S1.

a) 10 bytes.

b) 3 bytes

c) 12 bytes

d) 5 bytes.

S2.

0x 1001 0000	- A	L1
0x 1001 0001	U	
0x 1001 0002	L	
0x 1001 0003	A	
0x 1001 0004	S	
0x 1001 0005	5	
0x 1001 0006	6	
0x 1001 0007	T	
0x 1001 0008	10	

0x1001 0008 5 L2

0x1001 0009 8

0x1001 000A 23

0x1001 000B

0x1001 000C L3[0] L3

0x1001 000D L3[0]

0x1001 000F L3[0]

0x1001 000F L3[0]

0x1001 0010 L3[1]

0x1001 0011 L3[1]

0x1001 0012 L3[1]

0x1001 0013 L3[1]

0x1001 0014 L3[2]

0x1001 0015 L3[2]

0x1001 0016 L3[2]

0x1001 0017 L3[2]

0x1001 0018 • L4

0x1001 0019 •

0x1001 001A •

0x1001 001B •

0x1001 001C •

53.

L2: 0x1001 0006 L3: 0x1001 000C

L4: 0x1001 0018

54.

a) &b[0] ou b

b) lw $t0, 0x24[$a0] # $a0 = b

55. Codificado em complemento para 2, como um nº inteiro de 2 Bytes.

(Label - (PC + 4))/4 = Imm

Program Counter

56.

BTA = PC + offset

57.

"beq/bne" - I

"j", "jr" - J

58. J Target

Este valor tem 26 bits para especificar a posição para qual o programa deve pular.

Target/4 = endereço - alvo

só precisamos dos 4 bits mais significativos?

59 - R: A e D.

60. $t0 = f $t1 = g $t2 = h $t3 = i $t4 = j
$s2 = B

a) f = g + h + B[2]:

```
lw $t5, 8($s1)
add $t5, $t5, $t2
add $t0, $t5, $t1
```

j = g - A[B[2]]:

```
lw $t5, 8($s1)   # $t5 = B[2]
sll $t5, $t5, 2
```

addu $t6, $s0, $t5

lw $t5, 0($t6)

sub $t4, $t1, $t5

b)

1st: 3 linhas e 5 registos.

2nd: 5 linhas e 6 registos.

c) A[0] = 0x00000012

A[1] = 0x22ED3400

A[2] = 0x00000001

B[0] = 0xFFFFFFFE

B[1] = 0x00005002

B[2] = 0x00000002

d) g = -3 h = 2

f = -3 + 2 + d j = -3 - A[d]

IDK

61.

```
void troca(int *x, int *y)
{
        int aux;
        aux = *x;
       *x = *y;
       *y = aux;
}
```

62. $ra/4 = endereco - alvo.

63. 0x5A18F34C

$rd / 4 = ÷ 4

0x005A18F3

0000 0000 0101 1010 0001 1000 1111 0011

Menor endereço é 0x000000 = 0

Maior endereço é 2^{26}.

64.

Menor endereço é 0x000000 = 0

Maior endereço é 2^{16}.

65.

Menor endereço é 0x000000 = 0

Maior endereço é 2^{26}.

66. $[-2^{16-1}, 2^{16-1}+1] = [-32,768; 32,767]$

67. $[-32,768; 32,767]$

68. lui e ori são instruções já existentes com capacidade de carregar uma constante de

de 32 bits.

69. Com Usi e lui.

71. Uma sub-rotina é uma função que pode ser chamada usando "jal" para processar certos dados e obter um resultado em $vo e/ou $f0.

72. "jal"

73. A instrução "j" é utilizada para jumps para o endereço target. O "jal" muda o endereço de $ra para o endereço onde o jal está, desta forma quando a sub-rotina acabar a rotina "main" não terá Problema a continuar.

74. Endereço de retorno (PC+4) é armazenado em $ra.

Da jump para o Target.

A sub-rotina é executada.

No final da sub-rotina o comando "jr $ra" é executado para retornar.

75. $ra e o nº é $31

76. Guardar o endereço de retorno "$ra". Armazenas outras possíveis estados importantes. Executar a sub-rotina. Restaurar os estados. E utilizar "jr $ra" para retornar ao endereço de retorno.

77. "jr $ra"

78. Jump.

79. Usar stack para

7 (. Uma stack serve para guardar valores. O stack pointer serve para guardar valores entre sub-rotinas.

8 0. A instrução pop serve para remover elementos da stack, enquanto a push adiciona elementos à stack.

8 1. A codificação é little Endian, ou seja, vai do mais baixo para o mais alto.

8 2.

1 - O registo $sp contém o endereço da última posição ocupada da stack.

2 - A stack cresce no sentido decrescente dos endereços da memória.

3 - A stack está organizada em words de 32 bits.

83. $sp

84.

a) Para passar: "$a0" até "$a3"
"$f12" até "$f14"
Para devolver: "$v0" e "$f0"

b) Pode alterar os registos "$sn", "$fn", "$t" e "$an".

c) "$sn"

d) Sempre que é chamada uma sub-rotina e queremos usar os valores dos "$sn".

e) Os valores "tn" são usados para guardar valores irrelevantes ou temporários.

Os "an" para guardar/receber valores importantes e, se necessário, syscalls.

Os "vn" para syscall e guardar o valor que sai da sub-rotina.

85.

a) Os registos importantes para as operações da sub-rotina intermédia.

b) Não é necessário copiar nenhum valor, mas é boa prática copiar o $fra.

86.

$a0, $a1, $a2, $a3

87. $[-2^{n-1}, 2^{n-1}-1]$

3 bits: $[-2^2, 2^2-1] = [-4, +3]$

4 bits: $[-2^3, 2^3-1] = [-8, +7]$

5 bits: $[-2^4, 2^4-1] = [-16, +15]$

8 bits: $[-2^7, 2^7-1] = [-128, +127]$

16 bits: $[-2^{15}, 2^{15}-1] = [-32768, +32767]$

89.

a) $5_{10} = 0101_2 =$

$= 0000\ 0000\ 0000\ 0101$

b) $-3_{10} = 1101_{C_2}$

1111 1111 1111 1101
3 2 1 0

$-2^3 + 2^2 + 2^1 \times 0 + 2^0 \times 1 = -8 + 4 + 1 =$

$= -8 + 5 = -3$

c) $-128_{10} = 10000000_2$ (2)

1111 1111 1000 0000

90.

a) $0b\ 00101011$ (5 4 3 2 1 0)

$= 2^5 + 2^3 + 2^1 + 1 = 43_{10}$ (32 8 2 1)

b) $0xA5_{16} = 0b\ 10100101$ (7 6 5 4 3 2 1 0)

$= 2^7 + 2^5 + 2^2 + 1 =$

$= 128 + 32 + 4 + 1 =$

$= 165_{10}$

91

a) $43_{10} = 00101011 = 0x2B_{16}$

=

89 a)

$1111\ 1111\ 1111\ 1101 = 0xFFFD_{16}$

92 e 93)

Com sinal depende do carry out e carry in. Sem sinal depende só do carry out.

94.

a)

I: 0x7FFFFFFF

II: 0x80000000

b) São esferadores.

c)

I: 0x70000000 - 0x0FFFFFFF - 0x60000000

II: 0x00000000

d) São esperados

e) 0x7FFF FFFF

0x8FFFFFFE

f) São esperados

95. 64 bits.

97. 0x4FFFFFFF6

98.

div: 0x55555550

$\frac{15}{3}$

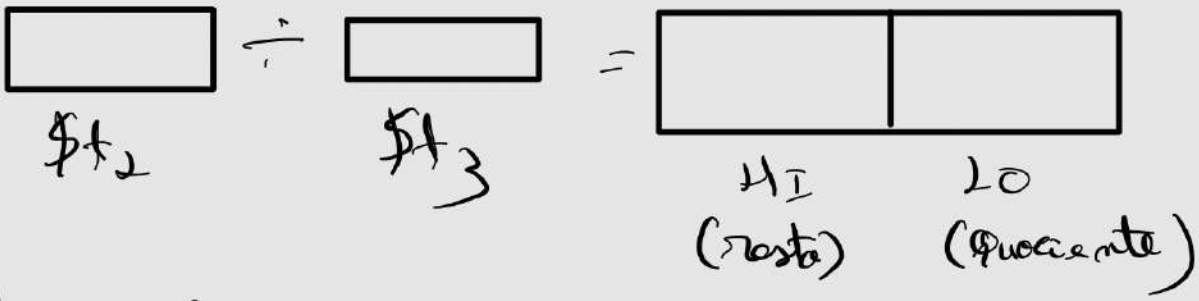
res: 0x00000000

99.

```
div $t2, $t3
mflo $t0
```
} div

```
div $t2, $t3
mfhi $t0
```
} res

100.



divide-se em módulo.

O quociente é negativo se os sinais

dos registos forem diferentes.

E o resto tem o mesmo sinal

que t_2.