



## Arquitectura de Computadores

ENGENHARIA INFORMÁTICA  
FACULDADE DE CIÊNCIAS E TECNOLOGIA  
UNIVERSIDADE DE COIMBRA

### – 2ª Frequência –

14 de Junho de 2019

Duração: 90 min. + 15 min. de tolerância

Nome: \_\_\_\_\_ Número: \_\_\_\_\_

#### Notas Importantes:

A fraude denota uma grave falta de ética e constitui um comportamento não admissível num estudante de ensino superior. Não serão admitidas eventuais tentativas de fraude, que provocarão a reprovação imediata, tanto do facilitador como do prevaricador.

Durante a prova pode consultar a bibliografia da disciplina (slides, livros, enunciados e material de apoio a trabalhos práticos). No entanto, não é permitido o uso de computadores, calculadoras ou qualquer outro dispositivo electrónico.

Este é um teste de escolha múltipla e deverá assinalar sem ambiguidades as respostas na tabela apresentada a baixo. Cada pergunta corretamente respondida vale cinco pontos; cada resposta errada desconta dois pontos; e cada pergunta não respondida vale zero pontos. Uma nota final abaixo de zero pontos, vale zero valores.

#### Respostas: (indicar resposta A, B, C ou D, debaixo do número da questão)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

1. Assumindo que a *label* `str` se refere a uma *string* de caracteres armazenada no endereço de memória `0x00008400`, quantas entradas na tabela de realocação gera o seguinte código *Assembly* do MIPS?

- a. 1 entradas na tabela de realocação.
- b. 2 entradas na tabela de realocação.
- c. 3 entradas na tabela de realocação.
- d. 4 entradas na tabela de realocação.

```
L1:
    lw    $t0, 8($sp)
    lw    $t1, 12($sp)
    addu   $t2, $t1, $t0
    sw     $t2, 16($sp)
    blt    $t0, 20, L1
    la     $a0, str
    lw     $t3, 4($a0)
    jal    func
    ...
```

2. Suponha a multiplicação, em *Assembly* do MIPS, de dois números armazenados nos registos `$t2` e `$t4`. Se quiser usar apenas instruções *TAL*, escolha qual dos seguintes excertos permite colocar o resultado da multiplicação no registo `$a0`.

MIPS 1  
...  
mult \$t2,\$t4  
mflo \$a0

MIPS 2  
...  
mult \$a0,\$t2,\$t4

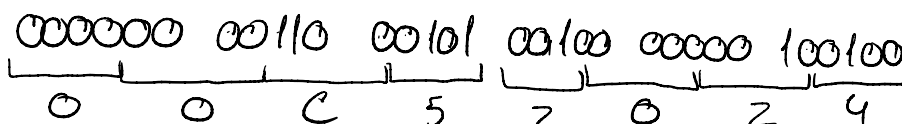
MIPS 3  
...  
mul \$a0,\$t2,\$t4

MIPS 4  
...  
mul \$t2,\$t4  
mflo \$a0

- a. MIPS 1      b. MIPS 2      c. MIPS 3      d. MIPS 4

3. Considere a instrução em *Assembly* do MIPS `and $a0, $a2, $a1`. Sabendo que os registos `$a0`, `$a1` e `$a2` são os registos #4, #5 e #6, respectivamente, indique qual dos seguintes códigos hexadecimais corresponde à codificação desta instrução:

- a. `0x00A62024`
- b. `0x00C52020`
- c. `0x00C52024`
- d. `0x00A43024`



4. Diga qual das afirmações é **VERDADEIRA**:

- a. O deslocamento relativo de uma instrução *branch* permite codificar saltos entre -32768 e +32767 instruções.
- b. As chamadas a funções são feitas com instruções do tipo *branch*
- c. Numa instrução do tipo *branch*, o campo *immediate* deve ser sempre um valor positivo.
- d. Uma operação de *branch* pode mover o *Program Counter* (PC) para qualquer zona da memória

5. Qual dos segmentos de código em C apresentado ao lado reproduz mais fielmente o ciclo em *Assembly* indicado em baixo?

```
loop:
    addiu    $s4,$s4,-1
    slt      $t0,$s3,$s4
    slt      $t1,$s4,$s5
    add      $t0,$t0,$t1
    bne      $t0,$0,out
    j        loop
out:
```

- a. `do{$s4=$s4-1;} while (($s3 < $s4) || ($s4 < $s5))`
- b. `do{$s4=$s4-1;} while (($s3 >= $s4) || ($s4 >= $s5))`
- c. `do{$s4=$s4-1;} while (($s3 < $s4) && ($s4 < $s5))`
- d. `do{$s4=$s4-1;} while (($s3 >= $s4) && ($s4 >= $s5))`

6. Relativamente ao MIPS, diga qual das afirmações é **FALSA**:

- a. Os registos \$a0-\$a3 são usados para a passagem de parâmetros para a função chamada.
- b. Os registos temporários \$t0-\$t7 podem ser alterados em qualquer ponto do programa.
- c. É obrigatório guardar o conteúdo do registo \$ra na pilha sempre que se inicia a execução de uma função.
- d. Os registos \$v0-\$v1 são usados para devolver valores provenientes da execução de funções.

7. Relativamente ao *Assembly* do MIPS, indique qual das seguintes afirmações é **VERDADEIRA**:

- a. As tabelas de símbolos e de realocação são criadas e resolvidas na fase da *linkagem*.
- b. A resolução de “*labels*” que dependem de endereços relativos são feitas na fase do “*assembling*”.
- c. Algumas instruções do tipo R não são totalmente resolvidas na fase do “*assembling*” e necessitam de realocação na fase da “*linkagem*”.
- d. As “*labels*” das instruções *j* e *jal* que sejam referências internas ao ficheiro não necessitam de realocação na fase da “*linkagem*”.

8. Como se decompõe a instrução em *Assembly* do MIPS `or $t0, $t0, 0x000180AB` em instruções TAL?

a) `lui $at, 0x0001`  
`ori $at, $at, 0x80AB`  
`or $t0, $t0, $at`

b) `lui $at, 0xFFB8`  
`andi $at, $at, 0xD7D5`  
`or $t1, $t0, $at`

c) `lui $at, 0x80AB`  
`ori $at, $at, 0x0001`  
`or $t0, $t0, $at`

d) `li $at, 0x180AB`  
`and $t0, $t0, $at`

9. Considere o excerto de código *Assembly* do MIPS listado abaixo. Relativamente às duas instruções `beq`, indique qual terá de ser o valor do imediato em cada uma das instruções de forma a que o código salte para a *label* `L1`.

- No primeiro `beq` o valor do imediato deverá ser 5, enquanto no segundo `beq` deverá ser 2
- No primeiro `beq` o valor do imediato deverá ser 5, enquanto no segundo `beq` deverá ser 3.
- No primeiro `beq` o valor do imediato deverá ser -5, enquanto no segundo `beq` deverá ser -2.
- No primeiro `beq` o valor do imediato deverá ser 7, enquanto no segundo `beq` deverá ser 2.

```

...
beq  $t3,$t1,L1
sb   $0,0($a0)
andi $t3,$t0,0x1000200
beq  $t3,$t2,L1
sb   $0,0($a1)
j     END
L1:
...
END:

```

10. Indique qual das seguintes afirmações é FALSA:

- A instrução `j r` (“*jump register*”) é uma instrução do tipo R.
- Como as instruções do tipo J acomodam endereços absolutos com dimensão máxima de 26 bits, isto significa que com estas instruções conseguimos cobrir uma gama de memória de tamanho máximo  $2^{26}$  words.
- Como o tamanho do campo “*immediate*” das instruções do tipo I é de 16 bits, isto significa que é possível com instruções “*branch*” dar saltos relativos ao “*Program Counter*” de  $\pm 2^{16}$  words.
- O “*Program Counter*” (PC) contém sempre o endereço de memória da próxima instrução a ser executada.

11. Considere o excerto de código apresentado ao lado. Indique qual das instruções apresentadas permite ler o valor 8 da tabela de inteiros para o registo `$t1`.

- `lw $t1,2($t0)`
- `lb $t1,4($t0)`
- `lw $t1,4($t0)`
- `lb $t1,2($t0)`

```

MIPS
.data
tabela: .byte 1,2,3,4,5,6,7,8,9,0
        inteiros ← 0 1 2 3 4 5 6 7 8 9
.text

la $t0,tabela
addi $t0,$t0,5
##instrução em falta

```

12. Considere a instrução `sw $s0,16($a0)` e um datapath com cinco etapas: 1- instruction fetch; 2- instruction decode; 3 – ALU; 4 – memory access; 5- register write. Qual das seguintes afirmações é VERDADEIRA?

- A instrução está inactiva na etapa 3 correspondente à unidade de lógica e aritmética (ALU).
- A instrução apenas está activa nas primeiras três etapas.
- A instrução está activa em todas as cinco etapas do *datapath*.
- Nenhuma das opções acima está correcta.

13. Assuma que para completar uma determinada tarefa é necessário executar sequencialmente um conjunto de 4 operações, sendo que duas dessas operações demoram 10 minutos cada a executar, enquanto que as restantes duas operações demoram 5 minutos e 15 minutos a executar respectivamente. Se pretendermos utilizar um pipeline para otimizar a realização de 20 tarefas, qual seria o incremento em termos de desempenho global que obteríamos em relação a uma execução meramente sequencial?

- 2.67x
- 2.32x
- 2.0x
- 7.0x

14. A execução da instrução `addi $sp, $sp, -16` escrita em Assembly do MIPS permite:

- a. Reservar espaço na pilha correspondente a 16 *words*.
- b. Libertar espaço na pilha correspondente a 16 *words*.
- c. Reservar espaço na pilha correspondente a 4 *words*.
- d. Libertar espaço na pilha correspondente a 4 *words*.

15. Assumindo que o registo `$a0` contém o valor `0x11223344`, indique qual é o valor armazenado no registo `$v0` após a execução do excerto de código *Assembly* do MIPS listado abaixo:

- a. `0x00223355`.
- b. `0x22334455`.
- c. `0x11223355`.
- d. `0x22330055`.

```
andi $t0, $a0, 0x00FFFFFF
srl  $t0, $t0, 8
sll  $t0, $t0, 16
ori  $v0, $t0, 0x00000055
```

16. Considere o excerto de código na caixa seguinte. Indique qual das seguintes opções é **VERDADEIRA**:

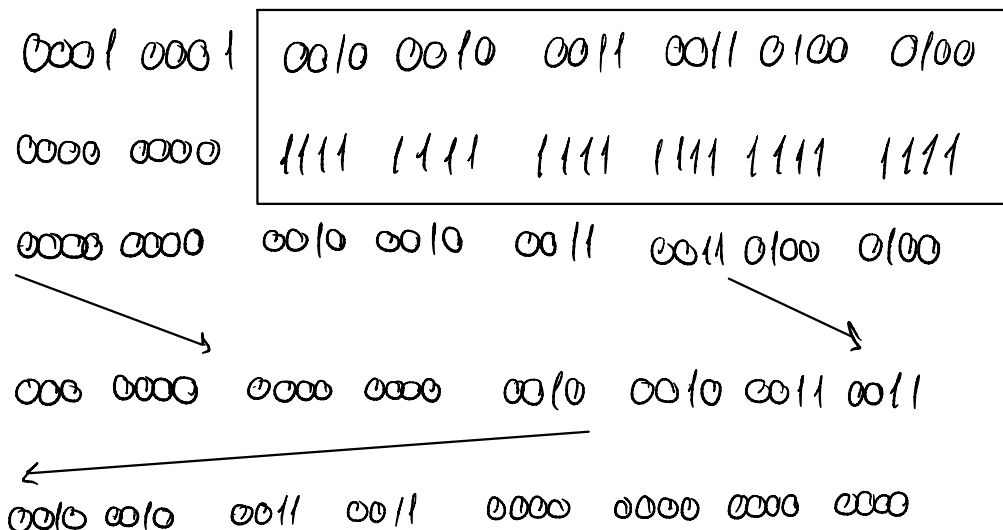
- a. As instruções I1 e I2 evidenciam uma situação de conflitos de dados.
- b. As instruções I1 e I4 exemplificam a ocorrência de um conflito estrutural.
- c. A instrução I3 evidencia uma situação de conflito de controlo.
- d. Todas as afirmações são verdadeiras.


```
I1: add $t0, $t0, $t1
I2: sub $t4, $t0, $t3
I3: beq $t1, $0, Label
I4: andi $t4, $t1, 0x00ff
I5: add $t0, $t1, $t5
Label:
```

17. Indique a razão pela qual se deve usar a instrução `jal` (“*jump and link*”) na chamada de funções:

- a. Porque uma instrução do tipo `j` (“*jump*”) apenas permite fazer saltos para zonas de código próximas da posição actual.
- b. A instrução `jal` guarda o endereço de retorno no registo `$ra` em vez deste ser calculado manualmente.
- c. Na chamada de funções não se deve usar a instrução `jal` (“*jump and link*”), mas sim a instrução `b` (“*branch*”).
- d. Nenhuma das opções anteriores está correcta.

15) 0x11223344



	<b>Arquitectura de Computadores</b> ENGENHARIA INFORMÁTICA FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE DE COIMBRA  <b>– 2ª Frequência –</b>  <b>Parte II</b>  14 de Junho de 2019 Duração: 20 min.	(a preencher pelo docente)

Nome: \_\_\_\_\_ Número: \_\_\_\_\_

**Notas Importantes:**

*A fraude denota uma grave falta de ética e constitui um comportamento não admissível num estudante de ensino superior. Não serão admitidas eventuais tentativas de fraude, que provocarão a reprovação imediata, tanto do facilitador como do prevaricador.*

*Durante a prova pode consultar a bibliografia da disciplina (slides, livros, enunciados e material de apoio a trabalhos práticos). No entanto, não é permitido o uso de computadores, calculadoras ou qualquer outro dispositivo electrónico.*

*Este é um teste de desenvolvimento e deverá produzir sem ambiguidades as respostas na caixa de texto abaixo indicada.*

**Considere a representação “pipelined” do programa apresentada abaixo (F="Instruction Fetch", D="Decode", A="Execute" ou "Arithmetic", M="Memory Access", R="Write Back"):**

```

#1 add $s0, $s1, $s0  F D A M R
#2 ori $t0, $t1, 100   F D A M R
#3 lw  $t2, 16($t1)    F D A M R
#4 sw  $t2, 8($t0)      F D A M R
#5 lw  $t3, 4($t2)      F D A M R
#6 sw  $s0, 24($t3)     F D A M R
#7 and $t1, $t3, $t3    F D A M R
#8 sw  $t3, -64($t1)    F D A M R

```

- i) **Desenhe setas para indicar a dependência de dados entre os diferentes níveis. A seta deve começar no nível em que os dados ficam pela primeira vez disponíveis e terminar onde os dados são absolutamente necessários. Indique também todos os conflitos detectados. Assuma que o processador usa todos os mecanismos de optimização disponíveis.**
- ii) **Resolva todos os conflitos detectados de forma a tornar o código o mais eficiente possível e determine quantos ciclos de relógio são necessários para a execução total do código (assuma que a pipeline se encontre cheia no momento da execução do código apresentado).**