



**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 caractere 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?**

Diretiva 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*
12 # ori: Registrador Destino, Registrador A, Imediato -> Dest = A | I
13 # and: Registrador Destino, Registrador A, Registrador B -> Dest = A & B
14 # xor: Registrador Destino, Registrador A, Registrador B -> Dest = A XOR B
15 # nor: Registrador Destino, Registrador A, Registrador B -> Dest = A NOR B
16
17 .text
18 .globl main
19
20
21 main:
22
23     # Atribuição de valores
24     ori $t0, $zero, 2 # a
25     ori $t1, $zero, 3 # b
26     ori $t2, $zero, 4 # c
27     ori $t3, $zero, 5 # d
28
29     # Somando valores
30     add $t0, $t0, $t1 # Atribuição -> t0: s0 + s1 (a + b)
31     add $t1, $t2, $t3 # Atribuição -> t1: s2 + s3 (c + d)
32     sub $t4, $t0, $t1 # Atribuição -> (x)s4: t0 - t1
33
34     sub $t3, $t0, $t1 # Atribuição -> t3: s0 - s1 (a - b)
35     add $t5, $t3, $t5 # Atribuição -> (y)s5: t3 - s4 (t3 + x)
36     sub $t1, $t4, $t5 # Atribuição -> s1 = x - y
37
38
```

Figura 1: Imagem do programa 1

#### Programa - 02

```
2
3
4 # Aritméticas
5 # add: Registrador Destino, Registrador A, Registrador B -> Dest = A + B
6 # addi: Registrador Destino, Registrador A, Imediato -> Dest = A + Imediato
7 # sub: Registrador Destino, Registrador A, Registrador B -> Dest = A - B
8
9
10 # Lógicas
11 # andi: Registrador Destino, Registrador A, Imediato -> Dest = A & I
12 # orisi: Registrador Destino, Registrador A, Imediato -> Dest = A | I
13 # andi: Registrador Destino, Registrador A, Registrador B -> Dest = A & B
14 # nori: Registrador Destino, Registrador A, Registrador B -> Dest = A XOR B
15 # nori: Registrador Destino, Registrador A, Registrador B -> Dest = A NOR B
16
17 .text
18 .globl main
19
20 main:
21
22     ori $t0, $zero, 1 # Atribuição -> x = 1
23     add $t0, $t0, $t0 # Atribuição -> t0 = 2x
24     add $t0, $t0, $t0 # Atribuição -> t0 = 4x
25     add $t0, $t0, $t0 # Atribuição -> t0 = 8x
26     addi $t1, $t0, 15 # Atribuição -> y = 5x + 15
27
28
```

Figura 2: Imagem do programa 2

#### Programa - 03

```
21 main:
22
23     ori $t0, $zero, 3 # Atribuição s0 = 15
24     ori $t1, $zero, 4 # Atribuição s1 = 4
25
26     add $t0, $t0, $t0 # Atribuição t0 = 2x
27     add $t0, $t0, $t0 # Atribuição t0 = 4x
28     add $t0, $t0, $t0 # Atribuição t0 = 8x
29     add $t0, $t0, $t0 # Atribuição t0 = 16x
30     sub $t0, $t0, $t0 # Atribuição t0 = 15x
31
32     add $t1, $t1, $t1 # Atribuição t1 = 2y
33     add $t1, $t1, $t1 # Atribuição t1 = 4y
34     add $t1, $t1, $t1 # Atribuição t1 = 8y
35     add $t1, $t1, $t1 # Atribuição t1 = 16y
36     add $t1, $t1, $t1 # Atribuição t1 = 32y
37     add $t1, $t1, $t1 # Atribuição t1 = 64y
38     add $t1, $t1, $t1 # Atribuição t1 = 128y
39     add $t1, $t1, $t1 # Atribuição t1 = 256y
40     add $t1, $t1, $t1 # Atribuição t1 = 512y
41
42     add $t2, $t0, $t1 # Atribuição t2 = t0 + t1
43     add $t2, $t2, $t2 # Atribuição t2 = 2 * t2
44     add $t2, $t2, $t2 # Atribuição t2 = 4 * t2
45
46     add $t3, $zero, $t2 # Atribuição z = (15x + 67y) * 4
47
```

Figura 3: Imagem do programa 3

## Programa - 04

```
programa_3.asm programma_4.asm
5
6
7 .text
8 .globl main
9
10 main:
11     ori $s0, $zero, 3 # -> s0 = 15
12     ori $s1, $zero, 4 # -> s1 = 4
13
14     sll $t0, $s0, 4 # -> t0 = 16x
15     sub $t0, $t0, $s0 # -> t0 = 15x
16
17     sll $t1, $s1, 6 # -> t1 = 64x
18     add $t1, $t1, $s1 # -> t1 = 65x
19     add $t1, $t1, $s1 # -> t1 = 66x
20     add $t1, $t1, $s1 # -> t1 = 67x
21
22     add $t2, $t1, $s0 # -> t2 = (15x + 67y)
23     sll $t2, $t2, 2 # -> t2 = (15x + 67y) * 4
24     add $t2, $zero, $t2 # -> z = t2
25
26
27
28
29
30
31
```

Figura 4: Imagem do programa 4

## Programa - 05

```
programa_5.asm* programma_6.asm*
1
2 .text
3 .globl main
4
5 main:
6
7     ori $s0, $zero, 0x1B6A # s0 = 6250
8     sll $s0, $s0, 4 # s0 = 6250 * 16
9
10
11    ori $s1, $zero, 0x30D4 # s1 = 12500
12    sll $s1, $s1, 4 # s1 = 12500 * 16
13
14    add $s2, $s0, $s1
15
16
17
18
19
20
21
22
```

Figura 5: Imagem do programa 5

## Programa - 06

```
programa_5.asm programma_6.asm*
1
2 .text
3 .globl main
4
5 main:
6
7     ori $s0, $zero, 0xFFFF # s0 = FFFF
8     sll $s0, $s0, 16 # s0 = FFFF 0000
9     ori $t0, $zero, 0XFFFF # t0 = 0000 FFFF
10    or $s0, $t0, $s0 # s0 = FFFF FFFF
11
12    ori $s1, $zero, 0x493E # s1 = 0x493E
13    sll $s1, $s1, 4 # s1 = 0x493E0
14
15    sll $t1, $s1, 2
16
17    sub $s2, $s0, $t1 # s2 = Maior possivel - 4y
18
19
20
21
22
23
24
25
```

Figura 6: Imagem do programa 6

## Programa - 07

A screenshot of a text editor window titled "programa\_7.asm". The code is as follows:

```
1 .text
2 .globl main
3
4 main:
5
6     ori $8, $0, 0xd1 # $8 = 0000 0001
7     sll $8, $8, 31 # $8 = 1000 0000
8     sra $8, $8, 31 # $8 = 1111 1111
9
10
11
12
13
14
15
16
17
18
```

Figura 7: Imagem do programa 7

## Programa - 08

A screenshot of a text editor window titled "programa\_8.asm". The code is as follows:

```
6 .text
7 .globl main
8
9 main:
10
11     ori $8, $0, 0x1234 # t0 = 0x1234
12     sll $8, $8, 16 # t0 = 0x12340000
13     or $8, 0x5678 # t0 = 0x12345678
14
15     srl $9, $8, 24 # t1 = 0x00000012
16
17     srl $10, $8, 16 # t2 = 0x00001234
18     andi $10, $10, 0xFF # t2 = 0x00000034
19
20     srl $11, $8, 8 # t3 = 0x00123456
21     andi $11, $11, 0xFF # t3 = 0x00000056
22
23     andi $12, $8, 0xFF # t3 = 0x00000056
24
25
26
27
28
29
30
31
```

Figura 8: Imagem do programa 8

## Programa - 09

A screenshot of a text editor window titled "programa\_9.asm". The code is written in assembly language. It starts with a .text section containing a main routine. Inside the routine, it performs several memory operations: loading values from memory into registers \$t0, \$t1, \$t2, and \$t3, and then adding them together to calculate a sum. The result is then stored back into memory. Following the routine, there is a .data section defining four variables (x1, x2, x3, x4) and one variable (soma). The assembly code uses labels like .text, .globl, .main, and .data, and instructions like lw, add, and sw.

```
programa_9.asm
10 .text
11 .globl main
12
13 main:
14     lw $t0, x1 # $t0 = 15
15     lw $t1, x2 # $t1 = 25
16     lw $t2, x3 # $t2 = 13
17     lw $t3, x4 # $t3 = 17
18
19     add $t4, $t0, $t1 # t4 = (t0 + t1)
20     add $t4, $t4, $t2 # t4 = (t0 + t1) + t2
21     add $t4, $t4, $t3 # t4 = ((t0 + t1) + t2) + t3
22
23     sw $t4, soma
24
25
26
27
28 .data
29 x1: .word 15
30 x2: .word 25
31 x3: .word 13
32 x4: .word 17
33 soma: .word -1
34
35
36
```

Line: 36 Column: 1  Show Line Numbers

Figura 9: Imagem do programa 9

## Programa - 10

A screenshot of a text editor window titled "programa\_10.asm". The code is written in assembly language. It starts with a .text section containing a main routine. Inside the routine, it performs several arithmetic operations: loading values from memory into registers \$t0, \$t1, \$t2, and \$t3, performing shifts (sll), subtractions (sub), additions (add), and then storing the result back into memory. Following the routine, there is a .data section defining three variables (x, z, y) and their corresponding word values. The assembly code uses labels like .text, .globl, .main, and .data, and instructions like lw, sll, sub, add, and sw.

```
programa_10.asm
9
10 .text
11 .globl main
12
13 main:
14
15     lw $t0, x # t0 = x
16     lw $t1, z # t1 = y
17
18     sll $t2, $t0, 7 # t2 = 128x
19     sub $t2, $t2, $t0 # t2 = 127x
20
21     sll $t3, $t1, 6 # t3 = 64z
22     add $t3, $t3, $t1 # t3 = 65z
23
24     sub $t4, $t2, $t3 # t4 = 127x - 65z
25     addi $t4, $t4, 1   # t4 = 127x - 65z + 1
26
27     sw $t4, y
28
29
30
31
32 .data
33 x: .word 5
34 z: .word 7
35 y: .word
```

Line: 1 Column: 1  Show Line Numbers

Figura 10: Imagem do programa 10

## Programa - 11

A screenshot of a text editor window titled "programa\_11.asm". The code is written in assembly language. It starts with a .text section containing a main routine. Inside the routine, it loads values from memory into registers \$t0, \$t1, and \$t2, performs arithmetic operations (subtraction and multiplication), and adds a constant value (300000) to \$t3. It then stores the result back into memory. The .data section defines variables x, z, and y with initial values 100000, 200000, and 0 respectively. A note at the end states that the value of y will be overwritten after execution. The code ends with a .size directive.

```
programa_11.asm
8
9
10 .text
11 .globl main
12
13 main:
14     lw $t0, x # t0 = x
15     lw $t1, z # t1 = z
16
17     sub $t2, $t0, $t1 # t2 = x - z
18
19     ori $t3, $zero, 0x493E # t3 = 0x493E
20     sll $t3, $t3, 4 # t3 = 0x493E0
21
22     add $t3, $t2, $t3 # t3 = (x - z) + 300000
23     sw $t3, y
24
25
26
27
28
29 .data
30 x: .word 100000
31 z: .word 200000
32 y: .word 0    # esse valor deverá ser sobreescrito após a execução do programa.
33
34
Line: 1 Column: 1 Show Line Numbers
```

Figura 11: Imagem do programa 11

## Programa - 12

A screenshot of a text editor window titled "programa\_12.asm". The code is written in assembly language. It includes a comment "# Descrição das instruções" and a section "# Memória" explaining memory access conventions. The .text section contains a main routine that performs a series of memory operations: loading from memory into registers \$t0, \$t1, and \$t2, then performing a shift-left operation on \$t0 (\$t0 = 1). Finally, it stores the result back into memory. The .data section defines variables x, ptr1, ptr2, and ptr3 with initial values 5, x, ptr1, and ptr2 respectively. The code ends with a .size directive.

```
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
Line: 1 Column: 1 Show Line Numbers
```

Figura 12: Imagem do programa 12

## Programa - 13

A screenshot of a text editor window titled "programa\_13.asm". The code is written in assembly language and includes comments in Portuguese. The code initializes register \$t0 with value A, performs a right shift by 31 bits, and then branches based on the result. If the result is zero, it goes to the 'positivo' label where \$t0 is set to zero. Otherwise, it goes to the 'negativo' label where \$t0 is set to -7. The code concludes with a global variable main and a data section.

```
programa_13.asm
1 # Descrição das instruções
2
3 # Desvio condicional
4 # beg Registrador1, Registrador2, Label
5 # bne Registrador1, Registrador2, Label
6
7 # Desvio incondicional
8 # j Label
9
10 .text
11 .globl main
12
13 main:
14     lw $t0, A # t0 = A
15
16     srl $t1, $t0, 31
17     beq $t1, $zero, positivo
18     sub $t0, $zero, $t0
19
20 positivo:
21     sw $t0, A
22
23
24 .data
25 A: .word -7

Line: 23 Column: 1 Show Line Numbers
```

Figura 13: Imagem do programa 13

## Programa - 14

A screenshot of a text editor window titled "programa\_14.asm". The code is written in assembly language and includes comments in Portuguese. It initializes registers \$t0 and \$t1 with values A and B respectively. It then checks if \$t1 is even or odd using an AND operation and a branch. If even, it branches to the 'par' label where \$t0 is multiplied by 2. If odd, it branches to the 'impar' label where \$t0 is multiplied by 3 and then incremented by 1. The code concludes with global variables A and B.

```
programa_14.asm
9
10 .text
11 .globl main
12
13 main:
14     lw $t0, A # t0 = A
15
16     andi $t1, $t0, 1 # t1 = t0 & 1
17     beq $t1, $zero, par # par
18     bne $t1, $zero, impar # impar
19
20 par:
21     la $t2, B
22     sw $zero, ($t2)
23
24 impar:
25     la $t2, B
26     addi $t3, $zero, 1
27     sw $t3, ($t2)
28
29 .data
30 A: .word 5
31 B: .word
32
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, $t0, 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
    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 0x1000A00
y: .word 0x13800
z: .word 0x1A80
resultado: .word 0
```

Figura 16: Imagem do programa 16

## Programa - 17

```
programa_12.asm* programa_17.asm

.text
.globl main

main:
    lw $t0, x
    lw $t1, y

    add $t0, $t0, $zero
    beq $t1, $zero, fim
    addi $t1, $zero, 1

loop:
    beq $t1, $t1, fim

    add $t0, $t0, $t0
    addi $t1, $t1, 1

    j loop

fim:
    sw $t0, k

.data
x: .word 0
y: .word 6
k: .word 0
```

Figura 17: Imagem do programa 17

## Programa - 18

```
programa_18.asm*
1
2 .globl main
3 main:
4     lw $t0, x # t0 = x
5     lw $t1, y # t1 = y
6     add $t2, $zero, $zero # k = 0
7     add $t3, $zero, $zero # i = 0
8     add $t4, $zero, $zero # j = 0
9     add $t5, $zero, $zero # soma parcial
10    add $t6, $zero, $zero # soma acumulada
11    beq $t1, $zero, store
12    addi $t2, $zero, 1
13 pot:
14    addi $t3, $t3, 1
15    add $t2, $t2, $t3
16    add $t3, $zero, $zero
17    bne $t1, $zero, multi
18    j multi
19    tim
20 mult:
21    add $t3, $t3, $t4
22    addi $t2, $t2, 1
23    bne $t2, $zero, multi
24    add $t4, $zero, $t3
25    j pot
26
27 fim:
28    add $t2, $zero, $t4
29 store:
30    sw $t2, k
31
32 .data
33 x: .word 2
34 y: .word 3
35 k: .word 0
```

Figura 18: Imagem do programa 18

## Programa - 19

```
programa_19.asm
11 .text
12 .globl main      # O valor presente em A é 2 em B é 4 para a execução deste programa -> Não foi possível exibir no print devido ao tamanho.
13 main:
14     lw $t0, A
15     lw $t1, B
16     add $t0, $zero, $zero
17     add $t1, $zero, $zero
18     add $t0, $t0, $zero
19     count0:
20         beq $t2, $zero, done0
21         srl $t2, $t2, 1
22         addi $t0, $t0, 1
23         j count0
24     done0:
25         add $t1, $zero, $zero
26         add $t1, $t1, $zero
27     count1:
28         beq $t2, $zero, done1
29         srl $t2, $t2, 1
30         addi $t1, $t1, 1
31         j count1
32     done1:
33         sub $t4, $t0, $t2
34         beq $t4, $zero, mul64
35         bne $t4, $zero, check1
36     check1:
37         sub $t4, $t1, $t2
38         beq $t4, $zero, mul64
39         bne $t4, $zero, mul32
40     mul32:
41         mul $t0, $t0, $t1
42     mul64:
43         mult $t0, $t1
44         mtlo $t2
45         mthi $t3
```

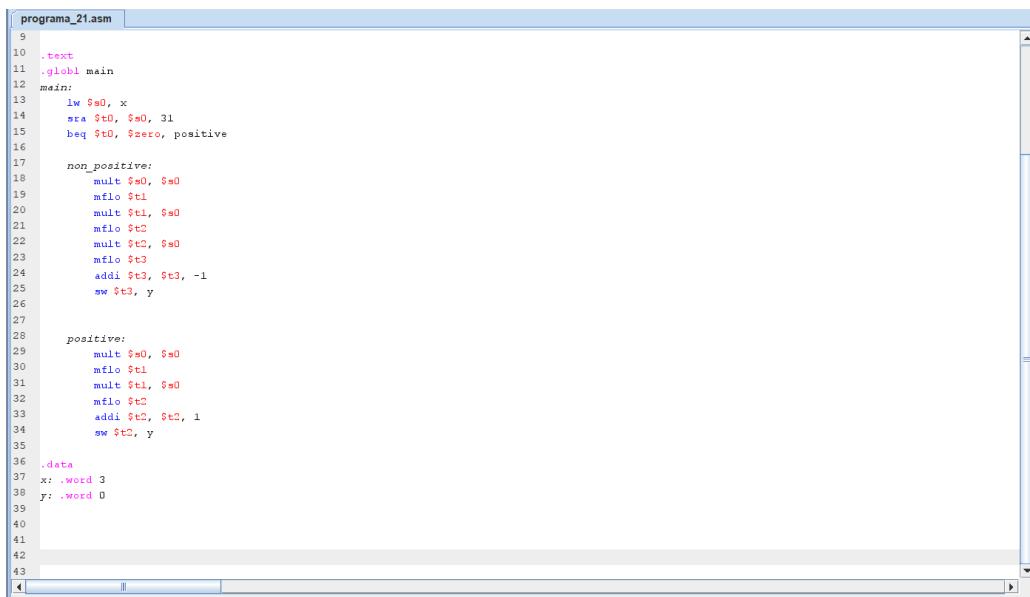
Figura 19: Imagem do programa 19

## Programa - 20

```
programa_20.asm
6
7 .text
8 .globl main
9 main:
10    lw $t0, x
11    andi $t0, $s0, 1
12    beq $t0, $zero, even
13
14    odd:
15        mul $t1, $s0, $s0
16        mul $t2, $t1, $s0
17        mul $t3, $t2, $t1
18        sub $t3, $t3, $t2
19        addi $t3, $t3, 1
20        sw $t3, y
21
22    even:
23        mul $t1, $s0, $s0
24        mul $t2, $t1, $s0
25        mul $t3, $t2, $s0
26        add $t4, $t3, $t2
27        sll $t1, $t1, 1
28        sub $t4, $t4, $t1
29        sw $t4, y
30
31 .data
32 x: .word 4
33 y: .word 0
34
35
36
37
38
39
```

Figura 20: Imagem do programa 20

## Programa - 21



A screenshot of a text editor window titled "programa\_21.asm". The code is written in assembly language for the MIPS architecture. It starts with the .text section containing the main function. The main function first loads the value 3 into register \$t0. It then checks if \$t0 is zero using a branch on equal (beq) instruction. If \$t0 is zero, it branches to the "positive" label. If \$t0 is not zero, it branches to the "non\_positive" label. The "non\_positive" block contains a loop that multiplies \$t0 by 9, divides the result by 10, and stores the remainder in \$t3. This loop continues until \$t3 becomes zero. Once \$t3 is zero, the program adds 1 to \$t2 and stores it back in \$t2. Finally, the value of \$t2 is stored in memory at address y. The "positive" block contains a similar loop, but instead of dividing by 10, it divides by 9. The .data section defines two variables: x and y, both of which are initialized to 0.

```
programa_21.asm
9
10 .text
11 .globl main
12 main:
13     lw $t0, x
14     sr $t0, $t0, 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

Name	Number	Value
Szero	0	0x00000000
Sat	1	0x00000000
Sr0	2	0x00000000
Sr1	3	0x00000000
Sr0	4	0x00000000
Sr1	5	0x00000000
Sr2	6	0x00000000
Sr3	7	0x00000000
Sr0	8	0x00000005
Sr1	9	0x00000009
Sr2	10	0x00000000
Sr3	11	0xffffffff
Sr4	12	0x00000000
Sr5	13	0x00000000
Sr6	14	0x00000000
Sr7	15	0x00000000
Sr0	16	0x00000002
Sr1	17	0x00000001
Sr2	18	0x00000004
Sr3	19	0x00000000
Sr4	20	0xffffffff
Sr5	21	0xffffffffb
Sr6	22	0x00000000
Sr7	23	0x00000000
Sr8	24	0x00000000
Sr9	25	0x00000000
Sr0	26	0x00000000
Sr1	27	0x00000000
Sr2	28	0x00000000
Sr3	29	0x7fffffc
Sr4	30	0x00000000
Sra	31	0x000400025
pc		0x00000000
hi		0x00000000
lo		0x00000000

Figura 22: Imagem da execução programa 1

## Execução - 02

Name	Number	Value
Szero	0	0x00000000
Sat	1	0x00000000
Sr0	2	0x00000000
Sr1	3	0x00000000
Sr0	4	0x00000000
Sr1	5	0x00000000
Sr2	6	0x00000000
Sr3	7	0x00000005
Sr0	8	0x00000000
Sr1	9	0x00000000
Sr2	10	0x00000000
Sr3	11	0x00000000
Sr4	12	0x00000000
Sr5	13	0x00000000
Sr6	14	0x00000000
Sr7	15	0x00000000
Sr0	16	0x00000000
Sr1	17	0x00000014
Sr2	18	0x00000000
Sr3	19	0x00000000
Sr4	20	0x00000000
Sr5	21	0x00000000
Sr6	22	0x00000000
Sr7	23	0x00000000
Sr8	24	0x00000000
Sr9	25	0x00000000
Sr0	26	0x00000000
Sr1	27	0x00000000
Sr2	28	0x00000000
Sr3	29	0xffffffff
Sr4	30	0x00000000
Sra	31	0x00000000
pc		0x000400014
hi		0x00000000
lo		0x00000000

Figura 23: Imagem do execução programa 2

## Execução - 03

The screenshot shows the Mars Simulation Environment during the execution of program 3. The registers pane displays the state of the processor, including general-purpose registers (\$r0 to \$r31) and floating-point registers (\$fp, \$sp, \$pc). The text segment pane shows assembly code with comments indicating register assignments. The data segment pane shows memory dump information. The mars messages pane displays log entries.

Name	Number	Value
\$r0	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	0x0000002d
\$t1	9	0x0000010c
\$t2	10	0x00000044
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$a0	16	0x00000003
\$a1	17	0x00000004
\$t2	18	0x00000044
\$t3	19	0x00000000
\$t4	20	0x00000000
\$t5	21	0x00000000
\$t6	22	0x00000000
\$t7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$t0	26	0x00000000
\$t1	27	0x00000000
\$sp	28	0x10000000
\$fp	29	0x7fffffc
\$ra	30	0x00000000
\$pc	31	0x00000000
hi		0x00000000
lo		0x00000000

Figura 24: Imagem do execução programa 3

## Execução - 04

The screenshot shows the Mars Simulation Environment during the execution of program 4. The registers pane displays the state of the processor, including general-purpose registers (\$r0 to \$r31) and floating-point registers (\$fp, \$sp, \$pc). The text segment pane shows assembly code with comments indicating register assignments. The data segment pane shows memory dump information. The mars messages pane displays log entries.

Name	Number	Value
\$r0	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	0x0000002d
\$t1	9	0x0000010c
\$t2	10	0x00000044
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$a0	16	0x00000003
\$a1	17	0x00000004
\$t2	18	0x00000044
\$t3	19	0x00000000
\$t4	20	0x00000000
\$t5	21	0x00000000
\$t6	22	0x00000000
\$t7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$t0	26	0x00000000
\$t1	27	0x00000000
\$sp	28	0x10000000
\$fp	29	0x7fffffc
\$ra	30	0x00000000
\$pc	31	0x00000000
hi		0x00000000
lo		0x00000000

Figura 25: Imagem do execução programa 4

## Execução - 05

Name	Number	Value
Szero	0	0x00000000
Sat	1	0x00000000
Sv0	2	0x00000000
Sv1	3	0x00000000
Sa0	4	0x00000000
Sa1	5	0x00000000
Sa2	6	0x00000000
Sa3	7	0x00000000
Sa0	8	0x00000000
Sa1	9	0x00000000
Sa2	10	0x00000000
Sa3	11	0x00000000
Sa4	12	0x00000000
Sa5	13	0x00000000
Sa6	14	0x00000000
Sa7	15	0x00000000
Sa0	16	0x000186a0
Sa1	17	0x00030d40
Sa2	18	0x0004593e
Sa3	19	0x00000000
Sa4	20	0x00000000
Sa5	21	0x00000000
Sa6	22	0x00000000
Sa7	23	0x00000000
Sa8	24	0x00000000
Sa9	25	0x00000000
Sa0	26	0x00000000
Sa1	27	0x00000000
Sgp	28	0x10000000
Sfp	29	0x7fffffc
Sra	30	0x00000000
pc	31	0x00400014
hi		0x00000000
lo		0x00000000

Figura 26: Imagem do execução programa 5

## Execução - 06

Name	Number	Value
Szero	0	0x00000000
Sat	1	0x00000000
Sv0	2	0x00000000
Sv1	3	0x00000000
Sa0	4	0x00000000
Sa1	5	0x00000000
Sa2	6	0x00000000
Sa3	7	0x00000000
Sa0	8	0x00000000
Sa1	9	0x00000000
Sa2	10	0x00000000
Sa3	11	0x00000000
Sa4	12	0x00000000
Sa5	13	0x00000000
Sa6	14	0x00000000
Sa7	15	0x00000000
Sa0	16	0xffffffff
Sa1	17	0xffffffff
Sa2	18	0xffffed807e
Sa3	19	0x00000000
Sa4	20	0x00000000
Sa5	21	0x00000000
Sa6	22	0x00000000
Sa7	23	0x00000000
Sa8	24	0x00000000
Sa9	25	0x00000000
Sa0	26	0x00000000
Sa1	27	0x00000000
Sgp	28	0x00000000
Sfp	29	0x7fffffc
Sra	30	0x00000000
pc	31	0x00400020
hi		0x00000000
lo		0x00000000

Figura 27: Imagem do execução programa 6

## Execução - 07

The screenshot shows the Mars Simulation Environment during the execution of program 7. The registers pane displays the state of various寄存器 (Registers) from \$zero to \$lo. The data segment pane shows memory starting at address 0x10010000, with values ranging from 0x00000000 to 0xffffffff. The text segment pane shows assembly code with comments explaining operations like ori, sll, sra, and andi. The Mars Messages window indicates the program has finished running.

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$s0	7	0x00000000
\$s1	8	0xffffffff
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$a0	16	0x00000000
\$a1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$t7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7ffffeffc
\$fp	30	0x00000000
\$ra	31	0x00000000
\$pc		0x00400000
\$hi		0x00000000
\$lo		0x00000000

Figura 28: Imagem do execução programa 7

## Execução - 08

The screenshot shows the Mars Simulation Environment during the execution of program 8. The registers pane displays the state of various寄存器 (Registers) from \$zero to \$lo. The data segment pane shows memory starting at address 0x10010000, with values ranging from 0x00000000 to 0xffffffff. The text segment pane shows assembly code with comments explaining operations like ori, sll, sra, and andi. The Mars Messages window indicates the program has finished running.

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$s0	7	0x00000000
\$t0	8	0x12345678
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$a0	16	0x00000000
\$a1	17	0x00000000
\$s1	18	0x00000000
\$s2	19	0x00000000
\$s3	20	0x00000000
\$s4	21	0x00000000
\$s5	22	0x00000000
\$s6	23	0x00000000
\$t7	24	0x00000000
\$t8	25	0x00000000
\$t9	26	0x00000000
\$k0	27	0x00000000
\$k1	28	0x00000000
\$gp	29	0x10000000
\$sp	30	0x00000000
\$fp	31	0x00000000
\$ra		0x00000000
\$pc		0x00400000
\$hi		0x00000000
\$lo		0x00000000

Figura 29: Imagem do execução programa 8

## Execução - 09

The screenshot shows the Mars Simulation Environment during the execution of program 9. The registers pane displays the state of various寄存器 (Registers) from \$zero to \$lo. The data segment pane shows memory starting at address 0x10010000, with values ranging from 0x00000000 to 0xffffffff. The text segment pane shows assembly code with comments explaining operations like lw, add, and sw. The Mars Messages window indicates the program has finished running.

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$s0	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000019
\$t2	10	0x00000000
\$t3	11	0x00000011
\$t4	12	0x00000044
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$a0	16	0x00000000
\$a1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$t3	25	0x00000000
\$t5	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7ffffeffc
\$fp	30	0x00000000
\$ra	31	0x00400034
\$pc		0x00400000
\$hi		0x00000000
\$lo		0x00000000

Figura 30: Imagem do execução programa 9

## Execução - 10

The screenshot shows the Mars Simulation Environment during the execution of program 10. The registers pane displays the state of various寄存器 (Registers) from \$zero to \$t0, \$t1, ..., \$t31. The data segment pane shows memory starting at address 0x10010000 with values ranging from 0x00000000 to 0xffffffff. The text segment pane shows assembly code with comments and addresses from 0x00000000 to 0x0000000f. The Mars Messages window indicates the program has finished running.

Figura 31: Imagem do execução programa 10

## Execução - 11

The screenshot shows the Mars Simulation Environment during the execution of program 11. The registers pane displays the state of various寄存器 (Registers) from \$zero to \$t0, \$t1, ..., \$t31. The data segment pane shows memory starting at address 0x10010000 with values ranging from 0x00000000 to 0xffffffff. The text segment pane shows assembly code with comments and addresses from 0x00000000 to 0x0000000f. The Mars Messages window indicates the program has finished running.

Figura 32: Imagem do execução programa 11

## Execução - 12

The screenshot shows the Mars Simulation Environment during the execution of program 12. The registers pane displays the state of various寄存器 (Registers) from \$zero to \$t0, \$t1, ..., \$t31. The data segment pane shows memory starting at address 0x10010000 with values ranging from 0x00000000 to 0xffffffff. The text segment pane shows assembly code with comments and addresses from 0x00000000 to 0x0000000f. The Mars Messages window indicates the program has finished running.

Figura 33: Imagem do execução programa 12

Execução - 13

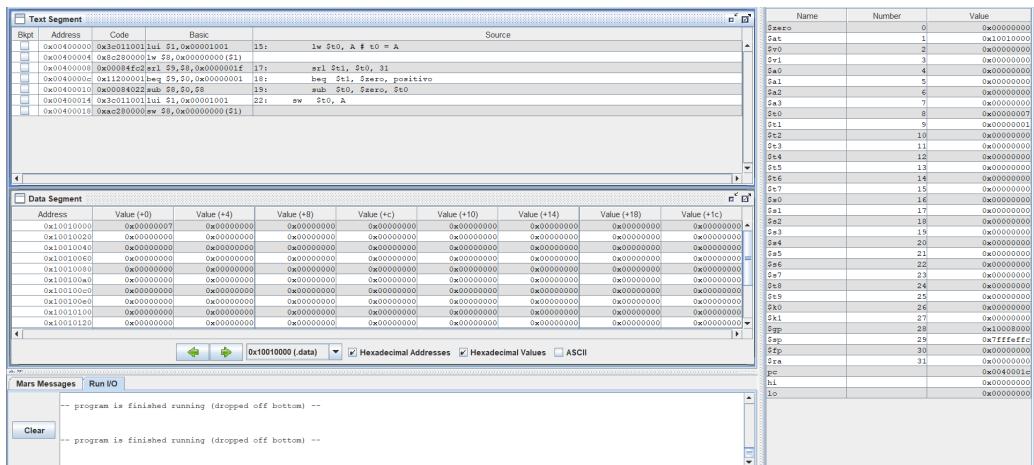


Figura 34: Imagem do execução programa 13

Execução - 14

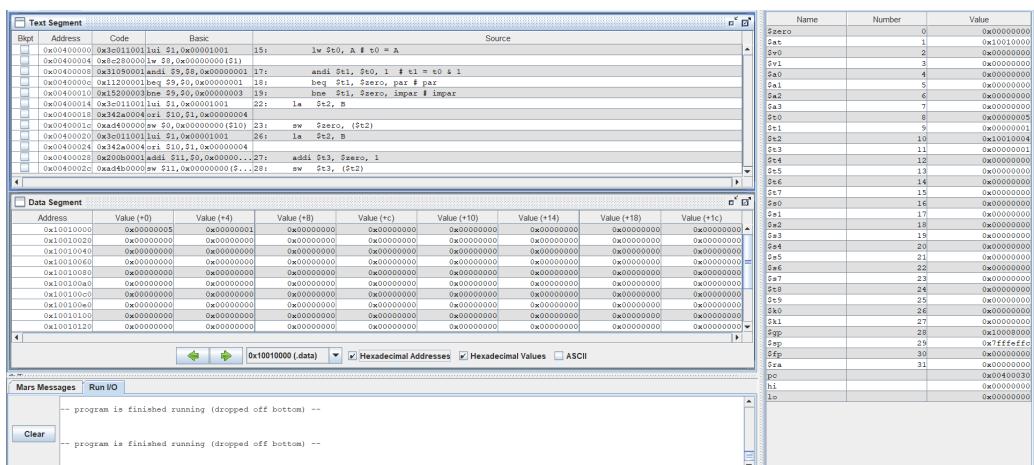


Figura 35: Imagem do execução programa 14

Execução - 15

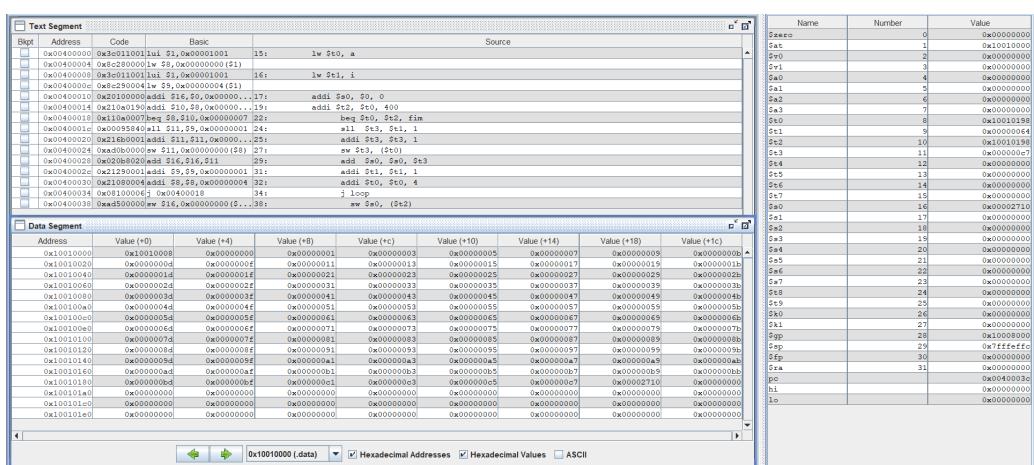


Figura 36: Imagem do execução programa 15

## Execução - 16

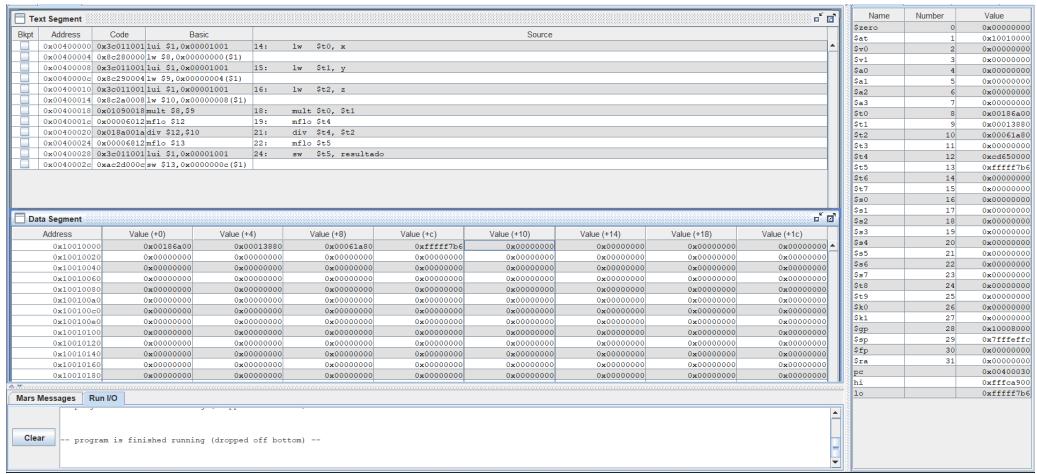


Figura 37: Imagem do execução programa 16

## Execução - 17

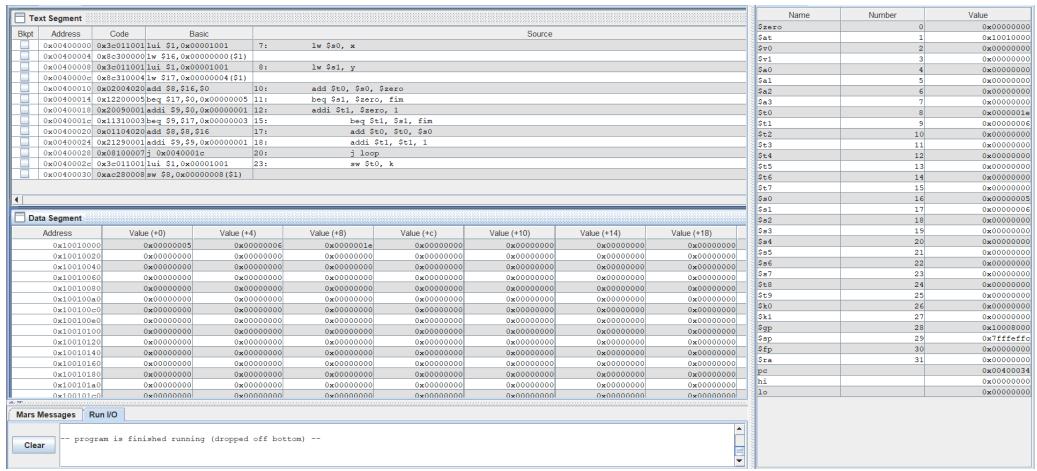


Figura 38: Imagem do execução programa 17

## Execução - 18

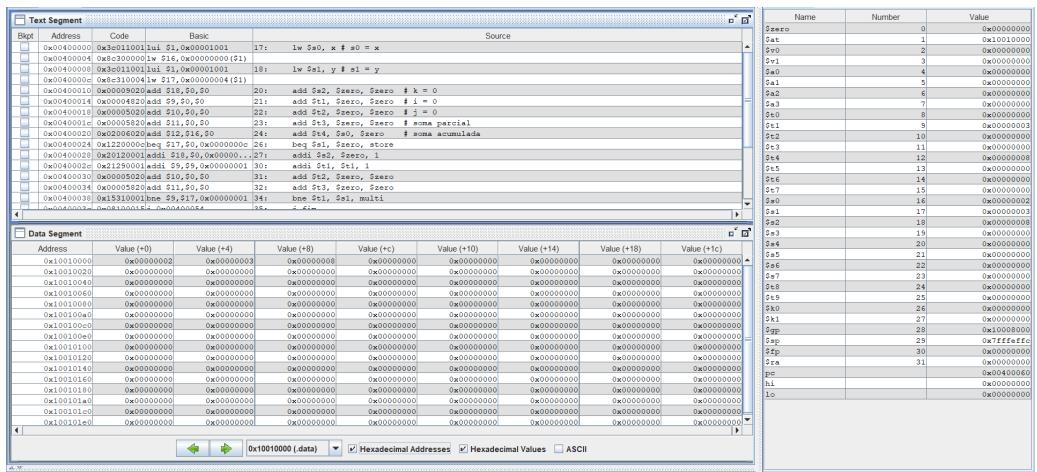


Figura 39: Imagem da execução programa 18

## Execução - 19

The screenshot shows the Mars SIMulator during the execution of program 19. The Text Segment window displays assembly code with columns for Blkt, Address, Code, Basic, and Source. The Data Segment window shows memory dump details. The Registers window lists various寄存器 (Registers) with their names, numbers, and values. The Run I/O window contains messages indicating the program has finished running.

Figura 40: Imagem do execução programa 19

## Execução - 20

The screenshot shows the Mars SIMULATOR during the execution of program 20. The Text Segment window displays assembly code with columns for Blkt, Address, Code, Basic, and Source. The Data Segment window shows memory dump details. The Registers window lists various registers with their names, numbers, and values. The Run I/O window contains messages indicating the program has finished running.

Figura 41: Imagem do execução programa 20

## Execução - 21

The screenshot shows the Mars SIMULATOR during the execution of program 21. The Text Segment window displays assembly code with columns for Blkt, Address, Code, Basic, and Source. The Data Segment window shows memory dump details. The Registers window lists various registers with their names, numbers, and values. The Run I/O window contains messages indicating the program has finished running.

Figura 42: Imagem do execução programa 21

## **Perguntas 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 será 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.