Syscalls

Código	Chamada	Argumentos	Resultados
1	print integer	$a0 = integer to print	
2	print float	$f12 = float to print	
3	print double	$f12 = float to print	
4	print string	$a0 = address of beginning of string	
5	read integer		integer stored in $v0
6	read float		float stored in $f0
7	read double		double stored in $f0
8	read string	$a0 = pointer to buffer, $a1 = length of buffer	string stored in buffer
9	sbrk (allocate memory buffer)	$a0 = size needed	$v0 = address of buffer
10	exit		
11	print character	$a0 = character to print	

Registradores

Notação	Número	Descrição
$zero	0	Constante zero
$at	1	Reservado para o Assembler
$v0-$v1	2-3	Valores para resultados e avaliação de expressões
$a0-$a3	4-7	Argumentos
$t0-$t7	8-15	Temporários (não preservados entre chamadas)
$s0-$s7	16-23	Salvos (preservados entre chamadas)
$t8-$t9	24-25	Outros temporários
$k0-$k1	26-27	Reservado para o Kernel do O.S.
$gp	28	Global Pointer
$sp	29	Stack Pointer
$fp	30	Float Pointer
$ra	31	Return Address

Instruções

Possuem 3 formatos:

R - Todos os dados utilizados na instrução estão em registradores.

opcode	rs	rt	rd	shift(shamt)	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- Opcode: The opcode is the machinecode representation of the instruction mnemonic. Several related instructions can have the same opcode. The opcode field is 6 bits long (bit 26 to bit 31).
- rs, rt, rd: The numeric representations of the source registers and the destination register. These numbers correspond to the $X representation of a register, such as $0 or $31. Each of these fields is 5 bits long. (25 to 21, 20 to 16, and 15 to 11, respectively). Interestingly, rather than rs and rt being named r1 and r2 (for source register 1 and 2), the registers were named "rs" and "rt" for register source, register target and register destination.
- Shift (shamt): Used with the shift and rotate instructions, this is the amount by which the source operand rs is rotated/shifted. This field is 5 bits long (6 to 10).
- Funct: For instructions that share an opcode, the funct parameter contains the necessary control codes to differentiate the different instructions. 6 bits long (0 to 5).

I - Algum dado utilizado na instrução é imediato (máximo de 16 bits).

opcode	rs	rt	IMM
6 bits	5 bits	5 bits	16 bits

- Opcode: The 6-bit opcode of the instruction. In I instructions, all mnemonics have a one-to-one correspondence with the underlying opcodes. This is because there is no funct parameter to differentiate instructions with an identical opcode. 6 bits (26 to 31).
- rs, rt: The source and target register operands, respectively. 5 bits each (21 to 25 and 16 to 20, respectively).
- IMM: The 16 bit immediate value. 16 bits (0 to 15). This value is usually used as the offset value in various instructions, and depending on the instruction, may be expressed in two's complement.

J - Usadas para realizar algum jump.

opcode	pseudo-address
6 bits	26 bits

- Opcode: The 6 bit opcode corresponding to the particular jump command. (26 to 31).
- Address: A 26-bit shortened address of the destination. (0 to 25). The full 32-bit destination address is formed by concatenating the highest 4 bits of the PC (the address of the instruction following the jump), the 26-bit pseudo-address, and 2 zero bits (since instructions are always aligned on a 32-bit word).

Instruções	Minemônico	Formato	Comentário
Adição	add $t0, $t1, $t2	R	$t0 = $t1 + $t2
Subtração	sub $t0, $t1, $t2	I	$t0 = $t1 - $t2
Adição Imediate	addi $t0, $t1, 5	I	$t0 = $t1 + 5
Load Word	lw $t0, $t1	I	
Store Word	sw	I	
Load Half	lh	I	
Load Half Unsigned	lhu	I	
Store Half	sh	I	
Load Byte	lb	I	
Load Byte Unsigned	lbu	I	
Store Byte	sb	I	
Load Linked	ll	I	
Store Conditional	sc	I	
Load Upper Immediate	lui	I	
And	and $t0, $t1, $t2	R	Comparação lógica entre $t1 e $t2, onde $t0 armazena o resultado final. Ex: 1001 e 0011 = 0001
Or	or $t0, $t1, $t2	R	Comparação lógica entre $t1 ou $t2, onde $t0 armazena o resultado. Ex: 1001 ou 0010 = 1011
Nor	nor $t0, $t1, $t2	R	Comparação lógica entre $t1 not or $t2, onde $t0 armazena o resultado final. Ex: 1001 nor 0010 = 0100
And Immediate	andi $t0, $t1, 5	I	$t0 = 1001(valor hipotético) & 0101(5) = 0001
Or immediate	ori $t0, $t1, 5	I	$t0 = 1001(valor hipotético) | 0101(5) = 1101
Shift Left Logical	sll $t0, $t1, 2	R	Descarta o bit mais a direita e acrescenta 0 a esquerda. Ex: <-2--101101 = 001011 (01) <-Descartado
Shift Right Logical	srl $t0, $t1, 2	R	Descarta o bit mais a esquerda e acrescenta 0 a direita. Ex: 101101--2-> (10) 110100
Shift Left Arithmetic	sll $t0, $t1, 2	R	Descarta o bit mais a direita e extende o bit a esquerda. Ex: <-2--101101 = 111011 (01) <-Descartado
Shift Right Arithmetic	srl $t0, $t1, 2	R	Descarta o bit mais a esquerda e extende o bit a direita. Ex: 101101--2-> (10) 110111
Branch on Equal	beq	I	
Branch on Not Equal	bne	I	
Set ~1~ on Less Than	slt $t0, $t1, $t2	R	$t1 < $t2 ? $t0 = 1 : $t0 = 0
Set on Less Than Immediate	slti $t0, $t1, 5	I	$t1 < 5 ? $t0 = 1 : $t0 = 0
Set on Less Than Immediate Unsigned	sltiu	I	Considera o bit de sinal como parte do número.
Jump	j	J	
Jump to Register	jr	R	
Jump and Link	jal label	J	Iguala $ra como a próxima linha e da jump para a label

Pseudoinstruções

Instruções	Minemônico	Formato
Move	move	R
Multiplicação	mult	R
Multiplicação Immediate	multi	I
Load Immediate	li	I
Branch on Less Than	blt	I
Branch on Less or Equals than	ble	I
Branch on Greater Than	bgt	I
Branch on Greater or Equals than	bge	I

Traduzindo Pseudoinstruções

Move

`move $t0, $v0` pode ser reescrito como `add $t0, $zero, $v0`

Multiplicação

```
    addi    $a0,        $zero,          3                   # Inicializando número de vezes que será multiplicado
    addi    $a1,        $zero,          3                   # Inicializando número a ser multiplicado

    addi    $t0,        $zero,          0                   # Inicializando contador
    addi    $t1,        $zero,          0                   # Inicializando resultado


multiplica:
    beq     $t0,        $a0,            exit                # $t0 > $a0 ? exit : continue
    add     $t1,        $t1,            $a1                 # $t1 += $a1

    addi    $t0,        $t0,            1                   # Incrementando contador
    j       multiplica                                      # Jump back to loop
```

Load Immediate

`li $t0, 5` pode ser reescrita como `addi $t0, $zero, 5`

Branch on Less Than

```
slt     $t0,    $s0,            $s1     # $t0 = ($s0 < $s1) ? 1 : 0
bne     $t0,    $zero,          exit    # if $t0 != $zero then goto exit
```

Branch on Greater Than

```
slt     $t0,    $s1,            $s0     # $t0 = ($s1 < $s0) ? 1 : 0
bne     $t0,    $zero,          exit    # if $t0 != $zero then goto exit
```

Branch on Less or Equals Than

```
slt     $t0,    $s1,            $s0     # $t0 = ($s1 < $s0) ? 1 : 0
beq     $t0,    $zero,          exit    # if $t0 == $zero then goto exit
```

Branch on Greater or Equals than

```
slt     $t0,    $s0,            $s1     # $t0 = ($s0 < $s1) ? 1 : 0
beq     $t0,    $zero,          exit    # if $t0 == $zero then goto exit
```

Questões

Formativa 1

Problema A

Comando

Você deve Imprimir uma única linha contendo a frase:

```
    Olá Mundo
```

Resolução

```
.data
ola_mundo: .asciiz "Ola Mundo\n"

.text
main:
    li      $v0,    4           # system call #4 - print string
    la      $a0,    ola_mundo
    syscall                     # execute

    li      $v0,    10
    syscall
```

Problema B

Comando

Bem vindo ao segundo exercício! No exercício anterior trabalhamos apenas com a impressão de uma única linha agora vamos interagir com a máquina!!!

Todos os exercícios com correção automática possuem um processamento de uma entrada e o seu resultado é impresso em uma ou mais linhas.

Para este exercício você deve ler 2 números da entrada padrão (geralmente o teclado) e imprimir uma única linha contendo a soma destes 2 números.

Resolução

```
.data
quebra_linha: .asciiz "\n"

.text
```

```
main:
    li      $v0,    5                       # Read Int
    syscall
    move    $t1,    $v0                     # $t1 = First Int

    li      $v0,    5                       # Read Int
    syscall
    move    $t2,    $v0                     # $t2 = Second Int

    add     $t0,    $t1,            $t2     # $t0 = $t1 + $t2


    li      $v0,    1                       # system call #1 - print int
    move    $a0,    $t0
    syscall                                 # execute

    li      $v0,    4                       # system call #4 - print string
    la      $a0,    quebra_linha
    syscall                                 # execute

    li      $v0,    10
    syscall
```

Entrada 1	Entrada 2	Saída
100	200	300

Problema C

Comando

Escreva um programa que, dada a pressão desejada digitada pelo motorista e a pressão do pneu lida pela bomba, indica a diferença entre a pressão desejada e a pressão lida

Resolução

```
.data
quebra_linha: .asciiz "\n"

.text
main:

    li      $v0,    5                       # Read Int
    syscall
    move    $t1,    $v0                     # $t1 = First Int

    li      $v0,    5                       # Read Int
    syscall
    move    $t2,    $v0                     # $t1 = Second Int

    sub     $t0,    $t1,            $t2     # $t0 = $t1 - $t2


    li      $v0,    1                       # system call #1 - print int
    move    $a0,    $t0
    syscall                                 # execute

    li      $v0,    4                       # system call #4 - print string
    la      $a0,    quebra_linha
    syscall                                 # execute

    li      $v0,    10                      # exit
    syscall
```

Entrada 1	Entrada 2	Saída
36	26	10

Formativa 2

Problema A

Comando

Determinar o maior número digitado.

Resolução

```
# Mapeamento de variaveis
# $t0 = quantidade de numeros
# $t1 = numero lido
# $s1 = maior numero = resultado
.data
quebra_linha: .asciiz "\n"

.text
main:

# ler quantidade de numeros
    li      $v0,    5
```

```
    syscall

    move    $t0,    $v0                     # Quantidade de numeros

    addi    $v0,    $0,             5       # system call #5 - input int
    syscall                                 # execute

    move    $s1,    $v0                     # $t1 = $v0 = primeiro numero

loop:
    addi    $t0,    $t0,            -1      # $t0 = $t0 + -1
    beq     $t0,    $zero,          exit    # if $t0 == $zero then goto print

    addi    $v0,    $0,             5       # system call #5 - input int
    syscall                                 # execute

    bgt     $v0,    $s1,            maior
    j       loop

maior:                                      # guardar maior numero
    move    $s1,    $v0
    j       loop

exit:
    li      $v0,    1
    move    $a0,    $s1
    syscall

    li      $v0,    4
    la      $a0,    quebra_linha
    syscall
    li      $v0,    10
    syscall
```

Entrada 1	Entrada 2	Saída
3	1 2 3 4	4

Problema B

Comando

Calcular o preço da água baseado na faixa de preço. Todos pagam R$7.00 por padrão.

Faixa	Preço
até 10	inluso na franquia
11 a 30	R$ 1
31 a 100	R$ 2
101 em diante	R$ 5

Resolução

```
# s3 = fator de multiplicacao de preco
# s2 = caso base
# s1 = consumo declarado
# s0 = resultado

.data
quebra_linha: .asciiz "\n"

.text
main:

    addi    $v0,        $0,             5               # system call #5 - input int
    syscall                                             # execute


    move    $s1,        $v0                             # $s1 = $v0 = consumo declarado

# inicializando resultado e var aux
    move    $s0,        $zero                           # $s0 = $zero
    move    $t7,        $zero                           # $t7 = $zero
    addi    $s2,        $zero,          10              # $s2 =  10 = caso base

# caso base
    ble     $s1,        $s2,            casoBase        # if $s1 <= $s2 casoBase

# settando valores para as branches
    addi    $t0,        $zero,          10              # $t0 = $zero + 10
    addi    $t1,        $zero,          30              # $t1 = $zero + 30
    addi    $t2,        $zero,          100             # $t2 = $zero + 100


while:
    beq     $s1,        $s2,            casoBase        # if $t0 == $t1 then goto target

    jal     setValue                                    # jump to set_value and save position to $ra

    add     $s0,        $s0,            $s3             # calculando preco

    addi    $s1,        $s1,            -1              # $s1 = $s1 - 1
```

```
    j       while                                   # loop back

faixa11a30:

    addi    $s3,        $zero,          1           # $t0 = $zero + 1
    jr      $ra                                     # jump to $ra


faixa31a100:

    addi    $s3,        $zero,          2           # $t0 = $zero + 2
    jr      $ra                                     # jump to $ra

faixa101:

    addi    $s3,        $zero,          5           # $t0 = $zero + 5
    jr      $ra                                     # jump to $ra

setValue:

    ble     $s1,        $t1,            faixa11a30      # if consumo declarado <= 30 then goto faixa11a30
    ble     $s1,        $t2,            faixa31a100     # if consumo declarado <= 100 then goto faixa31a100
    bgt     $s1,        $t2,            faixa101        # if $s1 > $t2 then goto faixa101

casoBase:

    addi    $s0,        $s0,            7           # $s0 = $s0 + 7

exit:

    addi    $v0,        $0,             1           # system call #1 - print int
    add     $a0,        $0,             $s0
    syscall                                         # execute

    li      $v0,        4                           # system call #4 - print string
    la      $a0,        quebra_linha
    syscall                                         # execute

    li      $v0,        10                          # exit
    syscall
```

Entrada	Saída
42	51

Problema C

Comando

Imprimir duas pirâmides conforme saída.

Resolução

```
# $t2 = coontador2
# $t1 = coontador
# $s0 = numero de linhas
# # Tenho certeza que tinha uma forma mais otimizada de ser feito, mas ano novo entao preguica
.data
quebra_linha: .asciiz "\n"
espaco: .asciiz " "

.text
main:

    addi    $v0,                        $0,             5                           # system call #5 - input int
    syscall                                                                         # execute

    move    $s0,                        $v0                                         # $s0 = $v0 = numero de linhas

# first piramid
    move    $t1,                        $zero                                       # $t1 = 0 = contador
whileLinha:
    move    $t2,                        $zero                                       # $t2 = $zero

    addi    $t1,                        $t1,            1                           # $t1 = $t1 + 1 -> Adicionando cont
    bgt     $t1,                        $s0,            secondPiramid               # if contador > numero de linhas th

whileColuna:

    beq     $t2,                        $t1,            endWhileColuna              # se o contador 2 for igual a conta

    move    $a1,                        $t1                                         # $a1 = $t1
    jal     zeroing                                                                 # jump to zeroing and save position

    addi    $v0,                        $0,             1                           # system call #1 - print int
    add     $a0,                        $0,             $t1
    syscall                                                                         # execute

    addi    $t2,                        $t2,            1                           # $t2 = $t2 + 1

    addi    $v0,                        $0,             4                           # system call #4 - print string
    la      $a0,                        espaco
    syscall                                                                         # execute
```

```
    j       whileColuna

endWhileColuna:
    li      $v0,                    4                                       # system call #4 - print string
    la      $a0,                    quebra_linha
    syscall                                                                 # execute

    j       whileLinha                                                      # loop back


# second piramid
secondPiramid:

    li      $v0,                    4                                       # system call #4 - print string
    la      $a0,                    quebra_linha
    syscall                                                                 # execute


    move    $t1,                    $zero                                   # $t1 = 0 = contador
whileSecondPiramidLinha:
    move    $t2,                    $zero                                   # $t2 = $zero

    addi    $t1,                    $t1,            1                       # $t1 = $t1 + 1 -> Adicionando cont
    bgt     $t1,                    $s0,            exit                    # if contador > numero de linhas th

whileSecondPiramidColuna:

    addi    $t2,                    $t2,            1                       # $t2 = $t2 + 1

    bgt     $t2,                    $t1,            endWhileSecondPiramidColuna     # se o contador 2 for maior que con

    move    $a1,                    $t2                                     # $a1 = $t2
    jal     zeroing                                                         # jump to zeroing and save position

    addi    $v0,                    $0,             1                       # system call #1 - print int
    add     $a0,                    $0,             $t2
    syscall                                                                 # execute



    addi    $v0,                    $0,             4                       # system call #4 - print string
    la      $a0,                    espaco
    syscall                                                                 # execute

    j       whileSecondPiramidColuna

endWhileSecondPiramidColuna:

    li      $v0,                    4                                       # system call #4 - print string
    la      $a0,                    quebra_linha
    syscall                                                                 # execute

    j       whileSecondPiramidLinha                                         # loop back


zeroing:
    addi    $t0,                    $zero,          10                      # $t0 = $zero + 10

    bge     $a1,                    $t0,            return                  # if a1 >= 10 then goto target
    addi    $v0,                    $0,             1                       # system call #1 - print int
    add     $a0,                    $0,             $zero
    syscall                                                                 # execute
return:
    jr      $ra                                                             # jump to $ra


exit:

    li      $v0,                    4                                       # system call #4 - print string
    la      $a0,                    quebra_linha
    syscall                                                                 # execute

    li      $v0,                    10                                      # exit
    syscall
```

Entrada:

```
1
```

Saída:

```
01
02 02
03 03 03
04 04 04 04
05 05 05 05 05



01
01 02
01 02 03
01 02 03 04
01 02 03 04 05
```

Problema D

Comando

Validar gabarito.

Resolução

```
# s0 = quantidade de questoes
# s1 = Gabarito
# s2 = Marcadas
# s3 = Acertos = Resultado

.data
quebra_linha: .asciiz "\n"
gabarito: .space 1024
marcadas: .space 1024

.text
main:

    addi    $v0,    $0,             5       # system call #5 - input int
    syscall                                 # execute

    move    $s0,    $v0                     # $s0 = $v0

    addi    $v0,    $0,             8       # system call #8 - input string
    la      $a0,    gabarito
    li      $a1,    1024
    syscall                                 # execute

    la      $s1,    gabarito


    addi    $v0,    $0,             8       # system call #8 - input string
    la      $a0,    marcadas
    li      $a1,    1024
    syscall                                 # execute

    la      $s2,    marcadas

init:
    move    $t0,    $zero                   # $t0 = 0 = contador
    move    $s3,    $zero                   # $s3 = 0 = resultado

check:

    beq     $t0,    $s0,            exit    # if $t0 == $s0 = maximo then goto exit


    lb      $t4,    0($s1)
    lb      $t5,    0($s2)

    jal     valid

    addi    $t0,    $t0,            1       # $t0 = $t0 + 1
    addi    $s1,    $s1,            1       # $s1 = $s1 + 1 -> Incrementando um byte para locomover o caracter
    addi    $s2,    $s2,            1       # $s2 = $t0 + 1 -> Incrementando um byte para locomover o caracter
    j       check

valid:
    bne     $t4,    $t5,            return  # erro
    addi    $s3,    $s3,            1       # $s3 = $s3 + 1

return:
    jr      $ra                             # jump to $ra


exit:

    addi    $v0,    $0,             1       # system call #1 - print int
    add     $a0,    $0,             $s3
    syscall                                 # execute

    li      $v0,    4                       # system call #4 - print string
    la      $a0,    quebra_linha
    syscall                                 # execute

    li      $v0,    10                      # exit
    syscall
```

Entrada 1	Entrada 2	Entrada 3	Saída
5	ABCAA	AACCA	3

Problema E

Comando

Um binário de 7 bits (2^7-1 = 127) deve possuir uma quantidade par de 1. Caso ele já tenha uma quantidade par, o oitavo bit deve ser 0. Caso ele seja ímpar, deve-se adicionar 1 na oitava casa (+128).

Resolução

```
.data
quebra_linha: .asciiz "\n"

.text
main:

    addi    $v0,            $0,             5           # system call #5 - input int
    syscall                                             # execute

    add     $a0,            $zero,          $v0         # $a0 = $zero + $v0

    jal     bitparidade                                 # jump to bitparidade and save position to $ra
    j       exit                                        # jump to exit

bitparidade:
    addi    $t2,            $zero,          7           # $t2 = $t2 + 7
    addi    $t1,            $zero,          0           # $t1 = $zero + 0
    addi    $t4,            $zero,          2           # $t4 = $zero + 2
    addi    $v0,            $zero,          0           # $v0 = $zero + 0
    add     $v1,            $zero,          $a0         # $v1 = $zero + $a0


loop:

    addi    $t2,            $t2,            -1          # $t2 = $t2 - 1

    blt     $t2,            $zero,          paridade    # if $t2 == $zero then goto return

    andi    $t0,            $a0,            1           # $t0 = $t1 & 1

    srl     $a0,            $a0,            1           # $a0 = $a0 << 1

    bne     $t0,            $zero,          count       # if $t0 != $zero then goto target

    j       loop

count:
    addi    $t1,            $t1,            1           # $t1 = $t1 + 1

    j       loop

paridade:

    div     $t1,            $t4                         # $t1 / 2
    mfhi    $t3                                         # $t3 = $t1 % 2

    beq     $t3,            $zero,          return      # if $t3 == $zero then goto return

    addi    $v1,            $v1,            128         # $v1 = $v1 + 128
    addi    $v0,            $zero,          1           # $v0 = $zero + 1

return:
    jr      $ra

exit:

    move    $t0,            $v0                         # $t0 = $v0
    move    $t1,            $v1                         # $t1 = $v1

    addi    $v0,            $0,             1           # system call #1 - print int
    add     $a0,            $0,             $t0
    syscall                                             # execute

    li      $v0,            4                           # system call #4 - print string
    la      $a0,            quebra_linha
    syscall                                             # execute

    addi    $v0,            $0,             1           # system call #1 - print int
    add     $a0,            $0,             $t1
    syscall                                             # execute

    li      $v0,            4                           # system call #4 - print string
    la      $a0,            quebra_linha
    syscall                                             # execute

    li      $v0,            10                          # exit
    syscall
```

Entrada 1	Saída 1	Saída 2
127	1	255

Questões Teóricas

O código fonte, em alto nível, passa por dois processos de transformação para viabilizar sua execução em um processador digitar moderno. Que nome se da aos processos de transformção de um código de alto nível para assembly e de assembly para código de máquina, respectivamente?

- Compilação e Montagem

Qual o comprimento de uma palavra (word) na arquiterura MIPS?

- 32 bits

Um processador digital moderno enquadra-se no conceito de sistema computacional universal, cabendo ao desenvolvedor apresentar uma listagem de instruções (software) compatíveis com a ISA.

- Verdadeiro

É possível sempre reverter um processo de compilação?

- Não

É possível sempre reverter um processo de montagem?

- Sim

Qual é a ferramenta utilizada para converter um código assembly em código de máquina?

- Assembler / Montador

Defina ISA

- Interface entre o processador digital (silício) e o software básico, que expõe os serviços básicos providos pelo processador

Uma máquina de estados finitos é uma solução para o tipo problema e que depende de um projeto eletrônico específico

- Verdadeiro

Para operações aritméticas envolvendo constantes pequenas, o valor da constante (imediato) poderá ser codificado na própria instrução. Isso traz ganho de desempenho, pois operações aritméticas com constantes são muito frequentes.

- Verdadeiro

Quais são as 2 áreas de memória (segmentos) básicas que estruturam um programa em MIPS?

- .text e .data

Como se chama a região da memória de dados dinâmicos que foram alocados na memória principal por um processador moderno?

- heap

Os operandos de instruções aritméticas podem ser posições na memória de linguagem de montagem MIPS?

- Falso

Quais são os princípios de projeto utilizados na concepção da arquitetura MIPS?

- simplicidade favorece regularidade
- menor significa mais rápido
- agilize os casos mais comuns

Suponha que temos em memória um array de inteiros A de 50 posições e que o endereço inicial deste vetor está no registrados $s0. Qual instrução devo utilizar para carregar no registrador $t0 o elemento A[24]?

- lw $t0, 96($s0)
- (offset = 96 pois cada inteiro tem 4 bits, então para chegar no início da vigésima quarta posição multiplica-se 24 por 4)

Linguagem de máquina é mais primitiva que linguagens de alto nível?

- Verdadeiro

Linguagem de montagem e linguagem de máquina não são a mesma coisa: a primeira é composta por instruções de uma ISA, enquanto a segunda é o código binário executado pelo processador.

- Verdadeiro

Linguagem C é uma linguagem de baixo nível

- Falso

Como multiplicar o valor armazenado em $t0 por 8 utilizando apenas operações lógicas? Assuma que o resultado deverá ser armazenado em $t1.

- sll $t1, $t0, 3

Qual é o tamanho máximo do shift representável numa instrução sll/srl?

- 31, pois são do tipo R

Para uma instrução do tipo I, quantos bits estão disponíveis para a sinalização/informação de um imediato?

- 16

O que acontece depois da execução do seguinte fragmento de código:

```
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp 8
```

- São lidos da memória valores para os registrados $ra e $a0

O que acontece depois da execução do seguinte fragmento de código:

```
minha_funcao:
    addi $sp, $sp, -8
    sw $ra, 4($sp)
    sw $a0, 0($s0)
```

- Os valores de $ra e $a0 são salvos nas posições de memória apontadas para $sp+4 e $sp, respectivamente