# Arquitectura de Computadores

ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

## – 1ª Frequência –

20 de Abril de 2018
Duração: 75 min. + 10 min. de tolerância

**Nome:**_____ **Número:**_____

**Notas Importantes:**

*A fraude denota uma grave falta de ética e constitui um comportamento não admissível num estudante do ensino superior. Não serão admitidas quaisquer tentativas de fraude, levando qualquer tentativa detectada à reprovação imediata, tanto do facilitador como do prevaricador.*

*Durante a prova pode consultar a bibliografia da disciplina (slides, livros, enunciados e materiais de apoio aos trabalhos práticos). No entanto, não é permitido o uso de computadores/máquinas de calcular e a consulta de exercícios previamente resolvidos.*

*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 pergunta errada desconta dois pontos; cada pergunta não respondida vale zero pontos. Um total abaixo de zero, conta como 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | |

1. **Assuma que quer aceder, num programa escrito em Linguagem C, ao conteúdo de uma variável inteira num. Considere a seguinte declaração/instrução int *ptr=&num. Indique qual das seguintes afirmações é VERDADEIRA:**

   **a.** A instrução *printf("%d\n",*ptr)* imprime o valor da variável *num*.
   **b.** A instrução *printf("%d\n",&ptr)* imprime o valor da variável *num*.
   **c.** A instrução *printf("%d\n",&num)* imprime o valor da variável *num*.
   **d.** Não é possível imprimir o valor da variável *num* usando a variável *ptr*.

2. **Considere o seguinte excerto de código Assembly do MIPs em que é carregado em memória o array de inteiros *num*:**

```
        .data
num:  .byte   10,20,30,40,50,60,70,80,90,100
        .text
main:
        la      $a0,num
        addi    $t0,$0,5
```

   **Indique qual das instruções permite substituir o número 70 na tabela pela informação que está no registo $t0.**

   **a.** sb $t0,24($a0)
   **b.** sw $t0,24($a0)
   **c.** lw $t0,6($a0)
   **d.** sb $t0,6($a0)

3. **Considere um processador que possui um CPI real igual a 5 ciclos. Assuma a existência de duas caches, uma para instruções e outra para dados. A hit rate na cache de instruções é de 90% e a miss rate na cache de dados é igual a 20%. A miss penalty é de 15 ciclos de relógio na cache de instruções e de 20 ciclos de relógio na cache de dados. Considere que 25% das instruções também envolvem um acesso à memória de dados. Qual seria o CPI se todos os acessos à memória envolvessem apenas a cache?**

   **a.** 3.5 **b.** 2.5 **c.** 3.25 **d.** 2.75

4. **Considere o seguinte código em Assembly do MIPS relativo a uma *string* armazenada em memória. Indique qual dos seguintes caracteres se encontra no registo** $t1**.**

```
.data
  frase: .asciiz "Bom dia!?"
.text
  la $t0,frase
  lbu $t1,2($t0)
  lbu $t2,5($t0)
  slt $t3,$t1,$t2
  bne $t3,$0,salta
  lbu $t1,0($t0)
salta:
  addi $t1,$t1,1
```

   **a.** "n"
   **b.** "m"
   **c.** "C"
   **d.** Nenhuma das outras opções.

5. **Determine o tempo médio de acesso (average access time) de um sistema em que o acesso à memória principal requer 32 ns e a hit rate na cache é de 80%. Considere que o *speedup* resultante da introdução da *cache* é igual a 4.**

   **a.** 3.2 ns **c.** 6.4 ns
   **b.** 8.4 ns **d.** 16 ns

6. **Considere o código em Assembly do MIPS apresentado à direita que manipula uma tabela de inteiros** `tab[]` **armazenada em memória. Após a execução do código, que valores ficarão armazenados na tabela** `tab[]`**?**

```
       .data
tab: .word   1,2,3,4,5,6,7,8,9,0
       .text
       la     $t0,tab
       li     $t1,10
       li     $t7,2
ciclo:
       beqz   $t1,out
       lw     $t2,0($t0)
       mul    $t2,$t2,$t7
       sw     $t2,0($t0)
       addiu $t0,$t0,8
       subi   $t1,$t1,2
       j ciclo
out:
```

   **a.** `tab = {1,4,3,8,5,12,7,16,9,0};`
   **b.** `tab = {2,4,6,8,10,12,14,16,18,0};`
   **c.** `tab = {1,2,3,4,5,6,7,8,9,0};`
   **d.** `tab = {2,2,6,4,10,6,14,8,18,0};`

7. **Considerando o trecho de programa indicado ao lado, indique qual das afirmações é <u>VERDADEIRA</u>:**

```
int tab[5] = [1,2,3,4,5};

void main(){
  int i,*aux;
  aux=(int*)malloc(5*sizeof(int));

  for(i=0;i<5;i++)
    aux[i]=tab[i];
}
```

   **a.** A variável ***tab*** vai ser armazenada na zona de dados estáticos, a variável ***i*** na pilha e a variável ***aux*** é armazenada no *heap*.
   **b.** As variáveis ***tab*** e ***aux*** vão ser armazenadas no *heap* porque correspondem a tabelas.
   **c.** A variável ***tab*** vai ser armazenada na zona de dados estáticos do programa, a variável ***aux*** e ***i*** na pilha. A variável ***aux*** contém um endereço localizado no *heap*.
   **d.** As variáveis ***tab, i*** e ***aux*** vão ser armazenadas na pilha, sendo que ***tab*** e ***aux*** apontam para zonas de memória no *heap*.

8. **Considere um *datapath* com cinco etapas: 1- "*instruction fetch*"; 2- "*instruction decode*"; 3 – ALU; 4 – "*memory access*"; 5- "*register write*". Relativamente à instrução srl $t1,$t0,4, diga qual das seguintes afirmações é VERDADEIRA?**
   - **a.** A instrução está inactiva na etapa 4 correspondente ao acesso à memória ("*memory access*").
   - **b.** A instrução está inactiva na etapa 3 correspondente à unidade lógica e aritmética (ALU).
   - **c.** A instrução está inactiva na etapa 5 de escrita nos registos ("*register write*").
   - **d.** A instrução está activa em todas as etapas do *datapath*.

9. **Considere o seguinte código em a*ssembly* do MIPS, que pretende implementar o código equivalente ao programa em linguagem C descrita ao lado. Escolha das opções disponíveis aquela que correctamente representa o par de instruções <...> em falta no código MIPS. Assuma que `$a1` e `$a2` contêm os endereços das tabelas `A` e `B`, respectivamente.**

**CÓDIGO ASSEMBLY**

```
      li $t0, 1
      li $t5, 100
  loop:
      lw $t1, 0($a1)
      lw $t2, 4($a2)
      add $t3, $t1, $t2
      <. . .>
      addi $a2,$a2,4
      addi $t0, 1
      bne $t0, $t5, loop
  halt:
      ...
```

**CÓDIGO C**

```
int A[100], B[100];
 for (i=1; i < 100; i++) {
       A[i] = A[i-1] + B[i];
 }
```

   - **a.** `lw $a1, 4($t3)` e `addi $t1,$t1,4`
   - **b.** `sw $t3, 4($a1)` e `addi $a1,$a1,4`
   - **c.** `sw $t3, 0($a1)` e `addi $a1,$a1,4`
   - **d.** `sw $t3, 0($a2)` e `addi $a1,$a1,1`

10. **Considere o seguinte programa em linguagem C em que o 1º elemento da tabela *tab* está armazenado no endereço de memória 0x00007000. Com base nisso, que valores são impressos no ecrã?**

```
#include <stdio.h>

int main(){
      int tab[5]={1,2,3,4,5};
      int *ptr = tab+2;

      printf("%d ", tab[3]);
      printf("%d ", *ptr);
      printf("%d ", *(ptr+2));
      printf("%x ", ptr+1);
      printf("%x \n", tab);
      return 0;
}
```

   - **a.** 4 3 5 0x0000700C 1
   - **b.** 3 5 4 0x00007008 1
   - **c.** 4 3 5 0x0000700C 0x00007000
   - **d.** 4 2 4 0x0000700C 0x00007000

11. **Diga qual a operação aritmética que o excerto de código em *assembly* do MIPS apresentado à direita realiza. Assuma que `a`→`$s0`, `b`→`$s1`, `c`→`$s2`, `d`→`$s3` e `f`→`$s4`.**

```
add   $t0,$s0,$s1
addi  $t1,$0,2
mul   $t2,$s2,$t1
sub   $t3,$t2,$s3
sub   $s4,$t0,$t3
```

   - **a.** `f=(a+b)+(2c-d)`
   - **b.** `f=(a+b)-(c-d)`
   - **c.** `f=(a+b)-(2c-d)`
   - **d.** `f=(a+b)-2(c-d)`

**12. Qual das seguintes afirmações, relativas ao ambiente de desenvolvimento em C para o MIPS que utilizou nas aulas práticas laboratoriais, é <u>VERDADEIRA</u>:**

- **a.** O gdb permite, entre outras opções, correr o programa passo a passo, ver o estado das variáveis, analisar o ponto em que o programa falhou e gerar um novo executável com as correcções efectuadas.
- **b.** O uso da flag –c e -g em simultâneo com o compilador gcc produz um ficheiro objecto com informação adicional para fins de debug.
- **c.** O uso da flag –o com o compilador gcc permite gerar ficheiros objecto.
- **d.** O gcc apenas consegue compilar ficheiros em linguagem C. Para compilar ficheiros em *assembly* terá que ser utilizado um *assembler*.

**13. No trabalho prático 3 utilizaram-se dois *displays* de 7 segmentos do simulador MARS. Para aceder ao *display* da direita bastava escrever um *byte* no endereço 0xFFFF0010. Qual seria o resultado do trecho de programa em *assembly* apresentado à direita?**

```
addi $a0, $0, 0xFFFF0010
addi $t0, $0, 0x77
sb   $t0,0($a0)
```

- **a.** Escreve um 0 no *display*.
- **b.** Escreve um A no *display*.
- **c.** Escreve um 8 no *display*.
- **d.** Escreve um C no *display*.

**14. Qual dos excertos de código em *assembly* do MIPS reproduz fielmente o ciclo abaixo escrito em linguagem C? Assuma que `a`→`$s0`, `b`→`$s1`, `r`→`$s2`.**

```
for(a=10;a>0;a--)
      r=a-b;
```

**(a)**
```
      addiu $s0,$0,10
loop: slti  $t0,$s0,1
      bne   $t0,$0,out
      subu  $s2,$s0,$s1
      addiu $s0,$s0,-1
      j     loop
out:
```

**(b)**
```
      addiu $s0,$0,10
loop: slti  $t0,$s0,1
      beq   $t0,$0,out
      subu  $s2,$s0,$s1
      addiu $s0,$s0,-1
      j     loop
out:
```

**(c)**
```
      addiu $s0,$0,10
loop: slti  $t0,$s0,1
      beq   $t0,$0,out
      subu  $s2,$s0,$s1
      addiu $s0,$s0,1
      j     loop
out:
```

**(d)**
```
      addiu $s0,$0,10
loop: slti  $t0,$s0,1
      bne   $t0,$0,out
      addu  $s2,$s0,$s1
      addiu $s0,$s0,-1
      j     loop
out:
```

**15. Considere uma hierarquia de memória do tipo *2-way set-associative* organizada em *bytes* e com a seguinte estrutura de endereçamento:** offset = 10 bits; set/index = 10 bits; tag = 10 bits**. Qual será a organização dessa hierarquia de memória?**

- **a.** Memória Principal: 1 GBytes; Memória Cache: 2 Mbytes; Tamanho Bloco: 2 KBytes
- **b.** Memória Principal: 1 GBytes; Memória Cache: 1 Mbytes; Tamanho Bloco: 1 KBytes
- **c.** Memória Principal: 2 GBytes; Memória Cache: 2 Mbytes; Tamanho Bloco: 2 KBytes
- **d.** Memória Principal: 1 GBytes; Memória Cache: 2 Mbytes; Tamanho Bloco: 1 KBytes