

# Exercício Prático 06

Aluno: Luís Augusto Starling Toledo

Matrícula: 761670

## Parte 1

### 1. O que é um arquivo fonte?

A. um arquivo de texto que contém instruções de linguagem de programação.

### 2. O que é um registrador?

A. parte do sistema de computador que mantém o controle dos parâmetros do sistema.

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

A. #

### 4. Quantos bits há em cada instrução de máquina MIPS?

C. 32

### 5. O que é o contador de programa?

D. parte do processador que contém o endereço da próxima instrução de máquina para ser obtida.

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

C. 4

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

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

### 8. O que é um endereço simbólico?

D. um nome usado no código-fonte em linguagem assembly para um local na memória.

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

B. 0x00400000

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

A. operando imediato

**11. Como é chamada uma operação lógica executada entre bits de cada coluna dos operandos para produzir um bit de resultado para cada coluna?**

B. operação bitwise

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

D. Cada um dos registradores deve possuir 32 bit.

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

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

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

C. ori \$5,\$0,48

**15. A instrução de ori pode armazenar o complemento de dois de um número em um registrador?**

B. Sim.

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

D. andi \$8,\$8,0xFF

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

A. Todos os bits em zero.

**18. Todas as instruções de máquina têm os mesmos campos?**

A. Não. Diferentes de instruções de máquina possuem campos diferentes.

## Parte 2

### Implementar em MIPS/MARS os seguintes programas

programa 1 (add, addi, sub, lógicas) {

```
a = 2;
b = 3;
c = 4;
d = 5;
x = (a+b) - (c+d);
y = a - b + x;
b = x - y;
```

}

The screenshot displays the MARS MIPS simulator interface. The top window shows the assembly code for the program, which implements the given C code. The code uses registers \$s0 through \$s5 for variables a through y, and \$t0 through \$t5 for temporary calculations. The program starts with a label 'inicio' and ends with 'fim'.

Below the code window, the 'Text Segment' and 'Data Segment' are visible. The 'Text Segment' shows the compiled instructions with their addresses and codes. The 'Data Segment' shows the memory layout with addresses and values.

On the right side, a table lists the memory state, including registers and memory locations. The table has three columns: Name, Number, and Value.

Name	Number	Value
\$zero	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	0xffffffff
\$t1	9	0x00000009
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000002
\$s1	17	0xffffffff
\$s2	18	0x00000004
\$s3	19	0x00000005
\$s4	20	0xffffffff
\$s5	21	0xffffffff
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$a0	26	0x00000000
\$t1	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7fffffc0
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400028
hi		0x00000000
lo		0x00000000

The 'Mars Messages' window at the bottom shows the message: "program is finished running (dropped off bottom) --".

```
//programa 2 (add, addi, sub, lógicas) {
    x = 1;
    y = 5*x + 15;
}
```

The screenshot shows the Mars MIPS simulator interface. The top panel displays the assembly code for 'mips1.asm'. The code implements Program 2, which calculates  $y = 5x + 15$  for  $x = 1$ . The code uses the `addi` instruction to set `$s4` to 1, then a loop of `add` instructions to calculate  $5x$  (by adding `$s4` to `$t0` four times), and finally `addi` to add 15 to `$t0` and store the result in `$s5`.

The middle panel shows the 'Text Segment' with the disassembled instructions. The bottom panel shows the 'Data Segment' with memory addresses and values. The right panel shows the register file with values for all registers. The 'Mars Messages' panel at the bottom shows the program running successfully.

Name	Number	Value
\$zero	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	0x00000005
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000001
\$s5	21	0x00000014
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc0
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400018
hi		0x00000000
lo		0x00000000

```
// programa 3 (add, addi, sub, lógicas) {
    x = 3;
    y = 4;
    z = ( 15*x + 67*y)*4
}
```

```
1 .text
2
3 addi $s0, $zero, 3    # x = 3
4 addi $s1, $zero, 4    # y = 4
5
6 # 15 * x
7 addi $t0, $s0, $s0    # t0 = 2 * x
8 addi $t0, $t0, $t0    # t0 = 4 * x
9 addi $t0, $t0, $t0    # t0 = 8 * x
10 addi $t0, $t0, $t0    # t0 = 16 * x
11 sub $t0, $t0, $s0     # t0 = 15 * x
12
13 # 67 * y
14 addi $t1, $s1, $s1    # t1 = 2 * y
15 addi $t1, $t1, $t1    # t1 = 4 * y
16 addi $t1, $t1, $t1    # t1 = 8 * y
17 addi $t1, $t1, $t1    # t1 = 16 * y
18 addi $t1, $t1, $t1    # t1 = 32 * y
19 addi $t1, $t1, $t1    # t1 = 64 * y
20 add $t1, $t1, $s1     # t1 = 67 * y
21
22 # (15 * x + 67 * y) * 4
23 add $t2, $t0, $t1     # t2 = (15 * x + 67 * y)
24 addi $s2, $t2, $t2    # s2 = 2 * (15 * x + 67 * y)
25 addi $t2, $t2, $t2    # t2 = 4 * (15 * x + 67 * y)
26 # Fim
27
```



```
// programa 5 {
    x = 100000;
    y = 200000;
    z = x + y;
}
```

```
1 .text
2 ori $s0, $zero, 0x186A
3 sll $s0, $s0, 4
4 ori $s1, $zero, 0x30D4
5 sll $s1, $s1, 4
6 add $s2, $s0, $s1
7 |
```

The screenshot shows the Mars MIPS simulator interface. The main window displays the assembly code for the program, with the following instructions:

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3410186a	ori \$16,\$0,0x0000186a	2: ori \$s0, \$zero, 0x186A
	0x00400004	0x00108100	sll \$16,\$16,0x00000004	3: sll \$s0, \$s0, 4
	0x00400008	0x341130d4	ori \$17,\$0,0x000030d4	4: ori \$s1, \$zero, 0x30D4
	0x0040000c	0x00118900	sll \$17,\$17,0x00000004	5: sll \$s1, \$s1, 4
	0x00400010	0x02119020	add \$18,\$16,\$17	6: add \$s2, \$s0, \$s1

Below the assembly code, the Data Segment is shown with a table of memory addresses and values. The registers are also listed on the right side of the window.

Name	Number	Value
\$zero	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	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x0000186a
\$s1	17	0x000030d4
<b>\$s2</b>	<b>18</b>	<b>0x000493e0</b>
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400014
hi		0x00000000
lo		0x00000000

```
// programa 6 {
    x = o maior inteiro possível;
    y = 300000;
    z = x - 4y
}
```

```
mips1.asm
1 # carregar o maior valor possível (32767) em $s0
2 ori $s0, $zero, 0x7FFF # x = 32767 (0x7FFF)
3
4 # carregar o valor de y (300000) em $s1 usando deslocamento e soma
5 ori $s1, $zero, 0x493E # carrega 4 em $s1
6
7 # calcular 4 * y (multiplicar y por 4)
8 sll $s2, $s1, 4 # desloca y para a esquerda por 2 bits (multiplica por 4). Resultado em $s2 = 1200000
9
10 # Calcular z = x - 4y
11 sub $s3, $s0, $s2 # z = x - 4y, o resultado será armazenado em $s3
12
```



// programa 8

Inicialmente escreva um programa que faça:

\$8 = 0x12345678.

A partir do registrador \$8 acima, usando apenas instruções lógicas (or, ori, and, andi, xor, xori) e instruções de deslocamento (sll, srl e sra), você deverá obter os seguintes valores nos respectivos registradores:

\$9 = 0x12

\$10 = 0x34

\$11 = 0x56

\$12 = 0x78

The screenshot shows the MIPS assembler interface with the following assembly code:

```
1  # atribuir $8 = 0x12345678
2  ori $8, $zero, 0x1234
3  sll $8, $8, 16
4  ori $8, $8, 0x5678
5
6  # obtem $9 = 12
7  srl $9, $8, 24 # $9 = 12
8
9  # obtem $10 = 34
10 srl $10, $8, 16
11 andi $10, $10, 0x00FF
12
13 # obtem $11 = 56
14 srl $11, $8, 8
15 andi $11, $11, 0x00FF
16
17 # obtem $12 = 78
18 andi $12, $8, 0x00FF
19
```

The interface also displays the Text Segment and Data Segment. The Text Segment shows the assembly code with addresses and basic instructions. The Data Segment shows the memory layout with addresses and values.

Name	Number	Value
\$zero	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	0x12345678
\$t1	9	0x00000012
\$t2	10	0x00000034
\$t3	11	0x00000056
\$t4	12	0x00000078
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400024
hi		0x00000000
lo		0x00000000



## Para os programas a seguir use instruções de Memória (lw e sw)

// programa 9

Considere a memória inicial da seguinte forma:

.text

.data

x1: .word 15

x2: .word 25

x3: .word 13

x4: .word 17

soma: .word -1

Escrever um programa que leia todos os números, calcule e substitua o valor da variável soma por este valor.

The screenshot displays the MARS MIPS simulator interface. The top window shows the assembly code for `mips1.asm`. The code declares variables `x1`, `x2`, `x3`, `x4`, and `soma` in the `.data` segment. It then loads these values into registers `$s0`, `$s1`, `$s2`, and `$s3` in the `.text` segment. The program calculates the sum of `x1`, `x2`, `x3`, and `x4` and stores the result in `soma`.

The bottom window shows the memory state. The `Text Segment` window displays the assembly code with the corresponding machine code and the address of each instruction. The `Data Segment` window shows the memory layout, with the value of `soma` (0x00000000) highlighted in green.

Name	Number	Value
\$zero	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	0x10010000
<b>\$t1</b>	<b>9</b>	<b>0x00000046</b>
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x0000000f
\$s1	17	0x00000019
\$s2	18	0x0000000d
\$s3	19	0x00000011
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffcfc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400024
hi		0x00000000
lo		0x00000000

// programa 10

Considere o seguinte programa:  $y = 127x - 65z + 1$

Faça um programa que calcule o valor de  $y$  conhecendo os valores de  $x$  e  $z$ . Os valores de  $x$  e  $z$

estão armazenados na memória e, na posição imediatamente a seguir, o valor de  $y$  deverá ser escrito, ou seja:

.data

x: .word 5

z: .word 7

y: .word 0 # esse valor deverá ser sobrescrito após a execução do programa.

The screenshot displays the MARS MIPS simulator interface. The top window shows the assembly code for 'mips1.asm'. The code defines variables x, z, and y in the .data section, then calculates y using the formula  $y = 127x - 65z + 1$  using MIPS instructions. The bottom window shows the memory state, with the Text Segment and Data Segment visible. The Data Segment shows the values of x, z, and y at their respective memory addresses.

**Assembly Code (mips1.asm):**

```
1 .data
2 x: .word 5
3 z: .word 7
4 y: .word 0 # esse valor deverá ser sobrescrito após a execução do programa.
5
6 # y = 127x - 65z + 1
7 # s0 -> x
8 # s1 -> y
9 # s2 -> z
10
11 .text
12 lui $t0, 0x1001
13 lw $s0, 0($t0) # MEM[0] = x
14 lw $s2, 4($t0) # MEM[1] = z
15
16 # calcular 127 * x
17 ori $t1, $zero, 127
18 mult $t1, $s0
19 mflo $t2 # t2 = 127 * x
20
21 # calcular 65 * z
22 ori $t3, $zero, 65
23 mult $t3, $s2
24 mflo $t4 # t2 = 127 * x
25
26 # fazer 127x - 65z
27 sub $t5, $t2, $t4
28 # adicionar mais 1
29 addi $t5, $t5, 1
30
31 # retornar valor de t2 pra memoria
32 sw $t5, 8($t0)
33
```

**Memory State:**

Name	Number	Value
\$zero	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	0x10010000
\$t1	9	0x0000007f
\$t2	10	0x0000027b
\$t3	11	0x00000041
\$t4	12	0x000001c7
\$t5	13	0x000000b5
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000005
\$s1	17	0x00000000
\$s2	18	0x00000007
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffc00
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400030
hi		0x00000000
lo		0x000001c7

// programa 11

Considere o seguinte programa:  $y = x - z + 300000$

Faça um programa que calcule o valor de y conhecendo os valores de x e z. Os valores de x e z

estão armazenados na memória e, na posição imediatamente a seguir, o valor de y deverá ser escrito, ou seja:

.data

x: .word 100000

z: .word 200000

y: .word 0 # esse valor deverá ser sobrescrito após a execução do programa.

The screenshot displays a MIPS assembler interface with the following components:

- Assembly Code (mips1.asm):**

```
1 .data
2 x: .word 100000
3 z: .word 200000
4 y: .word 0 # esse valor deverá ser sobrescrito após a execução do programa
5
6 # y = x - z + 300000
7 .text
8 # colocando 300000
9 ori $t1, $zero, 0x493E
10 sll $t1, $t1, 4
11
12 lui $t0, 0x1001
13 lw $s0, 0($t0) # MEM[0] = x
14 lw $s1, 4($t0) # MEM[1] = z
15 sub $t2, $s0, $s1
16 add $t2, $t2, $t1
17 sw $t2, 8($t0)
```
- Text Segment Table:**

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3409493e	ori \$9,\$0,0x0000493e	9: ori \$t1, \$zero, 0x493E
	0x00400004	0x00094900	sll \$9,\$9,0x00000004	10: sll \$t1, \$t1, 4
	0x00400008	0x3c081001	lui \$8,0x00001001	12: lui \$t0,0x1001
	0x0040000c	0x8d100000	lw \$16,0x00000000(\$8)	13: lw \$s0, 0(\$t0) # MEM[0] = x
	0x00400010	0x8d110004	lw \$17,0x00000004(\$8)	14: lw \$s1, 4(\$t0) # MEM[1] = z
	0x00400014	0x02115022	sub \$10,\$16,\$17	15: sub \$t2, \$s0, \$s1
	0x00400018	0x01495020	add \$10,\$10,\$9	16: add \$t2, \$t2, \$t1
	0x0040001c	0xad0a0008	sw \$10,0x00000008(\$8)	17: sw \$t2, 8(\$t0)
- Data Segment Table:**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x000186a0	0x00030d40	0x00030d40	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
- Registers Table:**

Name	Number	Value
\$zero	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	0x10010000
\$t1	9	0x000493e0
\$t2	10	0x00030d40
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x000186a0
\$s1	17	0x00030d40
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400020
hi		0x00000000
lo		0x00000000

// programa 12

Considere a seguinte situação:

```
int ***x;
```

onde x contem um ponteiro para um ponteiro para um ponteiro para um inteiro.

Nessa situação, considere que a posição inicial de memória contenha o inteiro em questão.

Coloque todos os outros valores em registradores, use os endereços de memória que quiser dentro do espaço de endereçamento do Mips.

Resumo do problema:

$k = \text{MEM} [ \text{MEM} [ \text{MEM} [ x ] ] ]$ .

Crie um programa que implemente a estrutura de dados acima, leia o valor de K, o multiplique por 2 e o reescreva no local correto conhecendo-se apenas o valor de x.

```
1 .data
2 ppp_x: .word 0x10010004
3 pp_x:  .word 0x10010008
4 p_x:   .word 0x1001000C
5 x:     .word 10
6
7 .text
8 lui $t0, 0x1001
9 lw $t1, 0($t0) # ponteiro de ponteiro de ponteiro
10 lw $t1, 0($t1) # ponteiro de ponteiro
11 lw $t1, 0($t1) # ponteiro
12 lw $s0, 0($t1) # valor
13 sll $s0, $s0, 1
14 sw $s0, 0($t1) # MEM[t1] = k
15
```

The screenshot shows a MIPS simulator interface. The main window displays the assembly code from the previous block. Below the code, there are three panels: Text Segment, Data Segment, and Registers.

**Text Segment:** This panel shows the assembly code and its basic blocks. The code is as follows:

```
8: lui $t0, 0x1001
9: lw $t1, 0($t0) # ponteiro de ponteiro de ponteiro
10: lw $t1, 0($t1) # ponteiro de ponteiro
11: lw $t1, 0($t1) # ponteiro
12: lw $s0, 0($t1) # valor
13: sll $s0, $s0, 1
14: sw $s0, 0($t1) # MEM[t1] = k
```

**Data Segment:** This panel shows the memory layout. The address 0x1001000C contains the value 0x00000014, which is the value of x multiplied by 2.

**Registers:** This panel shows the current state of the registers. The register \$s0 contains the value 0x00000014, which is the value of x multiplied by 2.

## Para os programas a seguir use instruções de desvio (beq, bne, j)

// programa 13:

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 A.

The screenshot displays the Mips1.asm editor and its execution interface. The editor shows the following assembly code:

```
1 .text
2 .globl main
3 main:
4     lui $t0, 0x1001
5     lw $s0, 0($t0) # x = A
6     srl $t1, $s0, 31 # armazenar o bit do sinal do numero ( em complemento de dois )
7     beq $t1, $zero, fim # se nao for negativo, ir para o fim
8
9 modulo:
10    sub $s0, $zero, $s0 # 0 - (-4) = 4
11    sw $s0, 0($t0) # retorna o valor pra memoria
12
13
14 fim:
15 .data
16 A: .word -54
17
```

The execution interface shows the registers and data segment. The registers table is as follows:

Name	Number	Value
\$zero	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	0x10010000
\$t1	9	0x00000001
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000036
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7ffffc00
\$fp	30	0x00000000
\$ra	31	0x00000000
\$pc		0x00400018
\$hi		0x00000000
\$lo		0x00000000

The data segment table is as follows:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000036	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

The execution interface also shows the Text Segment and Data Segment tables. The Text Segment table is as follows:

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c081001	lui \$8,0x00001001	4: lui \$t0, 0x1001
	0x00400004	0x8d100000	lw \$16,0x00000000(\$8)	5: lw \$s0, 0(\$t0) # x = A
	0x00400008	0x00104fc2	srl \$9,\$16,0x0000001f	6: srl \$t1, \$s0, 31 # armazenar o bit do sinal do numero ( em ...
	0x0040000c	0x11200002	beq \$9,\$0,0x00000002	7: beq \$t1, \$zero, fim # se nao for negativo, ir para o fim
	0x00400010	0x00108022	sub \$16,\$0,\$16	10: sub \$s0, \$zero, \$s0 # 0 - (-4) = 4
	0x00400014	0xad100000	sw \$16,0x00000000(\$8)	11: sw \$s0, 0(\$t0) # retorna o valor pra memoria

The Data Segment table is as follows:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000036	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

The execution interface also shows the Registers table, which is a subset of the registers table shown above.

// programa 14:

Escreva um programa que leia um valor A da memória, identifique se o número é par ou não. Um valor deverá ser escrito na segunda posição livre da memória (0 para par e 1 para ímpar).

mips1.asm

```
1 .data
2 A: .word 2
3 resp: .word -1
4
5 .text
6 lui $t0, 0x1001
7 lw $s0, 0($t0)
8 andi $t1, $s0, 0x1
9 sw $t1, 4($t0) # armazena 0 se for par e 1 se for ímpar
10
```

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c081001	lui \$t0, 0x00001001	6: lui \$t0, 0x1001
	0x00400004	0x8d100000	lw \$s0, 0x00000000(\$t0)	7: lw \$s0, 0(\$t0)
	0x00400008	0x32090001	andi \$t1, \$s0, 0x00000001	8: andi \$t1, \$s0, 0x1
	0x0040000c	0xad090004	sw \$t1, 0x00000004(\$t0)	9: sw \$t1, 4(\$t0) # armazena 0 se for par e 1 se for ímpar

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000002	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Registers

Name	Number	Value
\$zero	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	0x10010000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000002
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400010
hi		0x00000000
lo		0x00000000

0x10010000 (.data) [X] Hexadecimal Addresses [X] Hexadecimal Values [ ] ASCII

// programa 15:

Escrever um programa que crie um vetor de 100 elementos na memória onde  $\text{vetor}[i] = 2*i + 1$ .

Após a última posição do vetor criado, escrever a soma de todos os valores armazenados do vetor.

Use o MARS para verificar a quantidade de instruções conforme o tipo (ULA, Desvios, Mem ou Outras)

The image shows the MARS MIPS simulator interface. The top pane displays the assembly code for 'mips1.asm'. The code initializes registers \$t0, \$s0, \$s1, and \$t1, then enters a loop to calculate and store values in a vector, and finally prints the sum of all values in the vector.

```
1  # $t0 => vet[0]
2  # $s0 -> i
3  # $s1 -> acumulador
4  # $t1 -> TAM
5
6  .text
7  .globl main
8  main:
9      lui $t0, 0x1001
10     ori $s0, $zero, 0      # i = 0
11     ori $s1, $zero, 0      # acumulador = 0
12     ori $t1, $zero, 100    # TAM = 100
13
14     vetor:
15         sll $t2, $s0, 1      # t2 = 2*i
16         addi $t2, $t2, 1     # t2 = 2*i + 1
17         sw $t2, 0($t0)       # vet[i] = 2*i + 1
18         add $s1, $s1, $t2    # acumulador = acumulador + (2*i + 1)
19         addi $t0, $t0, 4     # endereço a ser guardado, aumenta 4 para cada iteração
20         addi $s0, $s0, 1     # i = i + 1
21         bne $s0, $t1, vetor
22
23     fim:
24         sw $s1, 0($t0)       # resultado da soma de todos os números é armazenado
25
```

The bottom pane shows the 'Text Segment' and 'Data Segment' memory views. The 'Text Segment' view shows the assembly code with addresses and hex values. The 'Data Segment' view shows the memory layout, with the vector values stored in memory starting at address 0x10010000.

Name	Number	Value
\$zero	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	0x10010190
\$t1	9	0x00000064
\$t2	10	0x000000c7
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000064
\$s1	17	0x00002710
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400030
hi		0x00000000
lo		0x00000000

// programa 16

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.

**mips1.asm**

```
1  # $s0 -> x
2  # $s1 -> y
3  # $s2 -> z
4
5  .text
6  .globl main
7  main:
8      ori $t0, $zero, 0x186A
9      sll $s0, $t0, 8      # x = 1.600.000
10     ori $t0, $zero, 0x1388
11     sll $s1, $t0, 4      # y = 80.000
12     ori $t0, $zero, 0x6A8
13     sll $s2, $t0, 4      # z = 400.000
14
15     div $s0, $s2        # lo = 1.600.000 / 400.000
16     mflo $t0            # t0 = 4
17     mult $t0, $s1       # lo = 4 * 80.000
18     mflo $s2            # s2 = lo
19
```

**Text Segment**

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3408186a	ori \$t0,\$zero,0x186a	8: ori \$t0, \$zero, 0x186A
	0x00400004	0x00088200	sll \$t0,\$t0,8	9: sll \$s0, \$t0, 8 # x = 1.600.000
	0x00400008	0x34081388	ori \$t0,\$zero,0x1388	10: ori \$t0, \$zero, 0x1388
	0x0040000c	0x00088900	sll \$t0,\$t0,4	11: sll \$s1, \$t0, 4 # y = 80.000
	0x00400010	0x340806a8	ori \$t0,\$zero,0x6a8	12: ori \$t0, \$zero, 0x6A8
	0x00400014	0x00089100	sll \$t0,\$t0,4	13: sll \$s2, \$t0, 4 # z = 400.000
	0x00400018	0x0212001a	div \$s0,\$s2	15: div \$s0, \$s2 # lo = 1.600.000 / 400.000
	0x0040001c	0x00004012	mflo \$t0	16: mflo \$t0 # t0 = 4
	0x00400020	0x01110018	mult \$t0,\$s1	17: mult \$t0, \$s1 # lo = 4 * 80.000
	0x00400024	0x00009012	mflo \$s2	18: mflo \$s2 # s2 = lo

**Data Segment**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

**Register File**

Name	Number	Value
\$zero	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	0x00000003a
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00186a00
\$s1	17	0x00013880
\$s2	18	0x0046cd00
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400028
hi		0x00000000
lo		0x0046cd00



// programa 17

Para a expressão a seguir, escreva um programa que calcule o valor de k:

$k = x * y$  (Você deverá realizar a multiplicação através de somas!)

O valor de x deve ser lido da primeira posição livre da memória e o valor de y deverá lido da segunda posição livre. O valor de k, após calculado, deverá ainda ser escrito na terceira posição livre da memória.

The screenshot displays a MIPS assembler interface with the following components:

- Assembly Code (mips1.asm):**

```
1  # $s0 -> endereço base
2  # $s1 -> x
3  # $s2 -> k
4  # $s3 -> i
5
6  .text
7  .globl main
8  main:
9      lui $t0, 0x1001          # Carrega o endereço base em $t0
10     lw $s0, 0($t0)           # x = MEM[$t0]
11     lw $s1, 4($t0)           # y = MEM[$t0 + 4]
12     ori $s3, $zero, 0        # i = 0
13
14     multiplicacao:
15         add $s2, $s2, $s0     # k = k + x
16         addi $s3, $s3, 1      # i = i + 1
17         bne $s3, $s1, multiplicacao # Se i != y, volta para "multiplicacao"
18
19     resultado:
20         sw $s2, 8($t0)        # Salva o resultado em MEM[$t0 + 8]
21
22     .data
23     x: .word 20
24     y: .word 10
25     k: .word -1              # Será sobrescrito no final
26
```
- Text Segment Table:**

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c081001	lui \$t0, 0x00001001	9: lui \$t0, 0x1001 # Carrega o endereço base em \$t0
	0x00400004	0x8d100000	lw \$s0, 0(\$t0)	10: lw \$s0, 0(\$t0) # x = MEM[\$t0]
	0x00400008	0x8d110004	lw \$s1, 4(\$t0)	11: lw \$s1, 4(\$t0) # y = MEM[\$t0 + 4]
	0x0040000c	0x34130000	ori \$s3, \$zero, 0	12: ori \$s3, \$zero, 0 # i = 0
	0x00400010	0x22509020	add \$s2, \$s2, \$s0	15: add \$s2, \$s2, \$s0 # k = k + x
	0x00400014	0x22730001	addi \$s3, \$s3, 1	16: addi \$s3, \$s3, 1 # i = i + 1
	0x00400018	0x1671ffff	bne \$s3, \$s1, 0xfffffdd	17: bne \$s3, \$s1, multiplicacao # Se i != y, volta para "multiplicacao"
	0x0040001c	0xad120008	sw \$s2, 8(\$t0)	20: sw \$s2, 8(\$t0) # Salva o resultado em MEM[\$t0 + 8]
- Data Segment Table:**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000014	0x0000000a	0x000000c8	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
- Registers Table:**

Name	Number	Value
\$zero	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	0x10010000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000014
\$s1	17	0x0000000a
\$s2	18	0x000000c8
<b>\$s3</b>	<b>19</b>	<b>0x0000000a</b>
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400020
hi		0x00000000
lo		0x00000000

// programa 18

Para a expressão a seguir, escreva um programa que calcule o valor de k:

$k = x^y$

Obs: Você poderá utilizar o exercício anterior.

O valor de x deve ser lido da primeira posição livre da memória e o valor de y deverá lido da segunda posição livre. O valor de k, após calculado, deverá ainda ser escrito na terceira posição livre da memória.

Dê um valor para x e y (dê valores pequenos !!) e use o MARS para verificar a quantidade de instruções conforme o tipo (ULA, Desvios, Mem ou Outras)

mips1.asm

```

1  .globl main
2  main:
3      # Carregar valores de x e y da memória
4      lui $t0, 0x1001      # Carregar base do endereço
5      lw $t1, 0($t0)       # Carregar x em $t1
6      lw $t2, 4($t0)       # Carregar y em $t2
7      ori $t3, $zero, 1    # Inicializar $t3 (resultado inicial = 1)
8      ori $t4, $zero, 0    # Inicializar contador de expoente
9
10     potencia_loop:
11         beq $t4, $t2, resultado_k # Se o contador for igual a y, terminou a exponenciação
12         mul $t3, $t3, $t1        # Multiplica o resultado atual por x
13         addi $t4, $t4, 1        # Incrementa o contador
14         j potencia_loop        # Volta para multiplicar novamente
15
16     resultado_k:
17         sw $t3, 8($t0)          # Armazena o resultado (k) na posição de memória 3
18         # Fim do programa
19         li $v0, 10              # Sair do programa
20         syscall
21
22     .data
23     x: .word 3                  # Exemplo de valor de x
24     y: .word 4                  # Exemplo de valor de y
25     z: .word -1                 # Para armazenar o resultado k
26

```

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c081001	lui \$t0, 0x00001001	4: lui \$t0, 0x1001 # Carregar base do endereço
	0x00400004	0x8d090000	lw \$t1, 0(\$t0)	5: lw \$t1, 0(\$t0) # Carregar x em \$t1
	0x00400008	0x8d0a0004	lw \$t2, 4(\$t0)	6: lw \$t2, 4(\$t0) # Carregar y em \$t2
	0x0040000c	0x340b0001	ori \$t3, \$zero, 1	7: ori \$t3, \$zero, 1 # Inicializar \$t3 (resultado inicial = 1)
	0x00400010	0x340c0000	ori \$t4, \$zero, 0	8: ori \$t4, \$zero, 0 # Inicializar contador de expoente
	0x00400014	0x118a0003	beq \$t4, \$t2, resultado_k	11: beq \$t4, \$t2, resultado_k # Se o contador for igual a y, ter...
	0x00400018	0x71695802	mul \$t3, \$t3, \$t1	12: mul \$t3, \$t3, \$t1 # Multiplica o resultado atual por x
	0x0040001c	0x218c0001	addi \$t4, \$t4, 1	13: addi \$t4, \$t4, 1 # Incrementa o contador
	0x00400020	0x08100005	j 0x00400014	14: j potencia_loop # Volta para multiplicar novamente
	0x00400024	0xad0b0008	sw \$t3, 8(\$t0)	17: sw \$t3, 8(\$t0) # Armazena o resultado (k) na posição de ...
	0x00400028	0x2402000a	addiu \$t2, \$t2, 0x0000000a	19: li \$v0, 10 # Sair do programa
	0x0040002c	0x0000000c	syscall	20: syscall

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000003	0x00000004	0x00000051	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x0000000a
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x10010000
\$t1	9	0x00000003
\$t2	10	0x00000004
\$t3	11	0x00000051
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400030
hi		0x00000000
lo		0x00000051

## Desafio:

Todos viram durante a parte aritmética que podemos utilizar a ULA e registradores para multiplicar dois números através de 3 algoritmos.

Você deverá escrever um programa que leia dois números da memória (primeira posição e segunda posição) os multiplique e coloque o resultado na terceira posição a memória.

Procure usar a versão 3 do algoritmo de multiplicação, pode ser mais simples !!

Atenção que, ao multiplicarmos dois números de 32 bits a resposta poderá ser um número de 64 bits, assim a resposta deverá estar contida em dois registradores temporários, um armazenará a parte superior do número e outro a parte inferior, portanto duas posições de memória serão escritas (a terceira e a quarta).

mips1.asm

```
1 .globl main
2 main:
3     # Carregar valores de x e y da memória
4     lui $t0, 0x1001 # Carregar base do endereço
5     lw $t1, 0($t0) # Carregar x em $t1
6     lw $t2, 4($t0) # Carregar y em $t2
7     ori $t3, $zero, 0 # Inicializar resultado inferior (parte baixa) em $t3
8     ori $t4, $zero, 0 # Inicializar resultado superior (parte alta) em $t4
9     ori $t5, $zero, 0 # Inicializar contador (bit de multiplicação) em $t5
10
11 # Algoritmo de multiplicação por shift-and-add
12 multiplicacao_loop:
13     beq $t5, 32, resultado_k # Se o contador de bits chegar a 32, terminou a multiplicação
14
15     andi $t6, $t2, 1 # Verifica o bit menos significativo de y (multiplicador)
16     beq $t6, $zero, shift_step # Se o bit é 0, apenas desloca
17
18     # Se o bit de y for 1, soma o valor de x (movido) ao resultado inferior
19     add $t3, $t3, $t1 # Soma o valor de x ao resultado inferior
20     # Verifica se houve overflow no resultado inferior (soma maior que 32 bits)
21     bltz $t3, carry_bit # Se houve overflow, ajuste no resultado superior
22     j shift_step
23
24 carry_bit:
25     addi $t4, $t4, 1 # Ajusta a parte superior (overflow)
26
27 shift_step:
28     srl $t1, $t1, 1 # Desloca x para a esquerda (multiplica por 2)
29     srl $t2, $t2, 1 # Desloca y para a direita (divide por 2)
30     addi $t5, $t5, 1 # Incrementa o contador de bits
31     j multiplicacao_loop # Continua o loop de multiplicação
32
33 resultado_k:
34     # Armazena a parte inferior do resultado
35     sw $t3, 0($t0) # Armazena a parte inferior (k baixa) na posição de memória 3
36     # Armazena a parte superior do resultado
37     sw $t4, 12($t0) # Armazena a parte superior (k alta) na posição de memória 4
38
39 # Fim do programa
40 li $v0, 10 # Sair do programa
41 syscall
42
43 .data
44 x: .word 3 # Exemplo de valor de x
45 y: .word 5 # Exemplo de valor de y
46 z1: .word -1 # Para armazenar a parte baixa do resultado k
47 z2: .word -1 # Para armazenar a parte alta do resultado k
48
```

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c081001	lui \$8,0x00001001	4: lui \$t0, 0x1001 # Carregar base do endereço
	0x00400004	0x8d090000	lw \$9,0x00000000(\$8)	5: lw \$t1, 0(\$t0) # Carregar x em \$t1
	0x00400008	0x8d0a0004	lw \$10,0x00000004(\$8)	6: lw \$t2, 4(\$t0) # Carregar y em \$t2
	0x0040000c	0x340b0000	ori \$11,\$0,0x00000000	7: ori \$t3, \$zero, 0 # Inicializar resultado inferior (parte b...
	0x00400010	0x340c0000	ori \$12,\$0,0x00000000	8: ori \$t4, \$zero, 0 # Inicializar resultado superior (parte a...
	0x00400014	0x340d0000	ori \$13,\$0,0x00000000	9: ori \$t5, \$zero, 0 # Inicializar contador (bit de multiplica...
	0x00400018	0x20010020	addi \$1,\$0,0x00000020	13: beq \$t5, 32, resultado_k # Se o contador de bits chegar a 32...
	0x0040001c	0x102d000a	beq \$1,\$13,0x0000000a	
	0x00400020	0x314e0001	andi \$14,\$10,0x0000...	15: andi \$t6, \$t2, 1 # Verifica o bit menos significativo de y...
	0x00400024	0x11c00004	beq \$14,\$0,0x00000004	16: beq \$t6, \$zero, shift_step # Se o bit é 0, apenas desloca
	0x00400028	0x016e5820	add \$11,\$11,\$9	19: add \$t3, \$t3, \$t1 # Soma o valor de x ao resultado inferior
	0x0040002c	0x05600001	bltz \$11,0x00000001	21: bltz \$t3, carry_bit # Se houve overflow, ajuste no resultado ...
	0x00400030	0x0810000e	j 0x00400038	22: j shift_step
	0x00400034	0x218c0001	addi \$12,\$12,0x0000...	25: addi \$t4, \$t4, 1 # Ajusta a parte superior (overflow)
	0x00400038	0x00094840	srl \$9,\$9,0x00000001	28: srl \$t1, \$t1, 1 # Desloca x para a esquerda (multiplica p...
	0x0040003c	0x000a5042	srl \$10,\$10,0x00000001	29: srl \$t2, \$t2, 1 # Desloca y para a direita (divide por 2)
	0x00400040	0x218d0001	addi \$19,\$13,0x0000...	30: addi \$t5, \$t5, 1 # Incrementa o contador de bits
	0x00400044	0x0810000e	j 0x00400018	31: j multiplicacao_loop # Continua o loop de multiplicação
	0x00400048	0xad0b0008	sw \$11,0x00000008(\$8)	35: sw \$t3, 0(\$t0) # Armazena a parte inferior (k baixa) na ...
	0x0040004c	0xad0c000c	sw \$12,0x0000000c(\$8)	37: sw \$t4, 12(\$t0) # Armazena a parte superior (k alta) na p...
	0x00400050	0x2402000a	addiu \$2,\$0,0x0000000a	40: li \$v0, 10 # Sair do programa
	0x00400054	0x0000000c	syscall	41: syscall

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000003	0x00000005	0x0000000f	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000020
\$v0	2	0x0000000a
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x10010000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x0000000f
\$t4	12	0x00000000
\$t5	13	0x00000020
\$t6	14	0x00000000
\$t7	15	0x00000000
\$a0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffcfc
\$fp	30	0x00000000
\$ra	31	0x00000000
\$PC		0x00400058
hi		0x00000000
lo		0x00000000

## Parte 3

- 1. Se tivermos 2 inteiros, cada um com 32 bits, quantos bits podemos esperar para o produto?**  
C. 64
- 2. Quais os registradores que armazenam os resultados na multiplicação?**  
B. hi e lo
- 3. Qual a operação usada para multiplicar inteiros em comp. de dois?**  
A. mult
- 4. Qual instrução move os bits menos significativos da multiplicação para o reg. 8?**  
C. mflo \$8
- 5. Se tivermos dois inteiros, cada um com 32 bits, quantos bits deveremos estar preparados para receber no quociente?**  
B. 32
- 6. Após a instrução div, qual registrador possui o quociente?**  
A. lo
- 7. Qual a inst. Usada para dividir dois inteiros em comp. de dois?**  
D. div
- 8. Faça um arithmetic shift right de dois no seguinte padrão de bits: 1001 1011**  
B. 0010 0110
- 9. Qual o efeito de um arithmetic shift right de uma posição?**  
A. Se o inteiro for unsigned, o shift o divide por 2. Se o inteiro for signed, o shift o divide por 2.
- 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

# Parte 4

```
// programa 19
```

Escrever um programa que leia dois números da memória, a primeira e segunda posições respectivamente (os coloque em \$s0 e \$s1) e determine a quantidade de bits significantes de cada um. Coloque as respostas em \$t0 e \$t1, a partir desse resultado faça a multiplicação. Caso o número de bits significantes de ambos seja menor do que 32 a resposta deverá estar apenas em \$s2, caso contrário a resposta estará em \$s2 e \$s3 (LO e HI respectivamente). Para os exercícios a seguir, considere as variáveis com números abaixo de 16 bits, salvo se mencionado ao contrário.

```

1  # inicio
2
3  .data
4  x: .word 3
5  y: .word 4
6  .text
7  .globl main
8  main:
9
10     #t0 -> 0x10010000 (first position)
11
12     #x -> s0
13     #y -> s1
14     #k -> s2
15
16     ori $t2, $0, 0x1001          #t2 = 0x1001
17     sll $t2, $t2, 16             #t2 = 0x10010000
18     lw $s0, 0($t2)               #s0 = MEM[$t0]
19     lw $s1, 4($t2)               #s1 = MEM[$t0+4]
20     or $t3, $s0, $s0             #t3 = x
21     or $t4, $s1, $s1             #t4 = y
22     or $t0, $0, $0               #t0 = 0 (contador de x)
23     or $t1, $0, $0               #t1 = 0 (contador de y)
24
25 if:  beq $t3, $0, if2             #if(t3==0){ goto fim1 }
26     addi $t0, $t0, 1             #t0 = t0 + 1
27     srl $t3, $t3, 1              #t3 >> 1
28     j if
29
30 if2: beq $t4, $0, fim             #if(t4==0){ goto fim2 }
31     addi $t1, $t1, 1             #t1 = t1 + 1
32     srl $t4, $t4, 1              #t4 >> 1
33     j if2                         #goto if
34
35 fim:  mult $t0, $t1                #t0 * t1
36     mflo $t5                      #t5 = t0
37     slli $t6, $t5, 32            #if(t5<32){ t6=1 ; else { t6 = 0 }
38     bne $t6, 1, maior            #if(t6!=1){ goto maior }
39     or $t2, $0, $t5              #s2 = t5
40     mflo $t2                      #s2 = t0
41     j fim2                       #goto fim
42
43 maior: mghi $t2                    #s2 = hi
44     mflo $t3                      #s3 = t0
45
46 fim2:
47
48 # fim

```

Text Segment					Name	Number	Value
Bkpt	Address	Code	Basic	Source			
<input type="checkbox"/>	0x00400000	0x340a1001	ori \$t0,\$0,0x00001001	13: ori \$t2,\$0,0x1001	\$zero	0	0x00000000
<input type="checkbox"/>	0x00400004	0x000a5400	sll \$t0,\$t0,0x00000010	14: sll \$t2,\$t2,\$t6	\$at	1	0x00000001
<input type="checkbox"/>	0x00400008	0x8d500000	lw \$t6,0x00000000(\$t0)	15: lw \$s0,0(\$t2)	\$v0	2	0x00000000
<input type="checkbox"/>	0x0040000c	0x8d510004	lw \$t7,0x00000004(\$t0)	16: lw \$s1,4(\$t2)	\$v1	3	0x00000000
<input type="checkbox"/>	0x00400010	0x00206858	or \$t1,\$t6,\$0	17: or \$t3,\$s0,\$0	\$v2	4	0x00000000
<input type="checkbox"/>	0x00400014	0x00206825	or \$t2,\$t7,\$0	18: or \$t4,\$s1,\$0	\$v3	5	0x00000000
<input type="checkbox"/>	0x00400018	0x00004025	or \$s0,\$0,\$0	19: or \$t0,\$0,\$0	\$v4	6	0x00000000
<input type="checkbox"/>	0x0040001c	0x00004825	or \$s0,\$0,\$0	20: or \$t1,\$0,\$0	\$v5	7	0x00000000
<input type="checkbox"/>	0x00400020	0x11600003	beq \$t1,\$0,0x00000003	21: if: beq \$t3,\$0,\$t2	\$v6	8	0x00000000
<input type="checkbox"/>	0x00400024	0x21080001	addi \$t8,\$t8,0x00000001	22: addi \$t0,\$t0,\$t	\$v7	9	0x00000000
<input type="checkbox"/>	0x00400028	0x000b5842	srl \$t1,\$t1,0x00000001	23: srl \$t3,\$t3,\$t	\$v8	10	0x00000000
<input type="checkbox"/>	0x0040002c	0x08100008	j 0x00400020	24: j if	\$v9	11	0x00000000
<input type="checkbox"/>	0x00400030	0x11900003	beq \$t2,\$0,0x00000003	25: if2: beq \$t4,\$0,\$t1	\$t6	12	0x00000001
<input type="checkbox"/>	0x00400034	0x21290001	addi \$t9,\$t9,0x00000001	26: addi \$t1,\$t1,\$t	\$t7	13	0x00000000
<input type="checkbox"/>	0x00400038	0x000b6042	srl \$t2,\$t2,0x00000001	27: srl \$t4,\$t4,\$t	\$t8	14	0x00000000
<input type="checkbox"/>	0x0040003c	0x0810000c	j 0x00400030	28: j if2	\$t9	15	0x00000000
<input type="checkbox"/>	0x00400040	0x01090018	mult \$t8,\$t9	29: fim: mult \$t0,\$t1	\$s2	16	0x00000000
<input type="checkbox"/>	0x00400044	0x00006812	mflw \$t3	30: mflw \$t5	\$s3	17	0x00000000
<input type="checkbox"/>	0x00400048	0x29aee020	sllti \$t4,\$t3,0x0000...	31: sllti \$t6,\$t5,\$	\$s4	18	0x00000000
<input type="checkbox"/>	0x0040004c	0x20010001	addi \$t1,\$t0,0x00000001	32: bne \$t6,\$t1,maior	\$s5	19	0x00000000
<input type="checkbox"/>	0x00400050	0x142e0003	bne \$t1,\$t4,0x00000003	33: or \$s2,\$0,\$t5	\$s6	20	0x00000000
<input type="checkbox"/>	0x00400054	0x0000405c	or \$t8,\$t0,\$t3	34: mflw \$s2	\$s7	21	0x00000000
<input type="checkbox"/>	0x00400058	0x00009012	mflw \$t5	35: j fim2	\$s8	22	0x00000000
<input type="checkbox"/>	0x0040005c	0x0810001a	j 0x00400068	36: maior: mflw \$s2	\$s9	23	0x00000000
<input type="checkbox"/>	0x00400060	0x00009010	mflw \$t8	37: mflw \$s3	\$s0	24	0x00000000
<input type="checkbox"/>	0x00400064	0x00009812	mflw \$t9		\$k1	26	0x00000000
					\$gp	28	0x10080000
					\$sp	29	0x7ffffeff
					\$fp	30	0x00000000
					\$ra	31	0x00000000
					pc		0x00400068
					hi		0x00000000
					lo		0x00000000

Data Segment				
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x0000000003	0x0000000004	0x0000000000	0x0000000000
0x10010020	0x0000000000	0x0000000000	0x0000000000	0x0000000000

```
// programa 20
```

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

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

```

1  # inicio
2
3  .data
4  x: .word 3
5  .text
6  .globl main
7  main:
8      ori $t0, $0, 0x1001      #t0 -> 0x10010000 (first position)
9      sll $t0, $t0, 16        #t0 << 16
10
11     lw $s0, 0($t0)           #s0 = MEM[$t0]
12     andi $t1, $s0, 1         #t1 = s0 & 1
13     mult $s0, $s0            #s0 * s0
14     mflo $t2                 #t2 = x^2
15     mult $t2, $s0            #t2 * s0
16     mflo $t3                 #t3 = x^4
17     mult $t3, $s0            #t3 * s0
18     mflo $t4                 #t4 = x^8
19     mult $t4, $s0            #t4 * s0
20     mflo $t5                 #t5 = x^16
21     bne $t1, $0, impar      #if(t1!=0){ goto impar }
22     add $t2, $t2, $t2        #t2 = t2 + t2
23     add $s1, $t4, $t3        #s1 = t4 + t3
24     sub $s1, $s1, $t2        #s1 = s1 - t2
25     j fim                   #goto fim
26 impar: sub $s1, $t5, $t3     #s1 = t5 - t3
27     addi $s1, $s1, 1         #s1 = s1 + 1
28 fim:   sw $s1, 4($t0)        #MEM[4+$t0] = s1
29
30 # fim
31

```

[illegible]

```
// programa 21
```

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

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

[illegible]