a
    add  $t3, $t3  , $t3  # t3 = 2a  + 2a  = 4a
    add  $s2, $zero, $t3  # z  = t3

# EOF
```



Programa 04:

P04.asm

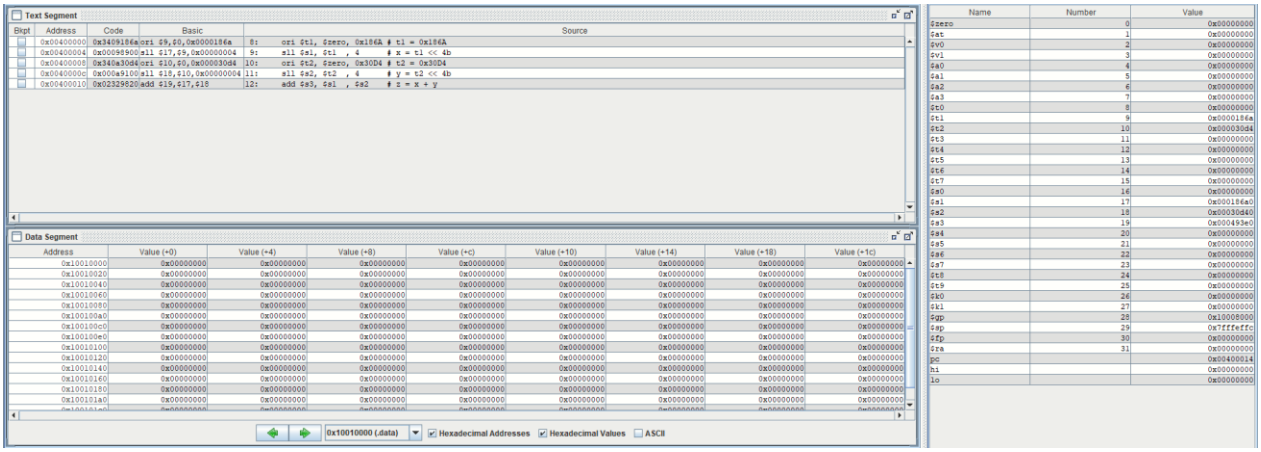
```
1   # x = 3;
2   # y = 4 ;
3   # z = ( 15*x + 67*y)*4
4   # sll, srl, sra
5
6   .text
7   main:
8       ori   $s0, $zero, 3    # x = 3
9       ori   $s1, $zero, 4    # y = 4
10      # 15*x
11      sll   $t1, $s0   , 4    #
12      sub   $t1, $t1   , $s0 # t1 = t1 - x
13      # 67*z
14      sll   $t2, $s1   , 6    #
15      add   $t2, $t2   , $s1 #
16      add   $t2, $t2   , $s1 #
17      add   $t2, $t2   , $s1 #
18
19  # EOF
```



Programa 05:

P05.asm

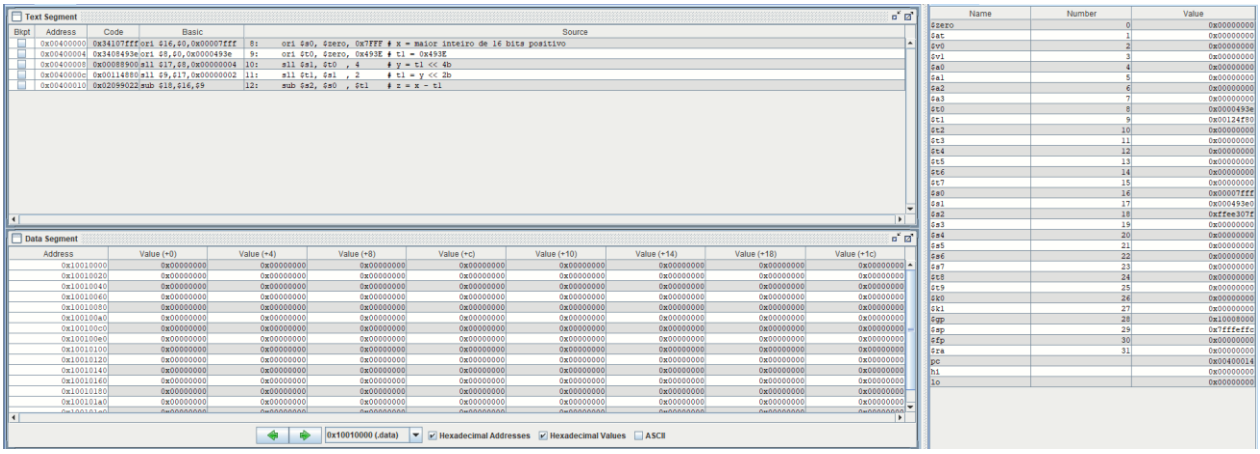
```
1   # x = 100000 = 0x186A0;
2   # y = 200000 = 0x30D40;
3   # z = x + y;
4   # sll, srl e sra
5
6   .text
7   main:
8       ori $t1, $zero, 0x186A # t1 = 0x186A
9       sll $s1, $t1  , 4      # x = t1 << 4b
10      ori $t2, $zero, 0x30D4 # t2 = 0x30D4
11      sll $s2, $t2  , 4      # y = t2 << 4b
12      add $s3, $s1  , $s2    # z = x + y
13
14  # EOF
```



Programa 06:

P06.asm

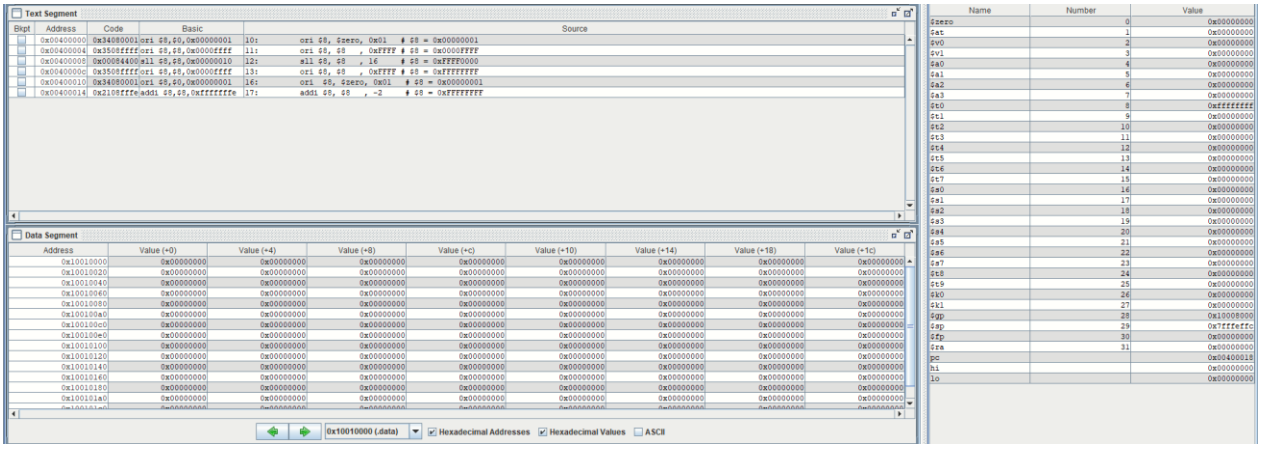
```
1   # x = o maior inteiro possível;
2   # y = 300000 = 493e0;
3   # z = x - 4y
4   # sll, srl e sra
5
6   .text
7   main:
8       ori $s0, $zero, 0x7FFF # x = maior inteiro de 16 bits positivo
9       ori $t0, $zero, 0x493E # t1 = 0x493E
10      sll $s1, $t0  , 4      # y = t1 << 4b
11      sll $t1, $s1  , 2      # t1 = y << 2b
12      sub $s2, $s0  , $t1    # z = x - t1
13
14  # EOF
```



Programa 07:

P07.asm

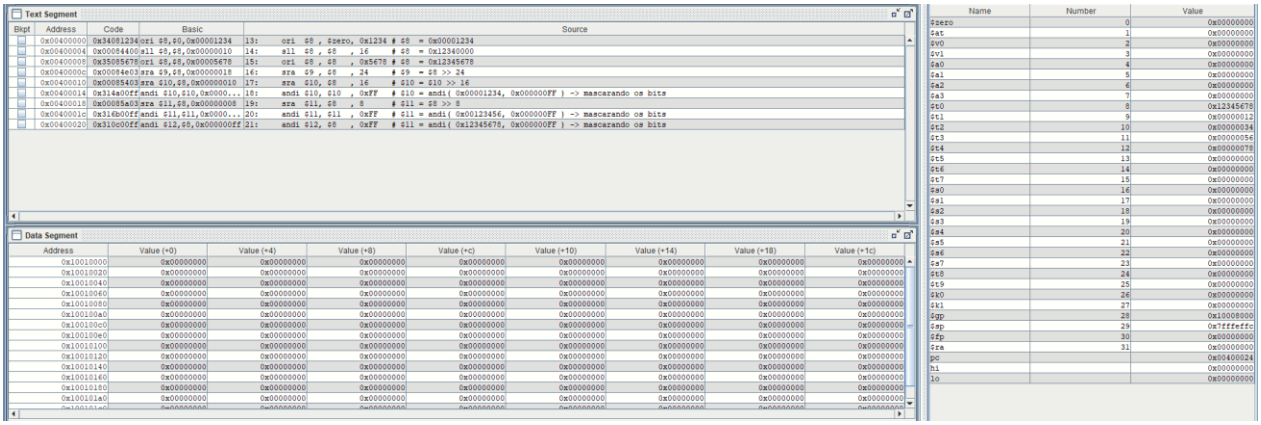
```
# Considere a seguinte instrução iniciando um programa:
#     ori $8, $0, 0x01
# Usando apenas instruções reg-reg lógicas e/ou instruções de deslocamento (sll, srl, sra), continuar o
# programa de forma que ao final, tenhamos o seguinte conteúdo no registrador $8:
#     $8 = 0xFFFFFFFF

.text
main:
    solucao_1:
        ori $8, $zero, 0x01    # $8 = 0x00000001
        ori $8, $8    , 0xFFFF # $8 = 0x0000FFFF
        sll $8, $8    , 16     # $8 = 0xFFFF0000
        ori $8, $8    , 0xFFFF # $8 = 0xFFFFFFFF

    solucao_2:
        ori  $8, $zero, 0x01    # $8 = 0x00000001
        addi $8, $8    , -2     # $8 = 0xFFFFFFFF

# EOF
```



Programa 08:

P08.asm

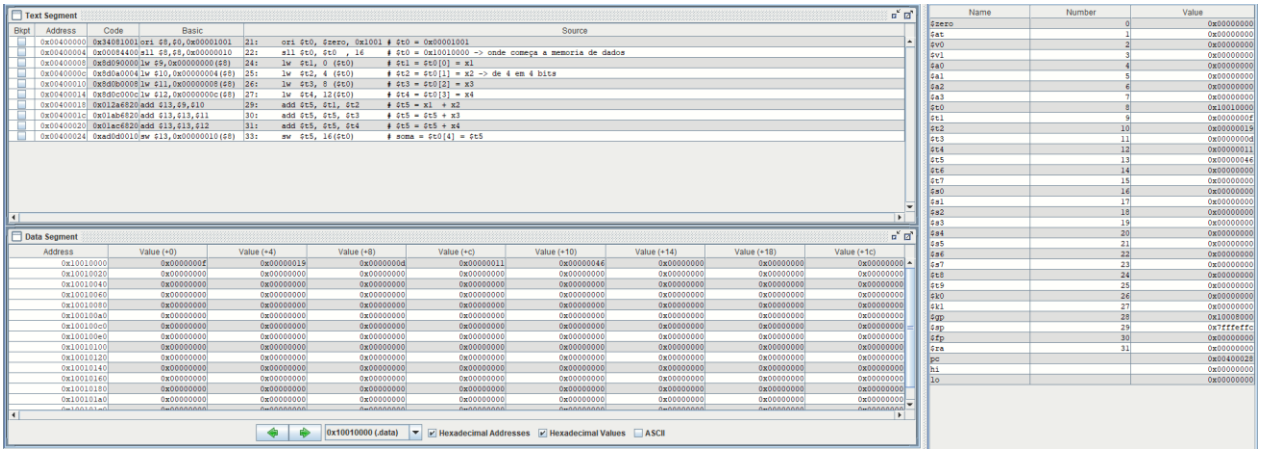
```
# Inicialmente escreva um programa que faça:
#     $8 = 0x12345678.
# A partir do registrador $8 acima, usando apenas instruções lógicas (or, ori, and, andi, xor, xori)
# e instruções de deslocamento (sll, srl e sra), você deverá obter os seguintes valores nos respectivos
# registradores:
#     $9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

.text
main:
    ori  $8 , $zero, 0x1234 # $8  = 0x00001234
    sll  $8 , $8   , 16     # $8  = 0x12340000
    ori  $8 , $8   , 0x5678 # $8  = 0x12345678
    sra  $9 , $8   , 24     # $9  = $8 >> 24
    sra  $10, $8   , 16     # $10 = $10 >> 16
    andi $10, $10  , 0xFF   # $10 = andi( 0x00001234, 0x000000FF ) -> mascarando os bits
    sra  $11, $8   , 8      # $11 = $8 >> 8
    andi $11, $11  , 0xFF   # $11 = andi( 0x00123456, 0x000000FF ) -> mascarando os bits
    andi $12, $8   , 0xFF   # $11 = andi( 0x12345678, 0x000000FF ) -> mascarando os bits

# EOF
```



Programa 09:

P09.asm

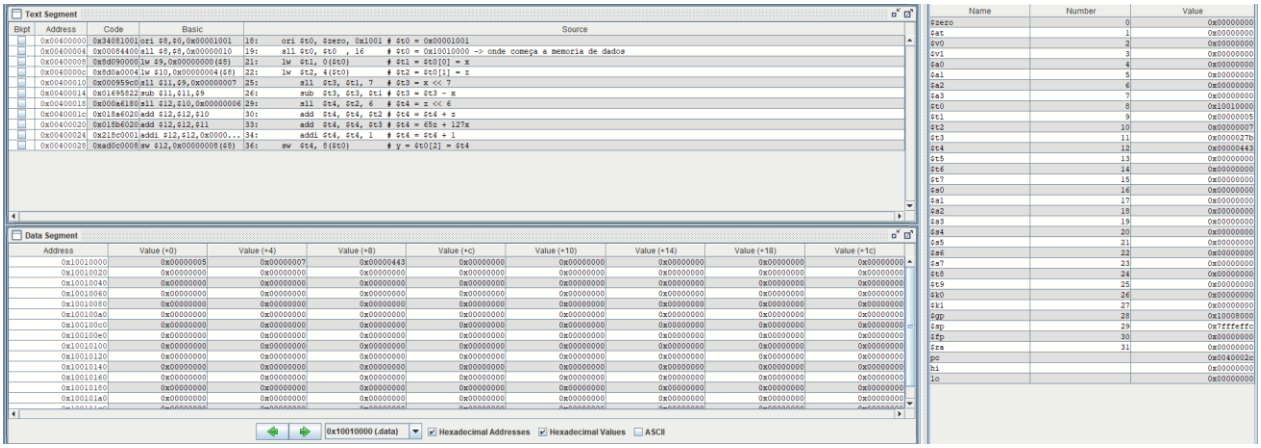
```
7  # x4: .word 17
8  # soma: .word -1
9  # Escrever um programa que leia todos os números, calcule e substitua o valor da variável soma por este valor.
10 # lw, sw
11
12 .data
13     x1:    .word 15
14     x2:    .word 25
15     x3:    .word 13
16     x4:    .word 17
17     soma:  .word -1
18
19 .text
20 main:
21     ori $t0, $zero, 0x1001 # $t0 = 0x00001001
22     sll $t0, $t0  , 16     # $t0 = 0x10010000 -> onde começa a memoria de dados
23
24     lw  $t1, 0 ($t0)       # $t1 = $t0[0] = x1
25     lw  $t2, 4 ($t0)       # $t2 = $t0[1] = x2 -> de 4 em 4 bits
26     lw  $t3, 8 ($t0)       # $t3 = $t0[2] = x3
27     lw  $t4, 12($t0)       # $t4 = $t0[3] = x4
28
29     add $t5, $t1, $t2      # $t5 = x1  + x2
30     add $t5, $t5, $t3      # $t5 = $t5 + x3
31     add $t5, $t5, $t4      # $t5 = $t5 + x4
32
33     sw  $t5, 16($t0)       # soma = $t0[4] = $t5
34
35 # EOF
```



Programa 10:

P10.asm

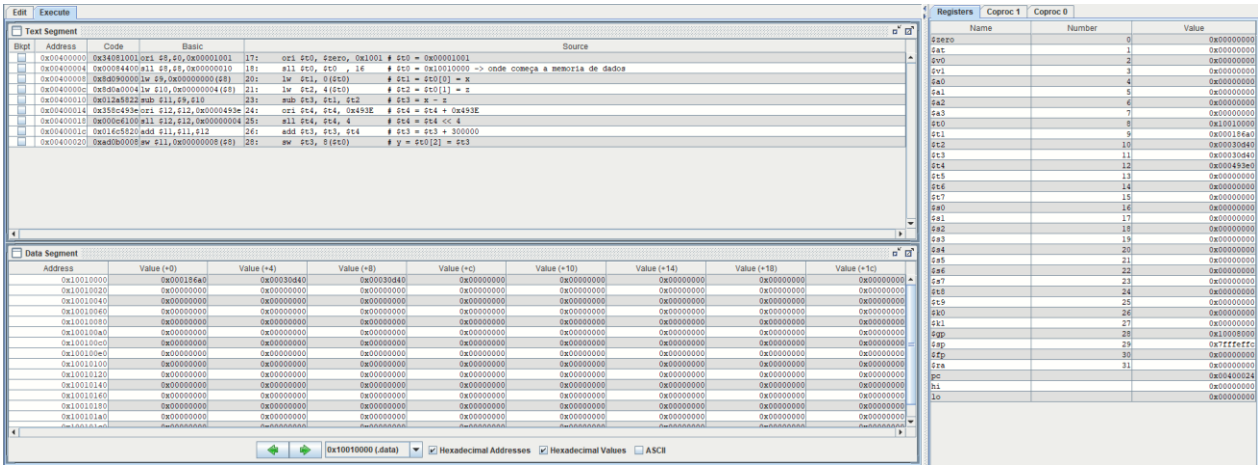
```
10
11  .data
12      x: .word 5
13      z: .word 7
14      y: .word 0
15
16  .text
17  main:
18      ori $t0, $zero, 0x1001 # $t0 = 0x00001001
19      sll $t0, $t0  , 16     # $t0 = 0x10010000 -> onde começa a memoria de dados
20
21      lw  $t1, 0($t0)        # $t1 = $t0[0] = x
22      lw  $t2, 4($t0)        # $t2 = $t0[1] = z
23
24      calcular_127x:
25          sll  $t3, $t1, 7   # $t3 = x << 7
26          sub  $t3, $t3, $t1 # $t3 = $t3 - x
27
28      calcular_65z:
29          sll  $t4, $t2, 6   # $t4 = z << 6
30          add  $t4, $t4, $t2 # $t4 = $t4 + z
31
32      calcular_exp:
33          add  $t4, $t4, $t3 # $t4 = 65z + 127x
34          addi $t4, $t4, 1   # $t4 = $t4 + 1
35
36      sw  $t4, 8($t0)        # y = $t0[2] = $t4
37
38  # EOF
```



Programa 11:

P11.asm

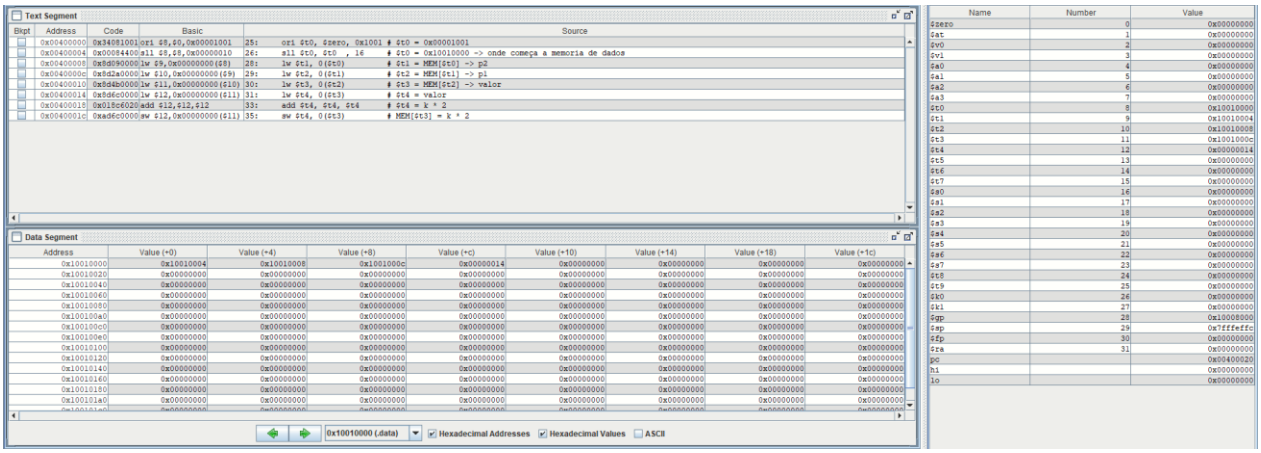
```
# Considere o seguinte programa:
#     y = x – z + 300000
# Faça um programa que calcule o valor de y conhecendo os valores de x e z. Os valores de x e z estão armazenados na
# memória e, na posição imediatamente a seguir, o valor de y deverá ser escrito, ou seja:
#     .data
#     x: .word 100000
#     z: .word 200000
#     y: .word 0 # esse valor deverá ser sobrescrito após a execução do programa.

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

.text
main:
    ori $t0, $zero, 0x1001 # $t0 = 0x00001001
    sll $t0, $t0  , 16     # $t0 = 0x10010000 -> onde começa a memoria de dados

    lw  $t1, 0($t0)        # $t1 = $t0[0] = x
    lw  $t2, 4($t0)        # $t2 = $t0[1] = z

    sub $t3, $t1, $t2      # $t3 = x - z
    ori $t4, $t4, 0x493E   # $t4 = $t4 + 0x493E
    sll $t4, $t4, 4        # $t4 = $t4 << 4
    add $t3, $t3, $t4      # $t3 = $t3 + 300000

    sw  $t3, 8($t0)        # y = $t0[2] = $t3

# EOF
```



Programa 12:

P12.asm

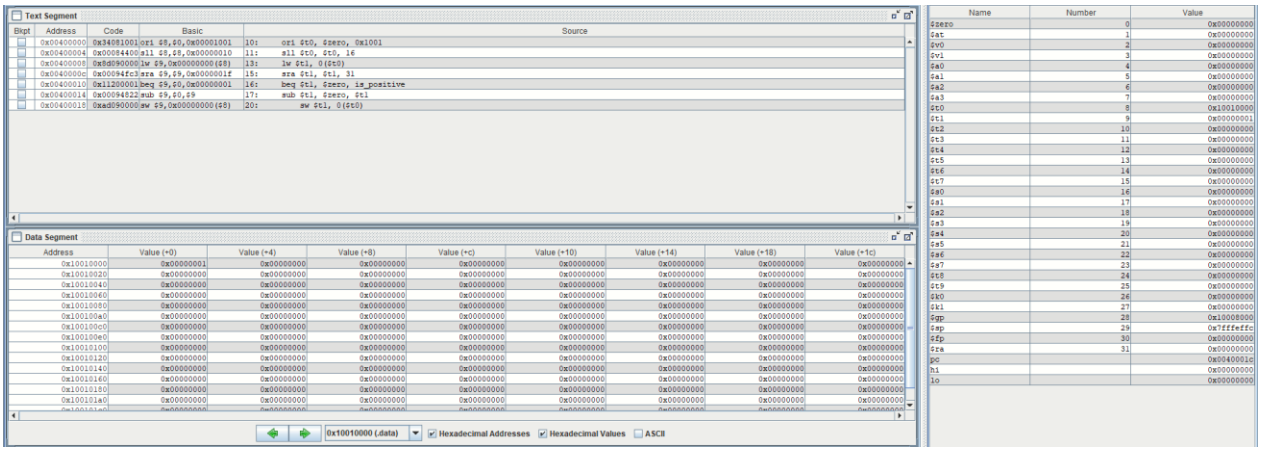
```
# Crie um programa que implemente a estrutura de dados acima, leia o valor de K, o multiplique por 2 e o reescreva no
# local correto conhecendo-se apenas o valor de x.

# *x     = &p2;
# *p2    = &p1;
# *p1    = &valor;
# *valor = valor;

.data
    x:      .word p2          # Terceiro ponteiro que aponta para 'p2' (***x)
    p2:     .word p1          # Segundo ponteiro que aponta para 'p1'
    p1:     .word valor       # Primeiro ponteiro que aponta para 'valor'
    valor: .word 10           # Inteiro inicial armazenado

.text
main:
    ori $t0, $zero, 0x1001 # $t0 = 0x00001001
    sll $t0, $t0  , 16     # $t0 = 0x10010000 -> onde começa a memoria de dados

    lw $t1, 0($t0)         # $t1 = MEM[$t0] -> p2
    lw $t2, 0($t1)         # $t2 = MEM[$t1] -> p1
    lw $t3, 0($t2)         # $t3 = MEM[$t2] -> valor
    lw $t4, 0($t3)         # $t4 = valor

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

    sw $t4, 0($t3)         # MEM[$t3] = k * 2

# EOF
```



Programa 13:

P13.asm

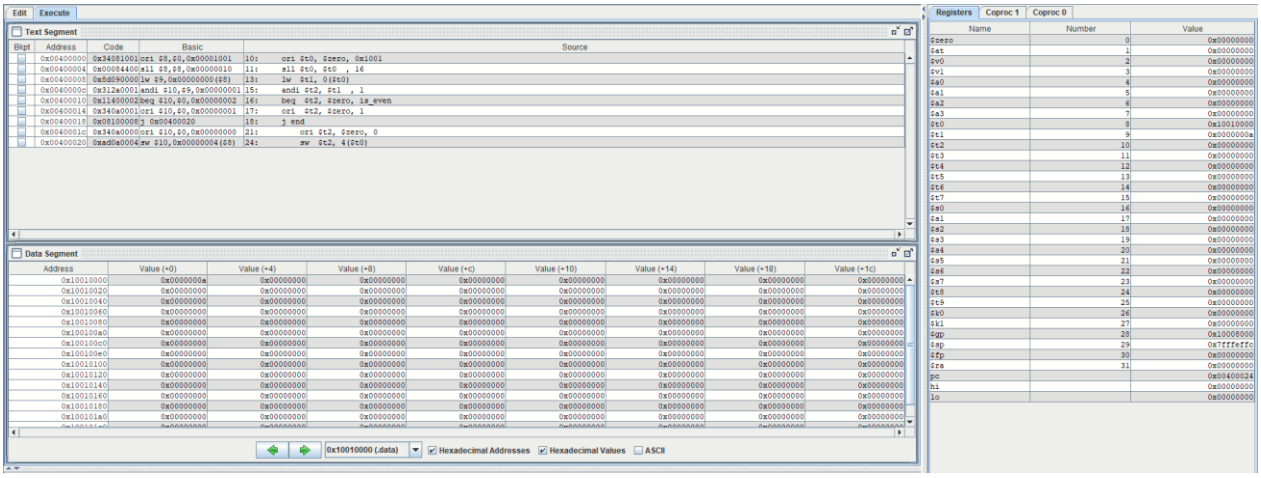
```
1  # Escreva um programa que leia um valor A da memória, identifique se o número é negativo ou não e encontre o seu módulo.
2  # O valor deverá ser reescrito sobre A.
3
4  .data
5      a: .word -10
6
7  .text
8  .globl main
9  main:
10     ori $t0, $zero, 0x1001
11     sll $t0, $t0, 16
12
13     lw $t1, 0($t0)
14
15     sra $t1, $t1, 31
16     beq $t1, $zero, is_positive
17     sub $t1, $zero, $t1
18
19     is_positive:
20         sw $t1, 0($t0)
21
22 # EOF
```



Programa 14:

P14.asm

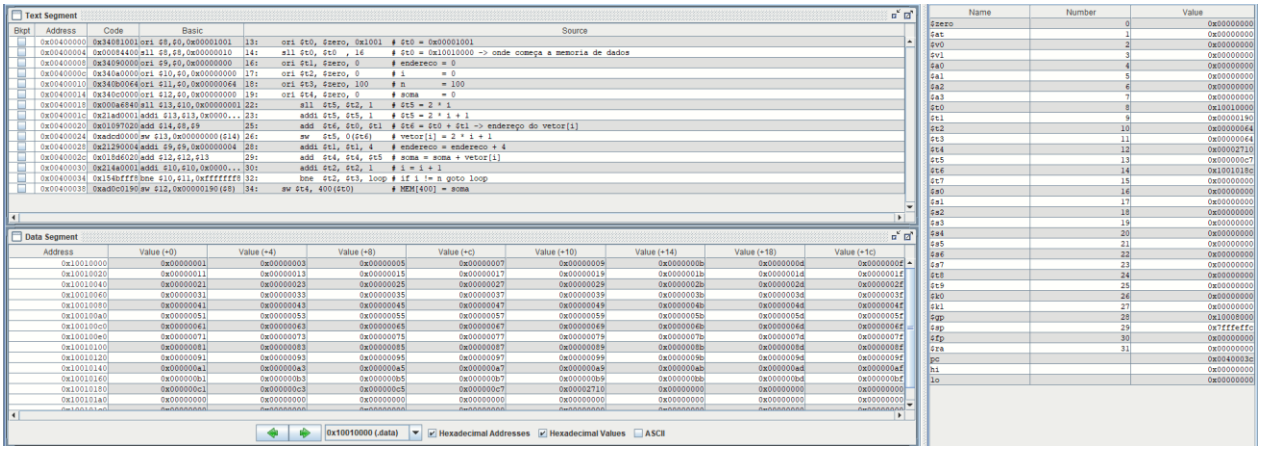
```
1  # Escreva um programa que leia um valor A da memória, identifique se o número é par ou não.
2  # Um valor deverá ser escrito na segunda posição livre da memória (0 para par e 1 para impar).
3
4  .data
5      a: .word 10
6
7  .text
8  .globl main
9  main:
10     ori $t0, $zero, 0x1001
11     sll $t0, $t0  , 16
12
13     lw  $t1, 0($t0)
14
15     andi $t2, $t1  , 1
16     beq  $t2, $zero, is_even
17     ori  $t2, $zero, 1
18     j end
19
20     is_even:
21         ori $t2, $zero, 0
22
23     end:
24         sw  $t2, 4($t0)
25
26 # EOF
```



Programa 15:

P15.asm

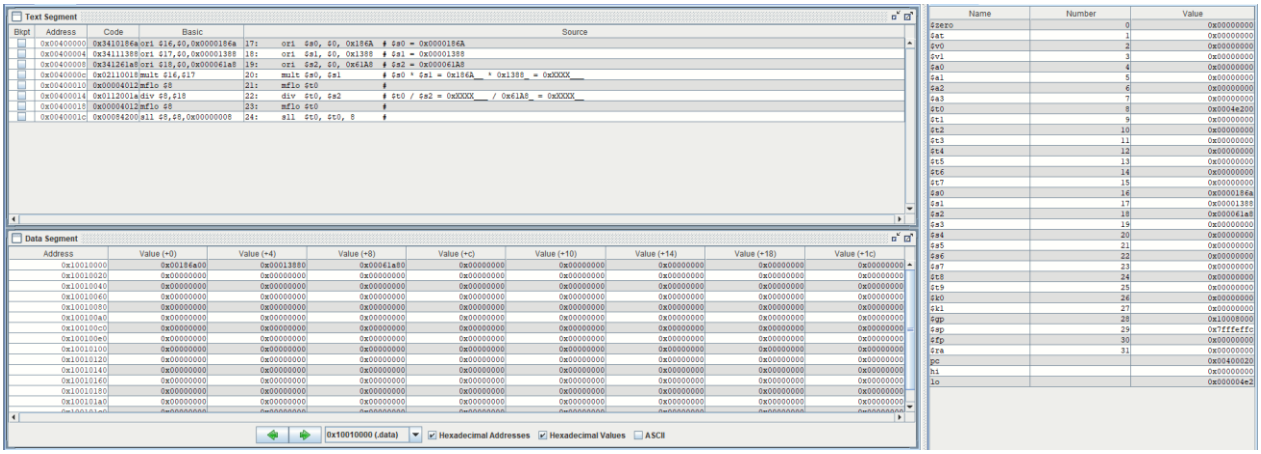
```
6   .data
7       vetor: .space 400
8       soma:  .word  0
9
10  .text
11  .globl main
12  main:
13      ori $t0, $zero, 0x1001  # $t0 = 0x00001001
14      sll $t0, $t0  , 16      # $t0 = 0x10010000 -> onde começa a memoria de dados
15
16      ori $t1, $zero, 0       # endereco = 0
17      ori $t2, $zero, 0       # i        = 0
18      ori $t3, $zero, 100     # n        = 100
19      ori $t4, $zero, 0       # soma     = 0
20
21      loop:
22          sll  $t5, $t2, 1    # $t5 = 2 * i
23          addi $t5, $t5, 1    # $t5 = 2 * i + 1
24
25          add  $t6, $t0, $t1  # $t6 = $t0 + $t1 -> endereço do vetor[i]
26          sw   $t5, 0($t6)    # vetor[i] = 2 * i + 1
27
28          addi $t1, $t1, 4    # endereco = endereco + 4
29          add  $t4, $t4, $t5  # soma = soma + vetor[i]
30          addi $t2, $t2, 1    # i = i + 1
31
32          bne  $t2, $t3, loop # if i != n goto loop
33
34      sw $t4, 400($t0)        # MEM[400] = soma
35
```



Programa 16:

P16.asm

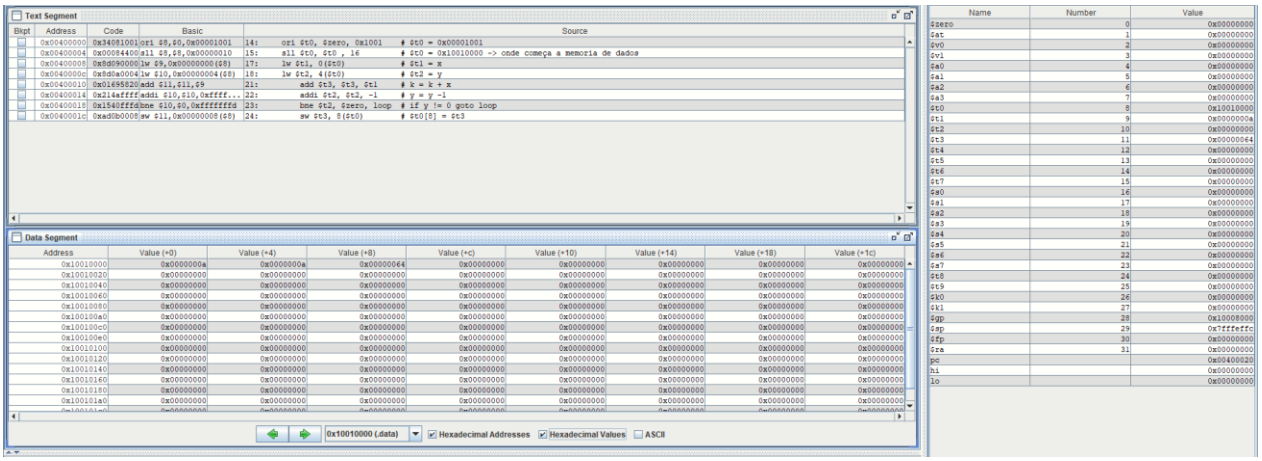
```
1  # Escreva um programa que avalie a expressão:
2  # (x * y) / z.
3  # Use
4  # x = 1600000 (=0x186A00),
5  # y = 80000 (=0x13880),
6  # z = 400000 (=0x61A80).
7  # Inicializar os registradores com os valores acima.
8
9  .data
10     x: .word 0x186A00
11     y: .word 0x13880
12     z: .word 0x61A80
13
14 .text
15 .globl main
16 main:
17     ori  $s0, $0, 0x186A  # $s0 = 0x0000186A
18     ori  $s1, $0, 0x1388  # $s1 = 0x00001388
19     ori  $s2, $0, 0x61A8  # $s2 = 0x000061A8
20     mult $s0, $s1         # $s0 * $s1 = 0x186A__ * 0x1388_ = 0xXXXX___
21     mflo $t0              #
22     div  $t0, $s2         # $t0 / $s2 = 0xXXXX___ / 0x61A8_ = 0xXXXX__
23     mflo $t0              #
24     sll  $t0, $t0, 8      #
25
26 # EOF
```



Programa 17:

P17.asm

```
1  # Para a expressão a seguir, escreva um programa que calcule o valor de k:
2  # k = x * y (Você deverá realizar a multiplicação através de somas!)
3  # O valor de x deve ser lido da primeira posição livre da memória e o valor de y deverá lido da segunda posição livre.
4  # O valor de k, após calculado, deverá ainda ser escrito na terceira posição livre da memória.
5
6  .data
7      x: .word 10
8      y: .word 10
9      k: .word -1
10
11 .text
12 .globl main
13 main:
14     ori $t0, $zero, 0x1001     # $t0 = 0x00001001
15     sll $t0, $t0 , 16          # $t0 = 0x10010000 -> onde começa a memoria de dados
16
17     lw $t1, 0($t0)             # $t1 = x
18     lw $t2, 4($t0)             # $t2 = y
19
20     loop:
21         add $t3, $t3, $t1      # k = k + x
22         addi $t2, $t2, -1      # y = y -1
23         bne $t2, $zero, loop   # if y != 0 goto loop
24         sw $t3, 8($t0)         # $t0[8] = $t3
25 # EOF
```

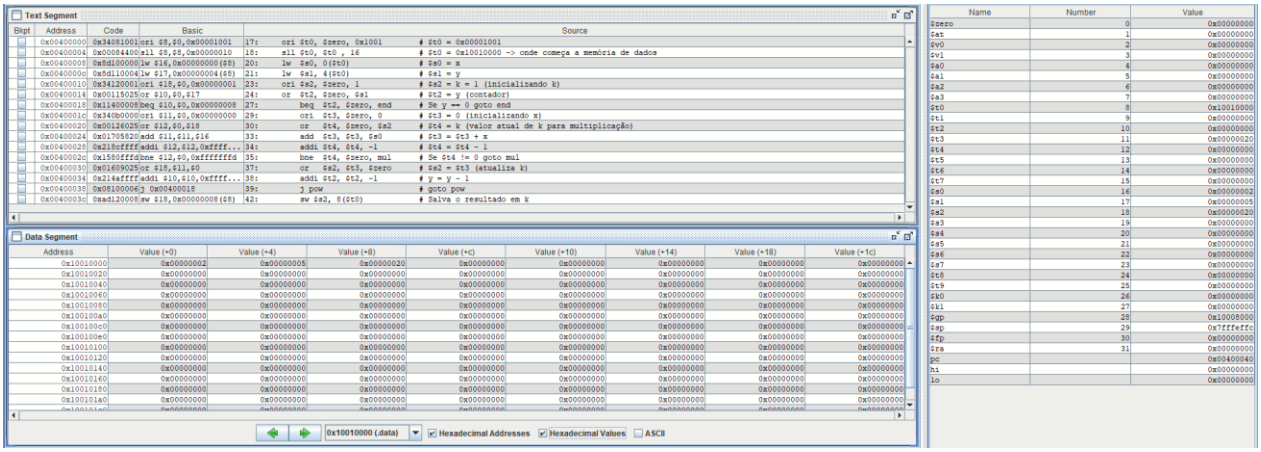


Programa 18:

Edit | Execute

P18.asm

```
.data
    x: .word 2
    y: .word 5
    k: .word -1

.text
.globl main
main:
    ori $t0, $zero, 0x1001          # $t0 = 0x00001001
    sll $t0, $t0 , 16               # $t0 = 0x10010000 -> onde começa a memória de dados

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

    ori $s2, $zero, 1               # $s2 = k = 1 (inicializando k)
    or  $t2, $zero, $s1             # $t2 = y (contador)

    pow:
        beq  $t2, $zero, end        # Se y == 0 goto end

        ori  $t3, $zero, 0          # $t3 = 0 (inicializando x)
        or   $t4, $zero, $s2        # $t4 = k (valor atual de k para multiplicação)

    mul:
        add  $t3, $t3, $s0          # $t3 = $t3 + x
        addi $t4, $t4, -1           # $t4 = $t4 - 1
        bne  $t4, $zero, mul        # Se $t4 != 0 goto mul

        or   $s2, $t3, $zero        # $s2 = $t3 (atualiza k)
        addi $t2, $t2, -1           # y = y - 1
        j pow                       # goto pow

    end:
        sw $s2, 8($t0)              # Salva o resultado em k

# EOF
```

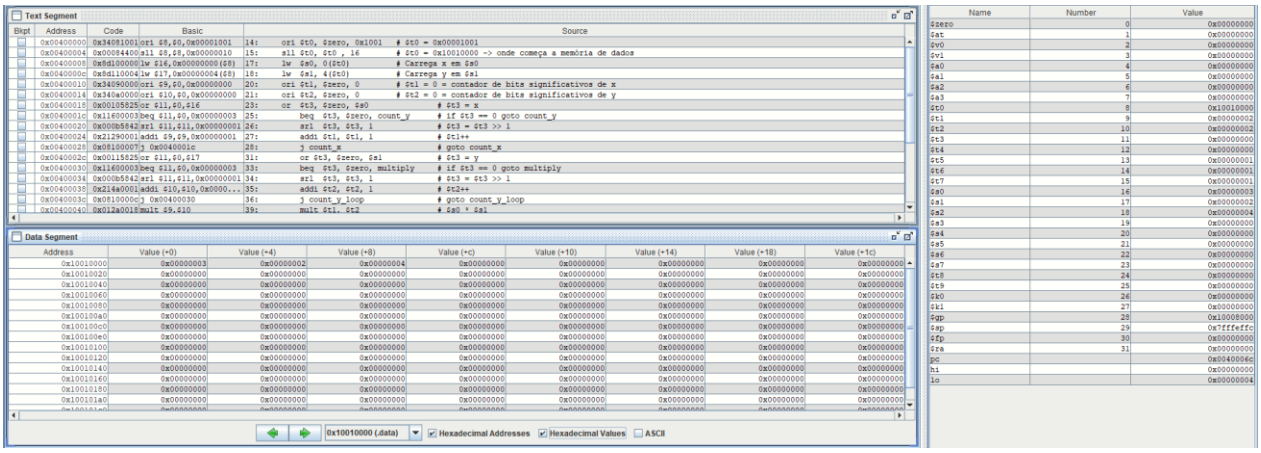


Programa 19:

Edit | Execute

P19.asm

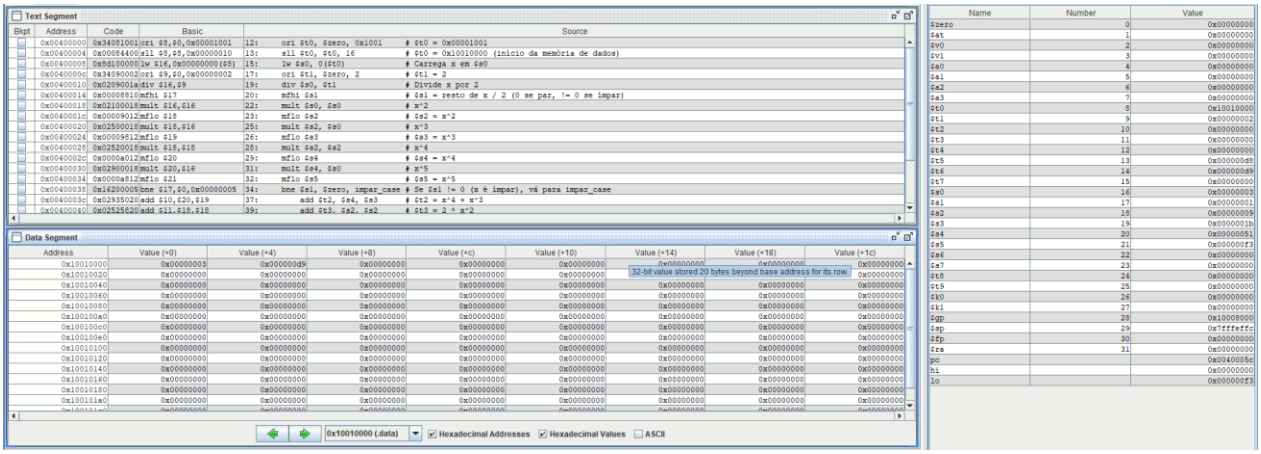
```
.data
    x: .word 3
    y: .word 2
    k: .word -1

.text
.globl main
main:
    ori $t0, $zero, 0x1001   # $t0 = 0x00001001
    sll $t0, $t0 , 16        # $t0 = 0x10010000 -> onde começa a memória de dados

    lw  $s0, 0($t0)          # Carrega x em $s0
    lw  $s1, 4($t0)          # Carrega y em $s1

    ori $t1, $zero, 0        # $t1 = 0 = contador de bits significativos de x
    ori $t2, $zero, 0        # $t2 = 0 = contador de bits significativos de y

    or  $t3, $zero, $s0              # $t3 = x
    count_x:
        beq  $t3, $zero, count_y     # if $t3 == 0 goto count_y
        srl  $t3, $t3, 1             # $t3 = $t3 >> 1
        addi $t1, $t1, 1             # $t1++
        j count_x                    # goto count_x

    count_y:
        or $t3, $zero, $s1           # $t3 = y
    count_y_loop:
        beq  $t3, $zero, multiply    # if $t3 == 0 goto multiply
        srl  $t3, $t3, 1             # $t3 = $t3 >> 1
        addi $t2, $t2, 1             # $t2++

        j count_y_loop               # goto count_y_loop

    multiply:
        mult $t1, $t2                # $s0 * $s1
        mflo $s2                     # $s2 = LO
        mfhi $s3                     # $s3 = HI

        slti $t5, $t1, 32            # Se $t1 < 32, $t5 = 1
        slti $t6, $t2, 32            # Se $t2 < 32, $t6 = 1
        and  $t7, $t5, $t6           # $t7 = 1 se ambos têm menos de 32 bits significativos

        beq  $t7, $zero, store_hi_lo # if $t7 == 0 goto store_hi_lo
        sw   $s2, 8($t0)             # Se $t7 é 1, armazena apenas em $s2
        j end                        # goto end

    store_hi_lo:
        sw   $s2, 8($t0)             # Armazena parte baixa em posição k
        sw   $s3, 12($t0)            # Armazena parte alta em posição k + 4

    end:

# EOF
```


Text Segment
Bkpt Address Code Basic Source
0x00400000 0x34081001 ori $8,$0,0x00001001 14: ori $t0, $zero, 0x1001 # $t0 = 0x00001001
0x00400004 0x00084400 sll $8,$8,0x00000010 15: sll $t0, $t0 , 16 # $t0 = 0x10010000 -> onde começa a memória de dados
0x00400008 0x8d100000 lw $16,0x00000000($8) 17: lw $s0, 0($t0) # Carrega x em $s0
0x0040000c 0x8d110004 lw $17,0x00000004($8) 18: lw $s1, 4($t0) # Carrega y em $s1
0x00400010 0x34090000 ori $9,$0,0x00000000 20: ori $t1, $zero, 0 # $t1 = 0 = contador de bits significativos de x
0x00400014 0x340a0000 ori $10,$0,0x00000000 21: ori $t2, $zero, 0 # $t2 = 0 = contador de bits significativos de y
0x00400018 0x00105825 or $11,$0,$16 23: or $t3, $zero, $s0 # $t3 = x
0x0040001c 0x11600003 beq $11,$0,0x00000003 25: beq $t3, $zero, count_y # if $t3 == 0 goto count_y
0x00400020 0x000b5842 srl $11,$11,0x00000001 26: srl $t3, $t3, 1 # $t3 = $t3 >> 1
0x00400024 0x21290001 addi $9,$9,0x00000001 27: addi $t1, $t1, 1 # $t1++
0x00400028 0x08100007 j 0x0040001c 28: j count_x # goto count_x
0x0040002c 0x00115825 or $11,$0,$17 31: or $t3, $zero, $s1 # $t3 = y
0x00400030 0x11600003 beq $11,$0,0x00000003 33: beq $t3, $zero, multiply # if $t3 == 0 goto multiply
0x00400034 0x000b5842 srl $11,$11,0x00000001 34: srl $t3, $t3, 1 # $t3 = $t3 >> 1
0x00400038 0x214a0001 addi $10,$10,0x0000... 35: addi $t2, $t2, 1 # $t2++
0x0040003c 0x08100003 j 0x00400030 36: j count_y_loop # goto count_y_loop
0x00400040 0x012a0018 mult $9,$10 39: mult $t1, $t2 # $s0 * $s1
Data Segment
Address Value (+0) Value (+4) Value (+8) Value (+c) Value (+10) Value (+14) Value (+18) Value (+1c)
0x10010000 0x00000003 0x00000002 0x00000004 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
0x10010120 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010140 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010160 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010180 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x100101a0 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x1001000 (.data)
Hexadecimal Addresses
Hexadecimal Values
ASCII
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 0x00000002
$t2 10 0x00000002
$t3 11 0x00000000
$t4 12 0x00000000
$t5 13 0x00000001
$t6 14 0x00000001
$t7 15 0x00000001
$s0 16 0x00000003
$s1 17 0x00000002
$s2 18 0x00000004
$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 0x0040006c
hi 0x00000000
lo 0x00000004

Programa 20:

P20.asm

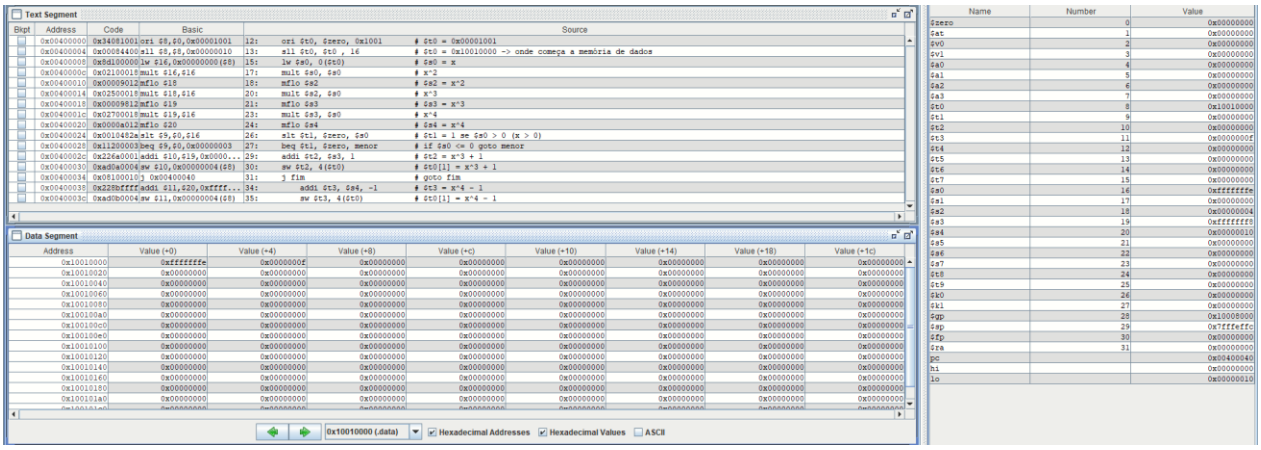
```
.data
    x: .word 3

.text
.globl main
main:
    ori $t0, $zero, 0x1001      # $t0 = 0x00001001
    sll $t0, $t0, 16            # $t0 = 0x10010000 (início da memória de dados)

    lw $s0, 0($t0)              # Carrega x em $s0

    ori $t1, $zero, 2           # $t1 = 2

    div $s0, $t1                # Divide x por 2
    mfhi $s1                    # $s1 = resto de x / 2 (0 se par, != 0 se impar)

    mult $s0, $s0               # x^2
    mflo $s2                    # $s2 = x^2

    mult $s2, $s0               # x^3
    mflo $s3                    # $s3 = x^3

    mult $s2, $s2               # x^4
    mflo $s4                    # $s4 = x^4

    mult $s4, $s0               # x^5
    mflo $s5                    # $s5 = x^5

    bne $s1, $zero, impar_case # Se $s1 != 0 (x é ímpar), vá para impar_case

    par_case:
        add $t2, $s4, $s3       # $t2 = x^4 + x^3

        add $t3, $s2, $s2       # $t3 = 2 * x^2

        sub $t4, $t2, $t3       # $t4 = x^4 + x^3 - 2*x^2
        sw $t4, 4($t0)          # Armazena y na segunda posição da memória
        j end                   # Fim do programa

    impar_case:
        sub $t5, $s5, $s3       # $t5 = x^5 - x^3
        addi $t6, $t5, 1        # $t6 = x^5 - x^3 + 1
        sw $t6, 4($t0)          # Armazena y na segunda posição da memória

    end:
# EOF
```


Text Segment
Bkpt Address Code Basic Source
0x00400000 0x34081001 ori $8,$0,0x00001001 12: ori $t0, $zero, 0x1001 # $t0 = 0x00001001
0x00400004 0x00084400 sll $8,$8,0x00000010 13: sll $t0, $t0, 16 # $t0 = 0x10010000 (início da memória de dados)
0x00400008 0x8d100000 lw $16,0x00000000($8) 15: lw $s0, 0($t0) # Carrega x em $s0
0x0040000c 0x34090002 ori $9,$0,0x00000002 17: ori $t1, $zero, 2 # $t1 = 2
0x00400010 0x0209001a div $16,$9 19: div $s0, $t1 # Divide x por 2
0x00400014 0x00008810 mfhi $17 20: mfhi $s1 # $s1 = resto de x / 2 (0 se par, != 0 se impar)
0x00400018 0x02100018 mult $16,$16 22: mult $s0, $s0 # x^2
0x0040001c 0x00009012 mflo $18 23: mflo $s2 # $s2 = x^2
0x00400020 0x02500018 mult $18,$16 25: mult $s2, $s0 # x^3
0x00400024 0x00009812 mflo $19 26: mflo $s3 # $s3 = x^3
0x00400028 0x02520018 mult $18,$18 28: mult $s2, $s2 # x^4
0x0040002c 0x0000a012 mflo $20 29: mflo $s4 # $s4 = x^4
0x00400030 0x02900018 mult $20,$16 31: mult $s4, $s0 # x^5
0x00400034 0x0000a812 mflo $21 32: mflo $s5 # $s5 = x^5
0x00400038 0x16200005 bne $17,$0,0x00000005 34: bne $s1, $zero, impar_case # Se $s1 != 0 (x é impar), vá para impar_case
0x0040003c 0x02935020 add $10,$20,$19 37: add $t2, $s4, $s3 # $t2 = x^4 + x^3
0x00400040 0x02525820 add $11,$18,$18 39: add $t3, $s2, $s2 # $t3 = 2 * x^2
Data Segment
Address Value (+0) Value (+4) Value (+8) Value (+c) Value (+10) Value (+14) Value (+18) Value (+1c)
0x10010000 0x00000003 0x000000d9
0x10010020
0x10010040
0x10010060
0x10010080
0x100100a0
0x100100c0
0x100100e0
0x10010100
0x10010120
0x10010140
0x10010160
0x10010180
0x100101a0
32-bit value stored 20 bytes beyond base address for its row.
0x10010000 (.data)
Hexadecimal Addresses
Hexadecimal Values
ASCII
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 0x00000002
$t2 10 0x00000000
$t3 11 0x00000000
$t4 12 0x00000000
$t5 13 0x000000d8
$t6 14 0x000000d9
$t7 15 0x00000000
$s0 16 0x00000003
$s1 17 0x00000001
$s2 18 0x00000009
$s3 19 0x0000001b
$s4 20 0x00000051
$s5 21 0x000000f3
$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 0x0040005c
hi 0x00000000
lo 0x000000f3

Programa 21:

P21.asm

```
5
6   .data
7       x: .word -2
8
9   .text
10  .globl main
11  main:
12      ori $t0, $zero, 0x1001          # $t0 = 0x00001001
13      sll $t0, $t0 , 16               # $t0 = 0x10010000 -> onde começa a memória de dados
14
15      lw $s0, 0($t0)                  # $s0 = x
16
17      mult $s0, $s0                   # x^2
18      mflo $s2                        # $s2 = x^2
19
20      mult $s2, $s0                   # x^3
21      mflo $s3                        # $s3 = x^3
22
23      mult $s3, $s0                   # x^4
24      mflo $s4                        # $s4 = x^4
25
26      slt $t1, $zero, $s0             # $t1 = 1 se $s0 > 0 (x > 0)
27      beq $t1, $zero, menor           # if $s0 <= 0 goto menor
28
29      addi $t2, $s3, 1                # $t2 = x^3 + 1
30      sw $t2, 4($t0)                  # $t0[1] = x^3 + 1
31      j fim                           # goto fim
32
33      menor:
34          addi $t3, $s4, -1           # $t3 = x^4 - 1
35          sw $t3, 4($t0)              # $t0[1] = x^4 - 1
36
37      fim:
38  # EOF
```



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

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

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

A. mult

B. multu

C. multi

D. mutt

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

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

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

A. lo

B. hi

C. high

D. $2

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

A. dv

B. divide

C. divu

D. div

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

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.

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
```

FIM