Nº de ordem: _____

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Mestrado Integrado em Engenharia Informática e Computação
Arquitetura e Organização de Computadores
Teste 2

1º ano
2017-01-16
Duração 1:45
Sem consulta

Nome: _____ **Nº de estudante:** _____

Atenção: Este teste tem 14 questões em 6 páginas, num total de 200 pontos.

Parte I — Questões de Escolha Múltipla

Cada questão tem uma resposta certa. Respostas erradas não descontam.
As respostas às questões de escolha múltipla devem ser assinaladas com × na grelha seguinte.
Apenas as respostas indicadas na grelha são consideradas para efeitos de avaliação.

	Questão									
Opção	1	2	3	4	5	6	7	8	9	10
A					×		×		×	
B		×	×			×				×
C	×							×		
D				×						

Pontos: _____ / 100

[10] 1. O código de uma instrução é `0x0137C825`. Esta instrução utiliza o registo:

A. `$t0` B. `$t2` **C. `$s7`** D. `$s6`

[10] 2. O código-máquina da instrução `sra $t2, $t1, 4` é:

A. `0x00095202` **B. `0x00095103`** C. `0x000A4903` D. `0x000A4A03`

[10] 3. Considere que no CPU estudado o sinal `ALUSrc` é sempre 1 devido a uma anomalia. Não sendo possível executar a instrução `xor $v0, $v0, $v0`, que alternativa a pode substituir levando ao mesmo resultado?

A. `add $v0, $zero, $zero`

B. `addi $v0, $zero, 0`

C. `sub $v0, $v0, $v0`

D. `lw $v0, 0($zero)`

[10] 4. A memória principal usada com um CPU de 2 GHz tem um tempo de acesso de 60 ns. O acesso à memória *cache* demora 0,5 ns. Qual deve ser o valor mínimo da taxa de faltas da memória *cache* para que o tempo médio de acesso a memória seja 10 ciclos?

A. 5 % B. 10 % C. 2,5 % **D. 7,5 %**

[10] 5. Um programa gasta 50 % do tempo a executar cálculos de vírgula flutuante. Qual é o ganho de rapidez *(speedup)* que se poderia obter, se a unidade de vírgula flutuante fosse 5 vezes mais rápida?

A. 5/3 B. 2,5 C. 10/3 D. 2

[10] 6. Considerar o seguinte fragmento de código:

```
li  $t0,0xABCD78EF
li  $t1,0x00002000
sw  $t0,0($t1)
lb  $t3,2($t1)
```

Assumindo que o valor inicial de `$t3` é 0, qual é o seu valor final?

A. `0x000000CD` **B. `0xFFFFFFCD`** C. Acesso ilegal D. `0x00CD0000`

[10] 7. A instrução `lw $t3, 0x80004400($t7)` não é uma instrução primitiva do processador MIPS-32. Qual dos seguintes fragmentos é equivalente a essa instrução?

A.
```
lui  $at, 0x8000
add  $at, $at, $t7
lw   $t3, 0x4400($at)
```

B.
```
and  $at, $at, $zero
addi $at, $t7, 0x4400
lw   $t3, 0x8000($at)
```

C.
```
lui  $at, 0x8000
or   $at, $at, $t7
lw   $t3, 0x4400($at)
```

D.
```
add  $at, $at, $zero
addi $at, $t7, 0x8000
lw   $t3, 0x4400($at)
```

[10] 8. Considerar o seguinte fragmento de código:

```
li   $t0,0xAABBCCDD
srl  $t1,$t0,0x10
sll  $t0,$t0,16
or   $t0, $t0,$t1
```

Qual é o valor final de `$t0`?

A. `0xCCFFFAEF3` B. `0xDDCCAABB` **C. `0xCCDDAABB`** D. `0xAACCBBDD`

[10] 9. Qual das seguintes afirmações sobre uma memória *cache* do tipo *write-back* é verdadeira?

A. Existe a possibilidade de serem feitos 2 acessos a memória principal para apenas um acesso a memória *cache*.

B. O conteúdo da memória principal está sempre atualizado.

C. Pode haver um acesso a memória principal mesmo que o acesso a memória *cache* seja um acerto *(hit)*.

D. Não pode ser usada como *D-cache*.

[10] 10. Considere duas versões do mesmo programa a correrem no mesmo processador. A versão A foi produzida pelo compilador C_A e executa 2×10^{10} instruções; a versão B foi produzida pelo compilador C_B e executa $1{,}25 \times 10^{10}$ instruções. Para a versão A, o valor de CPI_A é de 2. A versão A é 25 % mais rápida que a a versão B. Qual é o valor de CPI da versão B?

A. $CPI_B = 25/16$ **B. $\mathbf{CPI_B = 4}$** C. $CPI_B = 57/16$ D. $CPI_B = 3$

(Continua)

Nome: _____ **Nº de estudante:** _____

Parte II — Questões de Resposta Aberta

Atenção: Responder diretamente no enunciado. **Justificar** todas as respostas.

11. A sub-rotina `sumsel` retorna a soma dos elementos de uma sequência (de N *half-words*) que pertencem ao intervalo $[a; b]$. Os parâmetros da sub-rotina são, por ordem, os seguintes: 1) endereço-base da sequência; 2) número de elementos da sequência; 3) valor de a; 4) valor de b.

[20] (a) Completar a sub-rotina tendo em atenção as convenções relacionadas com o uso de registos.

```
sumsel: xor     $v0, $v0, __$v0__
ciclo:  beq     $a1, $zero, fim          # terminar?
        __lh__  $t1, 0($a0)
        slt     $t2, $t1, $a2            # limite inferior
        bne     $t2, $zero, cont
        slt     $t2, __$a3__, $t1        # limite superior
        __bne__ $t2, $zero, cont
        add     $v0, $v0, __$t1__
cont:   addi    $a0, $a0, __2__
        addi    $a1, $a1, __-1__
        j       ciclo
fim:    jr      __$ra__                  # retornar
```

[10] (b) Para a sequência {-3, 3, 6, 5, 0, -5, 8, 2, -1} e intervalo $[-1; 6]$, determinar quantas instruções são executadas pela sub-rotina `sumsel`.

(Nota: enunciado original tinha uma gralha: $[-1; 6[$ em vez de $[-1; 6]$. O intervalo $[-1; 6]$ é que corresponde ao código *assembley* apresentado. Foram consideradas corretas soluções que consideraram o intervalo $[-1; 6[$.)

- Iterações para elementos da sequência inferiores a a executam 7 instruções.
- Iterações para elementos da sequência superiores a b executam 9 instruções.
- Iterações para elementos da sequência dentro do intervalo executam 10 instruções.

Neste caso, temos N=9: Existem 2 elementos na primeira condição, 1 na segunda e 6 na terceira.
A primeira e última instruções do código são executadas apenas 1 vez cada. A instrução `beq` é executada ainda uma vez após as 9 iterações.
No total: $2 \times 7 + 1 \times 9 + 6 \times 10 + 3 = 86$ instruções executadas. (Nota: Para $[-1; 6[$, o resultado seria 85.)

[10] (c) O modelo do processador usada para a execução da sub-rotina emprega um sinal de relógio com a frequência de 1 GHz. O tempo de execução da sub-rotina com os dados da alínea (b) é de 170 ns. Determinar o valor médio de ciclos por instrução (CPI).

(Nota: Se não resolveu a alínea anterior, assuma que o número de instruções executadas é 100.)

Usando a fórmula para o desempenho: $T_{exec} = N \times CPI \times \frac{1}{F}$, pode determinar-se CPI por $CPI = F \times T_{exec} \times \frac{1}{N}$.
Com $N = 86$, obtém-se: $CPI = 1 \times 10^9 \times 170 \times 10^{-9} \times \frac{1}{86} = \frac{86}{43}$.
(Nota: Com $N = 85$, obtém-se: $CPI = 1 \times 10^9 \times 170 \times 10^{-9} \times \frac{1}{85} = 2$. Com $N = 100$, tem-se $CPI = 1{,}7$.)

12. Sobre uma instrução *assembly* é conhecida a seguinte informação relativa a sinais e componentes que fazem parte da organização do CPU:

    - 'Read register 1' e 'Read register 2' são iguais a, respetivamente, 15 e 2;
    - O código da instrução está na posição de memória com endereço `0x00000038`;
    - O valor presente na entrada 1 do multiplexador controlado por 'PCSrc' é `0x0000004C`;
    - Todos os multiplexadores são úteis para a execução da instrução.

    Determine qual a instrução completa em causa assumindo que se trata de uma instrução envolvendo dados de 32 bits com:

[10] (a) opcode=001000

O valor de opcode corresponde à instrução `addi`, restando identificar os operandos (rt, rs, imm).

Os valores de 'Read register 1' e 'Read register 2' referem-se aos registos `$t7` e `$v0`, respetivamente. Portanto, rs=`$t7` e rt=`$v0`.

O valor imediato é o que está presente na entrada 1 do multiplexador controlado por 'ALUSrc', podendo ser deduzido a partir do valor na entrada 1 do multiplexador controlado por 'PCSrc'. Este valor (`0x0000004C`) resulta da soma de (endereço da instrução + 4)=`0x0000003C` com o valor à saída de 'Shift left 2'. Portanto, o valor à saída de 'Shift left 2' é `0x00000010`=16, concluindo-se que o valor imediato à saída de 'Sign extend' é 4.

Através dos valores deduzidos conclui-se que a instrução é `addi $v0, $t7, 4`.

[10] (b) 'MemToReg'=1

Como todos os multiplexadores são utilizados exclui-se a hipótese de ser uma instrução de salto condicional, para a qual 'MemToReg' pode assumir um valor qualquer. Sendo 'MemToReg'=1 conclui-se ser uma instrução de leitura da memória, ou seja, `lw rt, imm(rs)`.
Através dos valores determinados na alínea anterior conclui-se que a instrução é `lw $v0, 4($t7)`.

[10] 13. Assuma que o caminho crítico das instruções `beq`, `and` e `andi` inclui o banco de registos. A diferença entre as latências de execução de `beq` e `andi` é 10 ps. Todos os multiplexadores têm a mesma latência.

Indique o caminho crítico da instrução `and` e relacione a respetiva latência com a de `andi`.

Nas condições indicadas, os caminhos críticos de `beq` e `andi` são, respetivamente,

beq: *Instruction memory* → *Registers* → *Mux* → *ALU* → *Mux*
`andi`: *Instruction memory* → *Registers* → *ALU* → *Mux*

concluindo-se que a diferença de 10 ps traduz a latência de um multiplexador.

O caminho crítico da instrução `and` é

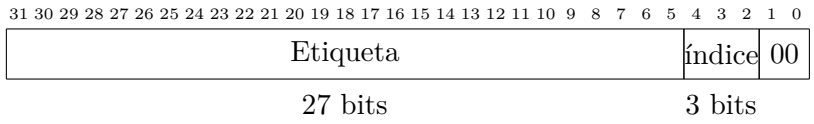
Instruction memory → *Registers* → *Mux* → *ALU* → *Mux*

O caminho crítico das instruções `beq` e `and`, embora não seja o mesmo, apresenta a mesma latência por integrar componentes iguais. Conclui-se portanto que a latência de `and` é superior em 10 ps à de `andi` porque tem um multiplexador a mais.

14. A tabela seguinte apresenta o conteúdo de uma memória *cache* do tipo *write-through* com 8 blocos de 4 bytes usada como *D-cache* num CPU com endereços de 32 bits.

	Conteúdo	Etiqueta	v
0	aa ff cc 33	123456a	0
1	12 34 56 78	7bcd001	1
2	88 b0 3c 2b	7fffd55	1
3	71 ab 3f 6d	7fffd55	1
4	34 ff 13 aa	07f9910	0
5	78 00 9c 23	0000893	1
6	7a 10 9f a3	2900002	1
7	99 43 65 b4	7f01d12	0

[10] (a) Como é decomposto o endereço para acesso à memória *cache*? Justifique.



Offset: 2 bits.
Índice: 3 bits. Como a cache possui 8 blocos são necessários 3 bits para representar os índices de 0 a 7.
Etiqueta: 27 bits. Como temos 32 bits e já gastamos 5, os restantes $(32 - 5 = 27)$ serão para a etiqueta.

[20] (b) Apresente, justificando, o conteúdo da memória cache após a execução do seguinte conjunto de instruções (se em algum caso não for possível conhecer o conteúdo escreva "indeterminado"):

```
1: li $t1,0xfafafafa
2: add $t4, $t3, $t2
3: li $t4, 0x0ff32210
4: lw $t5, 0($t4)
5: sub $t5, $t4, $t1
6: li $t6, 0xffffaaa8
7: sw $t1, 4($t6)
```

	Conteúdo	Etiqueta	v
0	aa ff cc 33	123456a	0
1	12 34 56 78	7bcd001	1
2	88 b0 3c 2b	7fffd55	1
3	fa fa fa fa	7fffd55	1
4	indeterminado	07f9910	1
5	78 00 9c 23	0000893	1
6	7a 10 9f a3	2900002	1
7	99 43 65 b4	7f01d12	0

As únicas instruções que acedem à memória de dados são:

- Instrução 4:
  Endereço `0x0ff32210 -> 00001111111100110010001000 | 100 | 00`
  Etiqueta: `0x07f9910`, bloco 4.
  Apesar de a etiqueta ser igual, tem-se $v = 0$ *(read miss)*. É necessário ler o valor da memória principal e escrevê-lo na memória cache, colocando ainda $v = 1$.
- Instrução 7:
  1º passo: Calcular o endereço: `0xffffaaa8+4=0xffffaaac`.
  2º passo: Obter etiqueta e bloco: `0xffffaaac` -> 11111111111111111010101010 | 011 | 00
  Etiqueta `0x7fffd55`, bloco 3.
  3º passo: Verificar se é para alterar a memória *cache*.
  O bloco 3 tem etiqueta igual e $v = 1$ *(write hit)*. O valor de `$t1` é escrito na memória principal e também na memória *cache*: o conteúdo do bloco 3 é alterado para `0xfafafafa`.