Folha de Exercícios Teórico-práticos N.º 1

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 blocos fundamentais de um CPU?
3. Qual a função do *Program Counter*?
4. Quais os passos mais importantes na execução de uma instrução no CPU?
5. O que é um compilador? O que é um assembler?
6. Quantos registos internos tem o MIPS? Qual a dimensão em bits de cada um?
7. Qual o formato de uma instrução aritmética no MIPS?
8. O que distingue a instrução **SRL** da instrução **SRA** do MIPS?
9. Se **$5=0x81354AB3**, qual o resultado das instruções:
   a. **srl $3,$5,1**
   b. **sra $4,$5,1**
10. O que é uma *system call*? No MIPS, qual o registo usado para identificar a *system call* a executar? Qual o registo ou registos usados para passar argumentos para *systems calls*? Qual o registo usado para obter o resultado produzido por uma *system call*?
11. O que é um endereço?
12. O que é o espaço de endereçamento de um processador?
13. Como se organiza internamente um processador? Quais são os blocos fundamentais da secção de dados? Para que serve a unidade de controlo?
14. O que é o conceito "stored-program"?
15. Como se codifica uma instrução? Que informação fundamental deverá ter o código de uma instrução?
16. O que é o ISA?
17. Quais são as classes de instruções que agrupam as instruções de uma arquitetura?
18. O que carateriza as arquiteturas "register-memory" e "load-store"? De que tipo é a arquitetura MIPS?
19. Com quantos bits são codificadas as instruções no MIPS? Quantos registos internos tem o MIPS? O que diferencia o registo **$0** dos restantes? Qual o número do registo interno do MIPS a que corresponde o registo **$ra**?

20. Quais os campos em que se divide o formato de codificação **R**? Qual o significado de cada um desses campos? Qual o valor do campo **opCode** nesse formato?
21. O que faz a instrução cujo código máquina é: **0x00000000**?
22. O símbolo >> da linguagem C significa deslocamento à direita e é traduzido por **SRL** ou **SRA** (no caso do MIPS). Quando é que é usado **SRL** e quando é que é usado **SRA**?
23. Qual a instrução nativa do MIPS em que é traduzida a instrução virtual "**move $4, $15**"?
24. Determine o código máquina das seguintes instruções:

    a. `xor $5,$13,$24`
    b. `sub $30,$14,8`
    c. `sll $3,$9,7`
    d. `sra $18,$9,8`

25. Traduza para instruções *assembly* do MIPS a seguinte expressão aritmética, supondo **x** e **y** 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;
```

26. 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;
```

27. Considere 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
    ```

28. Assumindo que g=1, h=2, i=3 e j= 4 qual o valor final das variáveis no final das sequências das alíneas da questão anterior?
29. Qual a função da instrução "**slt**"?
30. Qual o valor armazenado no registo **$1** na execução da instrução "**slt $1, $3, $7**", admitindo que:
    a. **a) $3=5 e $7=23**
    b. **b) $3=0xFE e $7=0x913D45FC**
31. Com que registo comparam as instruções "**bltz**", "**blez**", "**bgtz**" e "**bgez**"?

32. 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**
33. Na tradução para *assembly*, que diferenças encontra entre um ciclo do tipo "**while(…){…}**" e um do tipo "**do{…}while(…);**"
34. 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):

1)
```
if(a > b && b != 0)
        c = b << 2;
else
        c = (a & b) ^ (a | b);
```

2)
```
if(a > 3 || b <= c)
        c = c - (a + b);
else
        c = c + (a - 5);
```

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

1)
```
int i, k;
for(i=5, k=0; i < 20; i++, k+=5);
```

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

3)
```
unsigned int k=0;
for( ; ; )
{
     k += 10;
}
```

4)
```
int k=0, i=100;
do
{
     k += 5;
} while(--i >= 0);
```

36. Qual o modo de endereçamento usado pelo MIPS para acesso a quantidades residentes na memória externa?
37. Na instrução "**lw $3,0x24($5)**" qual a função dos registos **$3** e **$5** e da constante **0x24**?
38. Qual o formato de codificação das instruções de acesso à memória no MIPS e qual o significado de cada um dos seus campos?
39. Qual a diferença entre as instrução "**sw**" e "**sb**"? O que distingue as instruções "**lb**" e "**lbu**"?
40. O que acontece quando uma instrução **lw/sw** acede a um endereço que não é múltiplo de 4?
41. Sabendo que o *opcode* da instrução "**lw**" é **0x23**, determine o código máquina, expresso em hexadecimal, da instrução "**lw $3,0x24($5)**".
42. 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)**"?
43. Nas condições anteriores qual o valor armazenado no registo destino pelas instruções: "**lbu $3,0xA3($5)**" e "**lb $4,0xA3($5)**"
44. 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**
45. Desenhe esquematicamente a memória e preencha-a com o resultado das diretivas anteriores

46. 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**".

47. Suponha que "**b**" é um *array* declarado como "**int b[25]**;". Como é obtido o endereço inicial do *array*, i.e., o endereço da sua primeira posição? Supondo uma memória "*byte-addressable*", como é obtido o endereço do elemento "**b[6]**"?

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

```
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*.

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?

49. 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);
```

50. 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

51. Qual o formato de codificação de cada uma das seguintes instruções: "beq/bne", "j", "jr"?
52. O que é codificado no campo offset do código máquina das instruções "beq/bne" ?
53. A partir do código máquina de uma instrução "beq/bne", como é formado o endereço-alvo (Branch Target Address)?
54. A partir do código máquina de uma instrução "j", como é formado o endereço-alvo (Jump Target Address)?
55. Na instrução "jr $ra", como é obtido o endereço-alvo?
56. Qual o endereço mínimo e máximo para onde uma instrução "j", residente no endereço de memória 0x5A18F34C, pode saltar?
57. Qual o endereço mínimo e máximo para onde uma instrução "beq", residente no endereço de memória 0x5A18F34C, pode saltar?
58. Qual o endereço mínimo e máximo para onde uma instrução "jr", residente no endereço de memória 0x5A18F34C pode saltar?
59. Qual a gama de representação da constante nas instruções aritméticas imediatas?
60. Qual a gama de representação da constante nas instruções lógicas imediatas?
61. Porque razão não existe no ISA do MIPS uma instrução que permita manipular diretamente uma constante de 32 bits?
62. Como é que no MIPS se podem manipular constantes de 32 bits?

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

```
li      $6,0x8B47BE0F
xori    $3,$4,0x12345678
addi    $5,$2,0xF345AB17
beq     $7,100,L1
blt     $3,0x123456,L2
```

64. O que é uma sub-rotina? Qual a instrução do MIPS usada para saltar para uma sub-rotina? Porque razão não pode ser usada a instrução "j"?
65. Quais as operações realizadas, e relativa sequência, na execução de uma instrução "jal"? Qual o nome virtual e o número do registo associado à execução dessa instrução?
66. No caso de uma sub-rotina ser simultaneamente chamada e chamadora (sub-rotina intermédia) que operações é obrigatório realizar nessa sub-rotina?
67. Qual a instrução usada para retornar de uma sub-rotina? Que operação fundamental é realizada na execução dessa instrução?
68. 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? Quais os registos que uma sub-rotina chamadora tem a garantia que a sub-rotina chamada não altera?
    d. Em que situação devem ser usados registos $sn? Em que situação devem ser usados os restantes: $tn, $an e $vn?
69. O que é a *stack*? Qual a utilidade do *stack pointer*?
70. Como funcionam as operações de *push* e *pop*?
71. Porque razão a *stack* cresce no sentido dos endereços mais baixos?
72. Quais as regras para a implementação em software de uma *stack* no MIPS? Qual o registo usado como *stack pointer*?
73. De acordo com a convenção de utilização de registos do MIPS:
74. Que registos podem ter que ser copiados para a *stack* numa sub-rotina intermédia?
75. Que registos podem ter que ser copiados para a *stack* numa sub-rotina terminal?

76. 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)
```

77. 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;
}
```

78. 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).

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

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

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

001010112,	0xA5,
101011012,	0x6B,
0xFA,	0x80

81. Determine a representação das quantidades do exercício anterior em hexadecimal com 16 bits (também codificadas em complemento para 2).
82. Como é realizada a deteção de *overflow* em operações de adição com quantidades sem sinal? E com quantidades com sinal (codificadas em complemento para 2)?
83. 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` ?
É o resultado esperado ou ocorreu overflow?

b. Qual o resultado produzido pela instrução `sub $t0, $s0, $s1` ?
É o resultado esperado ou ocorreu overflow?

c. Qual o resultado produzido pelas instruções:
```
add $t0, $s0,$s1
add $t0, $t0,$t1 ?
```
É o resultado esperado ou ocorreu overflow?

84. Para a multiplicação de dois operandos de "**m**" e "**n**" bits, respetivamente, qual o número de bits necessário para o armazenamento do resultado?
85. Apresente a decomposição em instruções nativas da instrução virtual **mult $5,$6,$7**
86. Determine o resultado da instrução anterior, quando
$6=0xFFFFFFFE e **$7=0x00000005**.
87. Apresente a decomposição em instruções nativas das instruções virtuais
div $5,$6,$7 e **rem $5,$6,$7**
88. Determine o resultado das instruções anteriores, quando
$6=0xFFFFFFF0 e **$7=0x00000003**
89. As duas sub-rotinas seguintes permitem detetar overflow nas operações de adição com e sem sinal, no MIPS. Analise o código apresentado e determine o resultado produzido, pelas duas sub-rotinas, nas seguintes situações:
    - **$a0=0x7FFFFFF1, $a1=0x0000000E;**
    - **$a0=0x7FFFFFF1, $a1=0x0000000F;**
    - **$a0=0xFFFFFFF1, $a1=0xFFFFFFFF;**
    - **$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_unsigned: 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
```

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