

## Arquitetura de Computadores

ENGENHARIA INFORMÁTICA  
FACULDADE DE CIÊNCIAS E TECNOLOGIA  
UNIVERSIDADE DE COIMBRA

### – 2ª Frequência –

15 de Junho de 2018

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 |
| C | B | D | A | B | D | C | D | B | A  | C  | B  | A  | C  | D  | B  | D  |

1. Assumindo que, num programa em Assembly do MIPS, dispõe de um número inteiro com sinal num registo \$t0 e que pretende calcular a sua divisão por 8, indique qual das seguintes instruções está **CORRETA** para este propósito.

- a. srl \$t0, \$t0, 3
- b. sll \$t0, \$t0, 3

- c.** sra \$t0, \$t0, 3
- d. Nenhuma das restantes opções.

2. Observe o código ao lado, escrito em Linguagem C. Escolha qual das seguintes opções representa o resultado deste código:

- a. 4
- b.** 9
- c. 3
- d. Nenhuma das restantes opções.

```
...
int v[5];
int *p;

p=v;
for (k=0; k<5; k++, p++)
    *p=k*k;

printf("%d\n", v[3]);
```

3. Relativamente ao Assembly do MIPS, diga qual das afirmações é **VERDADEIRA**:

- a. O registo \$ra permanece inalterado entre chamadas de funções se usarmos a instrução jal.
- b. Para devolver um valor numa função podemos usar indiferentemente o registo \$s0 ou \$s1.
- c. O registo \$0 pode ser inicialmente usado como constante, mas ao longo da execução de uma função o seu valor pode mudar.
- d.** Uma função tem de guardar o conteúdo do registo \$ra na pilha apenas e só se durante a sua execução chamar outra função.

4. Escolha qual dos seguintes trechos de código em Assembly do MIPS tem o mesmo resultado do que o código C apresentado ao lado:

a.     blt    \$t2, \$t1, bloco2  
        beqz   \$t0, bloco2  
 bloco1:  
        ...  
        j fim  
 bloco2:  
        ...  
 fim:

c.     bgt    \$t2, \$t1, bloco1  
        bne    \$t2, \$0, bloco1  
 bloco2:  
        ...  
        j fim  
 bloco1:  
        ...  
 fim:

```
if (($t2 >= $t1) && ($t0!=0))
{
    BLOC01
}else{
    BLOC02
}
```

b.     blt    \$t1, \$t2, bloco2  
        beqz   \$t0, bloco2  
 bloco1:  
        ...  
        j fim  
 bloco2:  
        ...  
 fim:

d. Nenhuma das restantes opções.

5. Considerando a convenção de passagem de parâmetros no *Assembly* do MIPS, diga qual das afirmações é **VERDADEIRA**:

- a. O registo \$v1 pode ser usado quando a função precisa de retornar dois valores simultaneamente.
- b. O registo \$v1 só é utilizado quando o valor a retornar por certa função tem um tamanho superior a 32 bits.**
- c. Os registos \$t0-\$t7 podem ser usados para devolver valores entre funções.
- d. É possível utilizar o registo \$ra para devolver valores através pilha.

6. Assumindo que a “label” aux se refere a uma variável armazenada no endereço de memória 0x10000434, e que a “label” func corresponde a uma referência externa ao ficheiro, indique quantas entradas na tabela de realocação gerará o seguinte código Assembly do MIPS?

- a. 2 entradas na tabela de realocação.
- b. 3 entradas na tabela de realocação.
- c. 5 entradas na tabela de realocação.
- d. 4 entradas na tabela de realocação.**

```

      . . .
      la      $s0, aux
      lw      $a0, 0($s0)
      li      $v0, 0
L1:   slt     $t0, $0, $a1
      beq     $t0, $0, L2
      addi    $a1, $a1, -1
      j      L1
L2:   add     $a0, $t2, $a1
      jal     func
      . . .
```

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

- a. Nenhuma instrução do tipo I necessita de realocação na fase da linkagem.
- b. A resolução de “labels” de instruções “branches” é feita pelo “linker”.
- c. Instruções do tipo R são totalmente resolvidas na fase do “assembling” e assim nunca necessitam de realocação na fase do “linking”.**
- d. As tabelas de símbolos e de realocação são criadas e resolvidas na fase da linkagem.

`ori $t1,$t0,0xFFB8D7D5` em instruções TAL?

```
d) lui $at, 0xFFB8
    ori $at, $at, 0xD7D5
    or $t1, $t0, $at
```

**c.** 0xFFFC  
**d.** 0xF300

29 4 16 c. lw \$s0, 18(\$a0)  
1001 0000 1001 0000, 0000 0000 0001 1000 d. lbu \$a0, 24(\$s0)

```

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

        la $t0,tabela
        addi $t0,$t0,8
        ##instrução em falta

```

**a.** 2.5x                      **b.** 2.0x                      **c.** 3.0x                      **d.** 3.5x

14. Qual das seguintes afirmações, relativas ao gcc que utilizou nas aulas práticas laboratoriais, é **VERDADEIRA**:

- a. O uso da flag -O com o compilador gcc produz um ficheiro objecto.
- b. O uso da flag -g com o compilador gcc permite tornar a execução do nosso código mais rápido.
- c. O uso da flag -O com o compilador gcc seguida de um sufixo numérico permite definir o modo de optimização de código a utilizar.
- d. Nenhuma das outras opções é verdadeira.

15. Assumindo que o registo \$t0 contém o valor 0x12345678, indique qual é o valor armazenado no registo \$t2 após a execução do excerto de código *Assembly* do MIPS listado abaixo:

- a. 0x02345670.
- b. 0x00000000.
- c. 0x12345678.
- d. 0x02345678.

```
srl $t1,$t0,3  
sll $t1,$t1,7  
srl $t1,$t1,4  
and $t2,$t1,$t0
```

16. Indique qual das seguintes afirmações sobre conflitos numa arquitectura baseada no conceito de *pipelining* é **FALSA**:

- a. O conflito de dados resultante de uma dependência de dados de um *load* da memória para um registo e a sua consequente utilização na instrução seguinte não pode ser resolvido com o mecanismo de *forwarding* (é obrigatório a ocorrência de um *stall*).
- b. Os conflitos de controlo podem ser resolvidos através da antecipação da tomada de decisão dos saltos para a etapa de *fetch*.
- c. Alguns conflitos estruturais podem ser resolvidos através da utilização de caches para instruções e caches de dados
- d. A utilização de um *Register File* com um mecanismo de *Read-After-Write* resolve um conflito estrutural entre os estágios 2 e 5 do pipeline.

17. Considerando o trecho de programa indicado ao lado, indique qual das afirmações é **VERDADEIRA**:

- a. A variável *dados* vai ser armazenada na zona de dados estáticos, a variável *i* na pilha e a variável *temp* é armazenada no *heap*.
- b. As variáveis *dados* e *temp* vão ser armazenadas no *heap* porque contêm vários elementos.
- c. As variáveis *dados*, *i* e *temp* vão ser armazenadas na pilha, sendo que *dados* e *temp* apontam para zonas de memória no *heap*.
- d. A variável *dados* vai ser armazenada na zona de dados estáticos do programa, a variável *temp* e *i* na pilha. A variável *temp* vai conter um endereço localizado no *heap*.

```
double dados[3] = {1.0,2.0,3.0};  
void main(){  
    int i,*temp;  
    temp=(double*)malloc(3*sizeof(double));  
    for(i=0;i<3;i++)  
        temp[i]=dados[i];  
}
```


↳ dados estáticos

↳ pilha

↳ Heap

0 x 12345678

+3  
0'1  
00 01 00 10 00 11 01 00 01 01 01 10 01 11 10 00  
00 00 00 10 01 00 01 10 10 00 10 10 11 00 11 11  
00 10 00 11 01 00 01 01 01 10 01 11 10 00 00 00  
+4  
0'1  
00 00 00 10 00 11 01 00 01 01 01 10 01 11 10 00  
00 01 00 10 00 11 01 00 01 01 01 10 01 11 10 00  
00 00 00 10 00 11 01 00 01 01 01 10 01 11 10 00  
0 2 3 4 5 6 7 8


|   |  |                            |
|---|--|----------------------------|
|  | <b>Arquitectura de Computadores</b><br>ENGENHARIA INFORMÁTICA<br>FACULDADE DE CIÊNCIAS E TECNOLOGIA<br>UNIVERSIDADE DE COIMBRA<br><br><b>– 2ª Frequência –</b><br><br><b>Parte Prática</b> | (a preencher pelo docente) |
|   |  |                            |

Nome: \_\_\_\_\_ Número: \_\_\_\_\_

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

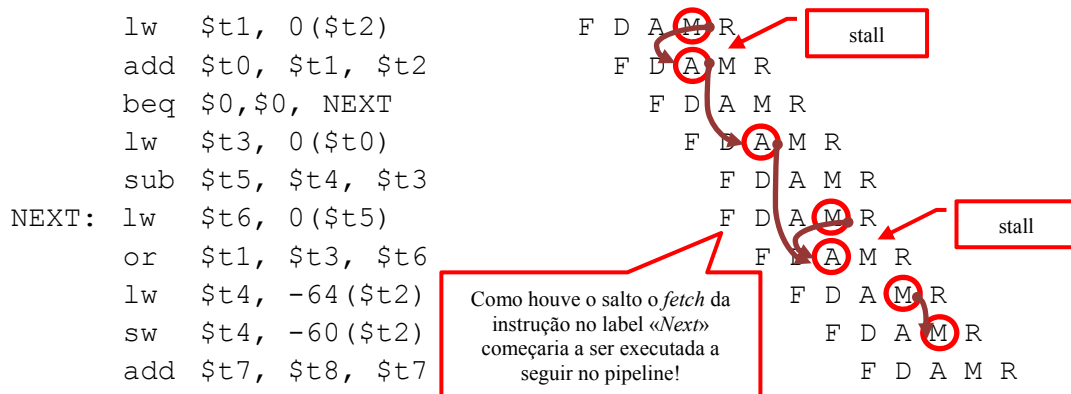
|          |                  |           |
|----------|------------------|-----------|
| lw       | \$t1, 0(\$t2)    | F D A M R |
| add      | \$t0, \$t1, \$t2 | F D A M R |
| beq      | \$0, \$0, NEXT   | F D A M R |
| lw       | \$t3, 0(\$t0)    | F D A M R |
| sub      | \$t5, \$t4, \$t3 | F D A M R |
| NEXT: lw | \$t6, 0(\$t5)    | F D A M R |
| or       | \$t1, \$t3, \$t6 | F D A M R |
| lw       | \$t4, -64(\$t2)  | F D A M R |
| sw       | \$t4, -60(\$t2)  | F D A M R |
| add      | \$t7, \$t8, \$t7 | F D A M R |

- 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, incluindo o «*delayed branch*».
- Tente resolver 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 (considere o pipeline cheio). As soluções mais eficientes (sem *stalls*) serão valorizadas.

|   |   |  |
|---|---|--|
|  | <p align="center"><b>Arquitectura de Computadores</b><br/>         ENGENHARIA INFORMÁTICA<br/>         FACULDADE DE CIÊNCIAS E TECNOLOGIA<br/>         UNIVERSIDADE DE COIMBRA</p> <p align="center"><b>– 2ª Frequência –</b><br/> <b>Parte Prática</b></p> | <p align="center">(a preencher pelo docente)</p> |
|---|---|--|

Nome: \_\_\_\_\_ Número: \_\_\_\_\_

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



- 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 otimização disponíveis, incluindo o «*delayed branch*».
- Tente resolver 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 (considere o pipeline cheio). As soluções mais eficientes (sem *stalls*) serão valorizadas.

Para resolver os dois *stalls* bastaria acrescentar dois *nops* no local, ou então de forma mais otimizada, bastaria trocar por exemplo as instruções indicadas (não interferem na execução do programa):

```

lw    $t1, 0($t2)
add    $t7, $t8, $t7
add    $t0, $t1, $t2
beq    $0,$0, NEXT
lw    $t3, 0($t0)
sub    $t5, $t4, $t3
NEXT:  lw    $t6, 0($t5)
        lw    $t4, -64($t2)
        or    $t1, $t3, $t6
        sw    $t4, -60($t2)

```

Com o pipeline cheio seriam necessários 9 ciclos de relógio para correr este programa (versus 11 no programa original).