Nome: _____ **Número:** _____

Atenção: Este teste tem 8 questões em 6 páginas, num total de 200 pontos.
Responda diretamente no enunciado. Fundamente todas as respostas.

[10] 1. Determinar a instrução MIPS cujo código-máquina é `0x1269fffd`.

Instrução em binário: `0001 0010 0110 1001 1111 1111 1111 1101`.
A instrução tem `opcode=000100`. Consultando a tabela, determina-se que se trata da instrução `beq`.
Como se trata de uma instrução do tipo I, o resto do código interpreta-se da seguinte forma: `imm=0xffdf` (equivalente a -3 em decimal), `rs=10011` (registo 19, `$s3`) e `rd=01001` (registo 9, `$t1`).
A instrução completa é: `beq $s3, $t1, -3`

2. Considere o seguinte fragmento de código *assembly* MIPS, que usa dois registos `$t0` e `$t1`.

```
xor $t0, $t0, $t1
xor $t1, $t1, $t0
xor $t0, $t0, $t1
```

[10] (a) O conteúdo inicial dos registos é `$t0=0x00000057` e `$t1=0x0000006a`. Determinar o valor final de ambos os registos.

A instrução `xor` calcula o ou-exclusivo (símbolo $\oplus$) dos seus operandos, bit a bit ($0 \oplus 0 = 1 \oplus 1 = 0$ e $1 \oplus 0 = 0 \oplus 1 = 1$). Inicialmente:

`$t0 = 00...0 0101 0111 = 0x00000057`;
`$t1 = 00...0 0110 1010 = 0x0000006a`

Depois da 1ª instrução: `$t0 = 00...0 0011 1101 = 0x0000003d`.
Depois da 2ª instrução: `$t1 = 00...0 0101 0111 = 0x00000057`.
Depois da 2ª instrução: `$t0 = 00...0 0110 1010 = 0x0000006a`.
Os valores finais são: `$t1=0x00000057`, `$t0=0x0000006a`.

[10] (b) Qual é o objetivo do fragmento? Para que valores de `$t0` e/ou `$t1` é que o fragmento não cumpre esse objetivo?

O fragmento troca os valores de dois registo sem usar um registo intermédio adicional.

Este método funciona para quaisquer valores dos operandos. Para ver isso, basta considerar os dois valores usados na alínea (a): Os bits 4-7 de `$t0` e `$t1` (sublinhados na resposta à alínea (a)) cobrem as quatro combinações possíveis de valores iniciais; em todos os casos, o resultado final é o pretendido, conforme foi calculado nessa alínea.

Quando os registos têm valores iguais, a troca também é efetuada, embora não resulte em alterações observáveis.

3. A sub-rotina `nocorr` conta quantas vezes é que um dado valor (uma palavra de 32 bits) ocorre numa sequência de palavras. Os argumentos da sub-rotina são, por ordem, os seguintes:
   1. endereço-base da sequência;
   2. número de elementos da sequência;
   3. o valor cujas ocorrências devem ser contadas.

[20] (a) Preencher o código da sub-rotina `nocorr` de forma a que a funcionalidade desejada seja implementada corretamente.

```
nocorr:   li    $v0, 0
loop:     beq   $a1, $zero, fim
          lw    $t0, 0($a0)
          bne   $t0, $a2, nc
          addi  $v0, $v0, 1
nc:       addi  $a0, $a0, 4
          addi  $a1, $a1, -1
          j     loop
fim:      jr    $ra
```

[10] (b) Indicar as alterações a efetuar no código para tratar sequências de meias palavras *(halfwords)* em vez de sequências de palavras.

1. Usar `lh` em vez de `lw` [na linha 3].
2. O registo `$a0` deve ser incrementado de 2 (em vez de 4) [na linha 6].

[10] (c) A sub-rotina `nocorr` é invocada para contar o número de vezes que o valor `10` aparece na sequência `{10,11,10,12,10,13,10,14,15}`.
Determinar o número de instruções executadas neste caso.

As instruções da primeira e última linhas são executadas apenas uma vez.
O ciclo (linhas 2-8) é executado 9 vezes, porque a sequência tem 9 valores.
Quando o elemento corrente da sequência é igual a 10, o ciclo executa 7 instruções. Isso acontece 4 vezes.
Quando o elemento corrente da sequência é diferente a 10, o ciclo executa 6 instruções (a instrução `addi $v0 , $v0, 1` não é executada). Isso acontece 5 vezes.
Quando a sequência chega ao fim, a instrução `beq` é ainda executada mais uma vez (é a instrução que deteta o fim dos elementos).
Ao todo, o número de instruções executadas é $1+4\times 7+5\times 6+1+1=3+28+30=61$.

Nome: _____ **Número:** _____

[20] 4. Considere o CPU MIPS simplificado apresentado na folha anexa. O conteúdo do registo i ($0 \leq i \leq 31$) é igual a $i \times 10$. Assumir que o CPU está a executar a instrução `addi $s3, $t2, 20`.

Indicar o valor dos sinais e portos seguintes [BR : Banco de registos]:

- entrada `Read register 1` do BR: 10
- entrada `Read register 2` do BR: 19
- saída `Read data 1` do BR: 100
- saída `Read data 2` do BR: 190
- entrada `Write Register` do BR: 19
- entrada `Write data` do BR: 120
- saída `Zero` da ALU: 0
- sinal de controlo `MemtoReg`: 0
- sinal de controlo `MemRead`: 0
- sinal de controlo `ALUSrc`: 1

[10] 5. Considerar o CPU MIPS apresentado na folha anexa. Determinar o tipo de instrução a ser executada quando:

i) `RegDst=1` e `RegWrite=1`;
ii) `MemRead` $\oplus$ `MemWrite` $= 1$.

i) Esta instrução atualiza um registo, porque tem `RegWrite=1`. O número do registo a atualizar é determinado pelos bits [15:11] da instrução, porque `RegDst=1`. Logo, trata-se de uma instrução do tipo R.

ii) A condição indicada implica que `MemRead=1` e `MemWrite=0` (leitura de memória, instrução `lw`) ou que `MemRead=0` e `MemWrite=1` (escrita em memória, instrução `sw`). Em qualquer dos casos, trata-se de uma instrução do tipo I.

6. A latência de vários componentes usados na implementação do CPU é a seguinte:

I-Mem	Add	Mux	ALU	Regs	D-Mem	Control	
400	150	45	110	220	?	90	(ps)

A latência dos restantes componentes é nula. Para esta implementação, a instrução `lw` é a mais longa.

[20] (a) Determinar a latência máxima do componente `D-MEM` que ainda permite que o CPU seja usado com um sinal de relógio de 1 GHz.

A instrução mais longa é *lw*. Neste caso, o caminho crítico é:

I-Mem $\rightarrow$ Regs $\rightarrow$ ALU $\rightarrow$ D-Mem $\rightarrow$ Mux.

Para satisfazer as condições do enunciado, o tempo desse caminho deve ser menor ou igual a 1 ns=1000 ps. Fazendo os cálculos em picossegundos:

$$\begin{aligned} 400 + 220 + 110 + x + 45 &\leq 1000 \\ 775 + x &\leq 1000 \\ x &\leq 225 \end{aligned}$$

[10] (b) Assumir agora que a latência de `D-MEM` é 300 ps. Está em planeamento uma nova versão do CPU usando uma tecnologia mais recente. Nessa nova tecnologia, todos os componentes, exceto as memórias, têm uma latência que é 80 % da original. A latência dos componentes `I-MEM` e `D-MEM` mantém-se.
Determinar quanto mais rápida é a nova versão.

Na versão original, o tempo de propagação associado ao caminho crítico de `lw` é:

$$T_{\text{original}} = 400 + 220 + 110 + 300 + 45 = 1075 = 700 + 375$$

Dos 1075 ps, 700 ps correspondem à latência das memórias. Usando a nova tecnologia, o caminho crítico passa a ser ($80\,\% = 4/5$):

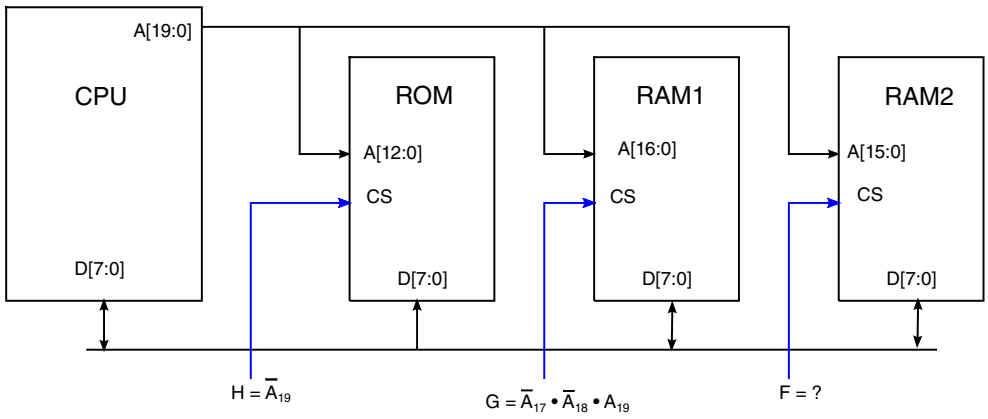
$$T_{\text{novo}} = 700 + 375 \times \frac{4}{5} = 700 + 4 \times 75 = 700 + 300 = 1000$$

O tempo de execução de um programa é $T_{\text{exec}} = N \times \text{CPI} \times T_p$. Neste caso, N e CPI são iguais. Logo, o desempenho relativo da nova versão é:

$$\frac{T_{\text{original}}}{T_{\text{novo}}} = \frac{1075}{1000} = 1.075$$

A nova versão é 1.075 vezes mais rápida (trata-se de um melhoramento insignificante).

7. Considerar o sistema de memória da figura.



[10] (a) Determinar a capacidade dos dispositivos de memória.

ROM: 13 bits de endereço → $2^{13} = 2^3$ Ki; (Ki = $2^{10} = 1024$)
como há 1 byte de dados por posição → capacidade total = 8 KiB.

RAM1: 17 bits de endereço → $2^{17} = 2^7$ Ki;
como há 1 byte de dados por posição → capacidade total = 128 KiB.

RAM2: 16 bits de endereço → $2^{16} = 2^6$ Ki;
como há 1 byte de dados por posição → capacidade total = 64 KiB.

[10] (b) O projetista pretende mapear de forma única a memória `RAM2` numa zona de memória que começa em `0xA0000`. Trata-se de uma escolha válida? Determinar a função de descodificação F correspondente à zona pretendida (seja esta válida ou não).

O endereço inicial é: `0xA0000 = 1010 0000 0000 0000 0000`. Como a memória RAM2 tem 16 linhas de endereço, a gama correspondente tem o formato `1010 xxxx xxxx xxxx xxxx`, i.e., as condições para seleção da memória RAM2 são:

$$A_{19} = 1,\ A_{18} = 0,\ A_{17} = 1,\ A_{16} = 0.$$

Para estes valores, temos sempre $G = H = 0$, pelo que a gama de endereços é válida (i.e., não coincide, nem mesmo parcialmente, com nenhuma das gamas dos outros circuitos de memória).

Como se pede descodificação total, a função F é:

$$F = A_{19} \cdot \overline{A_{18}} \cdot A_{17} \cdot \overline{A_{16}}$$

[10] (c) Determinar que posição de que circuito (`ROM`, `RAM1`, `RAM2` ou nenhum) é acedida quando o CPU usa o endereço `0x12345` para efetuar uma operação de leitura.

Em binário, o endereço indicado é `0001 0010 0011 0100 0101`. Como $A_{19} = 0$, este endereço é mapeado na memória `ROM`.

Os 13 bits menos significativos (sublinhados) são usados para selecionar a posição na memória `ROM`. Neste caso, o seu valor é `0 0011 0100 0101` = `0x0345` (em decimal: 837).

8. Considerar uma memória *cache write-through* de mapeamento direto com 64 blocos e 1 palavra por bloco. O CPU tem endereços de 24 bits.

[20] (a) Determinar a quantidade total de memória (em bytes) existente na *cache*.

Dados: $64 \times 4 = 256$ bytes.

Validade: 64 bits = 8 bytes.

Como *cache* tem $2^6 = 64$ blocos, uma etiqueta tem: $24 - 2 - 6 = 16$ bits (2 bytes).

Todas as etiquetas: $16\,\text{bits} \times 64 = 2\,\text{bytes} \times 64 = 128\,\text{bytes}$.

Total: $256 + 8 + 128 = 392$ bytes.

[10] (b) Determinar a taxa de acertos obtida quando o CPU acede repetidamente apenas à seguinte sequência de endereços: `0x012320,0x0ABC20,0x012320,0x0ABC20,...`?

O endereço `0x012320` deve ser decomposto como `0000 0001 0010 0011 | 0010 00|00`. Este endereço é mapeado no bloco dado pelo índice, que é `001000`, i.e., 8_{10}.

O endereço `0x0ABC20` deve ser decomposto como `0000 1010 1011 1100 | 0010 00|00`. Este endereço também é mapeado no bloco dado pelo índice, que é `001000`, i.e., 8_{10}.

O primeiro acesso guarda o valor obtido de memória no bloco 8, mas o segundo acesso substitui esse valor (falta). Por isso, o terceiro acesso também leva a uma substituição (falta). Este processo repete-se para todos os acessos subsequentes.

Logo, cada acesso resulta numa falta: a taxa de acertos *(hit rate)* é de 0 %.

[10] (c) Em geral, a taxa de falhas da memória *cache* é 5 %. Assumir que:

- CPU usa uma frequência $F = 2\,\text{GHz}$;
- um acesso a memória principal demora 50 ns;
- em média, 30 % de todas as instruções fazem um acesso a memória de dados.

Determinar o número médio de ciclos de protelamento causados por acessos a memória.

Pretende-se determinar $C_{\text{prot}} = n_a \times p_f \times t_f$, com

- número médio de acessos por instrução: $n_a = 1 + 0.3 = 1.3$;
- penalidade de acesso, em ciclos: $p_f = 50/0.5 = 100$ (um ciclo de relógio = 0.5 ps);
- taxa de faltas: $t_f = 0.05$.

$$C_{\text{prot}} = 1.3 \times 100 \times 0.05 = 1.3 \times 5 = 6.5 \text{ ciclos por instrução.}$$

Fim

Questão	1	2	3	4	5	6	7	8	Total
Pontos	10	20	40	20	10	30	30	40	200
Nota									