Relatório 10 - Laboratório de Arquitetura de Computadores

Luiz Junio Veloso - Matricula: 624037

1 de agosto de 2019

Exercícios

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

   (a) 16
   (b) 32
   (c) 64
   (d) 128

   R: c) 64

2. Quais os registradores que armazenam os resultados na multiplicação?

   (a) high e low
   (b) hi e lo
   (c) R0 e R1
   (d) $0 e $1

   R: b) hi e lo

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

   (a) mult
   (b) multu
   (c) multi
   (d) mutt

   R: a) mult

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

   (a) move $8,lo
   (b) mvlo $8,lo
   (c) mflo $8
   (d) addu $8,$0,lo

   R: c) mflo $8

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

   (a) 16
   (b) 32
   (c) 64
   (d) 128

   R: b) 32

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

   (a) lo
   (b) hi
   (c) high
   (d) $2

   R: a) lo

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

   (a) dv
   (b) divide
   (c) divu
   (d) div

   R: 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

   R: a) 1110 0110

9. Qual o efeito de um **arithmetic shift right** de uma posição?

   (a) Se o inteiro for unsigned, o shift divide por 2. Se o inteiro for signed, o shift o divide por 2.
   (b) Se o inteiro for unsigned, o shift o divide por 2. Se o inteiro for signed, o shift pode resultar em um valor errado.
   (c) Se o inteiro for unsigned, o shift pode ocasionar um valor errado. Se o inteiro for signed, o shift o divide por 2.
   (d) O shift multiplica o número por dois.

   R: 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.

10. Qual sequencia de instruções avalia 3x + 7, onde x é iniciado no reg. $8 e o resultado armazenado em $9?

    (a) ori $3,$0,3
        mult $8,$3
        mflo $9
        addi $9,$9,7

    (b) ori $3,$0,3
        mult $8,$3
        addi $9,$8,7

    (c) ori $3,$0,3
        mult $8,$3
        mfhi $9
        addi $9,$9,7

    (d) mult $8,$3
        mflo $9
        addi $9,$9,7

R: A)
ori $3,$0,3
mult $8,$3
mflo $9
addi $9,$9,7

Programas

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

   R:

```
.text
.globl main

main:
    # Colocar endereco base da memoria em t0
    addi $t0,$t0,0x1001
    sll  $t0,$t0,16

    # Obter valor da memoria
    lw   $s0,0($t0)

    # Verificar bit de sinal
    srl  $t1,$s0,31
    andi $t1,$t1,0x0001

    # Se diferente de 0, faz o modulo
    bne  $t1,$zero,nega
    j pos

nega:
    not $s0,$s0
    add $s0,$s0,1

pos:
    sw $s0,0($t0) # Salvar o resultado na memoria

.data
A: .word -42
```

programa13.asm

2. Escreva um programa que leia da memória um valor de temperatura TEMP. Se TEMP >= 30 e TEMP <= 50 uma variável FLAG, também na memória, deverá receber o valor 1, caso contrário, FLAG deverá ser zero.

R:

```
.text
.globl main

main:
    # Colocar endereco base da memoria em t0
    addi $t0,$t0,0x1001
    sll  $t0,$t0,16

    # Obter valor TEMP da memoria
    lw   $s0,0($t0)

    # Verifica se eh >= 30
    addi $t1,$zero,30
    slt $t1,$s0,$t1 # 1 se for < 30, 0 se for >= 30

    # Se igual a 0, faz o modulo
    beq  $t1,$zero,teste2
    j zero

teste2:
    # Verifica se eh <=50
    addi $t2,$zero,50
    slt $t2,$t2,$s0 # 1 se for 50 < TEMP, 0 se for 50 >= TEMP

    # Se igual a 0, FLAG = 1
    beq $t2,$zero,um
    j zero

um:
    # Colocar um em FLAG
    addi $t3,$zero,1
    sw $t3,4($t0)
    j fim

zero:
    # Colocar zero em FLAG
    sw $zero,4($t0)
    j fim

fim:

.data
TEMP: .word 35
FLAG: .word 0
```

programa14.asm

3. Escrever um programa que crie um vetor de 100 elementos na memória onde $vetor[i] = 2 * i + 1$. Após a ultima posição do vetor criado, escrever a soma de todos os valores armazenados do vetor.

R:

```
# i - $s0
# soma - $s1
.text
.globl main
main:
    # Colocar endereco base da memoria em t0
    addi $t0,$zero,0x1001
    sll  $t0,$t0,16

    addi $s0,$zero,0 # comeca i = 0
    addi $t3,$zero,100 # Colocando t3 = 100

do: # Preenche o vetor de 100
    # t1 = 2 * i + 1
    sll $t1,$s0,1
    addi $t1,$t1,1

    # vetor[i] = t1
    sll $t2,$s0,2
    add $t2,$t2,$t0
    sw $t1,0($t2)

    addi $s0,$s0,1 # i++

    # i != t3
    bne $s0,$t3,do
    j endWhile

endWhile:
    addi $s0,$zero,0 # Define i = 0
    addi $s1,$zero,0 # Define soma = 0

do2:
    # soma += vetor[i]
    sll $t2,$s0,2
    add $t2,$t2,$t0
    lw $t4, 0($t2)
    add $s1,$s1,$t4

    addi $s0,$s0,1 # i++

    # i != t3
    bne $s0,$t3,do2
    j endWhile2

endWhile2:
    sw $s1, 0($t2)
```

programa15.asm

4. Considere que a partir da primeira posição livre da memória temos um vetor com 100 elementos. Escrever um programa que ordene esse vetor de acordo com o algoritmo da bolha. Faça o teste colocando um vetor totalmente desordenado e verifique se o algoritmo funciona.

R:

```
# i - $s0
# j - $s1
# z - $s2
.text
.globl main
main:
    # Colocar endereco base da memoria em t0
    addi $t0,$zero,0x1001
    sll  $t0,$t0,16

    # Vetor desordenado
    addi $s2,$zero,100 # z = 100
    addi $t1,$zero,100
    do:
        sll $t9,$s2,2 # t9 = z * 4
        add $t9,$t9,$t0 # t9 = t9 + endereco base
        sub $t8,$t1,$s2 # t8 = 100 - z
        sw $t8,0($t9)
        addi $s2,$s2,-1 # z--
        bne $s2,$zero,do # while(z != 0)

BubbleSort:
    addi $s0,$zero,0 # comeca i = 0
    addi $t1,$zero,99 # Colocando t1 = 99 (100 - 1) = Tamanho do Array - 1
    for:
        slt $t7,$s0,$t1 # if (i < 99)
        beq $t7,$zero,endFor

        addi $s1,$zero,0 # j = 0
        for2:
            sub $t2,$t1,$s0 # t2 = 99 - i
            slt $t3,$s1,$t2 # if (j < 99 - i)
            beq $t3,$zero,endFor2

            sll $t4,$s1,2
            add $t4,$t4,$t0
            lw $t5,0($t4) # t5 = A[J]
            lw $t6,4($t4) # t6 = A[J + 1]
            slt $t7,$t6,$t5 # if (t5 > t6)
            beq $t7,$zero,pula
            # swap
            sw $t5,4($t4) # A[J + 1] = t5
            sw $t6,0($t4) # A[J] = t6
            pula:
            addi $s1,$s1,1 # j++
            j for2
            endFor2:
            addi $s0,$s0,1 # i++
            j for
    endFor:
```

programa16.asm

5.

$$y = \begin{cases} x^4 + x^3 - 2x^2 & \text{se x for par} \\ x^5 - x^3 + 1 & \text{se x for impar} \end{cases}$$

Os valores de x devem ser lidos da primeira posição livre da memória e o valor y deverá ser escrito na segunda posição livre.

R:

```
# endereço base - $s0
.text
.globl main
main:
    # Colocar endereco base da memoria em s0
    addi $s0,$zero,0x1001
    sll  $s0,$s0,16

    # Verifica se X é par ou impar
    lw $t1,0($s0) # t1 = X
    andi $t2,$t1,1 # Isola o primeiro bit
    beq $t2,$zero,par # if(t2 == 0) goto par
    j impar           # else goto impar

    par:
        mult $t1,$t1 # x * x
        mflo $t2 # t2 = x^2
        mult $t2,$t1 # x^2 * X
        mflo $t3 # t3 = x^3
        mult $t3,$t1 # x^3 * x
        mflo $t4 # t4 = x^4

        add $t5,$t4,$t3 # t5 = x^4 + x^3
        sll $t2,$t2,1   # t2 = x^2 * 2
        sub $t5,$t5,$t2 # t5 = t5 - t2
        sw $t5,4($s0)   # Y = t5
        j fim
    impar:
        mult $t1,$t1 # x * x
        mflo $t2 # t2 = x^2
        mult $t2,$t1 # x^2 * X
        mflo $t3 # t3 = x^3
        mult $t3,$t1 # x^3 * x
        mflo $t4 # t4 = x^4
        mult $t4,$t1 # x^4 * x
        mflo $t5 # t5 = x^5

        sub $t5,$t5,$t3 # t5 = x^5 - x^3
        addi $t5,$t5,1 # t5 = t5 + 1
        sw $t5,4($s0) # Y = t5
    fim:

.data
X: .word 42
Y: .word 0
```

programa17.asm

6.

$$y = \begin{cases} x^3 + 1 & \text{se x} > 0 \\ x^4 - 1 & \text{se x} <= 0 \end{cases}$$

Os valores de x devem ser lidos da primeira posição livre da memória e o valor y deverá ser escrito na segunda posição livre.

R:

```
# endereço base - $s0
.text
.globl main
main:
    # Colocar endereco base da memoria em s0
    addi $s0,$zero,0x1001
    sll  $s0,$s0,16

    # Verifica se X é par ou impar
    lw $t1,0($s0) # t1 = X
    slt $t2,$zero,$t1 # if (x > 0)
    bne $t2,$zero,maior # goto maior
    j menorIgual        # else goto menorIgual

    maior:
        mult $t1,$t1 # x * x
        mflo $t2 # t2 = x^2
        mult $t2,$t1 # x^2 * X
        mflo $t3 # t3 = x^3
        addi $t3,$t3,1 # t3 = x^3 + 1
        sw $t3,4($s0)
        j fim

    menorIgual:
        mult $t1,$t1 # x * x
        mflo $t2 # t2 = x^2
        mult $t2,$t1 # x^2 * X
        mflo $t3 # t3 = x^3
        mult $t3,$t1 # x^3 * x
        mflo $t4 # t4 = x^4
        addi $t4,$t4,-1 # t4 = x^4 - 1
        sw $t4,4($s0)

    fim:

.data
X: .word 42
Y: .word 0
```

programa18.asm

7. Escreva um programa que avalie a expressão: $(x * y)/z$.
   Use x = 1600000 (=0x186A00), y = 80000 (=0x13880),
   e z = 400000 (=0x61A80). Inicializar os registradores com os valores acima.

   R:

```
# x - $s0
# y - $s1
# z - $s2
# resultado - $s3
.text
.globl main
main:
    addi $t0,$zero,0x0018
    sll $t0,$t0,16
    ori $s0,$t0,0x6A00 # x = 0x186A00

    addi $t0,$zero,0x0001
    sll $t0,$t0,16
    ori $s1,$t0,0x3880 # y = 0x13880

    addi $t0,$zero,0x0006
    sll $t0,$t0,16
    ori $s2,$t0,0x1A80 # z = 0x61A80

    div $s0,$s2
    mflo $t1 # t1 = x / z
    mult $t1,$s1
    mflo $s3 # resultado = t1 * y
```

programa19.asm

8. Escreva um programa que gere um vetor com os números ímpares até 100. O valor 1 deverá estar na primeira posição livre da memória.
   Após gerar e armazenar o vetor, seu programa deverá varrer todo o vetor, ler cada termo, somar em uma variável auxiliar e armazenar a última posição a soma de todos os elementos.
   Mostre a tabela de porcentagens das instruções utilizadas.

   R:

```
# i - $s0
# j - $s1
# soma - $s2
.text
.globl main
main:
    # Colocar endereco base da memoria em t0
    addi $t0,$zero,0x1001
    sll  $t0,$t0,16
    addi $s0,$zero,0 # comeca i = 0
    addi $t3,$zero,99 # Colocando t3 = 99

do: # Preenche o vetor com impares até 100
    # t1 = 2 * i + 1
    sll $t1,$s0,1
    addi $t1,$t1,1

    # vetor[i] = t1
    sll $t2,$s0,2
    add $t2,$t2,$t0
    sw $t1,0($t2)

    addi $s0,$s0,1 # i++

    slt $s1,$t1,$t3 # j = t1 < 99
    bne $s1,$zero,do # while(j)
    j endWhile

endWhile:

    addi $s2,$zero,0 # Define soma = 0
    addi $s0,$s0,-1 # i-- para remover ultimo i++
do2:
    # soma += vetor[i]
    sll $t4,$s0,2
    add $t4,$t4,$t0
    lw $t5, 0($t4)
    add $s2,$s2,$t5

    addi $s0,$s0,-1 # i--
    # while(i >= 0)
    slt $t6,$s0,$zero
    beq $t6,$zero,do2
    j endWhile2

endWhile2:
    sw $s2,0($t2) # Escreve a soma na ultima posicao
```

programa20.asm

9. Escreva um programa que gere um vetor de inteiros até 100. Seu programa deverá armazenar na memória os números pares separados dos ímpares. Armazene primeiro os pares e logo a seguir os ímpares.
   Mostre a tabela de porcentagens das instruções utilizadas.

   R:

```
# i - $s0
# j - $s1
.text
.globl main
main:
    # Colocar endereco base da memoria em t0
    addi $t0,$zero,0x1001
    sll  $t0,$t0,16

    addi $s0,$zero,0 # comeca i = 0
    addi $t3,$zero,100 # Colocando t3 = 100

doPar: # Preenche o vetor de 100
    # t1 = 2 * i
    sll $t1,$s0,1

    # vetor[i] = t1
    sll $t2,$s0,2
    add $t2,$t2,$t0
    sw $t1,0($t2)

    addi $s0,$s0,1 # i++

    # t1 != 100
    bne $t1,$t3,doPar
    j endWhilePar

endWhilePar:

addi $s1,$zero,0 # comeca j = 0
addi $t3,$zero,99 # Colocando t3 = 99

doImpar: # Preenche o restante do vetor com impares até 100
    # t1 = 2 * j + 1
    sll $t1,$s1,1
    addi $t1,$t1,1

    add $t2,$s0,$s1 # i = i + j
    sll $t2,$t2,2
    add $t2,$t2,$t0
    sw $t1,0($t2)

    addi $s1,$s1,1 # j++

    slt $t4,$t1,$t3 # t4 = t1 < 99
    bne $t4,$zero,doImpar # while(j)
```

programa21.asm

10. Para a expressão a seguir, escreva um programa que calcule o valor de k: $k = x * y$
O valor de x deve ser lido da primeira posição livre da memória e o valor de y deverá ser lido da segunda posição livre. O valor de k, após calculado, deverá ainda ser escrito na terceira posição livre da memória.

R:

```
#Endereco base - $s0
.text
.globl main
main:
    # Colocar endereco base da memoria em s0
    addi $s0,$zero,0x1001
    sll  $s0,$s0,16

    lw $t0,0($s0) # t0 = A[0] = X
    lw $t1,4($s0) # t1 = A[1] = Y
    mult $t0,$t1
    mflo $t2      # t2 = t0 * t1
    sw $t2,8($s0) # K = A[2] = t2
.data
X: .word 4
Y: .word 2
K: .word 0
```

programa22.asm

11. O mesmo programa anterior, porém: $k = x^y$

R:

```
#Endereco base - $s0
# i - $s1
.text
.globl main
main:
    # Colocar endereco base da memoria em s0
    addi $s0,$zero,0x1001
    sll  $s0,$s0,16

    lw $t0,0($s0) # t0 = X
    lw $t1,4($s0) # t0 = Y

    addi $s1,$zero,1 # i = 1
    add $t2,$zero,$t0 # t2 = 0
    do:
        mult $t2,$t0
        mflo $t2 # t2 = t2 * X
        addi $s1,$s1,1 # i++
        slt $t3,$s1,$t1 # t3 = i < Y
        bne $t3,$zero,do # while(t3)
        sw $t2,8($s0) # K = t2
.data
X: .word 2
Y: .word 5
K: .word 0
```

programa23.asm

Imagens

Figura 1: Programa 13

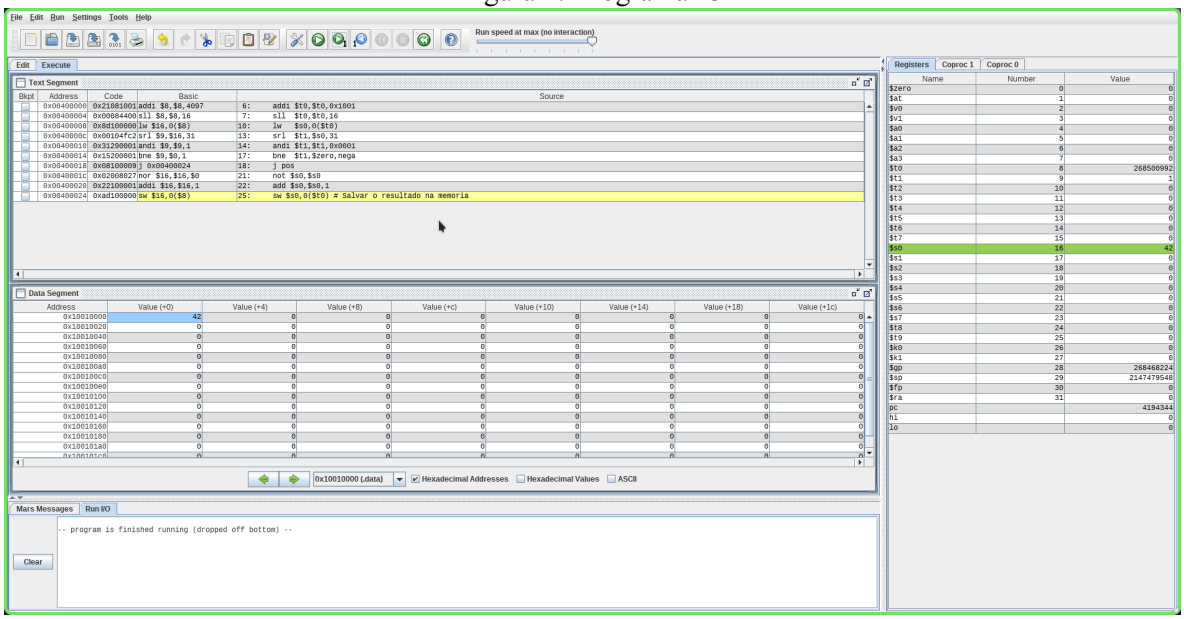

Figura 2: Programa 14

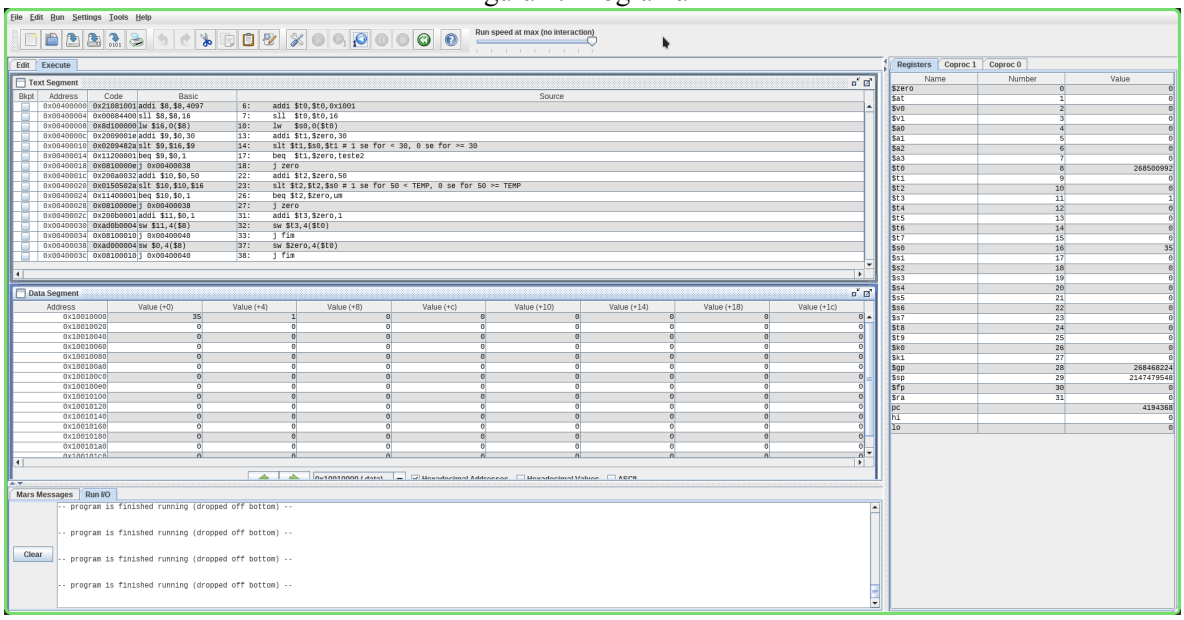

Figura 3: Programa 15

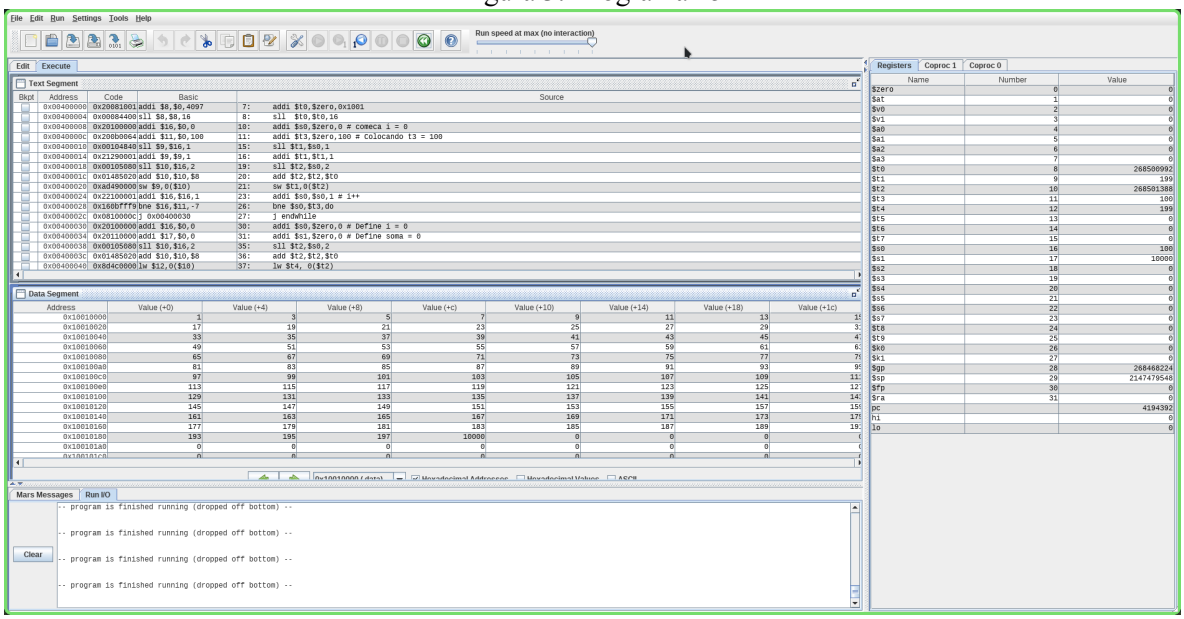

Figura 4: Programa 16

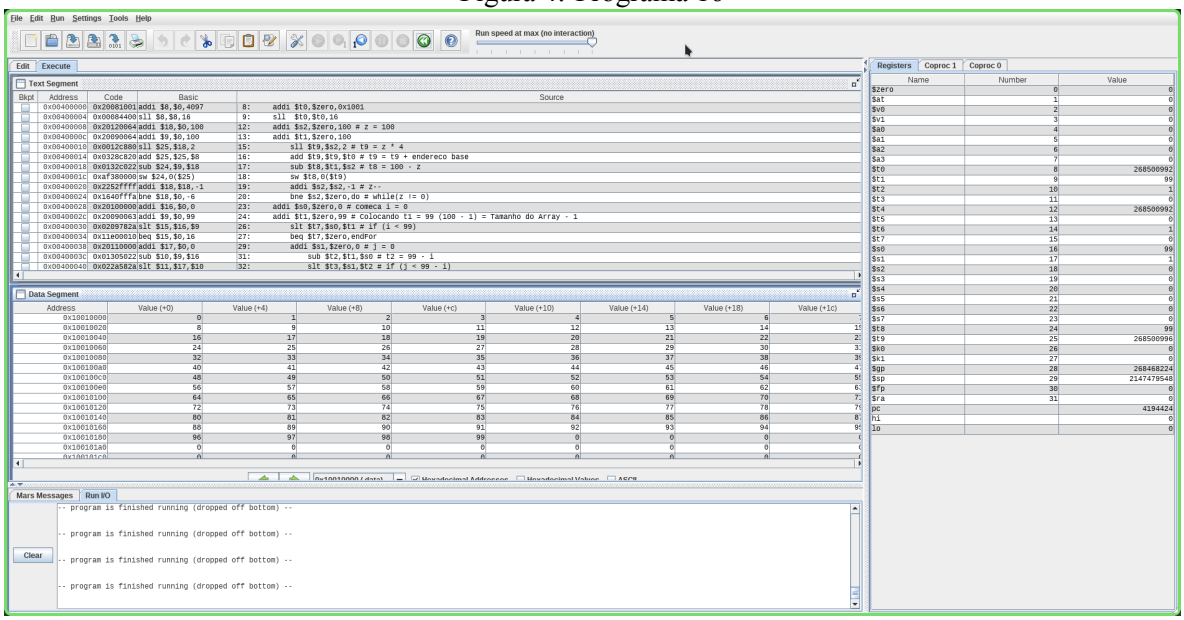

Figura 5: Programa 17

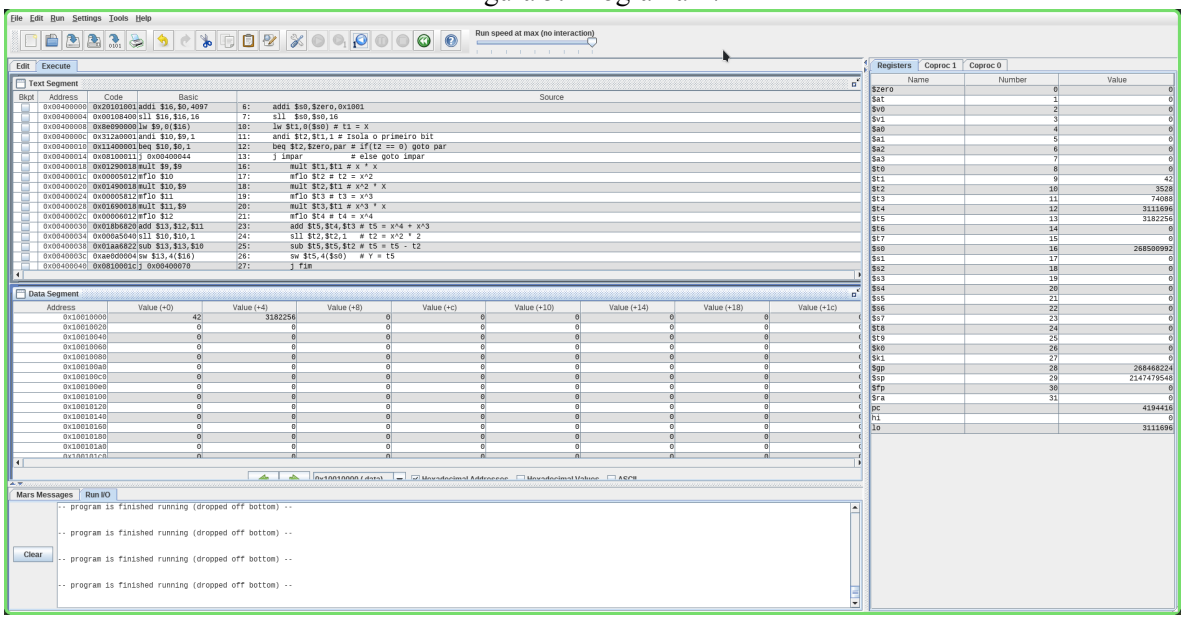

Figura 6: Programa 18

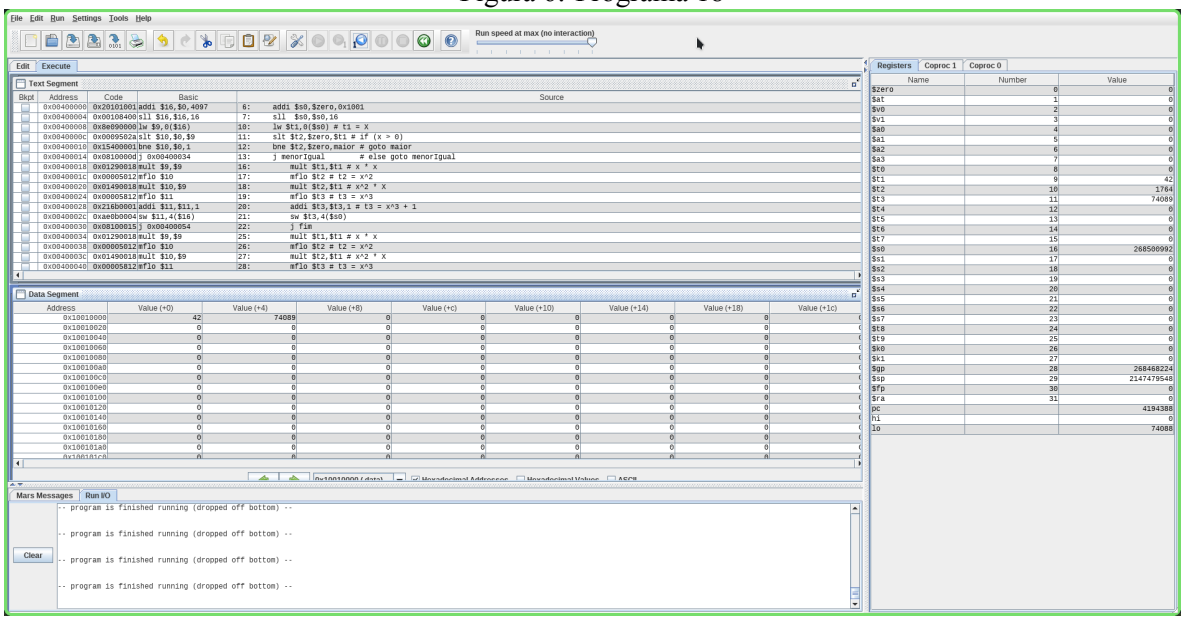

Figura 7: Programa 19

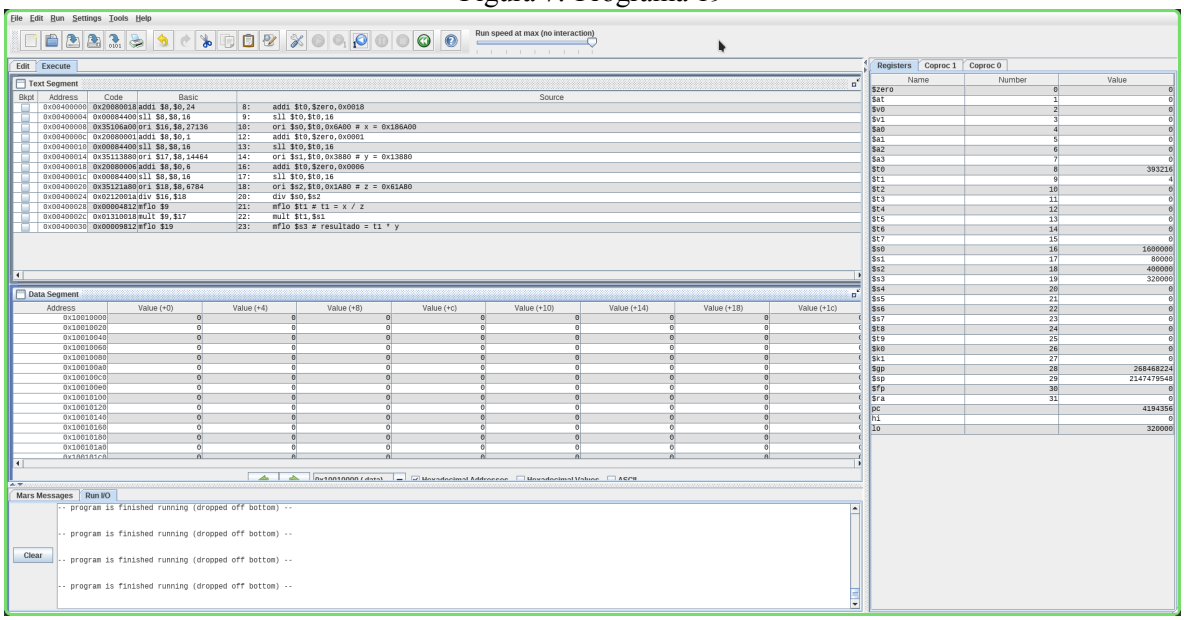

Figura 8: Programa 20

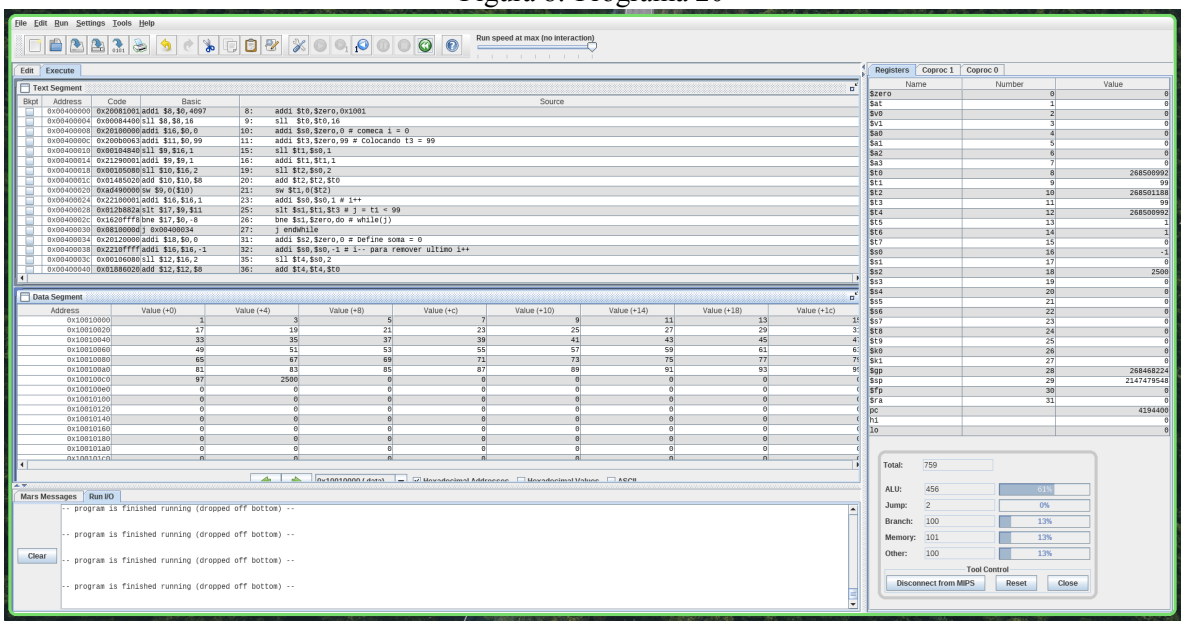

Figura 9: Programa 21

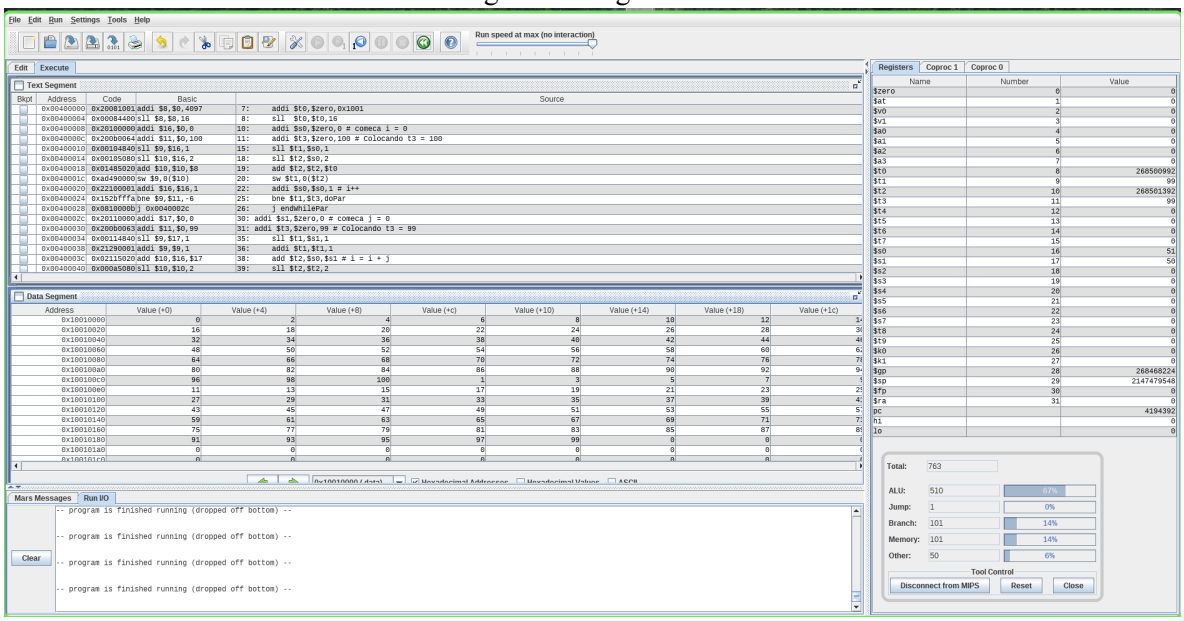

Figura 10: Programa 22

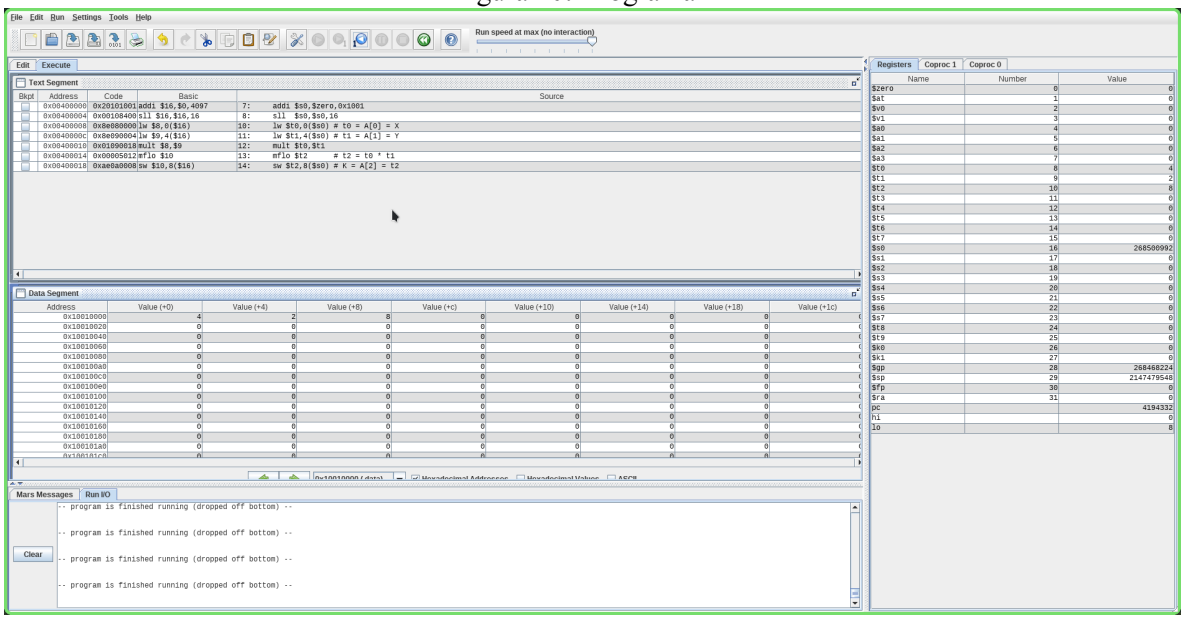

Figura 11: Programa 23