Exercício Pratico 05

Gabriela Colen Castelo Borges e Felipe Campolina Soares de Paula

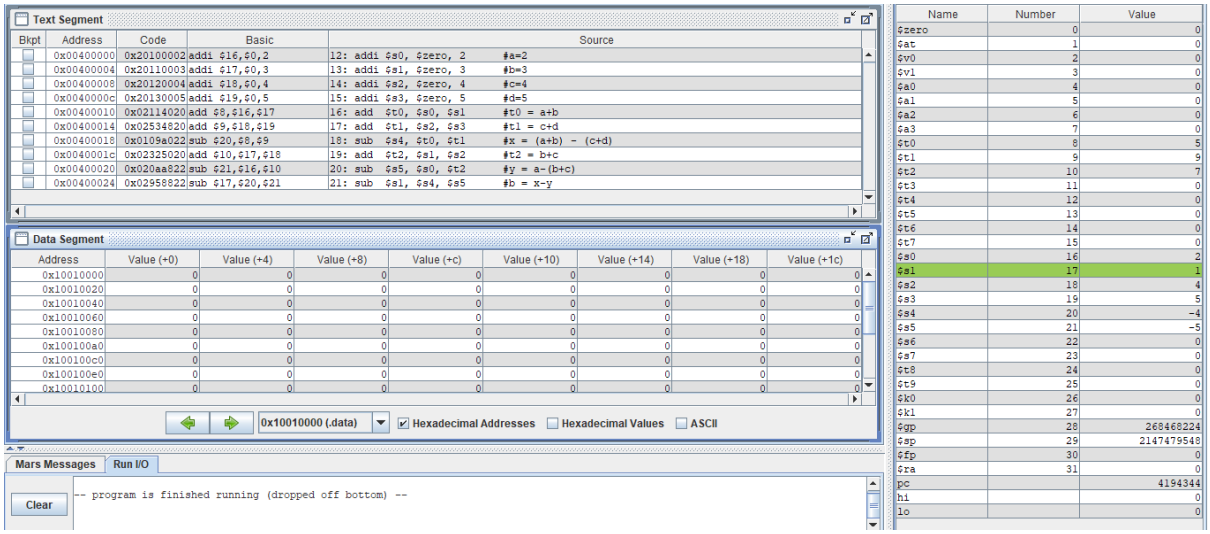
Parte 1

1. A
2. B
3. A
6. C
7. D
8. D
9. D
10. D
11. C
12. A

Parte 2

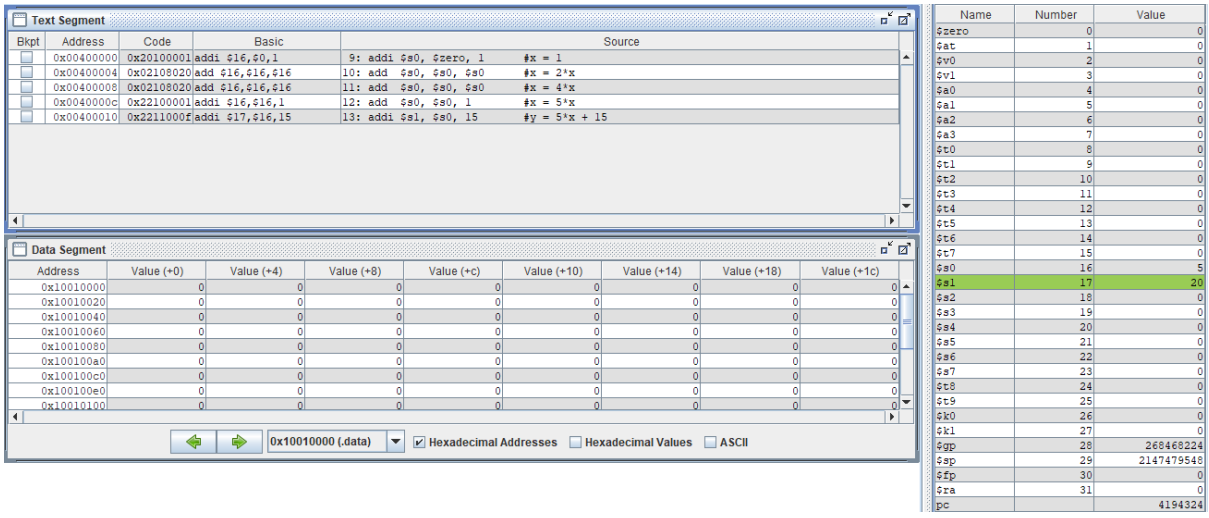
1.

```
# Exemplo 1:
# a -> $s0
# b -> $s1
# c -> $s2
# d -> $s3
# x -> $s4
# y -> $s5
.text
.globl main

main:
addi $s0, $zero, 2      #a=2
addi $s1, $zero, 3      #b=3
addi $s2, $zero, 4      #c=4
addi $s3, $zero, 5      #d=5
add  $t0, $s0, $s1      #t0 = a+b
add  $t1, $s2, $s3      #t1 = c+d
sub  $s4, $t0, $t1      #x = (a+b) - (c+d)
add  $t2, $s1, $s2      #t2 = b+c
sub  $s5, $s0, $t2      #y = a-(b+c)
sub  $s1, $s4, $s5      #b = x-y
```



2.

```
# Programa 2:
# x -> $s0
# y -> $s1

.text
.globl main

main:
addi $s0, $zero, 1        #x = 1
add  $s0, $s0, $s0        #x = 2*x
add  $s0, $s0, $s0        #x = 4*x
add  $s0, $s0, 1          #x = 5*x
addi $s1, $s0, 15         #y = 5*x + 15
```



Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20100001	addi $16,$0,1	9: addi $s0, $zero, 1 #x = 1
	0x00400004	0x02108020	add $16,$16,$16	10: add $s0, $s0, $s0 #x = 2*x
	0x00400008	0x02108020	add $16,$16,$16	11: add $s0, $s0, $s0 #x = 4*x
	0x0040000c	0x22100001	addi $16,$16,1	12: add $s0, $s0, 1 #x = 5*x
	0x00400010	0x2211000f	addi $17,$16,15	13: addi $s1, $s0, 15 #y = 5*x + 15

Data Segment

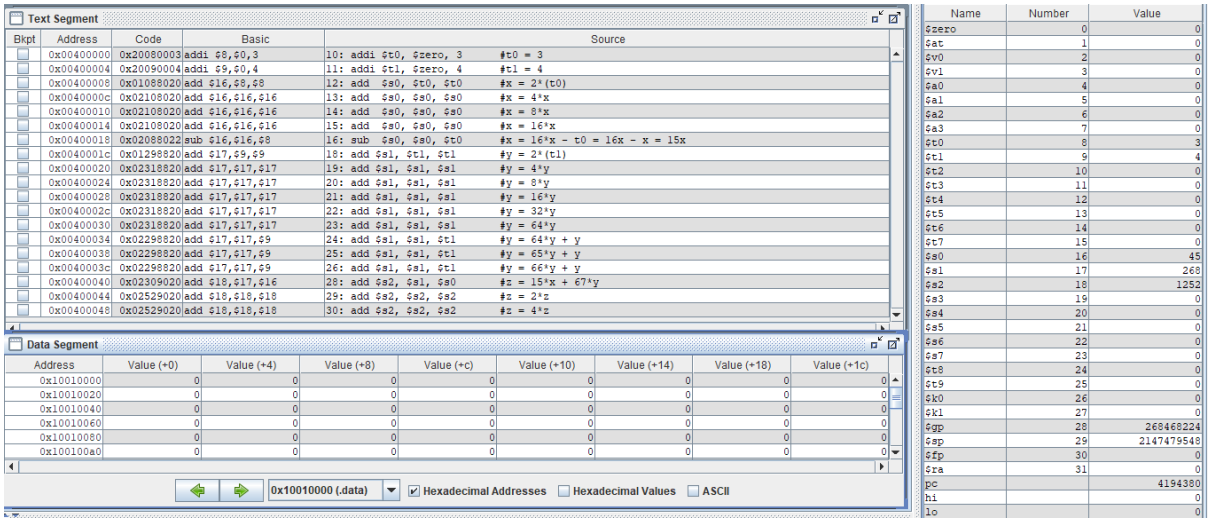
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0	0	0	0	0	0	0	0
0x10010020	0	0	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0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

0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

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	0
$t1	9	0
$t2	10	0
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	5
$s1	17	20
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	0
pc		4194324

3.

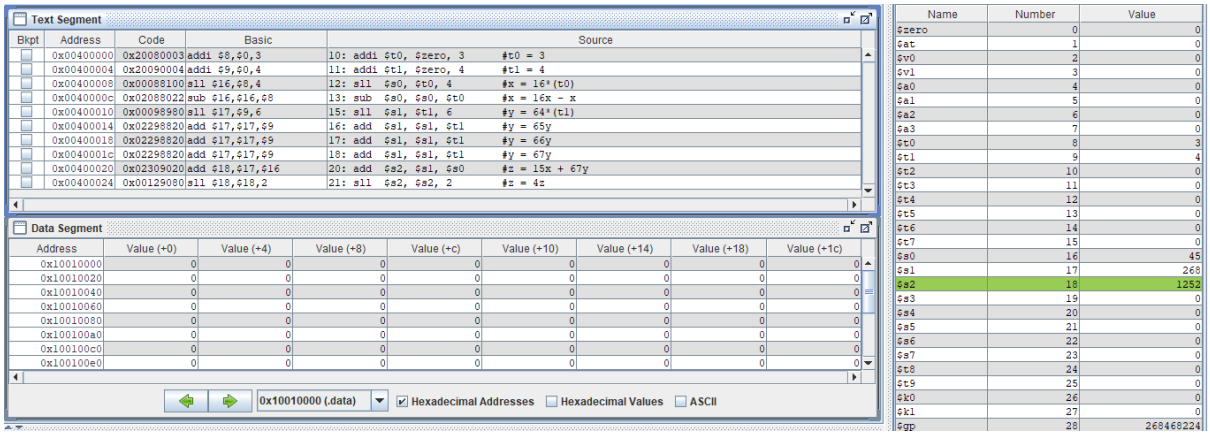
```
# Programa 3:
# x -> $s0
# y -> $s1
# z -> $s2

.text
.globl main

main:
addi $t0, $zero, 3       #t0 = 3
addi $t1, $zero, 4       #t1 = 4
add  $s0, $t0, $t0       #x = 2*(t0)
add  $s0, $s0, $s0       #x = 4*x
add  $s0, $s0, $s0       #x = 8*x
add  $s0, $s0, $s0       #x = 16*x
sub  $s0, $s0, $t0       #x = 16*x - t0 = 16x - x = 15x

add $s1, $t1, $t1        #y = 2*(t1)
add $s1, $s1, $s1        #y = 4*y
```



4.

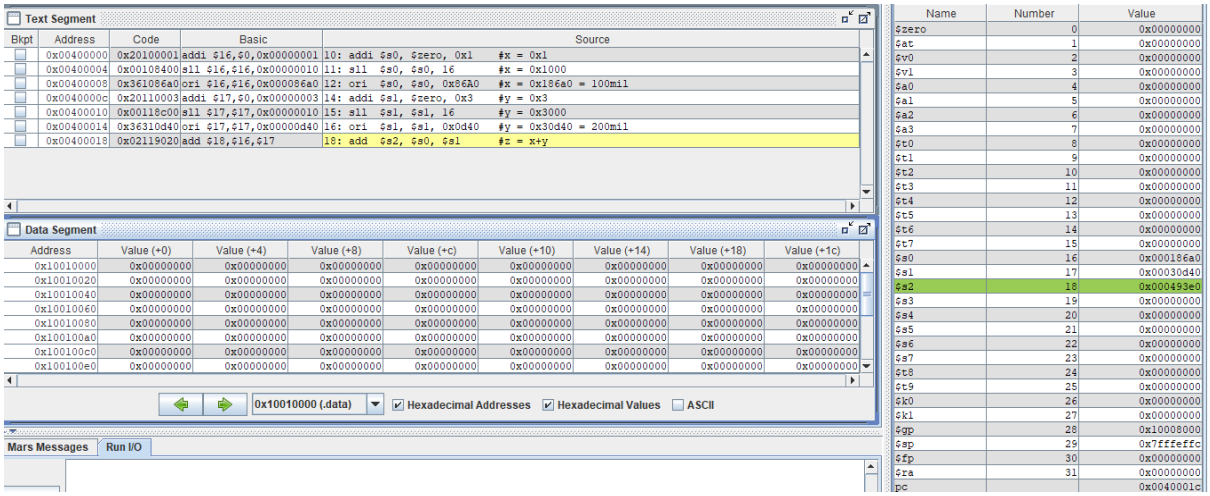
```
# Programa 4:
# x -> $s0
# y -> $s1
# z -> $s2

.text
.globl main

main:
addi $t0, $zero, 3       #t0 = 3
addi $t1, $zero, 4       #t1 = 4
sll  $s0, $t0, 4         #x = 16*(t0)
sub  $s0, $s0, $t0       #x = 16x - x
```

```
sll  $s1, $t1, 6         #y = 64*(t1)
add  $s1, $s1, $t1       #y = 65y
add  $s1, $s1, $t1       #y = 66y
add  $s1, $s1, $t1       #y = 67y

add  $s2, $s1, $s0       #z = 15x + 67y
sll  $s2, $s2, 2         #z = 4z
```



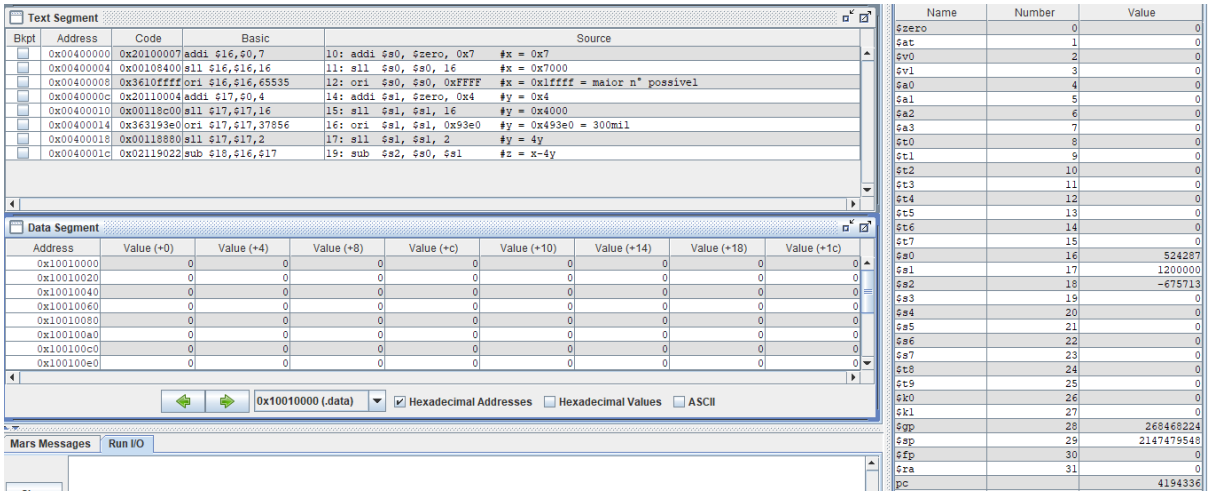
5.

```
# Programa 5:
# x -> $s0
# y -> $s1
# z -> $s2

.text
.globl main

main:
addi $s0, $zero, 0x1     #x = 0x1
sll  $s0, $s0, 16        #x = 0x1000
ori  $s0, $s0, 0x86A0    #x = 0x186a0 = 100mil

addi $s1, $zero, 0x3     #y = 0x3
sll  $s1, $s1, 16        #y = 0x3000
ori  $s1, $s1, 0x0d40    #y = 0x30d40 = 200mil

add  $s2, $s0, $s1       #z = x+y
```



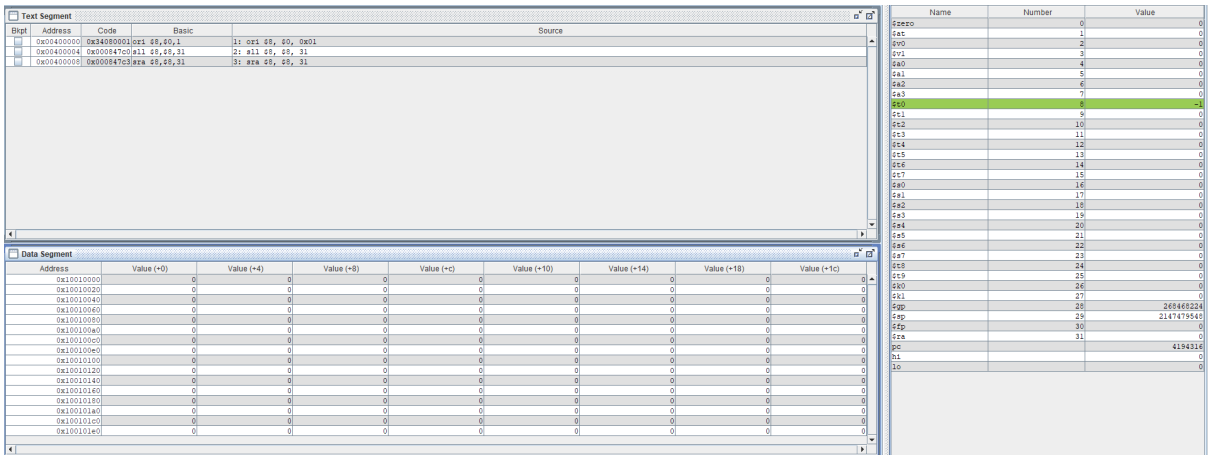
6.

```
# Programa 6:
# x -> $s0
# y -> $s1
# z -> $s2

.text
.globl main

main:
addi $s0, $zero, 0x7     #x = 0x7
sll  $s0, $s0, 16        #x = 0x7000
ori  $s0, $s0, 0xFFFF    #x = 0x1ffff = maior n° possivel
```



7.

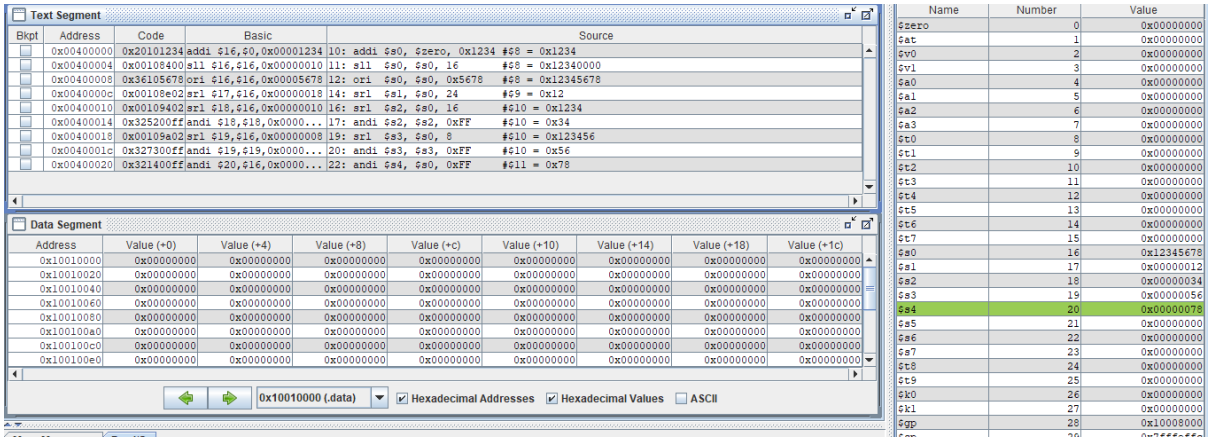
```
1  ori $8, $0, 0x01
2  sll $8, $8, 31
3  sra $8, $8, 31
```



8.

```
1  # Programa 8:
2  # $8 -> $s0
3  # $9 -> $s1
4  # $10 -> $s2
5  # $11 -> $s3
6
7  .text
8  .globl main
9
10 main:
11 addi $s0, $zero, 0x1234 #$8 = 0x1234
12 sll  $s0, $s0, 16       #$8 = 0x12340000
```

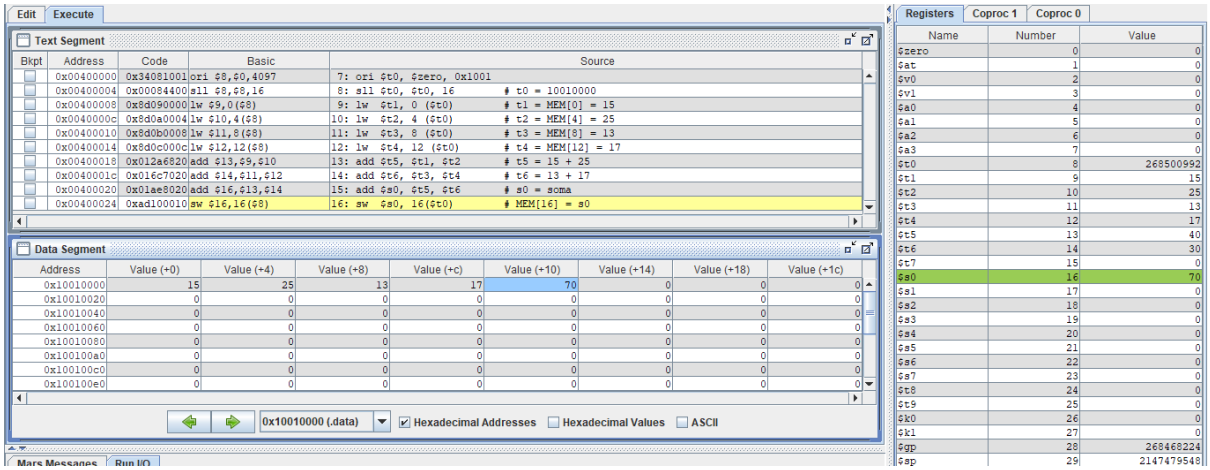
```
14
15  srl  $s1, $s0, 24       #$9 = 0x12
16
17  srl  $s2, $s0, 16       #$10 = 0x1234
18  andi $s2, $s2, 0xFF     #$10 = 0x34
19
20  srl  $s3, $s0, 8        #$10 = 0x123456
21  andi $s3, $s3, 0xFF     #$10 = 0x56
22
23  andi $s4, $s0, 0xFF     #$11 = 0x78
```



9.

```
1   # Programa 9
2   # s0 -> soma
3
4   .text
5   .globl main
6
7   main:
8   ori $t0, $zero, 0x1001
9   sll $t0, $t0, 16          # t0 = 10010000
10  lw  $t1, 0 ($t0)          # t1 = MEM[0] = 15
11  lw  $t2, 4 ($t0)          # t2 = MEM[4] = 25
12  lw  $t3, 8 ($t0)          # t3 = MEM[8] = 13
```

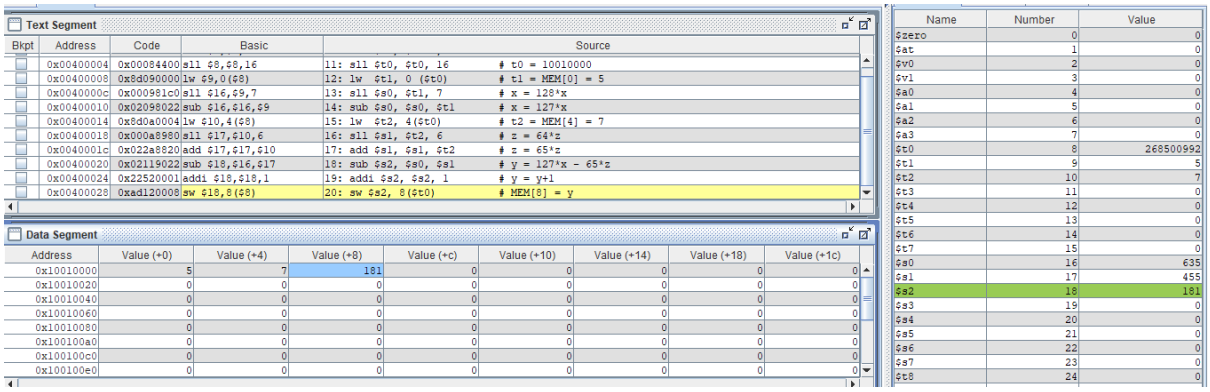
```
13  lw   $t4, 12 ($t0)        # t4 = MEM[12] = 17
14  add  $t5, $t1, $t2        # t5 = 15 + 25
15  add  $t6, $t3, $t4        # t6 = 13 + 17
16  add  $s0, $t5, $t6        # s0 = soma
17  sw   $s0, 16($t0)         # MEM[16] = s0
18
19  .data
20  x1: .word 15
21  x2: .word 25
22  x3: .word 13
23  x4: .word 17
24  soma: .word -1
```



10.

```
1   # Programa 10
2   # x -> s0
3   # z -> s1
4   # y -> s2
5
6   .text
7   .globl main
8
9   main:
10  ori $t0, $zero, 0x1001
11  sll $t0, $t0, 16          # t0 = 10010000
12  lw  $t1, 0 ($t0)          # t1 = MEM[0] = 5
13  sll $s0, $t1, 7           # x = 128*x
```

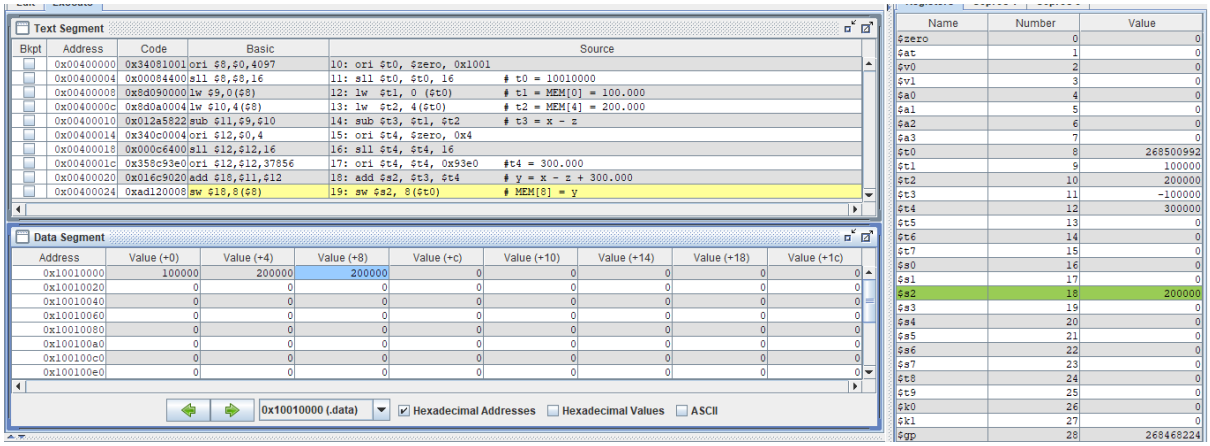
```
14  sub $s0, $s0, $t1       # x = 127*x
15  lw  $t2, 4($t0)         # t2 = MEM[4] = 7
16  sll $s1, $t2, 6         # z = 64*z
17  add $s1, $s1, $t2       # z = 65*z
18  sub $s2, $s0, $s1       # y = 127*x - 65*z
19  addi $s2, $s2, 1        # y = y+1
20  sw $s2, 8($t0)          # MEM[8] = y
21
22
23  .data
24  x: .word 5
25  z: .word 7
26  y: .word 0
```



11.

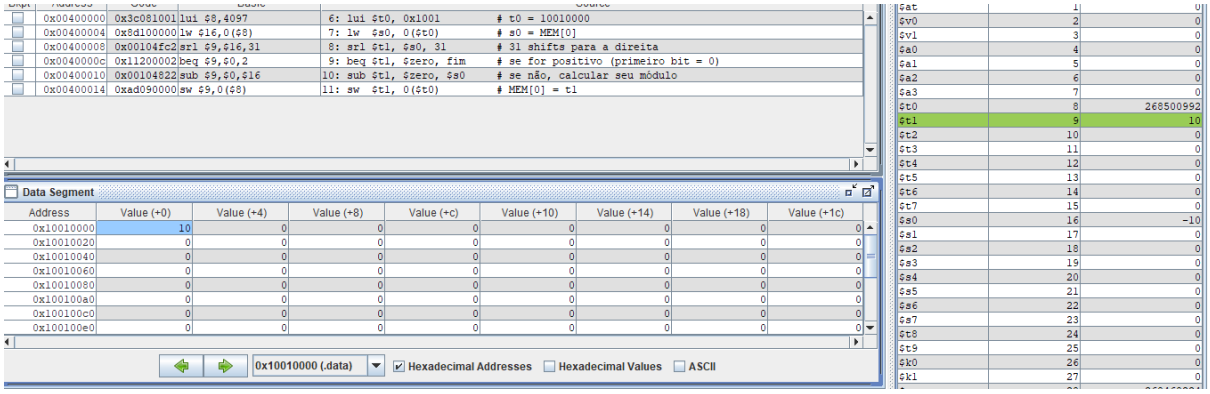
```
1   # Programa 11
2   # x -> s0
3   # z -> s1
4   # y -> s2
5
6   .text
7   .globl main
8
9   main:
10  ori $t0, $zero, 0x1001
11  sll $t0, $t0, 16          # t0 = 10010000
12  lw  $t1, 0 ($t0)          # t1 = MEM[0] = 100.000
```

```
13  lw  $t2, 4($t0)          # t2 = MEM[4] = 200.000
14  sub $t3, $t1, $t2        # t3 = x - z
15  ori $t4, $zero, 0x4
16  sll $t4, $t4, 16
17  ori $t4, $t4, 0x93e0     #t4 = 300.000
18  add $s2, $t3, $t4        # y = x - z + 300.000
19  sw $s2, 8($t0)           # MEM[8] = y
20
21  .data
22  x: .word 100000
23  z: .word 200000
24  y: .word 0
```



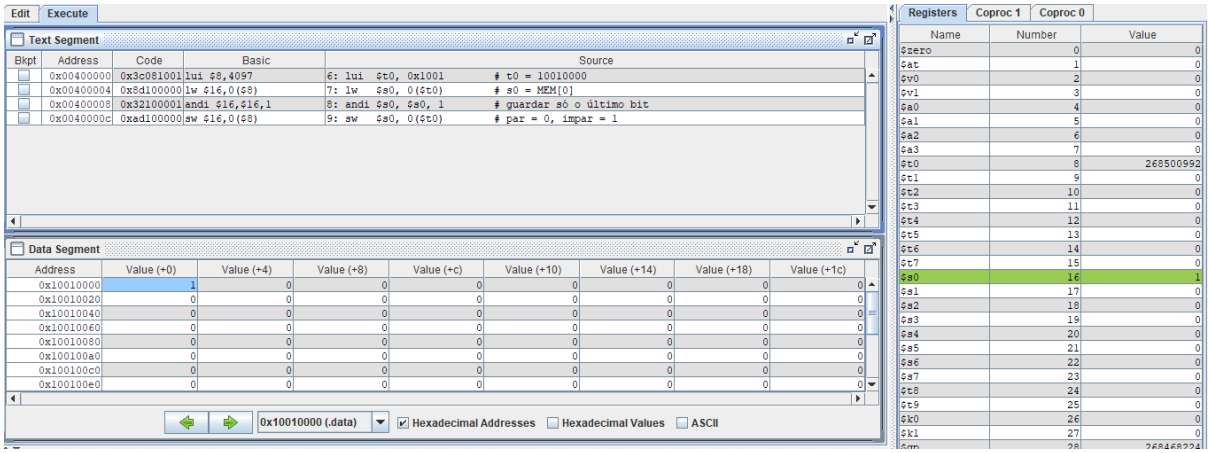
13.

```
1   # Programa 13:
2   .text
3   .globl main
4
5   main:
6   lui $t0, 0x1001          # t0 = 10010000
7   lw  $s0, 0($t0)          # s0 = MEM[0]
8   srl $t1, $s0, 31         # 31 shifts para a direita
9   beq $t1, $zero, fim      # se for positivo (primeiro bit = 0)
10  sub $t1, $zero, $s0      # se não, calcular seu módulo
11  sw  $t1, 0($t0)          # MEM[0] = t1
12
13  fim:
14
15  .data
16  a: .word -10
```



14.

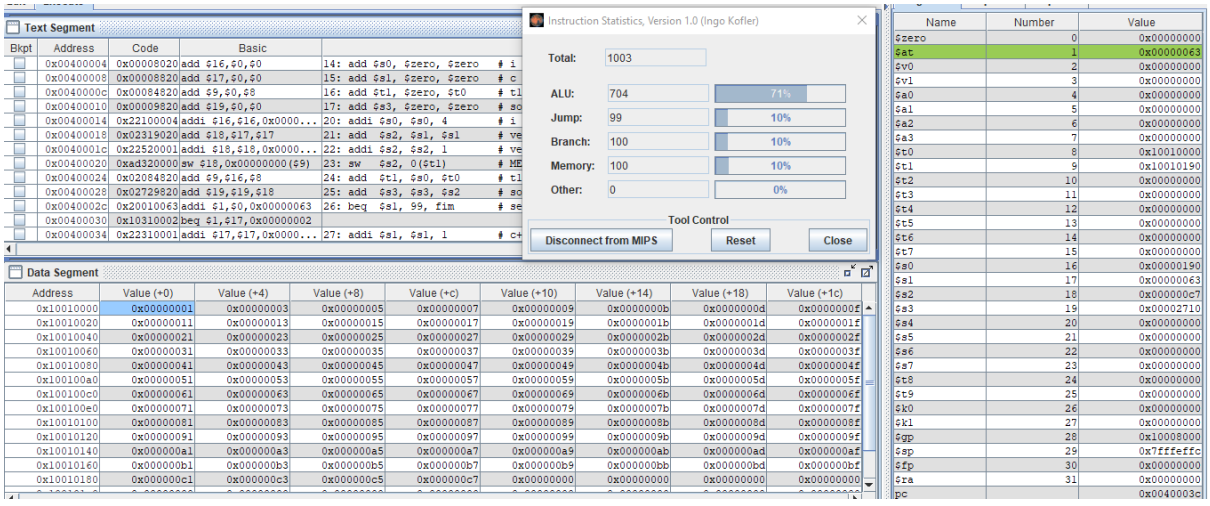
```
1   # Programa 14:
2   .text
3   .globl main
4
5   main:
6   lui  $t0, 0x1001        # t0 = 10010000
7   lw   $s0, 0($t0)        # s0 = MEM[0]
8   andi $s0, $s0, 1        # guardar só o último bit
9   sw   $s0, 0($t0)        # par = 0, impar = 1
10
11  .data
12  a: .word 100100011
```



15.

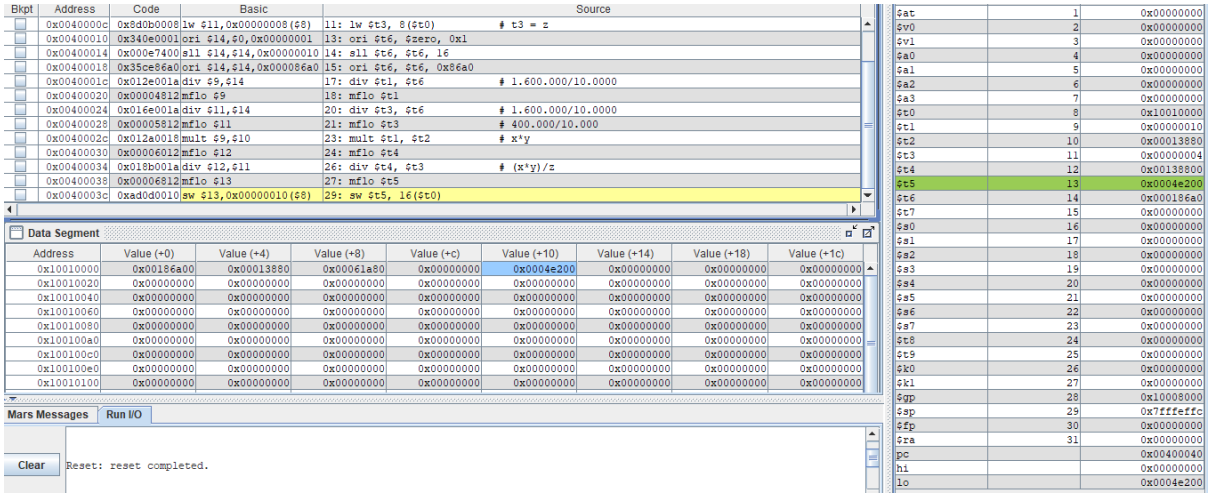
```
1   # Programa 15
2   # Vetor de 100 posições
3   # Vetor[i] = 2*i + 1
4   # Soma de todos os elementos
5   # i -> $s0
6   # contador (c)-> $s1
7   # conteudo (vetor) -> $s2
8   # soma -> $s3
9
10  .text
11  lui $t0, 0x1001          # t1 = 0x1001000
12
13  # inicializando as variáveis
14  add $s0, $zero, $zero    # i = 0
15  add $s1, $zero, $zero    # c = 0
```

```
16  add $t1, $zero, $t0      # t1 = t0
17  add $s3, $zero, $zero    # soma = 0
18
19  loop:
20  addi $s0, $s0, 4         # i = i+4
21  add  $s2, $s1, $s1       # vetor = 2*c
22  addi $s2, $s2, 1         # vetor = 2*c +1
23  sw   $s2, 0($t1)         # MEM[t1] = s1
24  add  $t1, $s0, $t0       # t1 = endereço + i
25  add  $s3, $s3, $s2       # soma = soma + vetor
26  beq  $s1, 99, fim        # se c == 99, vai para fim
27  addi $s1, $s1, 1         # c++
28  j loop
29
30  fim:
```



16.

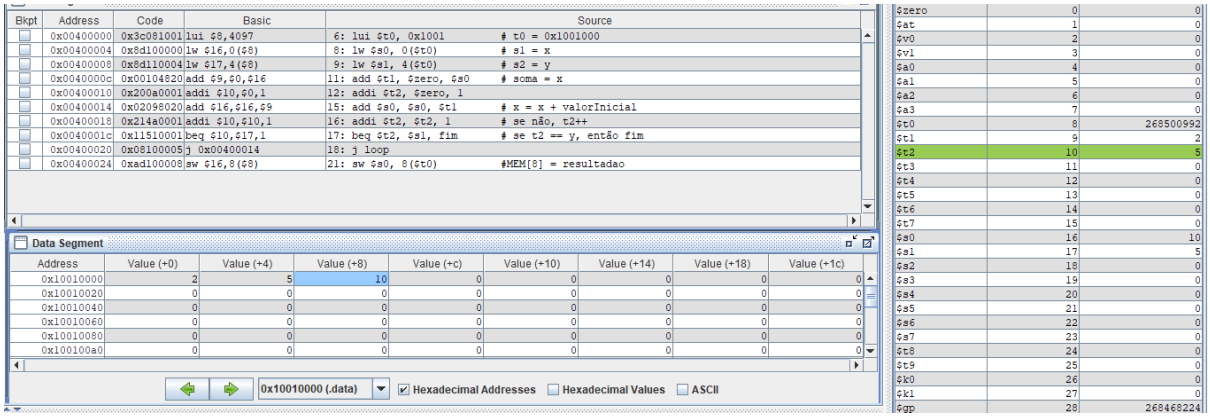
```
1  # Programa 16
2  # x = 1.600.000 -> $t1
3  # y = 80.000 -> $t2
4  # z = 400.000 -> $t3
5
6  .text
7  lui $t0, 0x1001          # t0 = 0x1001000
8
9  lw $t1, 0($t0)           # t1 = x
10 lw $t2, 4($t0)           # t2 = y
11 lw $t3, 8($t0)           # t3 = z
12
13 ori $t6, $zero, 0x1      # t6 = 100.000
14 sll $t6, $t6, 16
15 ori $t6, $t6, 0x86a0
16
17 div $t1, $t6             # 1.600.000/10.0000
18 mflo $t1
19
20 div $t3, $t6             # 1.600.000/10.0000
21 mflo $t3                 # 400.000/10.000
22
23 mult $t1, $t2            # x*y
24 mflo $t4
25
26 div $t4, $t3             # (x*y)/z
27 mflo $t5
28 sw $t5, 16($t0)          # MEM[16] = resultado
29
30 .data
31 x:   .word 0x186A00      # x = 1600000
32 y:   .word 0x13880       # y = 80000
33 z:   .word 0x61A80       # z = 400000
34 resp: .word 0            # armazenará o resultado final
```



17.

```
1   # Programa 17
2   # x -> $s0
3   # y -> $s1
4
5   .text
6   lui $t0, 0x1001          # t0 = 0x1001000
7
8   lw $s0, 0($t0)           # s1 = x
9   lw $s1, 4($t0)           # s2 = y
10
11  add $t1, $zero, $s0      # soma = x
12  addi $t2, $zero, 1
13
```

```
14  loop:
15  add $s0, $s0, $t1        # x = x + valorInicial
16  addi $t2, $t2, 1         # se não, t2++
17  beq $t2, $s1, fim        # se t2 == y, então fim
18  j loop
19
20  fim:
21  sw $s0, 8($t0)           #MEM[8] = resultadao
22
23  .data
24  x:   .word 2
25  y:   .word 5
26  k:   .word 0
```



Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c081001	lui $8,4097	6: lui $t0, 0x1001 # t0 = 0x1001000
	0x00400004	0x8d100000	lw $16,0($8)	8: lw $s0, 0($t0) # s1 = x
	0x00400008	0x8d110004	lw $17,4($8)	9: lw $s1, 4($t0) # s2 = y
	0x0040000c	0x00104820	add $9,$0,$16	11: add $t1, $zero, $s0 # soma = x
	0x00400010	0x200a0001	addi $10,$0,1	12: addi $t2, $zero, 1
	0x00400014	0x02098020	add $16,$16,$9	15: add $s0, $s0, $t1 # x = x + valorInicial
	0x00400018	0x214a0001	addi $10,$10,1	16: addi $t2, $t2, 1 # se não, t2++
	0x0040001c	0x11510001	beq $10,$17,1	17: beq $t2, $s1, fim # se t2 == y, então fim
	0x00400020	0x08100005	j 0x00400014	18: j loop
	0x00400024	0xad100008	sw $16,8($8)	21: sw $s0, 8($t0) #MEM[8] = resultadao

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	2	5	10	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

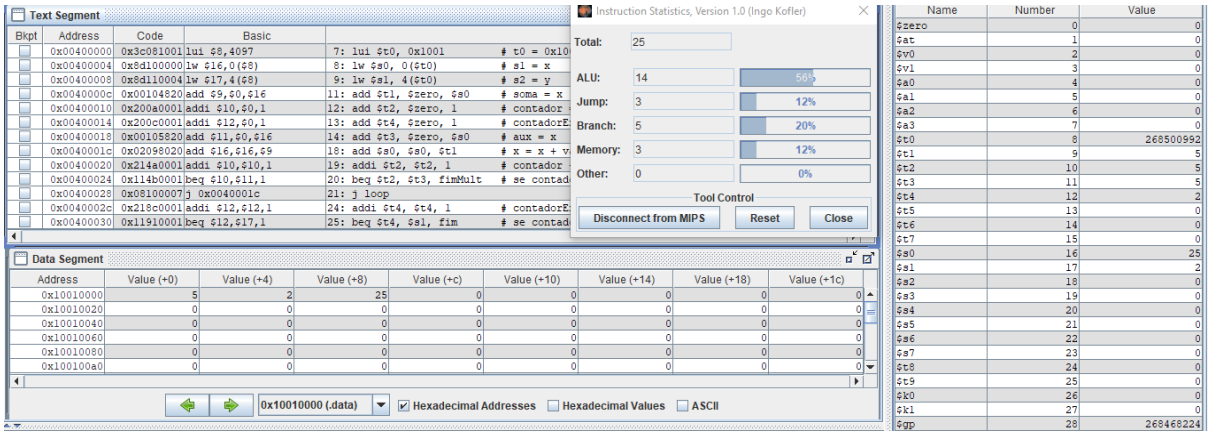
0x10010000 (.data) ☑ Hexadecimal Addresses ☐ Hexadecimal Values ☐ ASCII

$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	268500992
$t1	9	2
$t2	10	5
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	10
$s1	17	5
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224

18.

```
1  # Programa 18
2  # x -> $s0
3  # y -> $s1
4  # aux -> $t3
5
6  .text
7  lui $t0, 0x1001          # t0 = 0x1001000
8  lw $s0, 0($t0)           # s1 = x
9  lw $s1, 4($t0)           # s2 = y
10
11 add $t1, $zero, $s0      # soma = x
12 add $t2, $zero, 1        # contador = 1
13 add $t4, $zero, 1        # contadorExp = 1
14 add $t3, $zero, $s0      # aux = x
15
16 exponencial:
17 loop:
```

```
18  add $s0, $s0, $t1        # x = x + valorInicial
19  addi $t2, $t2, 1         # contador ++
20  beq $t2, $t3, fimMult    # se contador == x, então fim da multiplicação
21  j loop
22
23  fimMult:
24  addi $t4, $t4, 1         # contadorExp ++
25  beq $t4, $s1, fim        # se contadorExp == y, fim
26  j exponencial
27
28  fim:
29  sw $s0, 8($t0)           #MEM[8] = resultadao
30
31  .data
32  x:   .word 5
33  y:   .word 2
34  k:   .word 0
```



Parte 3

1. C
2. B
3. A
4. C
5. B
6. A
7. D
8. A

9. A

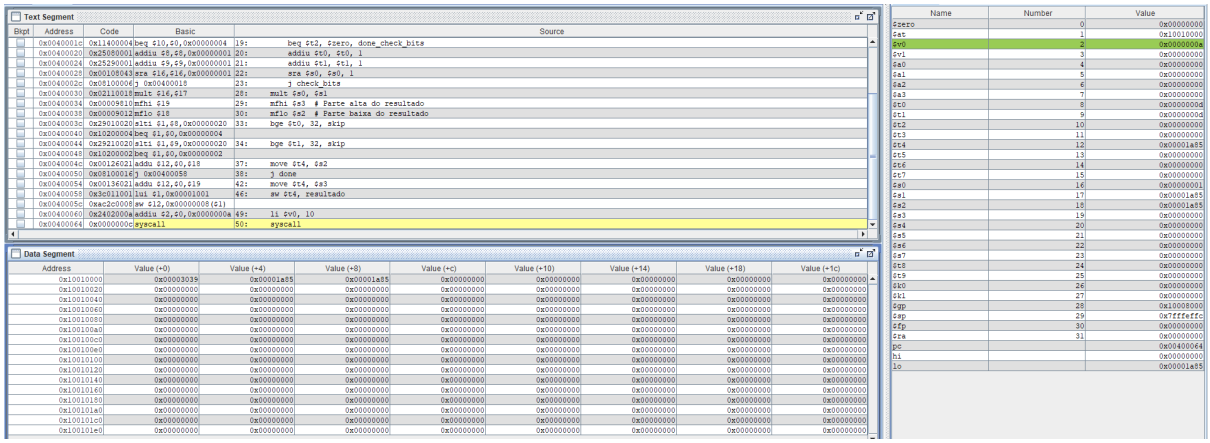
10. A

Parte 4

19.

```
.data
    num1: .word 12345   # Número 1 armazenado na posição de memória
    num2: .word 6789    # Número 2 armazenado na posição de memória
    resultado: .space 8 # Espaço de memória reservado para o resultado

.text
.globl main
main:
    # Carregar os números da memória para $s0 e $s1
    lw $s0, num1
    lw $s1, num2

    # Determinar a quantidade de bits significativos de cada número
    move $t0, $zero
    move $t1, $zero

    check_bits:
        srl $t2, $s0, 1
        beq $t2, $zero, done_check_bits
        addiu $t0, $t0, 1
        addiu $t1, $t1, 1
        sra $s0, $s0, 1
        j check_bits

    done_check_bits:

    # Multiplicação
    mult $s0, $s1
    mfhi $s3  # Parte alta do resultado
    mflo $s2  # Parte baixa do resultado

    # Verificar se a resposta deve ser apenas em $s2 ou em $s2 e $s3
    bge $t0, 32, skip
    bge $t1, 32, skip

    # Se o número de bits significantes for menor que 32, a resposta está apenas em $s2
    move $t4, $s2
    j done
```

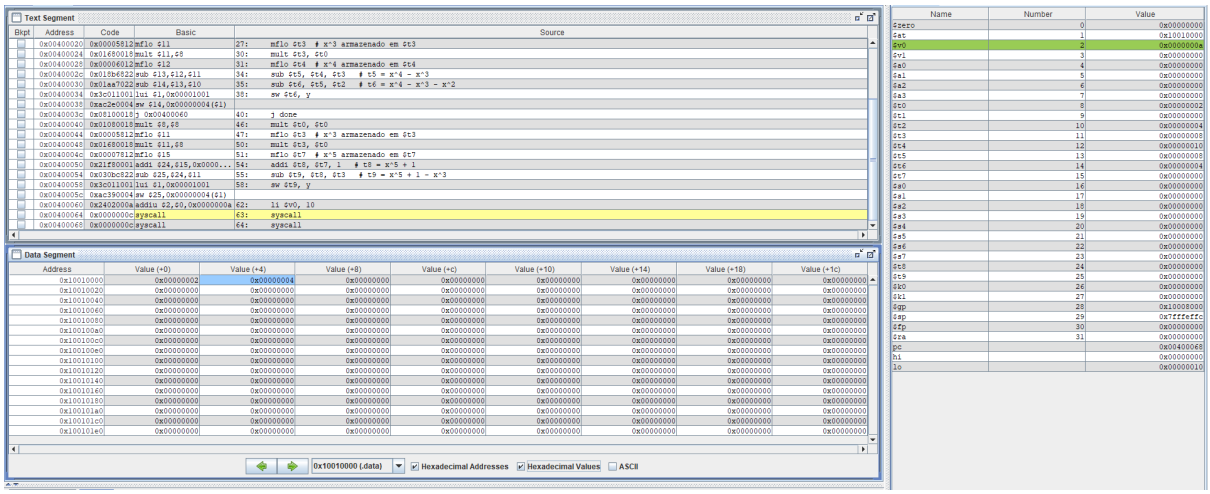
```
skip:
    # Caso contrário, a resposta está em $s2 e $s3
    move $t4, $s3

done:
    # Armazenar o resultado na posição de memória "resultado"
    sw $t4, resultado

    # Finalizar o programa
    li $v0, 10
    syscall
```



20.

```
#Programa 20

.data
    x: .word 2   # Variável x armazenada na posição de memória
    y: .word 0   # Variável y armazenada na posição de memória

.text
.globl main
main:
    # Carregar o valor de x da memória para o registrador $t0
    lw $t0, x

    # Verificar se x é par ou ímpar
    andi $t1, $t0, 1   # Verificar o bit menos significativo de x
    beq $t1, $zero, even  # Se o bit for zero, x é par
    j odd  # Caso contrário, x é ímpar

even:
    # Cálculo de y se x for par: y = x^4 + x^3 - 2x^2

    # Cálculo de x^2
    mult $t0, $t0
    mflo $t2  # x^2 armazenado em $t2

    # Cálculo de x^3
    mult $t0, $t2
    mflo $t3  # x^3 armazenado em $t3

    # Cálculo de x^4
    mult $t3, $t0
    mflo $t4  # x^4 armazenado em $t4

    # Cálculo de y = x^4 + x^3 - 2x^2
    sub $t5, $t4, $t3   # t5 = x^4 - x^3
    sub $t6, $t5, $t2   # t6 = x^4 - x^3 - x^2

    # Armazenar o resultado em y
```

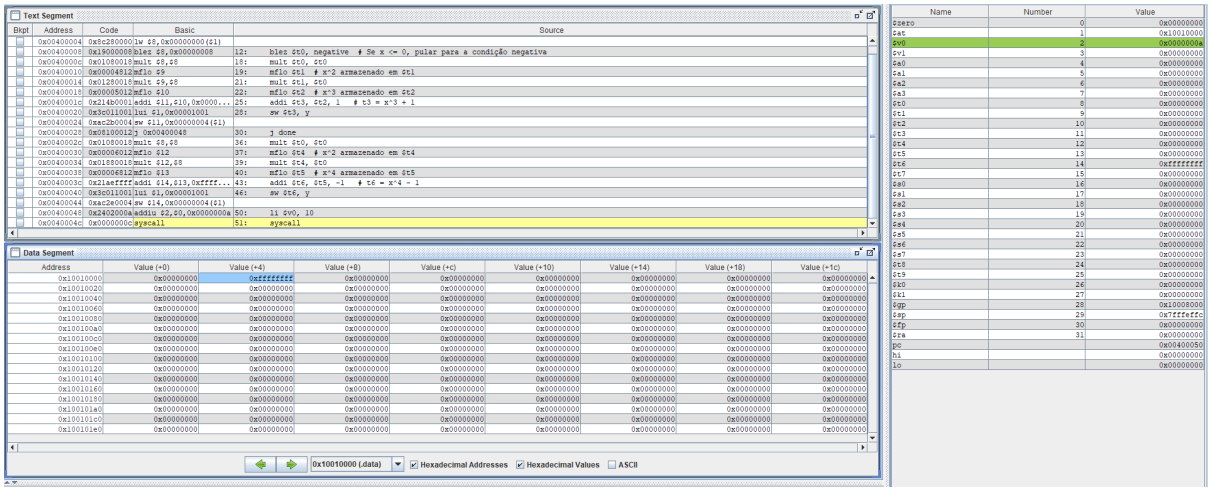
```
38      sw $t6, y
39
40      j done
41
42  odd:
43      # Cálculo de y se x for ímpar: y = x^5 - x^3 + 1
44
45      # Cálculo de x^3
46      mult $t0, $t0
47      mflo $t3  # x^3 armazenado em $t3
48
49      # Cálculo de x^5
50      mult $t3, $t0
51      mflo $t7  # x^5 armazenado em $t7
52
53      # Cálculo de y = x^5 - x^3 + 1
54      addi $t8, $t7, 1   # t8 = x^5 + 1
55      sub $t9, $t8, $t3   # t9 = x^5 + 1 - x^3
56
57      # Armazenar o resultado em y
58      sw $t9, y
59
60  done:
61      # Finalizar o programa
62      li $v0, 10
63      syscall
64      syscall
```



21.

```
.data
    x: .word 0   # Variável x armazenada na posição de memória
    y: .word 0   # Variável y armazenada na posição de memória

.text
.globl main
main:
    # Carregar o valor de x da memória para o registrador $t0
    lw $t0, x

    # Verificar se x > 0
    blez $t0, negative  # Se x <= 0, pular para a condição negativa

positive:
    # Cálculo de y se x > 0: y = x^3 + 1

    # Cálculo de x^3
    mult $t0, $t0
    mflo $t1  # x^2 armazenado em $t1

    mult $t1, $t0
    mflo $t2  # x^3 armazenado em $t2

    # Cálculo de y = x^3 + 1
    addi $t3, $t2, 1   # t3 = x^3 + 1

    # Armazenar o resultado em y
    sw $t3, y

    j done

negative:
    # Cálculo de y se x <= 0: y = x^4 - 1

    # Cálculo de x^4
    mult $t0, $t0
    mflo $t4  # x^2 armazenado em $t4
```

```
    mflo $t4  # x^2 armazenado em $t4

    mult $t4, $t0
    mflo $t5  # x^4 armazenado em $t5

    # Cálculo de y = x^4 - 1
    addi $t6, $t5, -1   # t6 = x^4 - 1

    # Armazenar o resultado em y
    sw $t6, y

done:
    # Finalizar o programa
    li $v0, 10
    syscall
```



Desafio:

```
.data
    num1: .word 0     # Primeiro número armazenado na memória
    num2: .word 0     # Segundo número armazenado na memória
    result_hi: .word 0    # Parte superior do resultado armazenado na memória
    result_lo: .word 0    # Parte inferior do resultado armazenado na memória

.text
.globl main
main:
    # Carregar o primeiro número da memória para o registrador $t0
    lw $t0, num1

    # Carregar o segundo número da memória para o registrador $t1
    lw $t1, num2

    # Multiplicação dos números
    mult $t0, $t1
    mfhi $t2   # Parte superior do resultado armazenada em $t2
    mflo $t3   # Parte inferior do resultado armazenada em $t3

    # Armazenar o resultado nas posições de memória result_hi e result_lo
    sw $t2, result_hi
    sw $t3, result_lo

    # Finalizar o programa
    li $v0, 10
    syscall
```