Tópicos:

- Introdução Sistemas de Computação de Uso Geral
- A arquitetura MIPS

Questões:

1. Quais são os 3 blocos fundamentais de um sistema computacional?
2. Quais são os 3 principais blocos funcionais que integram um CPU?
3. Qual a função do registo *Program Counter*?
4. Quais os passos mais importantes em que se decompõe a execução de uma instrução no CPU?
5. Descreva de forma sucinta a função de um compilador.
6. Descreva de forma sucinta a função de um assembler.
7. Quantos registos internos de uso geral tem o MIPS?
8. Qual a dimensão, em bits, que cada um dos registos internos do MIPS pode armazenar?
9. Qual a sintaxe, em *Assembly*, de uma instrução aritmética no MIPS?
10. O que distingue a instrução SRL da instrução SRA do MIPS?
11. Se **`$5=0x81354AB3`**, qual o resultado, expresso em hexadecimal, das instruções:
    a. **`srl $3,$5,1`**
    a. **`sra $4,$5,1`**
12. *System calls*:
    a. O que é uma *system call*?
    b. No MIPS, qual o registo usado para identificar a *system call* a executar?
    c. Qual o registo ou registos usados para passar argumentos para as *systems calls*?
    d. Qual o registo usado para obter o resultado devolvido por uma *system call* nos casos em que isso se aplica?
13. Em Arquitetura de Computadores, como definiria o conceito de endereço?
14. O que é o espaço de endereçamento de um processador?
15. Como se organiza internamente um processador? Quais são os blocos fundamentais da secção de dados? Para que serve a unidade de controlo?
16. Qual é o conceito fundamental por detrás do modelo de arquitetura "*stored-program*"?
17. Como se codifica uma instrução? Que informação fundamental deverá ter o código de uma instrução?
18. Descreva pelas suas próprias palavras o conceito de **ISA**.

19. Quantas e quais são as classes de instruções que agrupam as diferentes instruções de uma dada arquitetura?

20. O que carateriza e distingue as arquiteturas do tipo "*register-memory*" e "*load-store*"? De que tipo é a arquitetura MIPS?

21. O ciclo de execução de uma instrução é composto por uma sequência ordenada de operações. Quantas e quais são essas operações (passos de execução)?

22. Como se designa o barramento que permite identificar, na memória, a origem/destino da informação transferida?

23. Qual a finalidade do barramento normalmente designado por *Data Bus*?

24. Os processadores da arquitetura hipotética ZWYZ possuem 4 registos internos e todas as instruções são codificadas em 24 bits. Num dos formatos de codificação existem 5 campos: um *OpCode* com 5 bits, três campos para identificar registos internos em operações aritméticas e lógicas e um campo para codificar valores constantes imediatos em complemento para dois. Qual a gama de representação destas constantes?

25. A arquitetura hipotética ZPTZ tem um barramento de endereços de 32 bits e um barramento de dados de 16 bits. Se a memória desta arquitetura for ***bit_addressable***:

    a. Qual a dimensão do espaço de endereçamento desta arquitetura?

    b. Qual a dimensão máxima da memória suportada por esta arquitetura expressa em *bytes*?

26. Considere agora uma arquitetura em que o respetivo ISA especifica uma organização de memória do tipo ***word-addressable***, em que a dimensão da *word* é 32 bits. Tendo o espaço de endereçamento do processador 24 bits, qual a dimensão máxima de memória que este sistema pode acomodar expresso em *bytes*?

27. Relativamente à arquitetura MIPS:

    c. Com quantos bits são codificadas as instruções no MIPS?

    d. O que diferencia o registo **`$0`** dos restantes registos de uso geral?

    e. Qual o endereço do registo interno do MIPS a que corresponde a designação lógica **`$ra`**?

28. No MIPS, um dos formatos de codificação de instruções é designado por R:

    a. Quais os campos em que se divide este formato de codificação?

    b. Qual o significado de cada um desses campos?

    c. Qual o valor do campo *opCode* nesse formato?

    d. O que faz a instrução cujo código máquina é: **`0x00000000`**?

29. O símbolo ” >> “ da linguagem C significa deslocamento à direita e é traduzido por SRL ou SRA (no caso do MIPS). Em que casos é que o compilador gera um SRL e quando é que gera um SRA?

30. Qual a instrução nativa do MIPS em que é traduzida a instrução virtual "**`move $4,$15`**"?

31. Determine o código máquina das seguintes instruções (verifique a tabela na última página):

    a. xor $5,$13,$24

    b. sub $25,$14,$8

    c. sll $3,$9,7

    d. sra $18,$9,8

32. Traduza para instruções *Assembly* do MIPS a seguinte expressão aritmética, supondo **x** e **y** são inteiros e residentes em **$t2** e **$t5**, respetivamente (apenas pode usar instruções nativas e não deverá usar a instrução de multiplicação):

```
y = -3 * x + 5;
```

33. Traduza para instruções *assembly* do MIPS o seguinte trecho de código:

```
int a, b, c;                //a:$t0, b:$t1, c:$t2
unsigned int x, y, z;       //x:$a0, y:$a1, z:$a2
z = x >> 2 + y;
c = a >> 5 - 2 * b;
```

34. Considere que as variáveis **g**, **h**, **i** e **j** são conhecidas e podem ser representadas por uma variável de 32 bits num programa em C. Qual a correspondência em linguagem C às seguintes instruções:

```
a. add  h, i, j          #

b. addi j, j, 1          #
   add  h, g, j          #
```

35. Assumindo que **g=1**, **h=2**, **i=3** e **j= 4** qual o valor destas variáveis no final das sequências das alíneas da questão anterior?

36. Qual a operação realizada pela instrução "**slt**" e quais os resultados possíveis?

37. Qual o valor armazenado no registo **$1** na execução da instrução "**slt $1, $3, $7**", admitindo que:

    a. $3=5 e $7=23

    b. $3=0xFE e $7=0x913D45FC

38. Com que registo implícito comparam as instruções "**bltz**", "**blez**", "**bgtz**" e "**bgez**"?

39. Decomponha em instruções nativas do MIPS as seguintes instruções virtuais:

    a. blt $15,$3,exit

    b. ble $6,$9,exit

    c. bgt $5,0xA3,exit

    d. bge $10,0x57,exit

    e. blt $19,0x39,exit

    f. ble $23,0x16,exit

40. Na tradução e C para *assembly*, quais as principais diferenças entre um ciclo "**while(…){…}**" e um ciclo "**do{…}while(…);**" ?

41. Traduza para *assembly* do MIPS os seguintes trechos de código de linguagem C (admita que **a**, **b** e **c** residem nos registos **$4**, **$7** e **$13**, respetivamente):

```
a. if(a > b && b != 0)
     c = b << 2;
   else
     c = (a & b) ^ (a | b);


b. if(a > 3 || b <= c)
     c = c - (a + b);
   else
     c = c + (a - 5);
```

42. Qual o modo de endereçamento usado pelo MIPS para ter acesso a palavras residentes na memória externa?

43. Na instrução "**lw $3,0x24($5)**" qual a função dos registos **$3** e **$5** e da constante **0x24**?

44. Qual é o formato de codificação das instruções de acesso à memória no MIPS e qual o significado de cada um dos seus campos?

45. Qual a diferença entre as instruções "**sw**" e "**sb**"?

46. O que distingue as instruções "**lb**" e "**lbu**"?

47. O que acontece quando uma instrução **lw/sw** acede a um endereço que não é múltiplo de 4?

48. Traduza para *assembly* do MIPS os seguintes trechos de código de linguagem C (atribua registos internos para o armazenamento das variáveis **i** e **k**) :

```
a. int i, k;
   for(i=5, k=0; i < 20; i++, k+=5);

b. int i=100, k=0;
   for( ; i >= 0; )
   {
     i--;
     k -= 2;
    }

c. unsigned int k=0;
   for( ; ; )
   {
     k += 10;
   }

d. int k=0, i=100;
   do
   {
     k += 5;
   } while(--i >= 0);
```

49. Sabendo que o *OpCode* da instrução "**`lw`**" é **`0x23`**, determine o código máquina, expresso em hexadecimal, da instrução "**`lw $3,0x24($5`**)".

50. Suponha que a memória externa foi inicializada, a partir do endereço **`0x10010000`**, com os valores **`0x01,0x02,0x03,0x04,0x05`**,… .Suponha ainda que **`$3=0x1001`** e **`$5=0x10010000`**. Qual o valor armazenado no registo destino após a execução da instrução "**`lw $3,0x24($5)`**"?

51. Considere as mesmas condições da questão anterior. Qual o valor armazenado no registo destino pelas instruções:

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

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

52. Quantos *bytes* são reservados em memória por cada uma das diretivas:

    `a. L1: .asciiz "Aulas5&6T"`

    `b. L2: .byte 5,8,23`

    `c. L3: .word 5,8,23`

    `d. L4: .space 5`

53. Desenhe esquematicamente a memória e preencha-a com o resultado das diretivas anteriores admitindo que são interpretadas sequencialmente pelo *Assembler*.

54. Supondo que "**`L1:`**" corresponde ao endereço inicial do segmento de dados, e que esse endereço é **`0x10010000`**, determine os endereços a que correspondem os *labels* "**`L2:`**", "**`L3:`**" e "**`L4:`**".

55. Suponha que "**`b`**" é um *array* declarado como "**`int b[25];`**":

    a. Como é obtido o endereço inicial do *array*, i.e., o endereço a partir do qual está armazenado o seu primeiro elemento?

    b. Supondo uma memória "*byte-addressable*", como é obtido o endereço do elemento "**`b[6]`**"?

56. O que é codificado no campo offset do código máquina das instruções "**`beq/bne`**" ?

57. A partir do código máquina de uma instrução "**`beq/bne`**", como é formado o endereço-alvo (*Branch Target Address*)?

58. Qual o formato de codificação de cada uma das seguintes instruções: "**`beq/bne`**", "**`j`**", "**`jr`**"?

59. A partir do código máquina de uma instrução "**`j`**", como é formado o endereço-alvo (*Jump Target Address*)?

60. Dada a seguinte sequência de declarações:

```
int b[25];
int a;
int *p = b;
```

Identifique qual ou quais das seguintes atribuições permitem aceder ao elemento de índice 5 do *array* "**`b`**":

`a = b[5];`	`a = *p + 5;`	`a = *(p + 5);`	`a = *(p + 20);`

61. Assuma que as variáveis **`f, g, h, i`** e **`j`** correspondem aos registos **`$t0`**, **`$t1`**, **`$t2`**, **`$t3`** e **`$t4`** respetivamente. Considere que o endereço base dos *arrays* **A** e **B** está contido nos registos **`$s0`** e **`$s1`**. Considere ainda as seguintes expressões:

```
f = g + h + B[2]

j = g – A[B[2]]
```

a. Qual a tradução para *assembly* de cada uma das instruções C indicadas?

b. Quantas instruções *assembly* são necessárias para cada uma das instruções C indicadas? E quantos registos auxiliares são necessários?

c. Considerando a tabela seguinte que representa o conteúdo byte-a-byte da memória, nos endereços correspondentes aos *arrays* A e B, indique o valor de cada elemento dos *arrays* assumindo uma organização *little endian*.

Endereço	Valor
A+12	…
A+11	0x00
A+10	0x00
A+9	0x00
A+8	0x01
A+7	0x22
A+6	0xED
A+5	0x34
A+4	0x00
A+3	0x00
A+2	0x00
A+1	0x00
A+0	0x12

Endereço	Valor
B+12	…
B+11	0x00
B+10	0x00
B+9	0x00
B+8	0x02
B+7	0x00
B+6	0x00
B+5	0x50
B+4	0x02
B+3	0xFF
B+2	0xFF
B+1	0xFF
B+0	0xFE

A[0]=
A[1]=
A[2]=

B[0]=
B[1]=
B[2]=

d. Assumindo que **g = -3** e **h = 2**, qual o valor final das variáveis **f** e **j**?

62. Pretende-se escrever uma função para a troca do conteúdo de duas variáveis (troca(a, b);).
Isto é, se, antes da chamada à função, **`a=2`** e **`b=5`**, então, após a chamada à função, os valores de **`a`** e **`b`** devem ser: **`a=5`** e **`b=2`**

Uma solução incorreta para o problema é a seguinte:

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

Identifique o erro presente no trecho de código e faça as necessárias correções para que a função tenha o comportamento pretendido

63. Na instrução "**`jr $ra`**", como é obtido o endereço-alvo?

64. Qual é o menor e o maior endereço para onde uma instrução "**`j`**", residente no endereço de memória **`0x5A18F34C`**, pode saltar?

65. Qual é o menor e o maior endereço para onde uma instrução "**`beq`**", residente no endereço de memória **`0x5A18F34C`**, pode saltar?

66. Qual é o menor e o maior endereço para onde uma instrução "**`jr`**", residente no endereço de memória **`0x5A18F34C`** pode saltar?

67. Qual a gama de representação da constante nas instruções aritméticas imediatas?

68. Qual a gama de representação da constante nas instruções lógicas imediatas?

69. Por que razão não existe, no ISA do MIPS, uma instrução que permita manipular diretamente uma constante de 32 bits?

70. Como é que, no *assembly* do MIPS, se podem manipular constantes de 32 bits?

71. Apresente a decomposição em instruções nativas das seguintes instruções virtuais:

```
a. li      $6,0x8B47BE0F
b. xori    $3,$4,0x12345678
c. addi    $5,$2,0xF345AB17
d. beq     $7,100,L1
e. blt     $3,0x123456,L2
```

72. O que é uma sub-rotina?

73. Qual a instrução do MIPS usada para saltar para uma sub-rotina?

74. Por que razão não pode ser usada a instrução "j" para saltar para uma sub-rotina?

75. Quais as operações que são sequencialmente realizadas na execução de uma instrução "**`jal`**"?

76. Qual o nome virtual e o número do registo associado à execução dessa instrução?

77. No caso de uma sub-rotina ser simultaneamente chamada e chamadora (sub-rotina intermédia) que operações é obrigatório realizar nessa sub-rotina?

78. Qual a instrução usada para retornar de uma sub-rotina?

79. Que operação fundamental é realizada na execução dessa instrução?

80. O que é uma *stack* e qual a finalidade do *stack pointer*?

81. Como funcionam as operações de ***push*** e ***pop***?

82. Por que razão as *stacks* crescem normalmente no sentido dos endereços mais baixos?

83. Quais as regras para a implementação em software de uma *stack* no MIPS?

84. Qual o registo usado, no MIPS, como *stack pointer*?

85. De acordo com a convenção de utilização de registos no MIPS:
    a. Que registos são usados para passar parâmetros e para devolver resultados de uma sub-rotina?
    b. Quais os registos que uma sub-rotina pode livremente usar e alterar sem necessidade de prévia salvaguarda?
    c. Quais os registos que uma sub-rotina não pode alterar?
    d. Quais os registos que uma sub-rotina chamadora tem a garantia que a sub-rotina chamada não altera?
    e. Em que situação devem ser usados registos "**$sn**"?
    f. Em que situação devem ser usados os restantes registos: **$tn**, **$an** e **$vn**?

86. De acordo com a convenção de utilização de registos do MIPS:
    a. Que registos podem ter que ser copiados para a stack numa sub-rotina intermédia?
    b. Que registos podem ter que ser copiados para a stack numa sub-rotina terminal?

87. Para a função com o protótipo seguinte indique, para cada um dos parâmetros de entrada e para o valor devolvido, qual o registo do MIPS usado para a passagem dos respetivos valores:

```
char fun(int a,unsigned char b,char *c,int *d);
```

88. Para uma codificação em complemento para 2, apresente a gama de representação que é possível obter com **3**, **4**, **5**, **8** e **16** bits (indique os valores-limite da representação em binário, hexadecimal e em decimal com sinal e módulo).

89. Traduza para *assembly* do MIPS a seguinte função "**fun1()**", aplicando a convenção de passagem de parâmetros e salvaguarda de registos:

```
char *fun2(char *, char);

char *fun1(int n, char *a1, char *a2)
{
      int j = 0;
      char *p = a1;

      do
      {
            if((j % 2) == 0)
            fun2(a1++, *a2++);
      } while(++j < n);
      *a1='\0';
      return p;
}
```

90. Determine a representação em complemento para 2 com 16 bits das seguintes quantidades:

    5, -3, -128, -32768, 31, -8, 256, -32

91. Determine o valor em decimal representado por cada uma das quantidades seguintes, supondo que estão codificadas em complemento para 2 com 8 bits:

    0b00101011, 0xA5, 0b10101101, 0x6B, 0xFA, 0x80

92. Determine a representação das quantidades do exercício anterior em hexadecimal com 16 bits (também codificadas em complemento para 2).

93. Como é realizada a deteção de *overflow* em operações de adição com quantidades sem sinal?

94. Como é realizada a deteção de *overflow* em operações de adição com quantidades com sinal (codificadas em complemento para 2)?

95. Considere os seguintes pares de valores em **$s0** e **$s1**:

```
i.  $s0 = 0x70000000 $s1 = 0x0FFFFFFF
ii. $s0 = 0x40000000 $s1 = 0x40000000
```

    a. Qual o resultado produzido pela instrução **add $t0, $s0, $s1**?

    b. Para a alínea anterior os resultados são os esperados ou ocorreu *overflow*?

    c. Qual o resultado produzido pela instrução **sub $t0, $s0, $s1**?

    d. Para a alínea anterior os resultados são os esperados ou ocorreu *overflow*?

    e. Qual o resultado produzido pelas instruções:

```
add $t0, $s0,$s1
add $t0, $t0,$t1 ?
```

    f. Para a alínea anterior os resultados são os esperados ou ocorreu *overflow*?

96. Para a multiplicação de dois operandos de "**m**" e "**n**" bits, respetivamente, qual o número de bits necessário para o armazenamento do resultado?

97. Apresente a decomposição em instruções nativas da instrução virtual **mult $5,$6,$7** .

98. Determine o resultado da instrução anterior, quando

```
$6=0xFFFFFFFE e $7=0x00000005.
```

99. Apresente a decomposição em instruções nativas das instruções virtuais

```
div $5,$6,$7 e rem $5,$6,$7
```

100. Determine o resultado das instruções anteriores, quando

```
$6=0xFFFFFFF0 e $7=0x00000003
```

101. As duas sub-rotinas seguintes permitem detetar *overflow* nas operações de adição com e sem sinal, no MIPS. Analise o código apresentado e determine o resultado produzido, pelas duas sub-rotinas, nas seguintes situações:

a. $a0=0x7FFFFFF1, $a1=0x0000000E;

b. $a0=0x7FFFFFF1, $a1=0x0000000F;

c. $a0=0xFFFFFFF1, $a1=0xFFFFFFFF;

d. $a0=0x80000000, $a1=0x80000000;

```
# Overflow detection, signed
# int isovf_signed(int a, int b);
isovf_signed: ori $v0,$0,0
              xor $1,$a0,$a1
              slt $1,$1,$0
              bne $1,$0,notovf_s
              addu $1,$a0,$a1
              xor $1,$1,$a0
              slt $1,$1,$0
              beq $1,$0,notovf_s
              ori $v0,$0,1
notovf_s:     jr $ra

# Overflow detection, unsigned
# int isovf_unsigned(unsigned int a, unsigned int b);
isovf_unsig: ori $v0,$0,0
              nor $1,$a1,$0
              sltu $1,$1,$a0
              beq $1,$0,notovf_u
              ori $v0,$0,1
notovf_u:     jr $ra
```

102. Ainda no código das sub-rotinas da questão anterior, qual a razão para não haver salvaguarda de qualquer registo na stack?

Tabela de códigos de função (funct) e códigos de operação (OpCode) das principais instruções do MIPS

Arithm / Logical Instructions			Comparison Instructions	
Instruction	(funct)		Instruction	(OpCode)
add	100000 (0x20)		slt	101010 (0x2A)
addu	100001 (0x21)		sltu	101001 (0x29)
and	100100 (0x24)		slti	001010 (0x0A)
div	011010 (0x1A)		sltiu	001001 (0x09)
divu	011011 (0x1B)			
mult	011000 (0x18)		Branch Instructions	
multu	011001 (0x19)		beq	000100 (0x04)
nor	100111 (0x27)		bne	000101 (0x05)
or	100101 (0x25)		bgtz	000111 (0x07)
sll	000000 (0x00)		bgez	000001 (0x01)[1]
sra	000011 (0x03)		bltz	000001 (0x01)
srl	000010 (0x02)		blez	000110 (0x06)
sub	100010 (0x22)			
subu	100011 (0x23)		Jump Instructions	
xor	100110 (0x26)		j	000010 (0x02)
			jal	000011 (0x03)
Arithm / Logical Imm			jalr	001001 (0x09)
Instruction	(OpCode)		jr	001000 (0x08)
addi	001000 (0x08)			
addiu	001001 (0x09)		Load/Store Instructions	
andi	001100 (0x0C)		lb	100000 (0x20)
ori	001101 (0x0D)		lbu	100100 (0x24)
xori	001110 (0x0E)		lw	100011 (0x23)
			sb	101000 (0x28)
			sw	101011 (0x2B)
			Data Movement Instructions	
			mfhi	010000 (0x10)
			mflo	010010 (0x12)
			mthi	010001 (0x11)
			mtlo	010011 (0x13)

[1] O OpCode é igual ao da instrução **bltz** mas o valor de **rt** é igual a 00001_b