

Relatório 10 - Laboratório de Arquitetura de Computadores

Luiz Junio Veloso Dos Santos - Matricula: 624037

16 de maio 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:

```
1 .text
2 .globl main
3
4 main:
5     # Colocar endereço base da memória em t0
6     addi $t0,$t0,0x1001
7     sll  $t0,$t0,16
8
9     # Obter valor da memória
10    lw   $s0,0($t0)
11
12    # Verificar bit de sinal
13    srl  $t1,$s0,31
14    andi $t1,$t1,0x0001
15
16    # Se diferente de 0, faz o modulo
17    bne  $t1,$zero,nega
18    j    pos
19
20 nega:
21     not $s0,$s0
22     add $s0,$s0,1
23
24 pos:
25     sw  $s0,0($t0) # Salvar o resultado na memória
26
27 .data
28 A: .word -42
```

programa13.asm

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

R:

```
1 .text
2 .globl main
3
4 main:
5     # Colocar endereço base da memória em t0
6     addi $t0,$t0,0x1001
7     sll  $t0,$t0,16
8
9     # Obter valor TEMP da memória
10    lw   $s0,0($t0)
11
12    # Verifica se eh >= 30
13    addi $t1,$zero,30
14    slt  $t1,$s0,$t1 # 1 se for < 30, 0 se for >= 30
15
16    # Se igual a 0, faz o modulo
17    beq  $t1,$zero,teste2
18    j    zero
19
20 teste2:
21    # Verifica se eh <=50
22    addi $t2,$zero,50
23    slt  $t2,$t2,$s0 # 1 se for 50 < TEMP, 0 se for 50 >= TEMP
24
25    # Se igual a 0, FLAG = 1
26    beq  $t2,$zero,um
27    j    zero
28
29 um:
30    # Colocar um em FLAG
31    addi $t3,$zero,1
32    sw   $t3,4($t0)
33    j    fim
34
35 zero:
36    # Colocar zero em FLAG
37    sw   $zero,4($t0)
38    j    fim
39
40 fim:
41
42 .data
43 TEMP: .word 35
44 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:

```
1  # i - $s0
2  # soma - $s1
3  .text
4  .globl main
5  main:
6      # Colocar endereço base da memória em t0
7      addi $t0,$zero,0x1001
8      sll  $t0,$t0,16
9
10     addi $s0,$zero,0 # começa i = 0
11     addi $t3,$zero,100 # Colocando t3 = 100
12
13 do: # Preenche o vetor de 100
14     # t1 = 2 * i + 1
15     sll  $t1,$s0,1
16     addi $t1,$t1,1
17
18     # vetor[i] = t1
19     sll  $t2,$s0,2
20     add  $t2,$t2,$t0
21     sw  $t1,0($t2)
22
23     addi $s0,$s0,1 # i++
24
25     # i != t3
26     bne $s0,$t3,do
27     j   endWhile
28
29 endWhile:
30     addi $s0,$zero,0 # Define i = 0
31     addi $s1,$zero,0 # Define soma = 0
32
33 do2:
34     # soma += vetor[i]
35     sll  $t2,$s0,2
36     add  $t2,$t2,$t0
37     lw  $t4,0($t2)
38     add  $s1,$s1,$t4
39
40     addi $s0,$s0,1 # i++
41
42     # i != t3
43     bne $s0,$t3,do2
44     j   endWhile2
45
46 endWhile2:
47     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:

```

1  # i - $s0
2  # j - $s1
3  # z - $s2
4  .text
5  .globl main
6  main:
7      # Colocar endereço base da memória em t0
8      addi $t0,$zero,0x1001
9      sll  $t0,$t0,16
10
11     # Vetor desordenado
12     addi $s2,$zero,100 # z = 100
13     addi $t1,$zero,100
14     do:
15         sll $t9,$s2,2 # t9 = z * 4
16         add $t9,$t9,$t0 # t9 = t9 + endereço base
17         sub $t8,$t1,$s2 # t8 = 100 - z
18         sw  $t8,0($t9)
19         addi $s2,$s2,-1 # z--
20         bne $s2,$zero,do # while(z != 0)
21
22     BubbleSort:
23         addi $s0,$zero,0 # começa i = 0
24         addi $t1,$zero,99 # Colocando t1 = 99 (100 - 1) = Tamanho do Array - 1
25         for:
26             slt $t7,$s0,$t1 # if (i < 99)
27             beq $t7,$zero,endFor
28
29             addi $s1,$zero,0 # j = 0
30             for2:
31                 sub $t2,$t1,$s0 # t2 = 99 - i
32                 slt $t3,$s1,$t2 # if (j < 99 - i)
33                 beq $t3,$zero,endFor2
34
35                 sll $t4,$s1,2
36                 add $t4,$t4,$t0
37                 lw  $t5,0($t4) # t5 = A[J]
38                 lw  $t6,4($t4) # t6 = A[J + 1]
39                 slt $t7,$t6,$t5 # if (t5 > t6)
40                 beq $t7,$zero,pula
41                 # swap
42                 sw  $t5,4($t4) # A[J + 1] = t5
43                 sw  $t6,0($t4) # A[J] = t6
44                 pula:
45                 addi $s1,$s1,1 # j++
46                 j  for2
47             endFor2:
48             addi $s0,$s0,1 # i++
49             j  for
50     endFor:

```

programa16.asm

5.

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

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:

```
1 # endereço base - $s0
2 .text
3 .globl main
4 main:
5     # Colocar endereço base da memória em s0
6     addi $s0,$zero,0x1001
7     sll  $s0,$s0,16
8
9     # Verifica se X é par ou impar
10    lw $t1,0($s0) # t1 = X
11    andi $t2,$t1,1 # Isola o primeiro bit
12    beq $t2,$zero,par # if(t2 == 0) goto par
13    j impar # else goto impar
14
15    par:
16        mult $t1,$t1 # x * x
17        mflo $t2 # t2 = x^2
18        mult $t2,$t1 # x^2 * X
19        mflo $t3 # t3 = x^3
20        mult $t3,$t1 # x^3 * x
21        mflo $t4 # t4 = x^4
22
23        add $t5,$t4,$t3 # t5 = x^4 + x^3
24        sll $t2,$t2,1 # t2 = x^2 * 2
25        sub $t5,$t5,$t2 # t5 = t5 - t2
26        sw $t5,4($s0) # Y = t5
27        j fim
28    impar:
29        mult $t1,$t1 # x * x
30        mflo $t2 # t2 = x^2
31        mult $t2,$t1 # x^2 * X
32        mflo $t3 # t3 = x^3
33        mult $t3,$t1 # x^3 * x
34        mflo $t4 # t4 = x^4
35        mult $t4,$t1 # x^4 * x
36        mflo $t5 # t5 = x^5
37
38        sub $t5,$t5,$t3 # t5 = x^5 - x^3
39        addi $t5,$t5,1 # t5 = t5 + 1
40        sw $t5,4($s0) # Y = t5
41    fim:
42
43    .data
44    X: .word 42
45    Y: .word 0
```

programa17.asm

6.

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

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:

```
1 # endereço base — $s0
2 .text
3 .globl main
4 main:
5     # Colocar endereço base da memória em s0
6     addi $s0,$zero,0x1001
7     sll  $s0,$s0,16
8
9     # Verifica se X é par ou impar
10    lw  $t1,0($s0) # t1 = X
11    slt $t2,$zero,$t1 # if (x > 0)
12    bne $t2,$zero,maior # goto maior
13    j  menorIgual      # else goto menorIgual
14
15    maior:
16        mult $t1,$t1 # x * x
17        mflo $t2 # t2 = x^2
18        mult $t2,$t1 # x^2 * X
19        mflo $t3 # t3 = x^3
20        addi $t3,$t3,1 # t3 = x^3 + 1
21        sw  $t3,4($s0)
22        j  fim
23
24    menorIgual:
25        mult $t1,$t1 # x * x
26        mflo $t2 # t2 = x^2
27        mult $t2,$t1 # x^2 * X
28        mflo $t3 # t3 = x^3
29        mult $t3,$t1 # x^3 * x
30        mflo $t4 # t4 = x^4
31        addi $t4,$t4,-1 # t4 = x^4 - 1
32        sw  $t4,4($s0)
33
34    fim:
35
36    .data
37    X: .word 42
38    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:

```
1  # x - $s0
2  # y - $s1
3  # z - $s2
4  # resultado - $s3
5  .text
6  .globl main
7  main:
8      addi $t0,$zero,0x0018
9      sll $t0,$t0,16
10     ori $s0,$t0,0x6A00 # x = 0x186A00
11
12     addi $t0,$zero,0x0001
13     sll $t0,$t0,16
14     ori $s1,$t0,0x3880 # y = 0x13880
15
16     addi $t0,$zero,0x0006
17     sll $t0,$t0,16
18     ori $s2,$t0,0x1A80 # z = 0x61A80
19
20     div $s0,$s2
21     mflo $t1 # t1 = x / z
22     mult $t1,$s1
23     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:

```
1 # i - $s0
2 # j - $s1
3 # soma - $s2
4 .text
5 .globl main
6 main:
7     # Colocar endereço base da memória em t0
8     addi $t0,$zero,0x1001
9     sll $t0,$t0,16
10    addi $s0,$zero,0 # começa i = 0
11    addi $t3,$zero,99 # Colocando t3 = 99
12
13 do: # Preenche o vetor com impares até 100
14    # t1 = 2 * i + 1
15    sll $t1,$s0,1
16    addi $t1,$t1,1
17
18    # vetor[i] = t1
19    sll $t2,$s0,2
20    add $t2,$t2,$t0
21    sw $t1,0($t2)
22
23    addi $s0,$s0,1 # i++
24
25    slt $s1,$t1,$t3 # j = t1 < 99
26    bne $s1,$zero,do # while(j)
27    j endwhile
28
29 endwhile:
30
31    addi $s2,$zero,0 # Define soma = 0
32    addi $s0,$s0,-1 # i-- para remover ultimo i++
33 do2:
34    # soma += vetor[i]
35    sll $t4,$s0,2
36    add $t4,$t4,$t0
37    lw $t5,0($t4)
38    add $s2,$s2,$t5
39
40    addi $s0,$s0,-1 # i--
41    # while(i >= 0)
42    slt $t6,$s0,$zero
43    beq $t6,$zero,do2
44    j endwhile2
45
46 endwhile2:
47    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:

```
1  # i - $s0
2  # j - $s1
3  .text
4  .globl main
5  main:
6      # Colocar endereço base da memória em t0
7      addi $t0,$zero,0x1001
8      sll  $t0,$t0,16
9
10     addi $s0,$zero,0 # começa i = 0
11     addi $t3,$zero,100 # Colocando t3 = 100
12
13 doPar: # Preenche o vetor de 100
14     # t1 = 2 * i
15     sll  $t1,$s0,1
16
17     # vetor[i] = t1
18     sll  $t2,$s0,2
19     add  $t2,$t2,$t0
20     sw   $t1,0($t2)
21
22     addi $s0,$s0,1 # i++
23
24     # t1 != 100
25     bne  $t1,$t3,doPar
26     j    endWhilePar
27
28 endWhilePar:
29
30 addi $s1,$zero,0 # começa j = 0
31 addi $t3,$zero,99 # Colocando t3 = 99
32
33 doImpar: # Preenche o restante do vetor com ímpares até 100
34     # t1 = 2 * j + 1
35     sll  $t1,$s1,1
36     addi $t1,$t1,1
37
38     add  $t2,$s0,$s1 # i = i + j
39     sll  $t2,$t2,2
40     add  $t2,$t2,$t0
41     sw   $t1,0($t2)
42
43     addi $s1,$s1,1 # j++
44
45     sll  $t4,$t1,$t3 # t4 = t1 < 99
46     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:

```

1 #Endereco base — $s0
2 .text
3 .globl main
4 main:
5     # Colocar endereco base da memoria em s0
6     addi $s0,$zero,0x1001
7     sll  $s0,$s0,16
8
9     lw  $t0,0($s0) # t0 = A[0] = X
10    lw  $t1,4($s0) # t1 = A[1] = Y
11    mult $t0,$t1
12    mflo $t2      # t2 = t0 * t1
13    sw  $t2,8($s0) # K = A[2] = t2
14
15 .data
16 X: .word 4
17 Y: .word 2
18 K: .word 0

```

programa22.asm

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

R:

```

1 #Endereco base — $s0
2 # i — $s1
3 .text
4 .globl main
5 main:
6     # Colocar endereco base da memoria em s0
7     addi $s0,$zero,0x1001
8     sll  $s0,$s0,16
9
10    lw  $t0,0($s0) # t0 = X
11    lw  $t1,4($s0) # t0 = Y
12
13    addi $s1,$zero,1 # i = 1
14    add  $t2,$zero,$t0 # t2 = 0
15    do:
16        mult $t2,$t0
17        mflo $t2 # t2 = t2 * X
18        addi $s1,$s1,1 # i++
19        slt  $t3,$s1,$t1 # t3 = i < Y
20        bne  $t3,$zero,do # while(t3)
21        sw  $t2,8($s0) # K = t2
22
23 .data
24 X: .word 2
25 Y: .word 5
26 K: .word 0

```

programa23.asm

Imagens

Figura 1: Programa 13

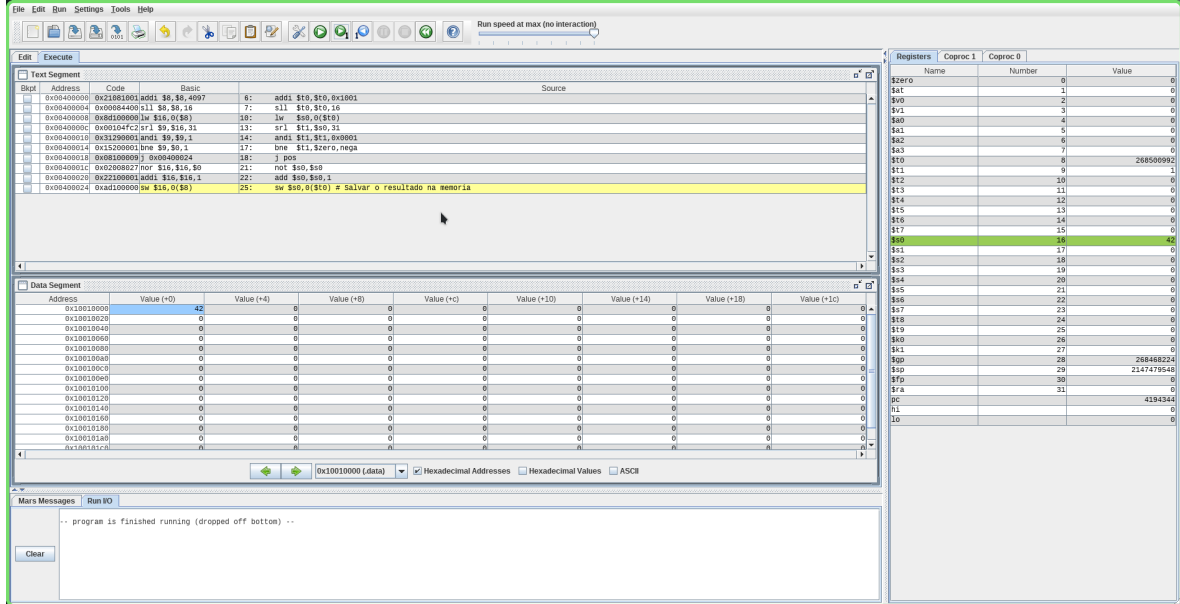


Figura 2: Programa 14

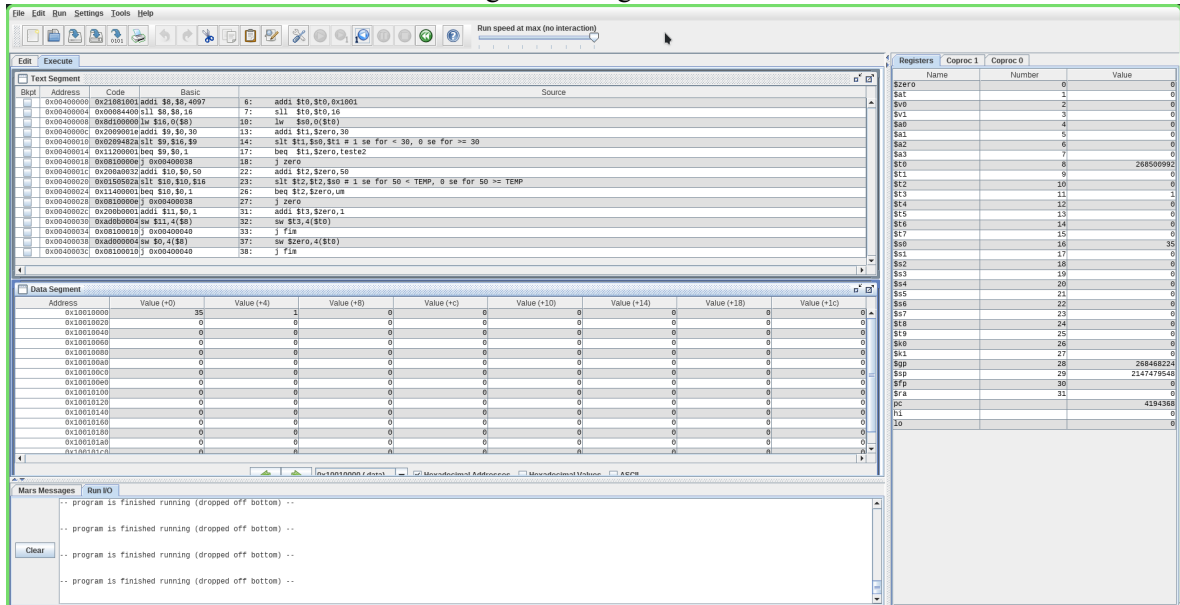


Figura 3: Programa 15

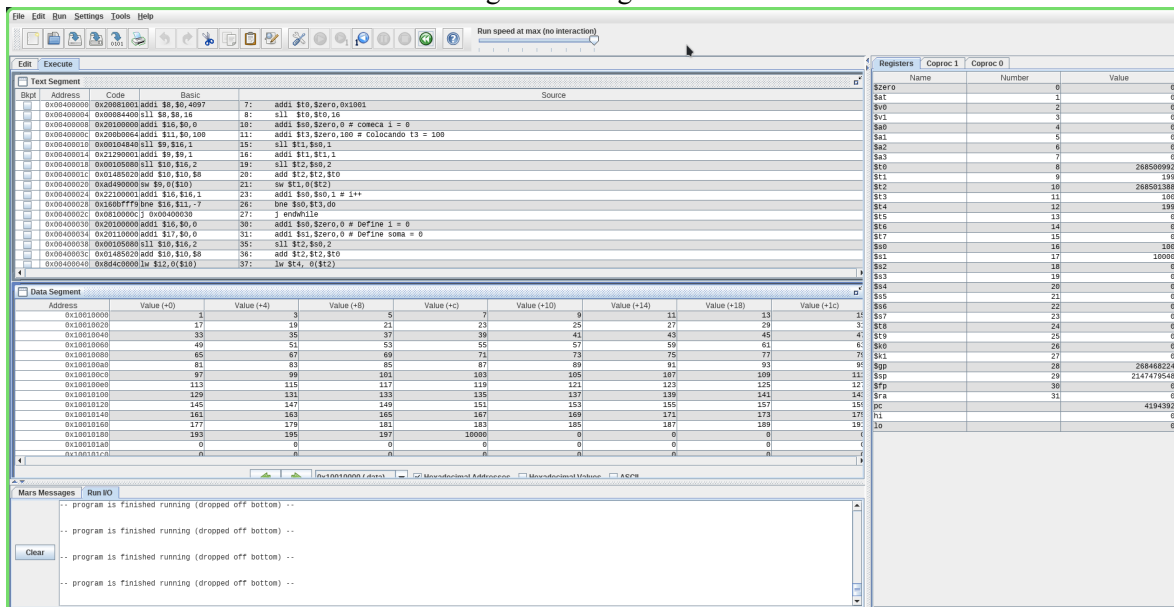


Figura 4: Programa 16

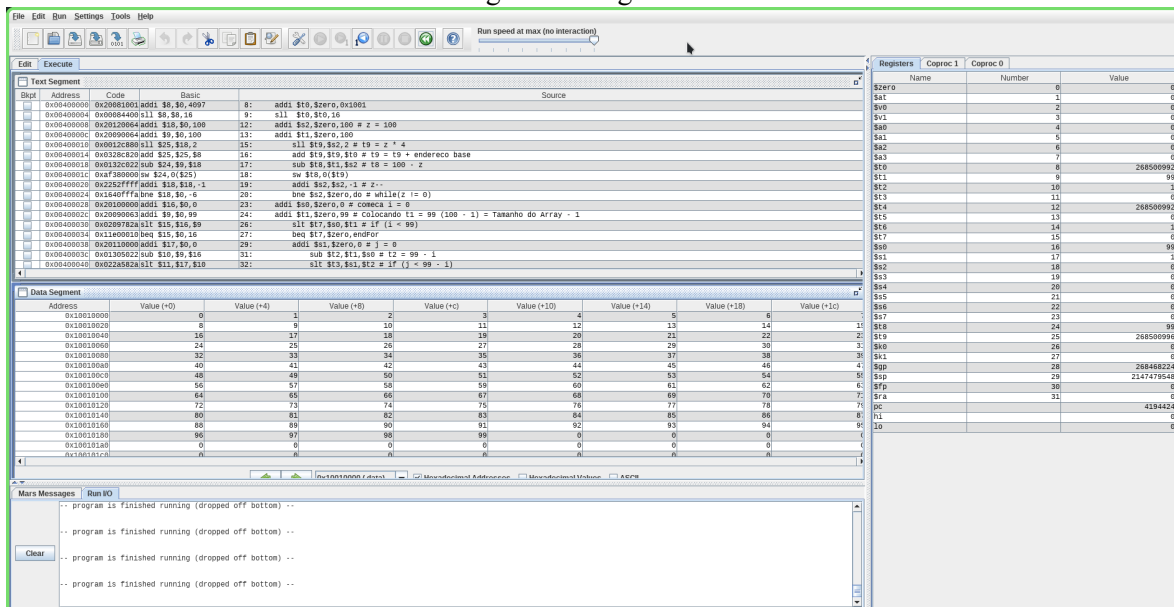


Figura 5: Programa 17

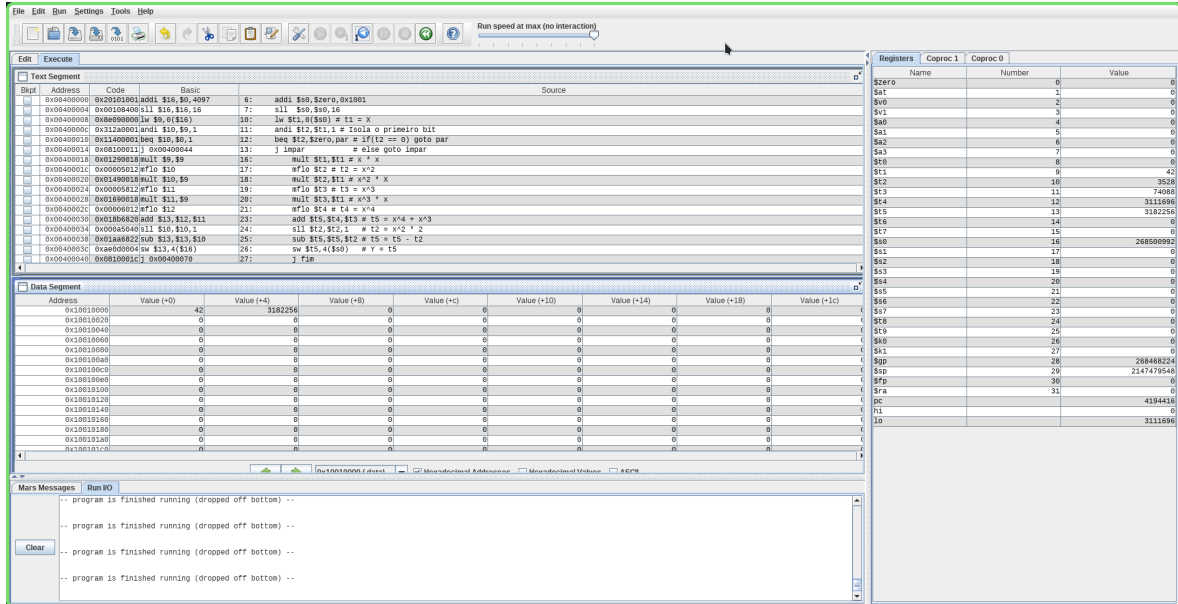


Figura 6: Programa 18

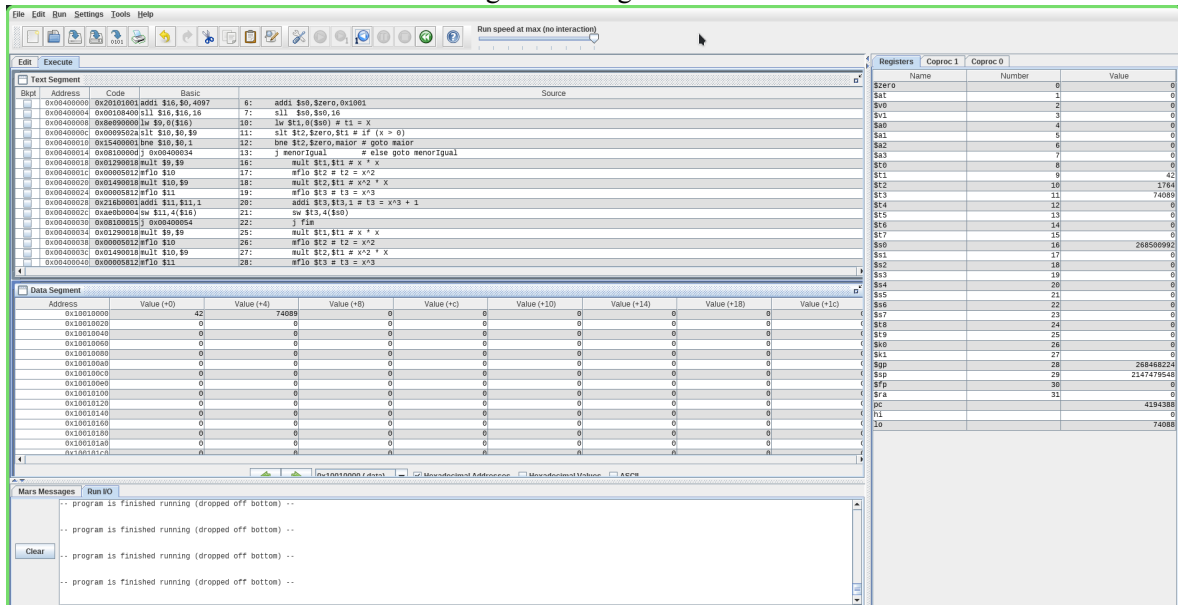


Figura 7: Programa 19

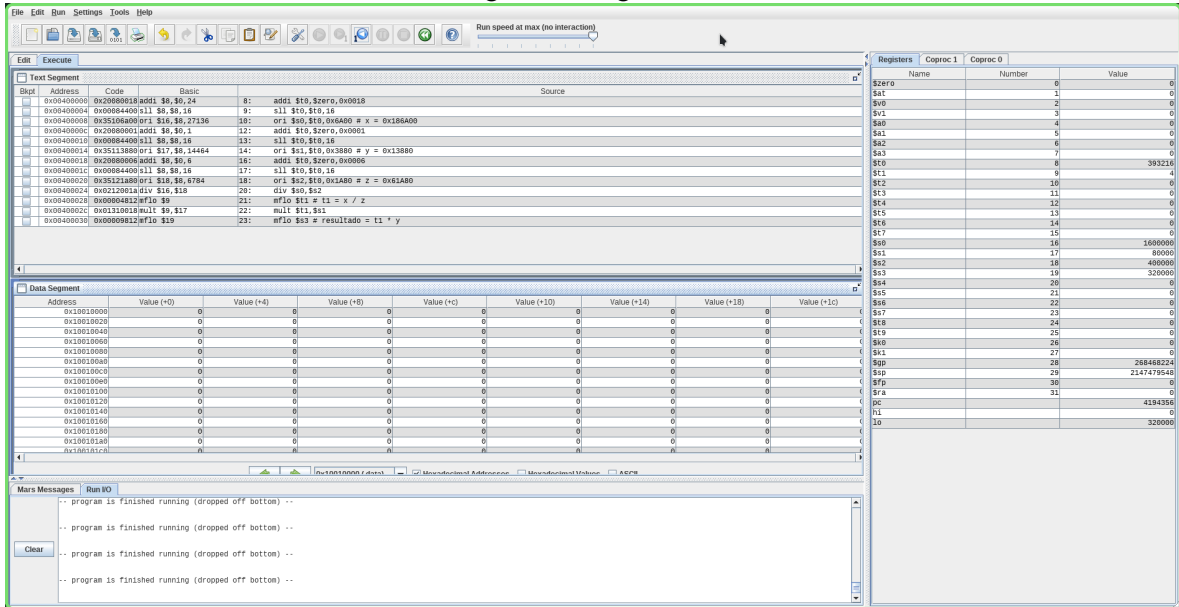


Figura 8: Programa 20

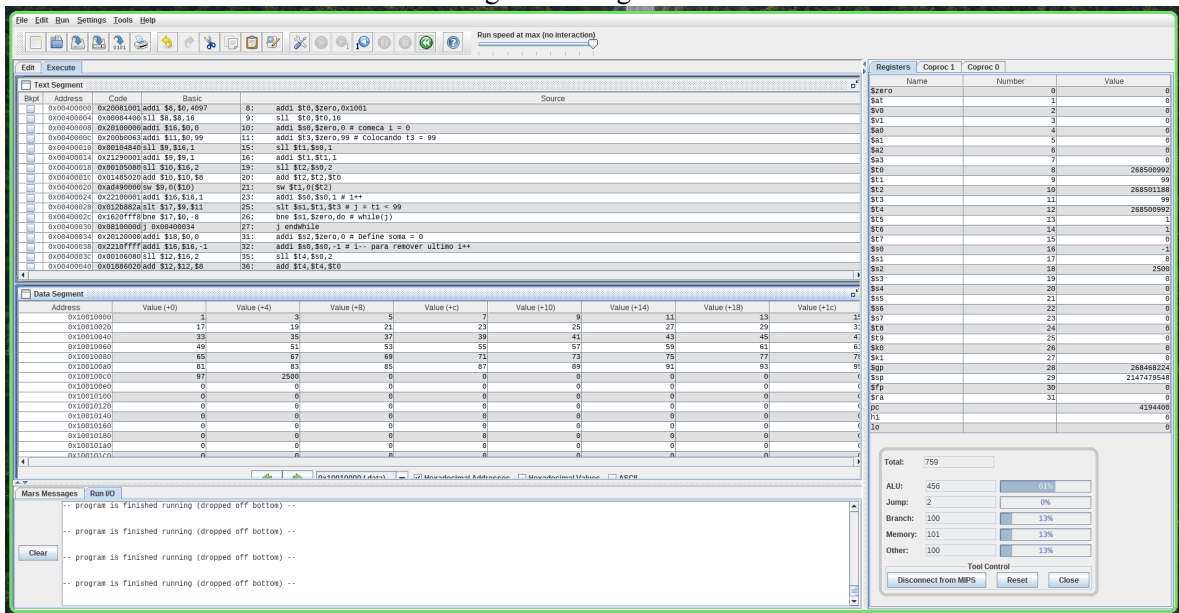


Figura 9: Programa 21

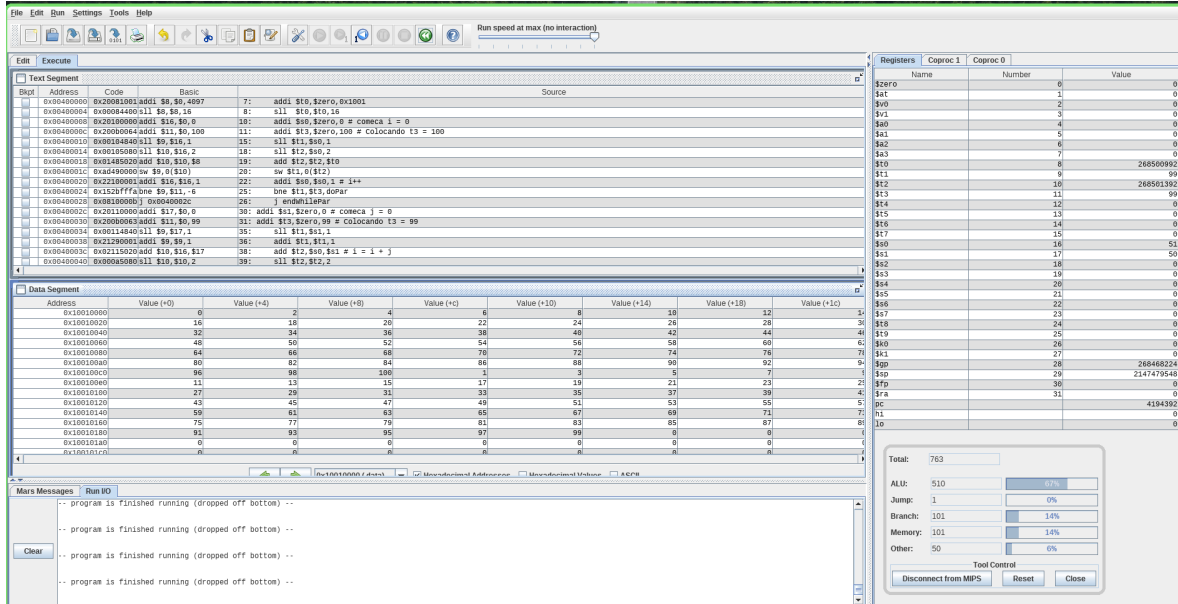


Figura 10: Programa 22

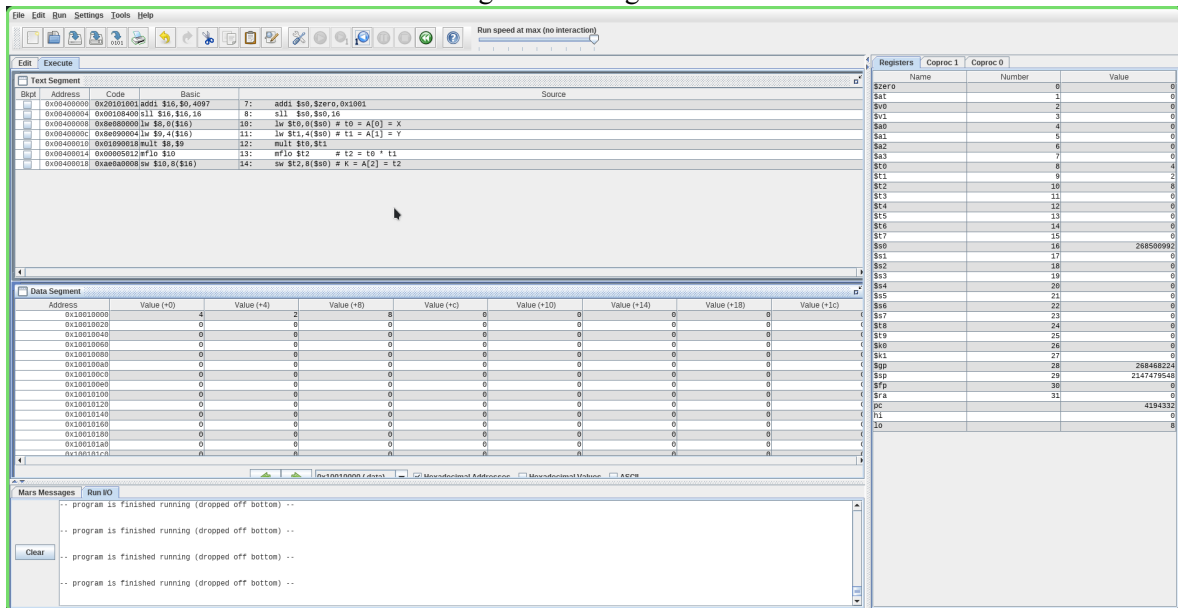


Figura 11: Programa 23

The screenshot displays a MIPS simulator window titled "Run speed at max (no interaction)". The interface is divided into several sections:

- Menu Bar:** File, Edit, Run, Settings, Tools, Help.
- Toolbar:** Contains icons for file operations, execution, and debugging.
- Text Segment:** Shows assembly code with columns for Skip, Address, Code, Basic, and Source. The code includes instructions like `addi $t0, $zero, 0x1061`, `slt $t0, $t0, 16`, `lw $t0, 0($t0)`, `lw $t1, 4($t0)`, `addi $t1, $zero, 1`, `add $t2, $zero, $t2 + 12 + 0`, `mult $t2, $t0`, `mflo $t2 = $t2 * X`, `addi $t1, $t1, 1`, `slt $t3, $t1, $t1`, `sw $t2, 0($t3)`, and `sw $t2, 8($t3)`.
- Data Segment:** A table showing memory addresses and their corresponding values. Most values are 0, with some non-zero values at higher addresses (e.g., 26850992, 2147479544, 4194352).
- Registers:** A table showing the state of MIPS registers. The Zero register is 0, and the PC register is 4194352.
- Message Log:** At the bottom, it shows the message "program is finished running (dropped off bottom)" repeated four times.