45. SW → a palavra armazenada na memória é de 32 bits (4 bytes)

SB → a palavra armazenada na memória é de 8 bits (1 byte)

46. lb → carrega num registo uma palavra de 8 bits; os restantes 24 bits (de modo a "fazer" 32) têm o valor igual ao do MSB dos 8 bits, i.e, é colocado em prática a extensão com sinal.

lbu → é armazenada uma palavra de 8 bits em que os restantes 24 são colocados a '0'; ignora o sinal do MSB dos 8 bits.

47. MIPS gera uma exceção, terminando aí a execução do programa.

48. C para assembly

a) int i, k; i:$2 k:$3

for(i=5, k=0; i<20; i++, k*=5);

```
    li $2,5
    li $3,0
while: bge $2,20, endw
    (...)
    addi $2,$2,1
    addi $3,$3,5
    j while
endw: (...)
```

b) int i=100, k=0; i:$2 k:$3

for(; i>=0;)
{
  i--;
  k-=2;
}

```
    li $2,100
    li $3,0
while: blt $2,0, endw
    (...)
    sub $2,$2,1
    sub $3,$3,2
    j while
endw: (...)
```

c) unsigned int K=0;
```
for(;;)
{
K+=10;
}
```
K: $3

```
li: $3, 0
while: addi $3, $3, 10
j while
```

d) int K=0, i=100;
```
do{
  K+=5;
}while(--i>=0);
```
i: $2
K: $3

```
li: $3, 0
li: $2, 100
do: addi $3, $3, 5
sub $2, $2, 1
bgez $2, do
(...)
```

49. opcode LW → 0x23 <=> $00\underbrace{10\,0011}_{6\text{ bits}}_2$

lw $3, 0x24($5)

$\underbrace{100011}_{}\;\underbrace{00101}_{}\;\underbrace{00011}_{}\;\underbrace{0000\,0000\,0010\,0100}_{} = 0x8CA30024$

50. lw $3, 0x24($5) $5: 0x10010000

0x10010024

$3 = 0x28272625

little-endian

0x10010000	0x01
..... 1	0x02
..... 2	0x03
⋮	⋮
24	0x25
25	0x26
26	0x27
27	0x28

51. a) lbu $3, 0xA3($5)

0x1001 00A3 : 0xA4
A4 : 0xA5
A5 : 0xA6
A6 : 0xA7

$3 = 0x000000A4

little-endian
24 MSB = 0

b) lb $4, 0xA3($5)

0x1001 00A3 : 0xA4 <=> $1010\,0100_2$

$3 = 0xFFFFFFA4

52. a) 10 bytes b) 3 bytes c) 12 bytes d) .space 5
5 bytes

53.

L1:	0x10010000	'A'
	0x10010001	'n'
	0x10010002	'l'
	0x10010003	'a'
	0x10010004	'l'
	0x10010005	'i'
	0x10010006	'z'
	0x10010007	'a'
	0x10010008	'r'
	0x10010009	'\0'
L2:	0x1001000A	0x05
	0x1001000B	0x08
	0x1001000C	0x17
	0x1001000D	nem used
	0x1001000E	nem used
	0x1001000F	nem used
L3:	0x10010010	0x05
	0x10010011	0x00
	0x10010012	0x00
	0x10010013	0x00
	0x10010014	0x08
	0x10010015	0x00
	0x10010016	0x00
	0x10010017	0x00
	0x10010018	0x17
	0x10010019	0x00
	0x1001001A	0x00
	0x1001001B	0x00
L4:	0x1001001C	??
	0x1001001D	??
	0x1001001E	??
	0x1001001F	??
	0x10010020	??

L1: .ascii "Anlos 5&6 T"
L2: .byte 5, 8, 23
L3: .word 5, 8, 23
L4: .space 5

54. L1: 0x10010000
L2: 0x1001000A
L3: 0x10010010
L4: 0x1001001C

55. int S [25]

a) É obtido mone próprio do array: ex: la $t0, S

Deste modo o registo $t0 contém o endereço do 1º elemento.

b) &S[6] = endereço inicial do array + índice × dimensão em bytes de cada posição do array

6 × 4

56. É o nº de endereços que, a partir da instrução imediatamente a seguir ao branch, se têm de saltar para chegar ao endereço alvo. Pode ser um endereço menor ao do branch (offset negativo) ou maior (offset positivo)

57. BTA = Pc-atual + (offset << 2)
(PC+4)

A constante de 16 bits do código máquina é estendida para 32, de seguida é multiplicada por 4. Será somada com o PC+4.

58. beq/bne → tipo I
j → tipo J
jr → tipo R

59. Os 26 LSB da inst. são multiplicados 4 (<<2) obtendo-se assim 28 bits
Os 4 MSB do PC são concatenados com os restantes 28 bits.
Pc+4 (já atualizado)

60. a = S[5] a = *(f+5)

61. f: $t0 g: $t1 h: $t2 i: $t3 j: $t4 A: $s0 B: $s1
Sendo A, B arrays de ints.

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

lw $t0, 8($s1) # $t0 = B[2]
add $t0, $t2, t0 # $t0 = h + B[2]
add $t0, $t1, $t0 # $t0 = g + h + B[2]

61. (continuação)

a) j = g - A[B[2]]

```
lw $t4, B($s1)      # $t4 = B[2]
sll $t4, $t4, 2     # $t4 = [$t4] * 4
add $t4, $s3, $t4   # $t4 = &A[B[2]]
lw $t4, 0($t4)      # $t4 = A[B[2]]
sub $t4, $t1, $t4   # $t4 = g - A[B[2]]
```

5) Para 1ª inst. → 3

Para 2ª inst. → 5

0 registos auxiliares

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 = g + h + B[2] = -3 + 2 + 2 = 1

j = g - A[B[2]] = -3 - A[2] = -3 - 1 = -4

62.

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

63. jr $ra - tipo R

O endereço-alvo está armazenado no registo $ra.

64. Menor endereço:

j PC_atual = 0x5A18F350 4 MSB [PC] = 0101_2

28 LSB do endereço:

0x0000000

0x50000000

Maior endereço:

0x5FFFFFFC

28 LSB do endereço

0xFFFFFFC

65. beq Endereço: 0x5A18F34C

Menor endereço: menor offset possível → offset = 0x8000 → EXT32 → 0xFFFF8000 → <<2 → 0xFFFE0000

PC+4 = 0x5A18F350

BTA = 0x5A18F350 + 0xFFFE0000

= 0x5A16F350

Maior endereço: maior offset possível → offset = 0x7FFF → EXT32 → 0x00007FFF → <<2 → 0x0001FFFC

BTA = 0x5A18F350 + 0x0001FFFC

= 0x5A1AF34C

66. jr Endereço: 0x5A18F34C

Menor endereço: 0x00000000

Maior endereço: 0xFFFFFFFC

67. gama de representação da constante em inst. imediatas.

$$\left[-2^{16-1} ; 2^{16-1}-1\right] = \left[-32768 ; 32767\right]$$ aritméticas

68. nas inst. lógicas imediatas

$$\left[0 ; 2^{16}-1\right] = \left[0 ; 65535\right]$$

69. Como as próprias instruções do MIPS já têm 32 bits para a sua codificação, não existe uma inst. que manipule constantes de 32 bits.

70. Os 16 MSB da constante serão carregados num registo com a instrução LUI (load upper immediate), de seguida, noutro registo, será "concatenado" o conteúdo deste 1º registo com os 16 LSB da constante. É uma concatenação virtual pois a instrução a usar é a OR e será feita entre um valor diferente de 0 e igual a 0.

Ex: Constante = 0x12345678

```
lui $t1, 0x1234
ori $t2, $t1, 0x5678
```

```
$t1: 0x12340000
or  0x00005678
$t2: 0x12345678
```

71. Decompor em nativos

a) li $6, 0x8B47BE0F

```
lui $1, 0x8B47
ori $6, $1, 0xBE0F
```

b) xori $3, $4, 0x12345678

```
lui $1, 0x1234
ori $2, $1, 0x5678   # $2 = 0x12345678
xor $3, $4, $2
```

c) addi $5, $2, 0xF345AB17

```
lui $1, 0xF345
ori $3, $1, 0xAB17
add $5, $2, $3
```

d) beq $7, 100, L1

```
ori $1, $0, 100
beq $7, $1, L1
```

e) blt $3, 0x123456, L2

```
lui $1, 0x0012
ori $1, $1, 0x3456
slt $2, $1, $3
bne $1, $0, L2
```

72. Sequência de instruções de um programa que executa uma tarefa.

73. JAL (jump-and-link)

74. A instrução "j" não liga o chamador e a sub-rotina. O conteúdo do PC na instrução JAL é armazenado no registo $ra; deste modo, quando termina a sub-rotina chamada, o PC é atualizado com o valor guardado em $ra.

75. JAL guarda o valor do PC atual em $ra; PC é atualizado com o endereço-alvo da inst JAL (onde está armazenada a 1ª instrução da sub-rotina). No fim da sub-rotina, ao ser executado, o PC é atualizado com o valor guardado em $ra.

76. $ra (return address) $31

77. Salvaguardar valores de certos registos: o de retorno ($ra) e registos que serão precisos depois da chamada à função.

78. Retornar sub-rotina: jr (jump register)

79. É feito o salto para o endereço de retorno.

80. Espaço de armazenamento temporário (filha) que serve para guardar informação importante enquanto decorrem outras operações.

Stack pointer é o ponteiro da stack que aponta sempre para o topo da stack (TOS), i.e, o registo $sp contém o endereço da TOS (último endereço ocupado)

81. push → operação que permite acrescentar informação à stack.

Para fazer push é necessário pré-atualizar o stack-pointer antes de copiar informação para a stack.

Pop → operação que permite retirar informação da stack.

Para fazer pop é necessário 1º fazer a leitura da informação da stack do stack-pointer atual e depois este é atualizado.

82. Esta implementação permite-nos alocar muito mais informação; o crescimento da stack no sentido dos endereços mais baixos disponibiliza-nos todo o espaço disponível; como não sabemos o espaço que precisaremos no começo do programa, temos assim uma "boa margem" para salvaguardar informação. Permite uma gestão simplificada na fronteira entre os segmentos de dados e de stack.

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

A stack cresce no sentido decrescente dos endereços de memória

A stack está organizada em words de 32 bits

84. Registo do stack pointer $29

85. a) Passar parâmetros:
$a0, $a1, $a2 e $a3
Para floats | doubles
$f12, $f14

Devolver resultados:
$v0 e $v1
32 LSB 32 MSB
Para floats | doubles
$f0

b) $t0 — $t9 ; $v0, $v1 ; $a0 — $a3

c) $s0 — $s7

d) $s0 — $s7

e) Numa sub-rotina chamadora; Numa sub-rotina chamada não irá alterar os valores pois já foram salvaguardados.

f) Numa sub-rotina chamada pois o conteúdo destes registos já foi pré-salvaguardados

86. a) $ra e os $sn que sejam utilizados na sub-rotina

b) Nenhum

87. char fun(int a; unsigned char b; char *c; int *d);
$a0 $a1 $a2 $a3
valor devolvido $v0

88. 3 bits
[100; 011] bin
[0x4; 0x3] hex
[-4; 3] decimal

4 bits
[1000; 0111] bin
[0x8; 0x7] hex
[-8; 7] decimal

5 bits
[10000; 01111] bin
[0x10; 0x0F] hex
[-16; 15] decimal

8 bits
[10000000; 01111111] bin
[0x80; 0x7F] hex
[-128; 127] decimal

16 bits

[1000000000000000; 0111111111111111] bin

[0x8000; 0x7FFF] hex

[-32768; 32767] decimal

89. (Feito no computador)

ZZZZZZ

90.

5 → 0000 0000 0000 0101

-3 → 1111 1111 1111 1101

-128 → 1111 1111 1000 0000

-32768 → 1000 0000 0000 0000

31 → 0000 0000 0001 1111

-8 → 1111 1111 1111 1000

256 → 0000 0001 0000 0000

-32 → 1111 1111 1110 0000