PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Instituto de Ciências Exatas e de Informática

Exercício Prático — Arquitetura de Computadores II

Vitor Costa Oliveira Rolla

Pontifícia Universidade Católica de Minas Gerais

19 de outubro de 2025

Parte 1 - Exercícios

O que é um arquivo fonte?

Um arquivo de texto que contém instruções de linguagem de programação.

O que é registrador?

É uma parte do processador que possui um padrão de bits.

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

O caractere que inicia um comentário em assembly é o caractere "#".

Quantos bits há em cada instrução de máquina MIPS?

Em cada instrução de máquina MIPS há um total de 32 bits.

O que é o contador de programa?

O contador de programa (program counter) é parte do processador que contém o endereço da próxima instrução de máquina para ser obtida.

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

Ao executarmos uma instrução, será adicionado 4 para o contador de programa.

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

Deretiva é uma declaração que diz o montador algo sobre o que o programador quer, mas não corresponde diretamente a uma instrução de máquina.

O que é um endereço simbólico?

O endereço simbólico é um nome usado no código-fonte em linguagem assembly para um local na memória.

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

Ao executar, a primeira instrução se localiza no endereço 0x00400000.

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

Tal operando é nomeado como operando imediato.

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 utilizada nesse contexto é a operação bitwise.

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

Ambos operandos devem vir de registros.

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

Os dados são estendidos em zero à esquerda por 16 bits.

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

ori $5,$0, 48

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

A resposta é não, pois se trata de operações lógicas e não aritméticas.

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

A instrução que realiza tal operação é andi $8,$8,0xFF, pois ao realizar o and entre os dados, ele mantém apenas o que for 1 entre os operandos.

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

O resultado é o contrário do original.

Todas as instruções de máquina têm os mesmos campos?

Não. Diferentes de instruções de máquina possuem campos diferentes.

Parte 2 - Exercícios

Programas

Programa - 01

programa_1.asm*

```
# ori:  Registrador Destino, Registrador A, Imediato -> Dest = A | I
# and:  Registrador Destino, Registrador A, Registrador B -> Dest = A & B
# xor:  Registrador Destino, Registrador A, Registrador B -> Dest = A XOR B
# nor:  Registrador Destino, Registrador A, Registrador B -> Dest = A NOR B

.text
.globl main


main:

        # Atribuição de valores
        ori $s0, $zero, 2 # a
        ori $s1, $zero, 3 # b
        ori $s2, $zero, 4 # c
        ori $s3, $zero, 5 # d

        # Somando valores
        add $t0, $s0, $s1 # Atribuição -> t0: s0 + s1 (a + b)
        add $t1, $s2, $s3 # Atribuição -> t1: s2 + s3 (c + d)
        sub $s4, $t0, $t1 # Atribuição -> (x)s4: t0 - t1

        sub $t3, $s0, $s1 # Atribuição -> t3: s0 - s1 (a - b)
        add $s5, $t3, $s4 # Atribuição -> (y)s5: t3 - s4 (t3 + x)
        sub $s1, $s4, $s5 # Atribuição -> s1 = x - y
```

Figura 1: Imagem do programa 1

Programa - 02

```

# Aritméticas
# add:  Registrador Destino, Registrador A, Registrador B -> Dest = A + B
# addi: Registrador Destino, Registrador A, Imediato -> Dest = A + Imediato
# sub:  Registrador Destino, Registrador A, Registrador B -> Dest = A - B

# Lógicas
# andi: Registrador Destino, Registrador A, Imediato -> Dest = A & I
# ori:  Registrador Destino, Registrador A, Imediato -> Dest = A | I
# and:  Registrador Destino, Registrador A, Registrador B -> Dest = A & B
# xor:  Registrador Destino, Registrador A, Registrador B -> Dest = A XOR B
# nor:  Registrador Destino, Registrador A, Registrador B -> Dest = A NOR B

.text
.globl main

main:

        ori  $s0, $zero, 1 # Atribuição -> x = 1
        add  $t0, $s0, $s0 # Atribuição -> t0 = 2x
        add  $t0, $t0, $t0 # Atribuição -> t0 = 4x
        add  $t0, $t0, $s0 # Atribuição -> t0 = 5x
        addi $s1, $t0, 15  # Atribuição -> y = 5x + 15
```

Figura 2: Imagem do programa 2

Programa - 03

```
main:

        ori $s0, $zero, 3 # Atribuição s0 = 15
        ori $s1, $zero, 4 # Atribuição s1 = 4

        add $t0, $s0, $s0 # Atribuição t0 = 2x
        add $t0, $t0, $t0 # Atribuição t0 = 4x
        add $t0, $t0, $t0 # Atribuição t0 = 8x
        add $t0, $t0, $t0 # Atribuição t0 = 16x
        sub $t0, $t0, $s0 # Atribuição t0 = 15x

        add $t1, $s1, $s1 # Atribuição t1 = 2y
        add $t1, $t1, $t1 # Atribuição t1 = 4y
        add $t1, $t1, $t1 # Atribuição t1 = 8y
        add $t1, $t1, $t1 # Atribuição t1 = 16y
        add $t1, $t1, $t1 # Atribuição t1 = 32y
        add $t1, $t1, $t1 # Atribuição t1 = 64y
        add $t1, $t1, $s1 # Atribuição t1 = 65y
        add $t1, $t1, $s1 # Atribuição t1 = 66y
        add $t1, $t1, $s1 # Atribuição t1 = 67y

        add $t2, $t0, $t1 # Atribuição t2 = t0 + t1
        add $t2, $t2, $t2 # Atribuição t2 = 2 * t2
        add $t2, $t2, $t2 # Atribuição t2 = 4 * t2

        add $s2, $zero, $t2 # Atribuição z = (15x + 67y) * 4
```

Figura 3: Imagem do programa 3

Programa - 04

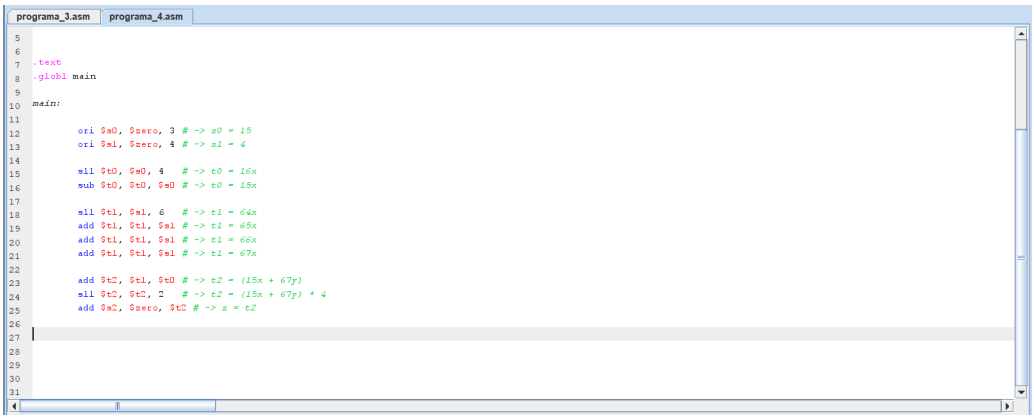

Figura 4: Imagem do programa 4

Programa - 05

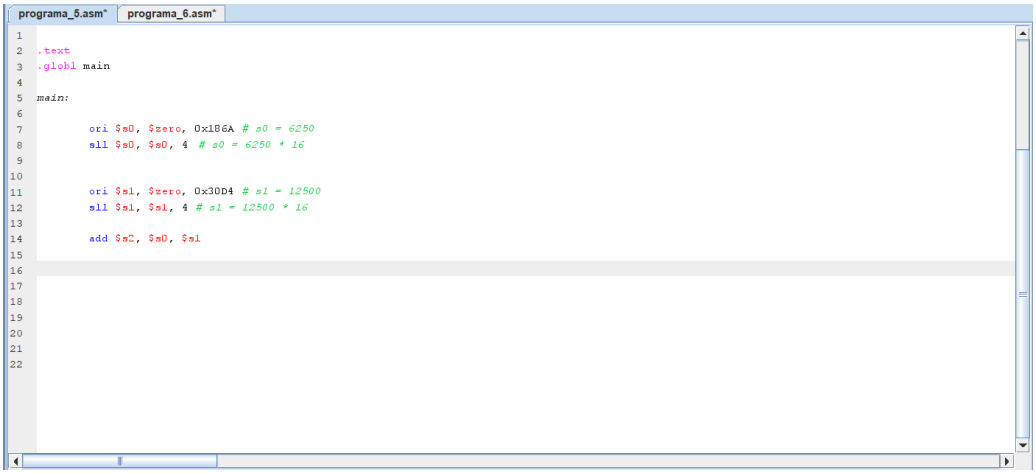

Figura 5: Imagem do programa 5

Programa - 06

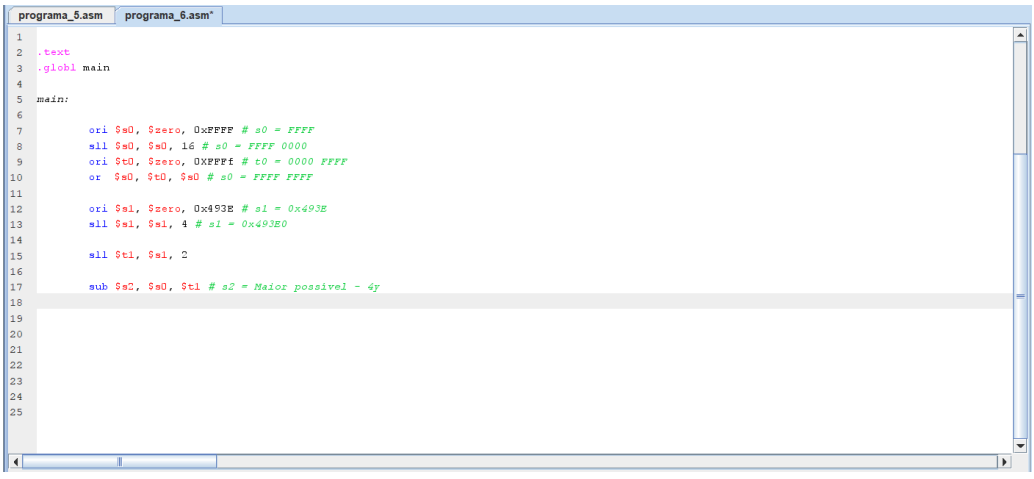

Figura 6: Imagem do programa 6

Programa - 07

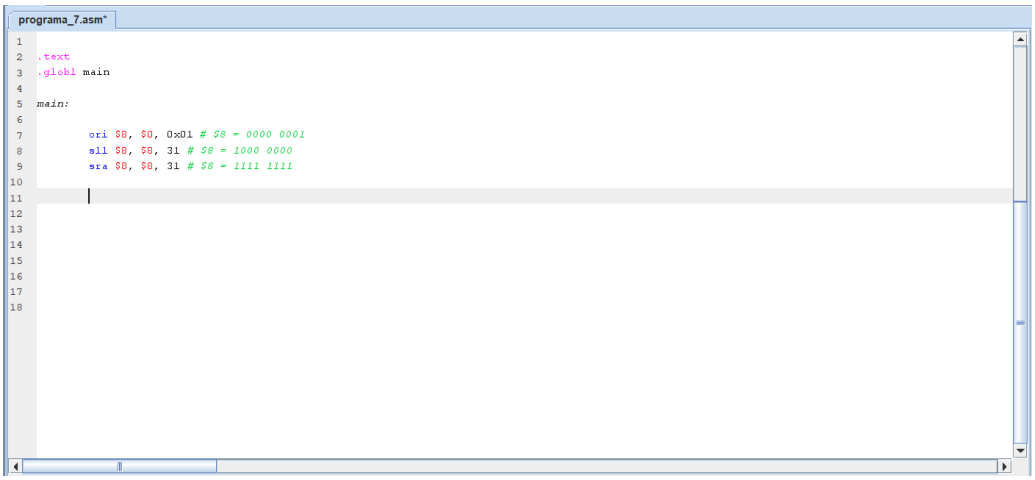

Figura 7: Imagem do programa 7

Programa - 08

Figura 8: Imagem do programa 8

Programa - 09

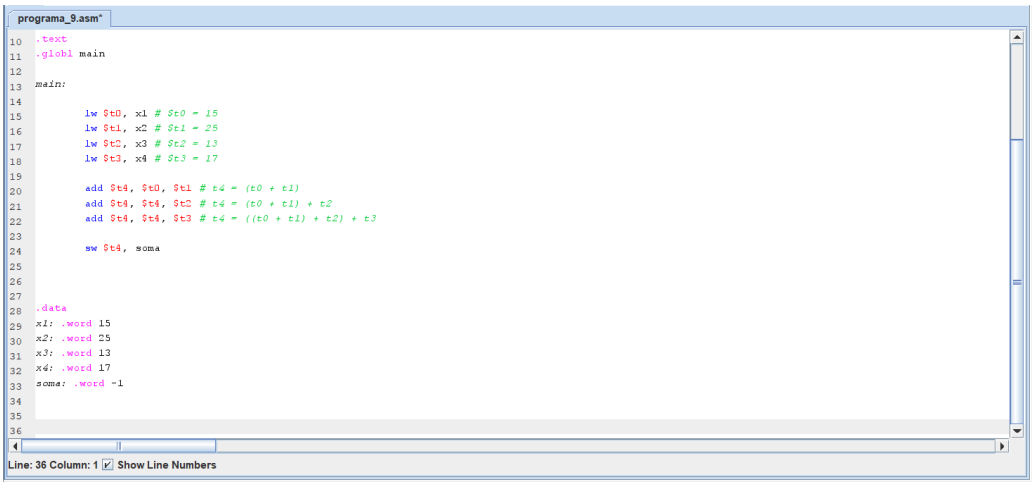

Figura 9: Imagem do programa 9

Programa - 10

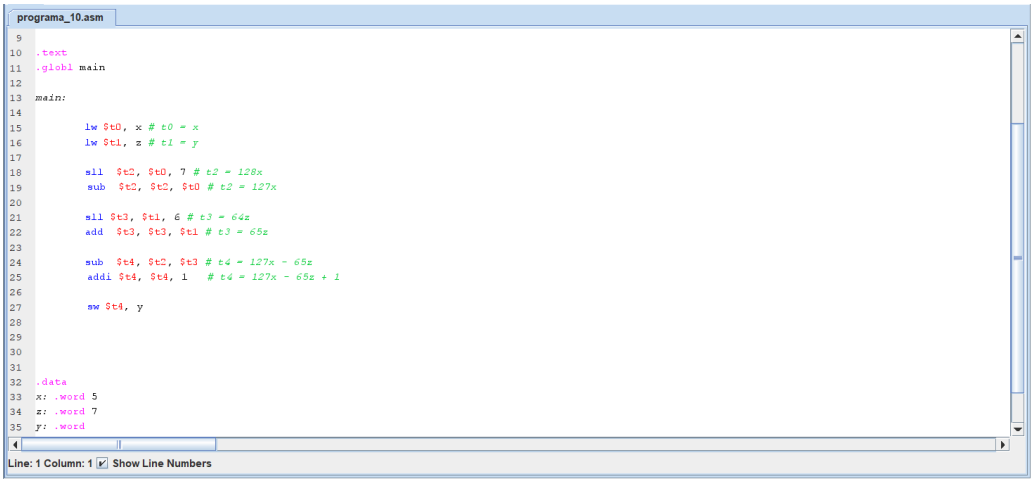

Figura 10: Imagem do programa 10

Programa - 11

programa_11.asm

```
8
9
10  .text
11  .globl main
12
13  main:
14
15          lw $t0, x # t0 = x
16          lw $t1, z # t1 = z
17
18          sub $t2, $t0, $t1 # t2 = x - z
19
20          ori $t3, $zero, 0x493E # t3 = 0x493E
21          sll $t3, $t3, 4 # t3 = 0x493E0
22
23          add $t3, $t2, $t3 # t3 = (x - z) + 300000
24          sw $t3, y
25
26
27
28
29  .data
30  x: .word 100000
31  z: .word 200000
32  y: .word 0    # esse valor deverá ser sobrescrito após a execução do programa.
33
34
```

Line: 1 Column: 1 Show Line Numbers

Figura 11: Imagem do programa 11

Programa - 12

programa_12.asm

```
# Descrição das instruções


# Memória
# lw Registrador, deslocamento(Endereço ou Label)
# sw Registrador, deslocamento(Endereço ou Label)

.text
.globl main

main:

        lw $t0, ptr3
        lw $t1, ($t0)
        lw $t2, ($t1)
        lw $s0, ($t2)

        sll $s0, $s0, 1

        sw $s0, x

.data
x: .word 5
ptr1: .word x
ptr2: .word ptr1
ptr3: .word ptr2
```

Figura 12: Imagem do programa 12

Programa - 13

programa_13.asm

```
.text
.globl main

main:

        lw $t0, A # t0 = A

        sra $t1, $t0, 31
        beq  $t1, $zero, positivo
        sub  $t0, $zero, $t0
        sw   $t0, A

positivo:
    # FIM

.data
A: .word -7
```

Figura 13: Imagem do programa 13

Programa - 14

programa_14.asm

```
9
10  .text
11  .globl main
12
13  main:
14
15          lw $t0, A # t0 = A
16
17          andi $t1, $t0, 1  # t1 = t0 & 1
18          beq  $t1, $zero, par # par
19          bne  $t1, $zero, impar # impar
20
21  par:
22      la   $t2, B
23      sw   $zero, ($t2)
24
25  impar:
26      la   $t2, B
27      addi $t3, $zero, 1
28      sw   $t3, ($t2)
29
30  .data
31  A: .word 5
32  B: .word
33
34
35
```

Line: 35 Column: 1 Show Line Numbers

Figura 14: Imagem do programa 14

Programa - 15

programa_15

```
.text
.globl main

main:

        lw $t0, a
        lw $t1, i
        addi $s0, $0, 0

        addi $t2, $t0, 400

        loop:
              beq $t0, $t2, fim

              sll  $t3, $t1, 1
              addi $t3, $t3, 1

              sw $t3, ($t0)

              add  $s0, $s0, $t3

              addi $t1, $t1, 1
              addi $t0, $t0, 4

              j loop

        fim:

                sw $s0, ($t2)

.data
a: .word 0x10010008
i: .word 0
```

Figura 15: Imagem do programa 15

Programa - 16

programa_16.asm

```
.text
.globl main
main:

    lw   $t0, x
    lw   $t1, y
    lw   $t2, z

    mult $t0, $t1
    mflo $t4

    div  $t4, $t2
    mflo $t5

    sw   $t5, resultado


.data
x: .word 0x186A00
y: .word 0x13880
z: .word 0x61A80
resultado: .word 0
```

Figura 16: Imagem do programa 16

Programa - 17

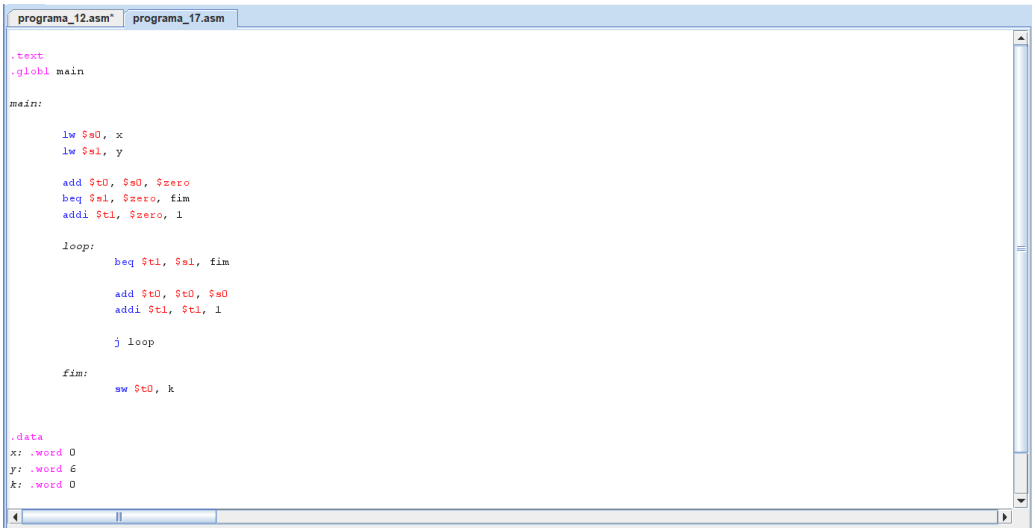

Figura 17: Imagem do programa 17

Programa - 18

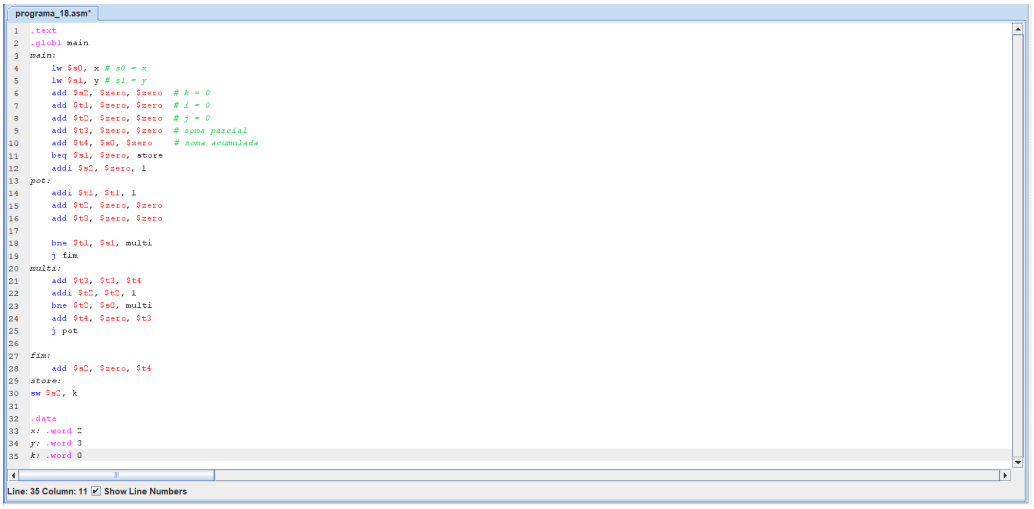

Figura 18: Imagem do programa 18

Programa - 19

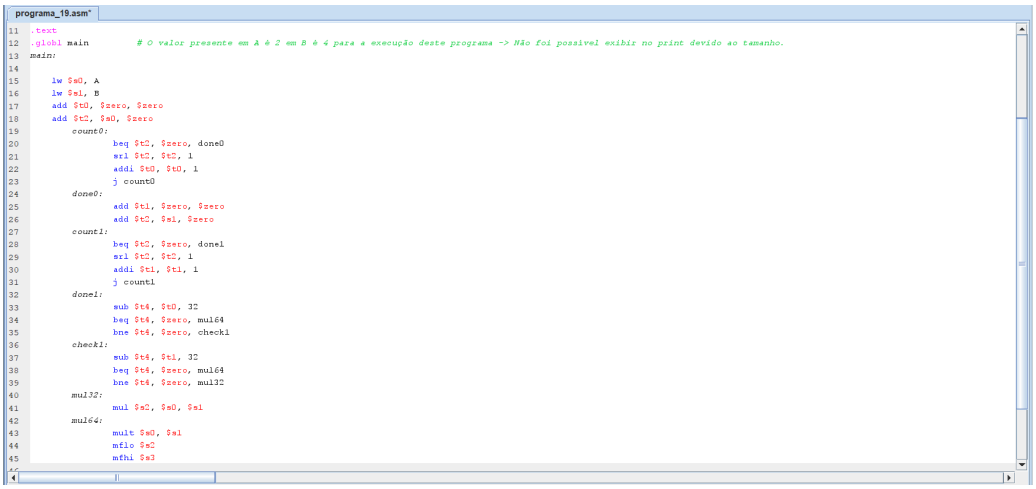

Figura 19: Imagem do programa 19

Programa - 20

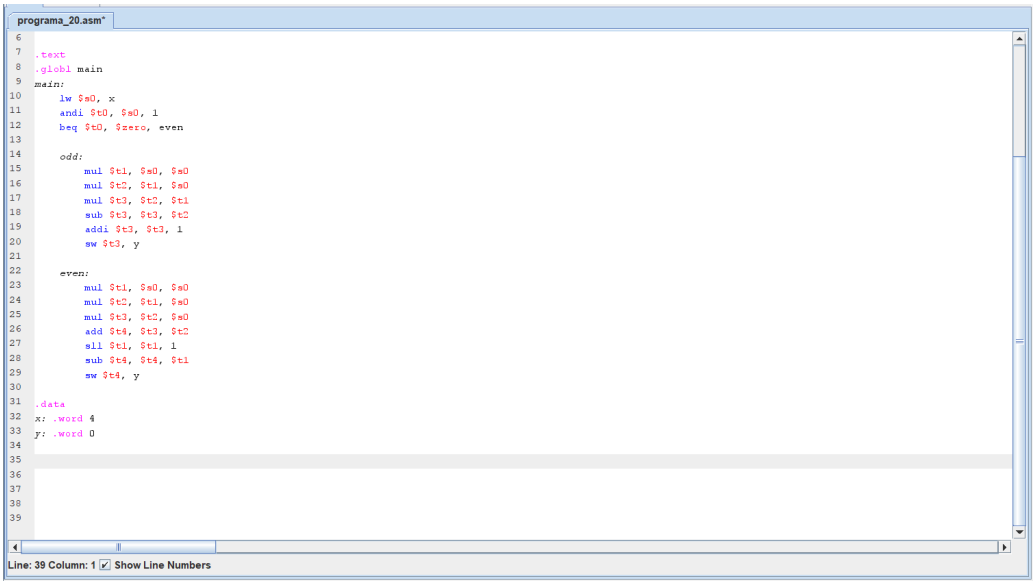

Figura 20: Imagem do programa 20

Programa - 21

programa_21.asm

```
9
10  .text
11  .globl main
12  main:
13      lw $s0, x
14      sra $t0, $s0, 31
15      beq $t0, $zero, positive
16
17      non_positive:
18          mult $s0, $s0
19          mflo $t1
20          mult $t1, $s0
21          mflo $t2
22          mult $t2, $s0
23          mflo $t3
24          addi $t3, $t3, -1
25          sw $t3, y
26
27
28      positive:
29          mult $s0, $s0
30          mflo $t1
31          mult $t1, $s0
32          mflo $t2
33          addi $t2, $t2, 1
34          sw $t2, y
35
36  .data
37  x: .word 3
38  y: .word 0
39
40
41
42
43
```

Figura 21: Imagem do programa 21

Execuções

Execução - 01

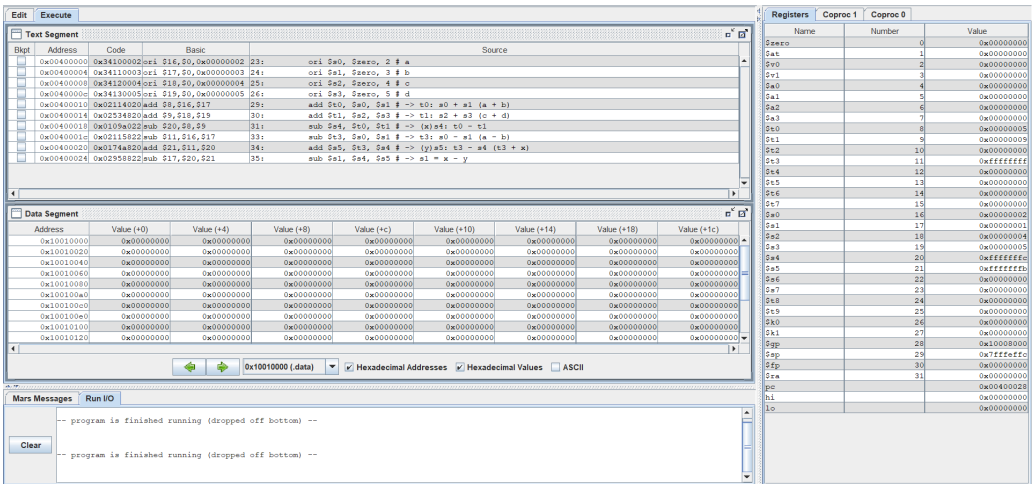

Figura 22: Imagem da execução programa 1

Execução - 02

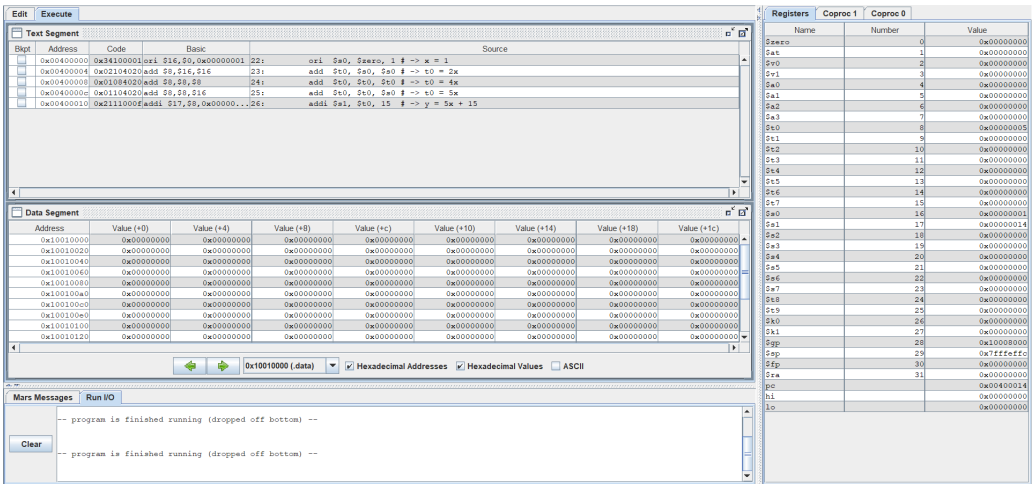

Figura 23: Imagem do execução programa 2

Execução - 03

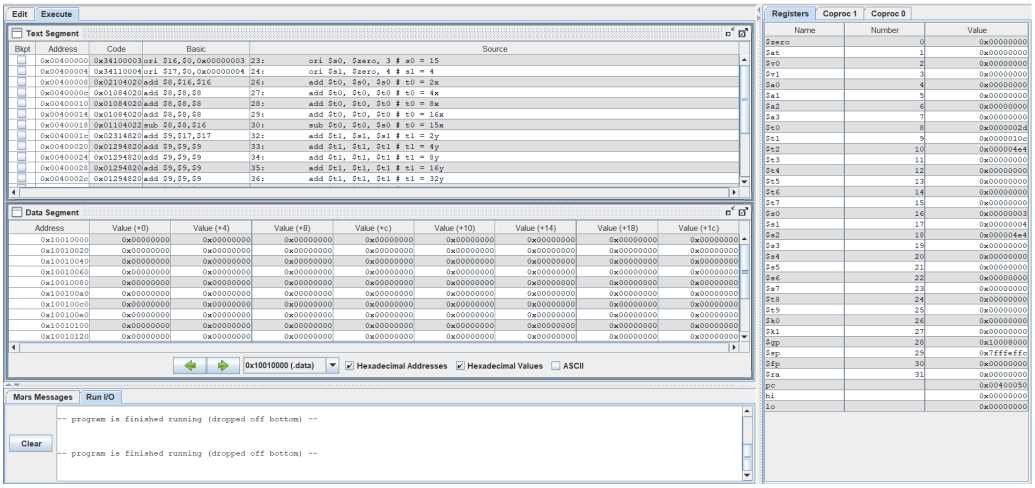

Figura 24: Imagem do execução programa 3

Execução - 04

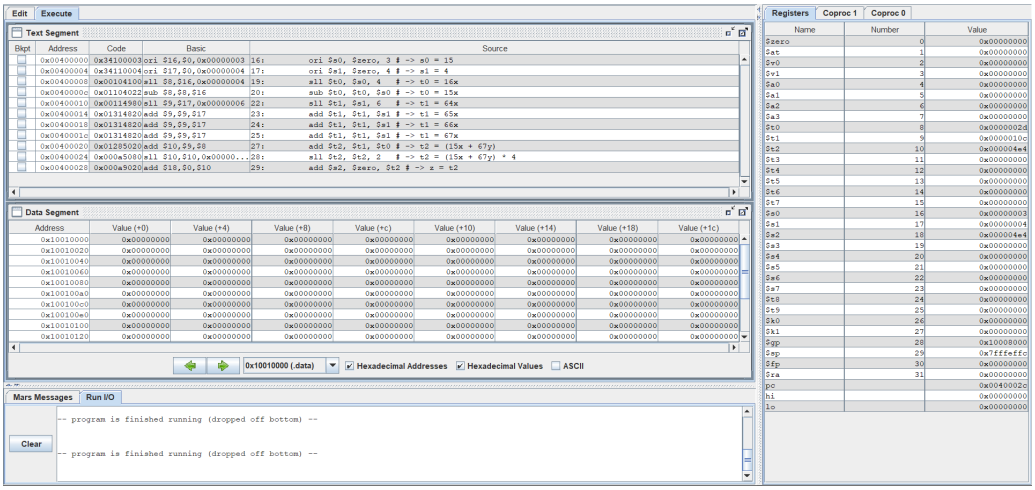

Figura 25: Imagem do execução programa 4

Execução - 05

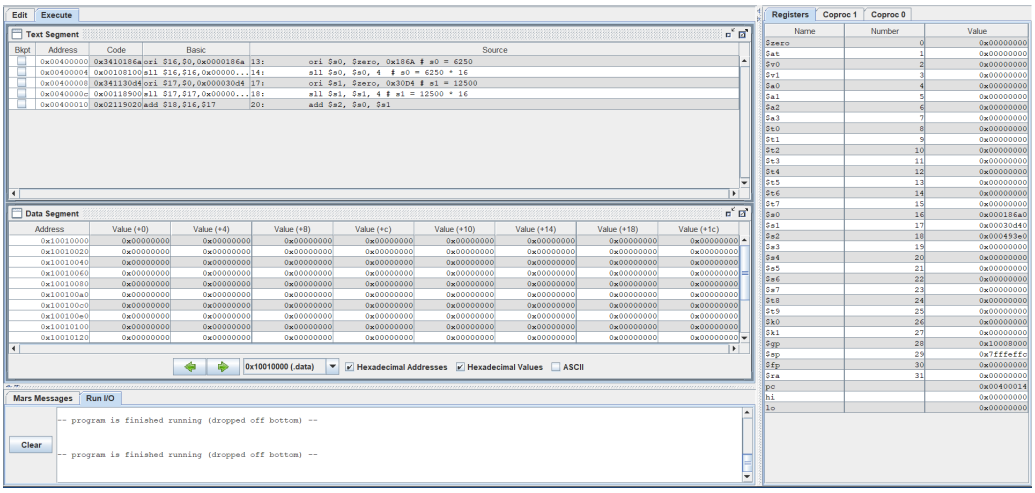

Figura 26: Imagem do execução programa 5

Execução - 06

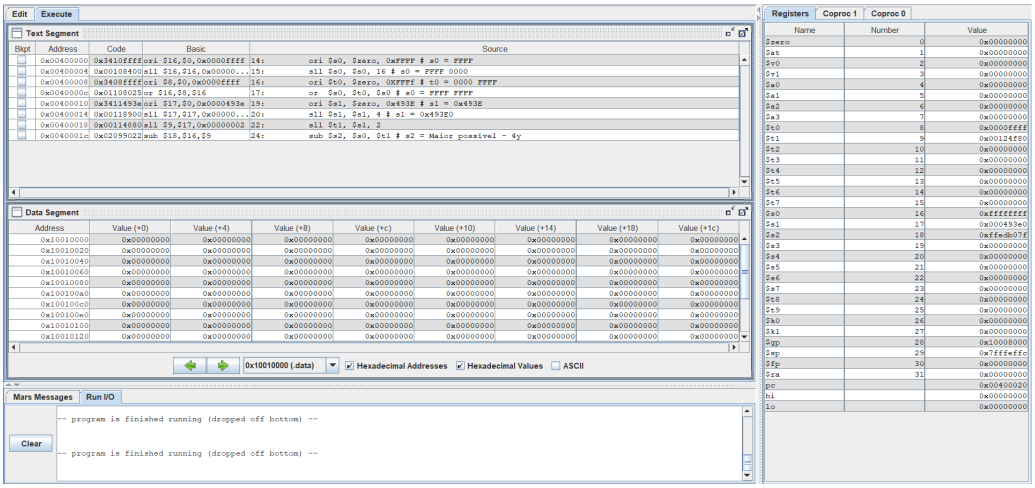

Figura 27: Imagem do execução programa 6

Execução - 07

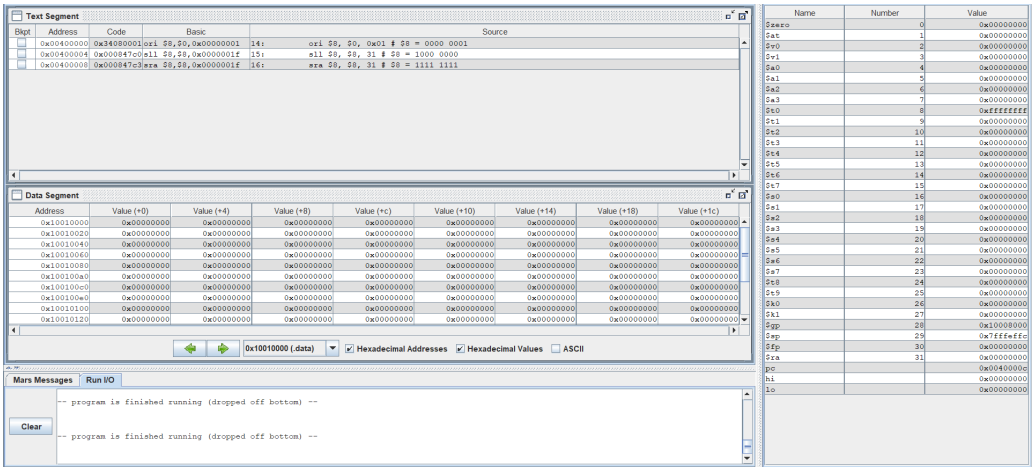

Figura 28: Imagem do execução programa 7

Execução - 08

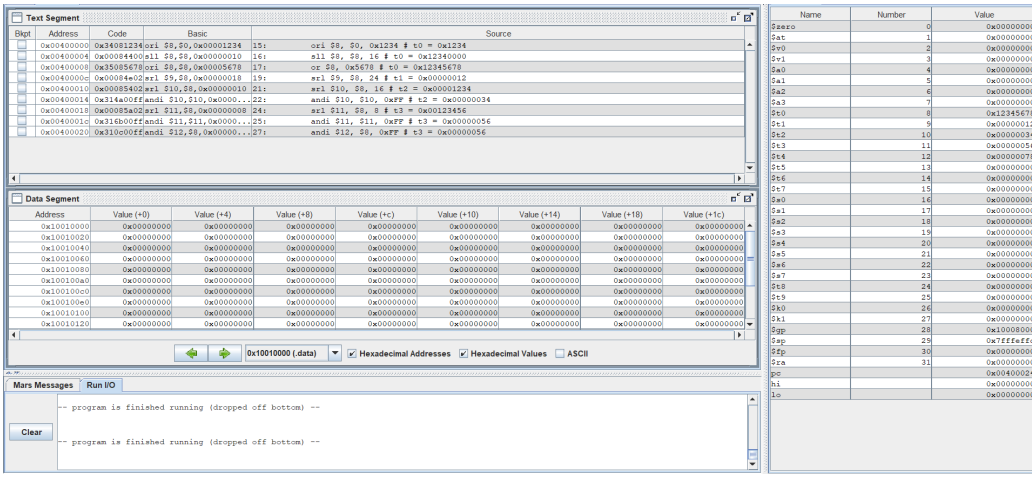

Figura 29: Imagem do execução programa 8

Execução - 09

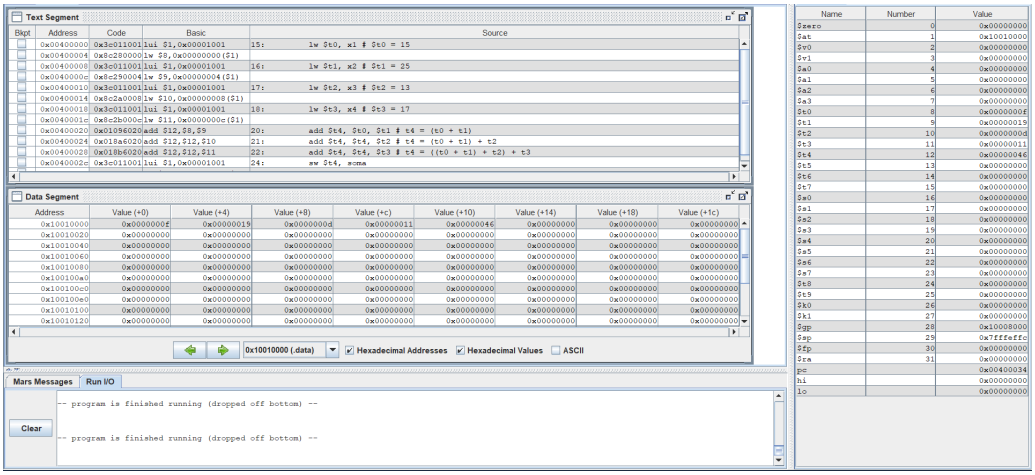

Figura 30: Imagem do execução programa 9

Execução - 10

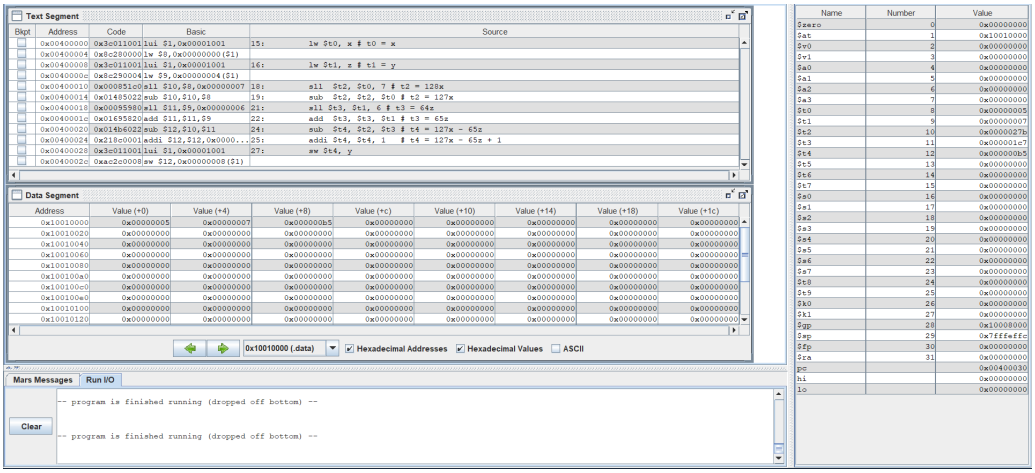

Figura 31: Imagem do execução programa 10

Execução - 11

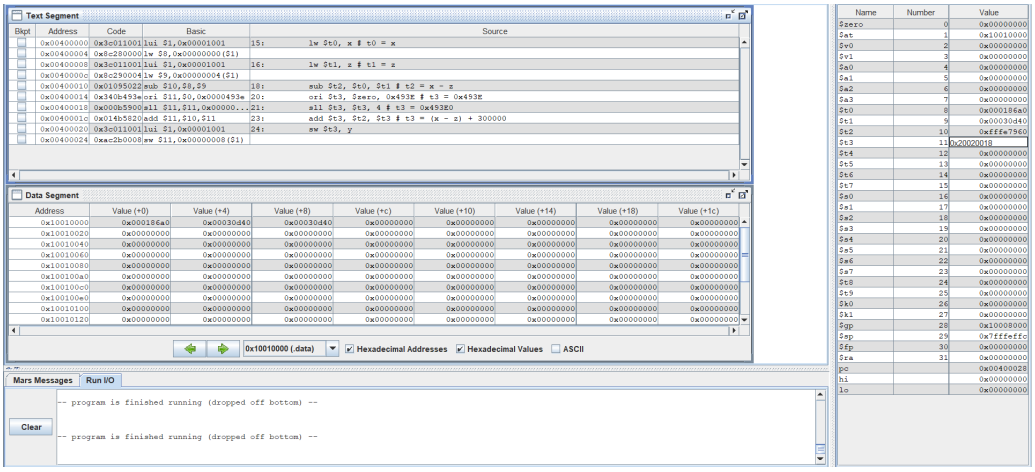

Figura 32: Imagem do execução programa 11

Execução - 12

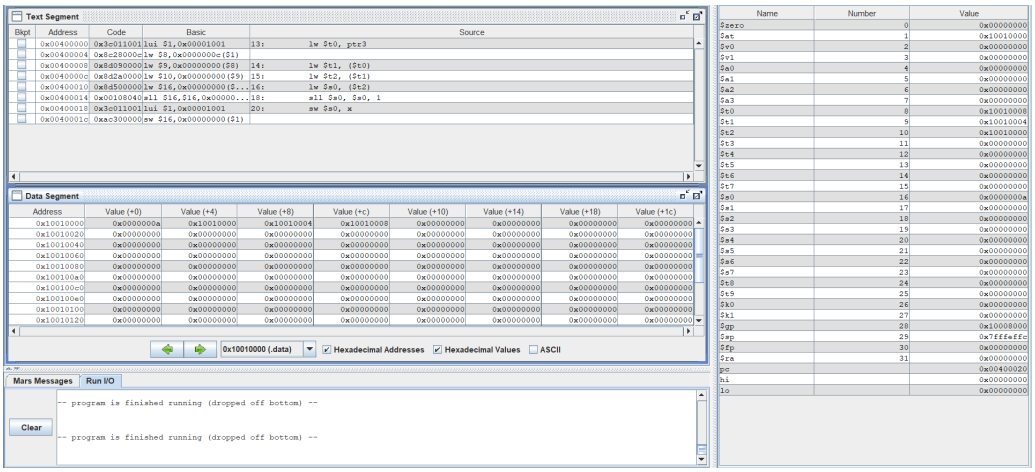

Figura 33: Imagem do execução programa 12

Execução - 13

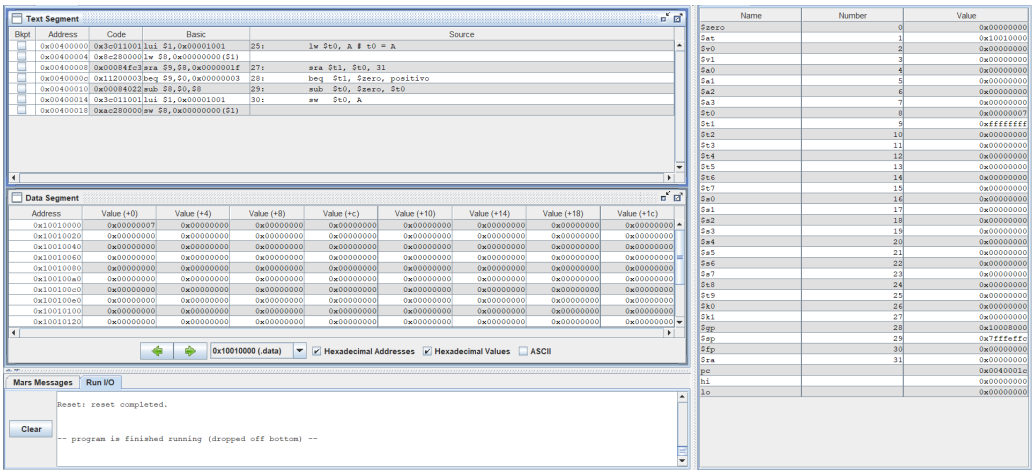

Figura 34: Imagem do execução programa 13

Execução - 14

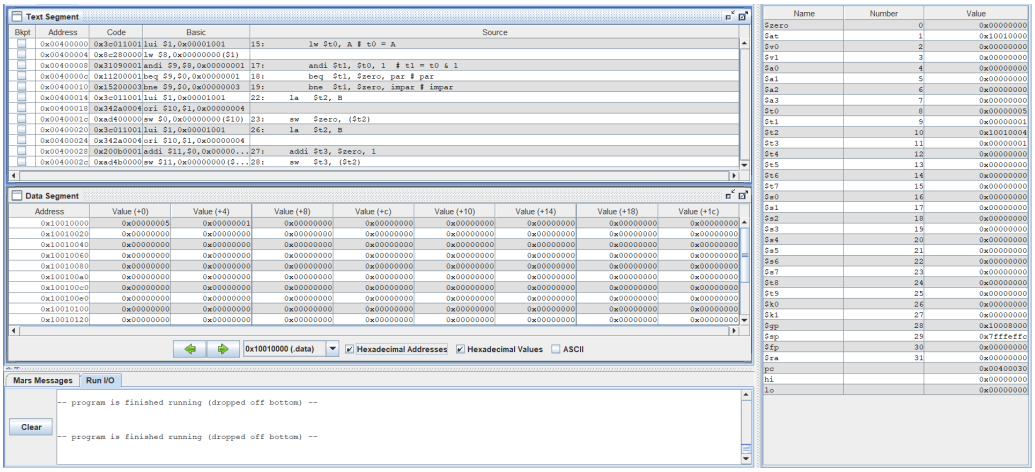

Figura 35: Imagem do execução programa 14

Execução - 15

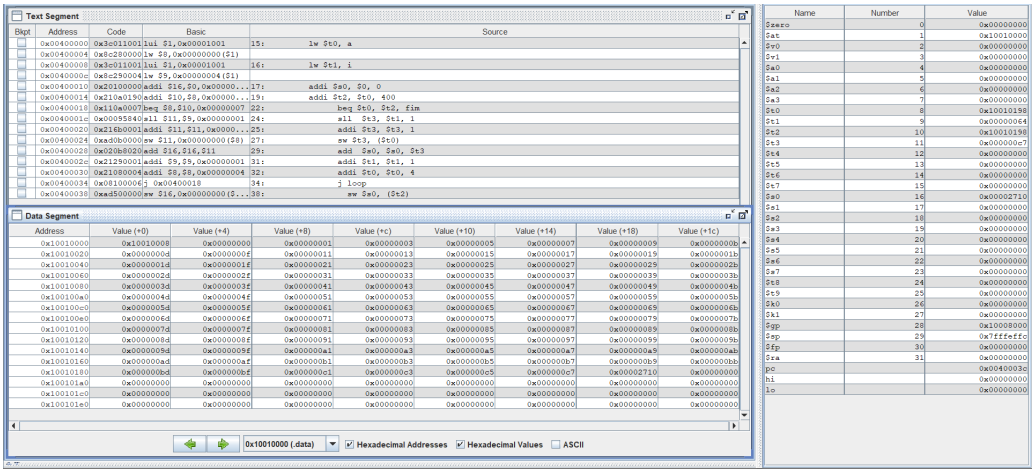

Figura 36: Imagem do execução programa 15

Execução - 16

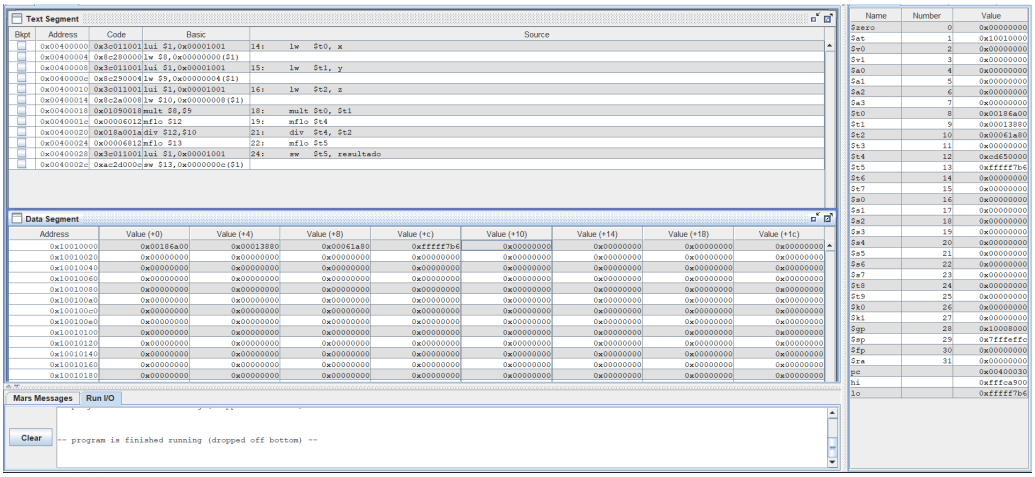

Figura 37: Imagem do execução programa 16

Execução - 17

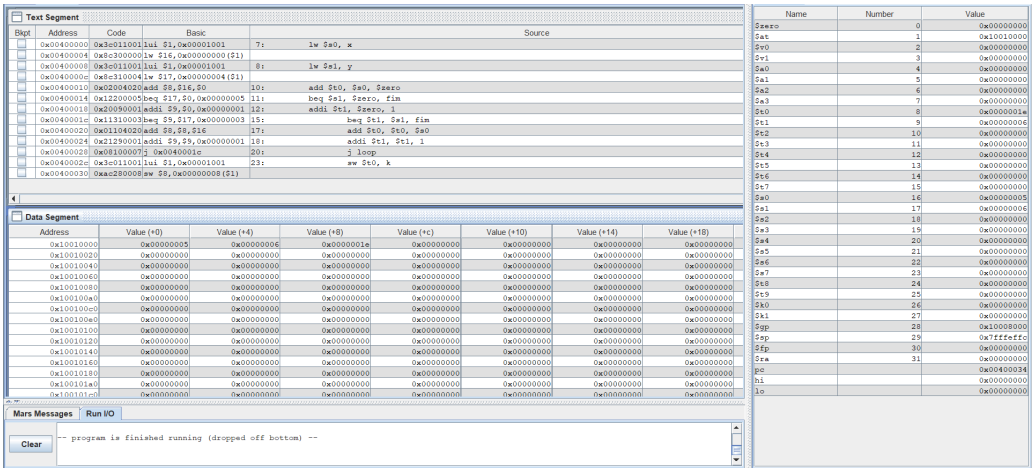

Figura 38: Imagem do execução programa 17

Execução - 18

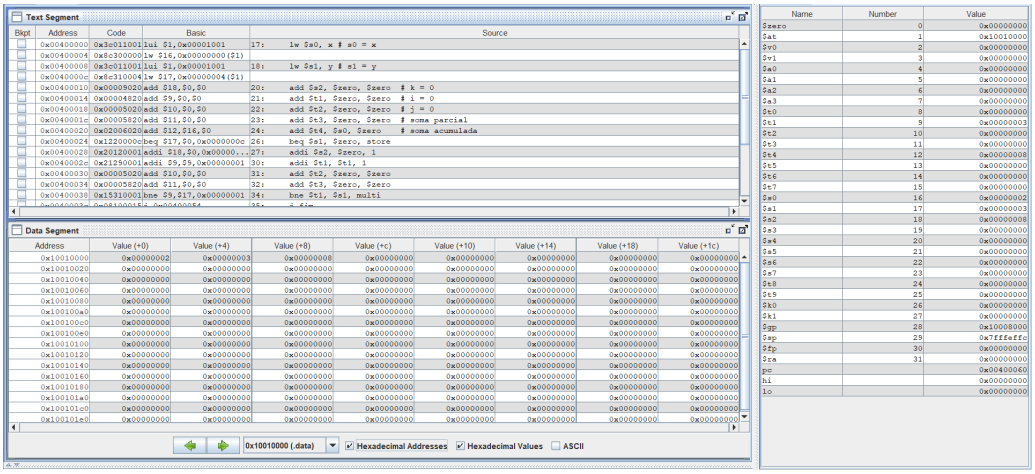

Figura 39: Imagem da execução programa 18

Execução - 19

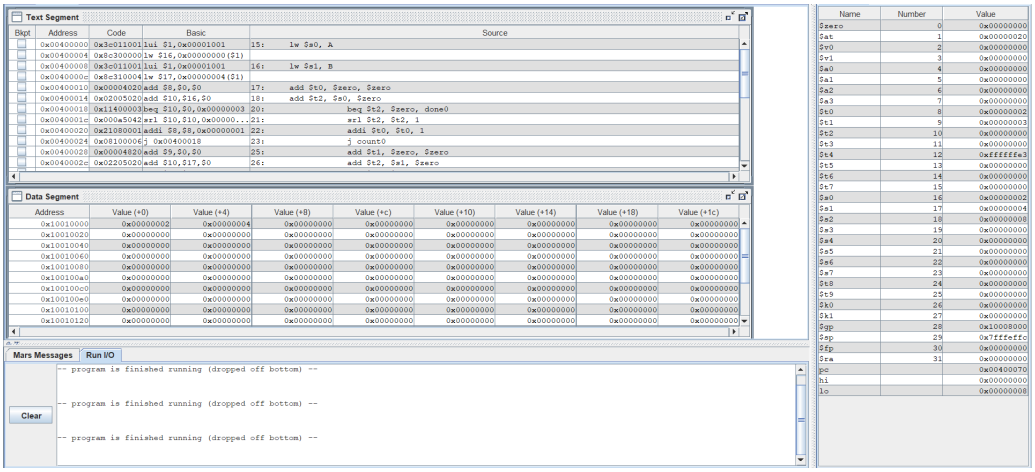

Figura 40: Imagem do execução programa 19

Execução - 20

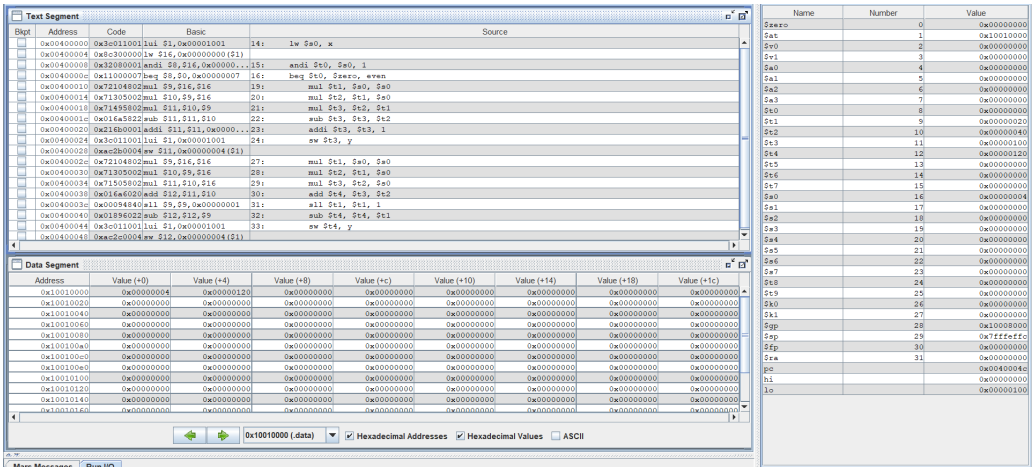

Figura 41: Imagem do execução programa 20

Execução - 21

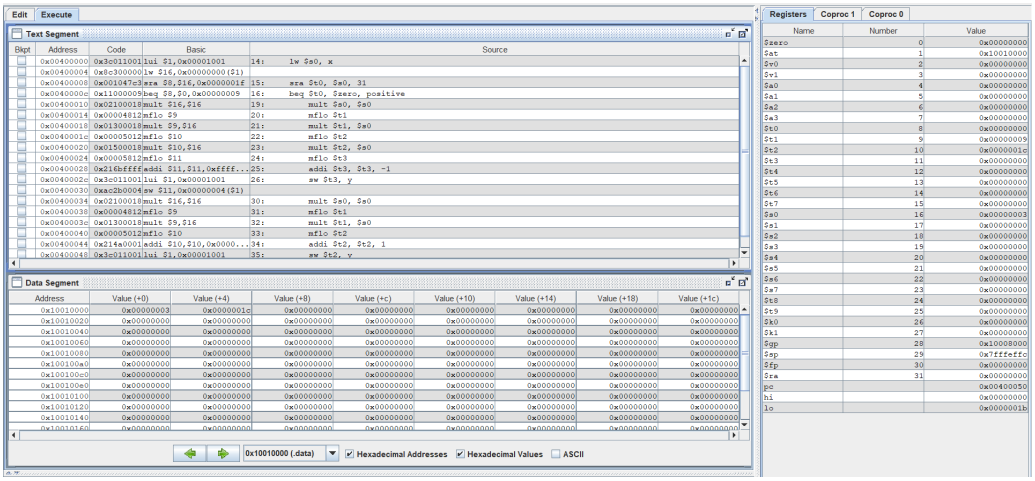

Figura 42: Imagem do execução programa 21

Peguntas a serem respondidas

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

A resposta correta é 64 bits

Quais os registradores que armazenam os resultados na multiplicação?

Os registradores que armazenam os resultados de uma multiplicação são o hi e o lo, que guardam os bits mais significativos e menos significativos, respectivamente.

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

Em MIPS, a operação realizada para multiplicar inteiros em complemento de dois é a operação mult.

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

Para mover o bit menos significativo para o registrador 8, é necessário utilizar mflo.

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

Para uma divisão inteira de 32 bits, o quociente pode armazenar até os 32 bits.

Após a instrução div, qual registrador possui o quociente?

O registrador que armazenará o quociente é o registrador lo.

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

Tal instrução diz a respeito sobre a instrução div.

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

O resultado de uma instrução sra aplicada ao número 1001 1011, deslocando-se 2 bits, resulta em: 1110 0110

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

Se o inteiro for unsigned, o shift pode ocasionar um valor errado. Se o inteiro for signed, o shift o divide por 2.