

Nome: Victor Hugo Lopes Mota

Matrícula: 13/0136581

Trabalho 01

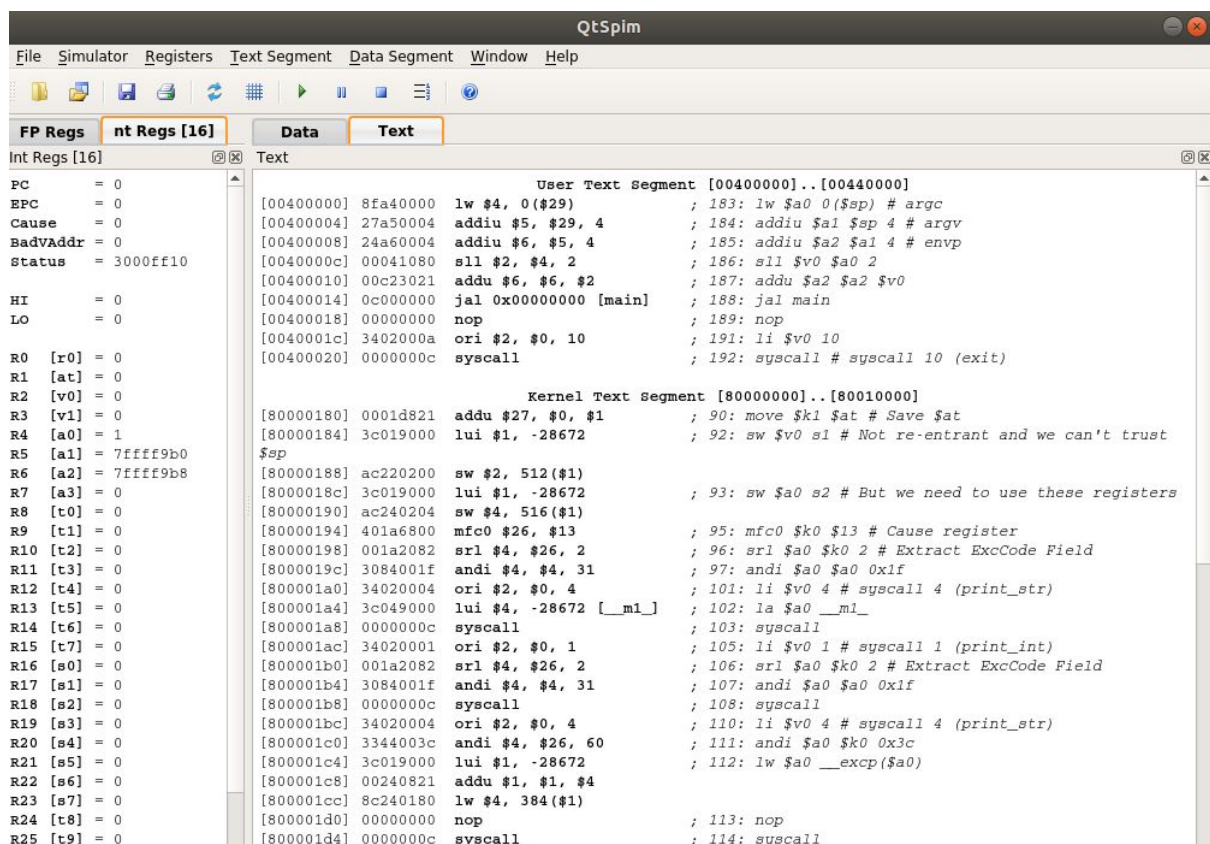
Sistema Operacional: Ubuntu 18.04

Simulador: QtSpim

Editor: Visual Studio

Passo a Passo:

O primeiro passo foi procurar um simulador para interpretar o assembly, com os .spim e .asm. O simulador escolhido foi o QtSpim:



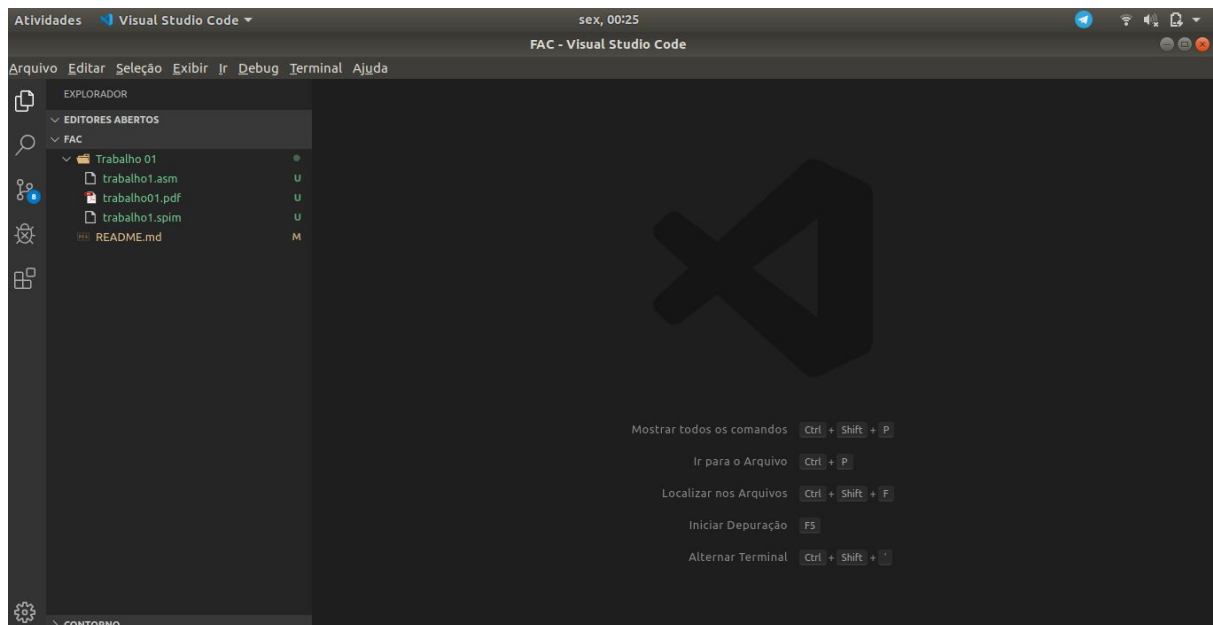
The screenshot shows the QtSpim MIPS simulator window. The 'Registers' tab is active, displaying the state of various registers. The 'Text' tab is also visible, showing the assembly code being executed. The assembly code is divided into two segments: 'User Text Segment' and 'Kernel Text Segment'. The 'User Text Segment' contains instructions for setting up the environment, including loading arguments and jumping to the main function. The 'Kernel Text Segment' contains instructions for handling system calls and managing the stack.

```
PC = 0
EPC = 0
Cause = 0
BadVAddr = 0
Status = 3000fff10
HI = 0
LO = 0
R0 [r0] = 0
R1 [at] = 0
R2 [v0] = 0
R3 [v1] = 0
R4 [a0] = 1
R5 [a1] = 7ffff9b0
R6 [a2] = 7ffff9b8
R7 [a3] = 0
R8 [t0] = 0
R9 [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0

User Text Segment [00400000]..[00440000]
[00400000] 8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080 sll $2, $4, 2 ; 186: sll $v0 $a0 2
[00400010] 00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
[00400014] 0c000000 jal 0x00000000 [main] ; 188: jal main
[00400018] 00000000 nop ; 189: nop
[0040001c] 3402000a ori $2, $0, 10 ; 191: li $v0 10
[00400020] 0000000c syscall ; 192: syscall # syscall 10 (exit)

Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821 addu $27, $0, $1 ; 90: move $k1 $at # Save $at
[80000184] 3c019000 lui $1, -28672 ; 92: sw $v0 $1 # Not re-entrant and we can't trust
$sp
[80000188] ac220200 sw $2, 512($1) ; 93: sw $a0 $2 # But we need to use these registers
[8000018c] 3c019000 lui $1, -28672 ; 96: srl $a0 $k0 2 # Extract ExcCode Field
[80000190] ac240204 sw $4, 516($1) ; 97: andi $a0 $a0 0x1f
[80000194] 401a6800 mfc0 $26, $13 ; 95: mfc0 $k0 $13 # Cause register
[80000198] 001a2082 srl $4, $26, 2 ; 96: srl $a0 $k0 2 # Extract ExcCode Field
[8000019c] 3084001f andi $4, $4, 31 ; 97: andi $a0 $a0 0x1f
[800001a0] 34020004 ori $2, $0, 4 ; 101: li $v0 4 # syscall 4 (print_str)
[800001a4] 3c049000 lui $4, -28672 [__m1] ; 102: la $a0 __m1_
[800001a8] 0000000c syscall ; 103: syscall
[800001ac] 34020001 ori $2, $0, 1 ; 105: li $v0 1 # syscall 1 (print_int)
[800001b0] 001a2082 srl $4, $26, 2 ; 106: srl $a0 $k0 2 # Extract ExcCode Field
[800001b4] 3084001f andi $4, $4, 31 ; 107: andi $a0 $a0 0x1f
[800001b8] 0000000c syscall ; 108: syscall
[800001bc] 34020004 ori $2, $0, 4 ; 110: li $v0 4 # syscall 4 (print_str)
[800001c0] 3344003c andi $4, $26, 60 ; 111: andi $a0 $k0 0x3c
[800001c4] 3c019000 lui $1, -28672 ; 112: lw $a0 __excp($a0)
[800001c8] 00240821 addu $1, $1, $4
[800001cc] 8c240180 lw $4, 384($1)
[800001d0] 00000000 nop ; 113: nop
[800001d4] 0000000c syscall ; 114: syscall
```

O segundo passo foi ler o trabalho e separar o que deveria ser feito inicialmente. O segundo passo foi escolher um editor para escrever o código assembly, e por escolha própria, foi escolhido o Visual Studio:



Como terceiro passo, foi o de ler um inteiro do teclado do usuário três vezes, pois o enunciado falou que seriam necessários três números.

```
.text
main:

    li $v0, 5 # Lê do teclado o número inserido
    syscall

    move $t0, $v0

    li $v0, 5 # Lê do teclado o número inserido
    syscall

    move $t1, $v0

    li $v0, 5 # Lê do teclado o número inserido
    syscall

    move $t2, $v0
```

Após ler os valores do teclado pelo usuário, as variáveis temporárias \$t0, \$t1 e \$t2, foram devidamente adicionadas com os valores fornecidos pelo usuário:

```
add $t0, $zero, $t0 # Primeiro valor: t0
add $t1, $zero, $t1 # Segundo valor: t1
add $t2, $zero, $t2 # Terceiro valor: t2 |
```

Após definir os valores das variáveis temporárias, o passo seguinte foi o de escrever as “Strings” pedidas no enunciado da questão:

```
.data
$add: .ascii "ADD: "
$sub: .ascii "\nSUB: "
$and: .ascii "\nAND: "
$or: .ascii "\nOR: "
$xor: .ascii "\nXOR: "
$mask: .ascii "\nMASK: "
$sll: .ascii "\nSLL("
$srl: .ascii "\nSRL("
$msg: .ascii "): "
$newline: .ascii "\n"
```

Para realizar o item 1, primeiro passo foi somar e subtrair os dois primeiros números informados e mostrar o resultado na tela:

```
# Questão 1

add $s0, $t0, $t1 # Soma dos dois primeiros valores
sub $s1, $t0, $t1 # Subtração dos dois primeiros valores

li $v0,4 # Comando de impressão de string na tela
la $a0, $add # Coloca o texto soma para ser impresso
syscall # Efetua a chamada ao sistema

li $v0,1 # Comando de impressão de inteiro na tela
la $a0, ($s0) # Coloca o registrador $s0 para ser impresso
syscall # Efetua a chamada ao sistema

li $v0,4 # Comando de impressão de string na tela
la $a0, $sub # Coloca o texto soma para ser impresso
syscall # Efetua a chamada ao sistema

li $v0,1 # Comando de impressão de inteiro na tela
la $a0, ($s1) # Coloca o registrador $s1 para ser impresso
syscall # Efetua a chamada ao sistema
```

No item 02, o enunciado pediu que fosse realizado as operações “AND, OR e XOR” e mostrados os resultados na tela:

```

# Questão 2

and $s2, $t0, $t1 # AND entre os dois primeiros valores
or $s3, $t0, $t1 # OR entre os dois primeiros valores
xor $s4, $t0, $t1 # XOR Entre os dois primeiros valores

li $v0,4 # Comando de impressão de string na tela
la $a0, $and # Coloca o texto and para ser impresso
syscall # Efetua a chamada ao sistema

li $v0,1 # Comando de impressão de inteiro na tela
la $a0, ($s2) # Coloca o registrador $s2 para ser impresso
syscall # Efetua a chamada ao sistema

li $v0,4 # Comando de impressão de string na tela
la $a0, $or # Coloca o texto or para ser impresso
syscall # Efetua a chamada ao sistema

li $v0,1 # Comando de impressão de inteiro na tela
la $a0, ($s3) # Coloca o registrador $s3 para ser impresso
syscall # Efetua a chamada ao sistema

li $v0,4 # Comando de impressão de string na tela
la $a0, $xor # Coloca o texto xor para ser impresso
syscall # Efetua a chamada ao sistema

li $v0,1 # Comando de impressão de inteiro na tela
la $a0, ($s4) # Coloca o registrador $s4 para ser impresso
syscall # Efetua a chamada ao sistema

```

No item 03, foi pedido que realizasse uma operação “AND” entre \$t2 e 31. Isso é chamada de máscara. Também foi pedido que fosse mostrado o resultado dessas operações na tela:

```

# Questão 3

and $s5, $t2, 31 # AND entre os dois primeiros valores

li $v0,4 # Comando de impressão de string na tela
la $a0, $mask # Coloca o texto mask para ser impresso
syscall # Efetua a chamada ao sistema

li $v0,1 # Comando de impressão de inteiro na tela
la $a0, ($s5) # Coloca o registrador $s5 para ser impresso
syscall # Efetua a chamada ao sistema

```

Por fim, no item 04, foi pedido que houvesse um deslocamento para a direita e para a esquerda no valor de \$s5. Após isso o programa devia ser encerrado.


```
# Questão 4
```

```
sll $s6, $t0, $s5
```

```
li $v0, 4 # Comando de impressão de string na tela  
la $a0, $sll # Coloca o texto and para ser impresso  
syscall # Efetua a chamada ao sistema
```

```
li $v0, 1 # Comando de impressão de inteiro na tela  
la $a0, ($s5) # Coloca o registrador $s5 para ser impresso  
syscall # Efetua a chamada ao sistema
```

```
li $v0, 4 # Comando de impressão de string na tela  
la $a0, $msg # Coloca o texto msg para ser impresso  
syscall # Efetua a chamada ao sistema
```

```
li $v0, 1 # Comando de impressão de inteiro na tela  
la $a0, ($s6) # Coloca o registrador $s5 para ser impresso  
syscall # Efetua a chamada ao sistema
```

```
srl $s7, $t1, $s5
```

```
li $v0, 4 # Comando de impressão de string na tela  
la $a0, $srl # Coloca o texto and para ser impresso  
syscall # Efetua a chamada ao sistema
```

```
li $v0, 1 # Comando de impressão de inteiro na tela  
la $a0, ($s5) # Coloca o registrador $s5 para ser impresso  
syscall # Efetua a chamada ao sistema
```

```
li $v0, 4 # Comando de impressão de string na tela  
la $a0, $msg # Coloca o texto msg para ser impresso  
syscall # Efetua a chamada ao sistema
```

```
li $v0, 1 # Comando de impressão de inteiro na tela  
la $a0, ($s7) # Coloca o registrador $s5 para ser impresso  
syscall # Efetua a chamada ao sistema
```

```
li $v0, 4  
la $a0, $newline  
syscall
```

```
li $v0, 10 # Comando de exit  
syscall # Efetua a chamada ao sistema
```