

### Grupo 1

1. Considere um sistema baseado numa arquitetura em que o respetivo ISA especifica uma organização de memória do tipo *byte-addressable*. Se uma variável de 64 bits, usada num dado programa, se encontrar armazenada em memória a partir do endereço `0x25A1C`, pode concluir-se que a dimensão mínima do espaço de endereçamento dessa arquitetura é:

a.  $2^6$   
b.  $2^{18}$   
c.  $2^{32}$   
d.  $2^{16}$

2. Considere uma arquitetura em que o respetivo ISA especifica uma organização de memória do tipo *word-addressable*, em que a dimensão da *word* é 32 bits. Sendo o espaço de endereçamento do processador 24 bits, a máxima memória que este sistema pode acomodar é:

a.  $2^{24}$  bytes  
b.  $2^{32}$  bytes  
c.  $2^{26}$  bytes  
d.  $2^{24}$  bits

Exatidão 100%

3. Uma arquitetura hipotética baseada num modelo *single cycle* em que os operandos das instruções aritméticas e lógicas podem residir em registos internos ou na memória externa, pode ser classificada como:

a. uma arquitetura Harvard do tipo "Register-Memory"  
b. uma arquitetura Harvard do tipo "Load-store"  
c. uma arquitetura Von Neumann do tipo "Register-Memory"  
d. uma arquitetura Von Neumann do tipo "Load-store"

4. Numa operação de adição de 2 quantidades codificadas em complemento para 2, se o *carry out* do resultado é "0":
- pode concluir-se que o resultado é válido, não sendo necessário fazer qualquer outra verificação
  - pode concluir-se que o resultado não é válido
  - não é possível, por si só, tirar conclusões quanto à validade do resultado
  - deve ser calculado o complemento para 2 do valor obtido para se ter o resultado correto

Programa de Computadores I

16-01-2020

5. Considere o código máquina `0x0C0000D4` correspondente a uma instrução *jal* do MIPS armazenada no endereço de memória `0x0000A034`. Após a execução dessa instrução:

a. `$ra=0x0000A038`    `PC=0x000001A8`  
b. `$ra=0x0000A034`    `PC=0x000000D4`  
c. `$ra=0x0000A038`    `PC=0x00000035`  
d. `$ra=0x0000A038`    `PC=0x00000350`

6. Nas instruções de salto condicional da arquitetura MIPS, para a obtenção do endereço alvo é usado endereçamento:
- indireto por registo com deslocamento
  - imediato
  - relativo ao PC com deslocamento
  - pseudo-direto

7. Considere o seguinte segmento de código *assembly*:

```
lui    $4, 0x36AC
addi   $4, $4, 0x6F80
sw     $4, 4($4)
lb     $4, 6($4)
```

No final da execução do código anterior num MIPS *little endian*, o valor de \$4 é:

- a. 0x000036AC      b. 0xFFFFF80      c. 0x0000006F      d. 0xFFFFFAC

8. Na arquitetura MIPS, uma instrução de *branch* pode saltar para qualquer endereço múltiplo de 4:

- a. na gama  $[-2^{17}, 2^{17} - 1]$ , relativamente ao endereço da instrução seguinte  
b. na gama  $[-2^{15}, 2^{15} - 1]$ , relativamente ao endereço da instrução de *branch*  
c. num segmento de memória contíguo de  $2^{28}$  endereços  
d. na gama  $[-2^{15}, 2^{15} - 1]$ , relativamente ao endereço da instrução seguinte

9. Considere que \$2=0xFFFFFFFF e \$3=0x00000003. O conteúdo do registo LO após a execução da instrução *mult \$2, \$3* é:

- a. 0x00000003  
b. 0x00000006  
c. 0xFFFFFFFF  
d. 0xFFFFF8FA

10. Considere que \$2=0xFFFFFFFF e \$3=0x00000003. O conteúdo do registo HI após a execução da instrução *div \$2, \$3* é:

- a. 0x00000002  
b. 0xFFFFFFFF  
c. 0xFFFFFFFF  
d. 0x00000003

11. A instrução virtual "*bge \$5, 0xA3, target*" da arquitetura MIPS decompõe-se na seguinte sequência ordenada de instruções nativas:

- |                               |  |
|-------------------------------|--|
| a. <i>slti \$1, \$5, 0xA3</i> | <i>beq \$1, \$0, target</i>                        |
| b. <i>slti \$1, \$5, 0xA3</i> | <i>bne \$1, \$0, target</i>                        |
| c. <i>addi \$1, \$0, 0xA3</i> | <i>slt \$1, \$1, \$5      beq \$1, \$0, target</i> |
| d. <i>addi \$1, \$0, 0xA3</i> | <i>slt \$1, \$1, \$5      bne \$1, \$0, target</i> |

12. Os endereços mínimo e máximo para os quais uma instrução *jr* da arquitetura MIPS, presente no endereço 0x3843F0EC, pode saltar são, respetivamente:

- a. 0x38430000, 0x3843FFFC  
b. 0x30000000, 0x3FFFFFFC  
c. 0x0843F0EC, 0xF843F0EC  
d. 0x00000000, 0xFFFFF8FC

$$130_{10} + 128_{10} + 2^7_{10} = 286_{10}$$

$$\frac{OP}{8} \mid \frac{rs}{3} \mid \frac{rt}{3} \mid \frac{im}{16}$$

$$2^{m-1} = 32K$$

$$2^{m-1} = 2^{15} \Leftrightarrow m = 16$$

13. Os processadores da arquitetura hipotética A20 implementam um total de 130 instruções, todas codificadas com 30 bits. Um dos formatos de codificação tem 4 campos: opcode, dois campos para identificar registros internos e um campo para codificar valores imediatos na gama [-32768, 32767]. Pode então concluir-se que o número de registros internos dessa arquitetura é:

- a. 16
- b. 8
- c. 32
- d. 4

$$2^3 = 8$$

14. Suponha que se pretende negar os 8 bits menos significativos do registo \$3 e colocar a zero os 24 bits restantes. Para obter esse resultado pode ser usada a seguinte sequência de instruções:

- a. `addi $1,$0,0x00FF`      `nor $3,$3,$1`
- b. `addi $1,$0,0x00FF`      `xor $3,$3,$1`
- c. `addi $1,$0,0x00FF`      `and $3,$3,$1`
- d. `addi $1,$0,0xFF00`      `nor $3,$3,$1`

$$\$1 = 0xFFFFFFFF$$

$$0xFF \neq 0xFF00$$

nor		
0	0	1
0	1	0
1	0	0
1	1	0

15. Considere que o segmento de dados de um programa contém as seguintes diretivas:

L1: .word 0, 1, 2      0:100    1:104    2:108  
 L2: .ascii "NATAL"    10C → +6 → 112  
 .align 3      → 112 (110 era o anterior)  
 L3: .space 20      → 118

- Se o endereço correspondente ao label L1 for 0x00000100, o endereço a que corresponde o label L3 é:
- a. 0x00000118
  - b. 0x00000114
  - c. 0x00000108
  - d. 0x00000110

16. Em linguagem C, o código que permite obter a soma dos elementos do array "a" é:

- a. `int a[]={1,5};`  
`int s;`  
`int *p;`  
  
`p=a[0];`  
`s=*p + *(p+1);`
- b. `int a[]={1,5};`  
`int s;`  
`int *p;`  
  
`p=a;`  
`s=*p + *(p+4);`
- c. `int a[]={1,5};`  
`int s;`  
`int *p;`  
  
`p=a;`  
`s=*p + *(p+1);`
- d. `int a[]={1,5};`  
`int s;`  
`int *p;`  
  
`p=&a;`  
`s=*p + *(p+4);`

17. No formato de vírgula flutuante IEEE 754, precisão dupla, o expoente da quantidade a representar é codificado em:

- a. complemento para 2
- b. excesso de 1023
- c. excesso de 127
- d. sinal e módulo

18. No formato de vírgula flutuante IEEE 754, precisão simples, com mantissa normalizada, a menor quantidade positiva representável é codificada como:

- a. 0x00C00000
- b. 0x00000001
- c. 0x80000001
- d. 0x00800000

$$0 \times 00000001$$



19. Numa implementação *single cycle* da arquitetura MIPS, o valor corrente do registo PC representa:
- o endereço de memória da instrução que está em execução
  - o endereço de memória da instrução que vai ser executada no ciclo de relógio seguinte
  - o código-máquina da instrução que está em execução
  - o endereço de memória onde vai ser escrito o código máquina da instrução que está em execução
20. Na implementação *single cycle* de uma arquitetura usam-se 2 memórias com o objetivo de:
- separar os segmentos de dados e de código do programa
  - melhorar o desempenho da arquitetura, ao permitir a duplicação da frequência de trabalho
  - permitir o acesso à memória de dados e à memória de instruções no mesmo ciclo de relógio
  - permitir que o programa em execução possa alterar os valores dos códigos-máquina das instruções
21. Na implementação *multi-cycle* da arquitetura MIPS, na execução da instrução **J**, a unidade de controlo:
- ativa o sinal **PCWrite** no primeiro e no terceiro ciclo de relógio
  - ativa o sinal **MemWrite** no primeiro ciclo de relógio e o **PCWrite** no terceiro ciclo de relógio
  - ativa o sinal **PCWriteCond** no primeiro ciclo de relógio e o **MemRead** no terceiro ciclo de relógio
  - ativa o sinal **PCWrite** no primeiro ciclo de relógio e o **RegWrite** no terceiro ciclo de relógio
22. O valor disponível na saída do registo **AluOut** no quarto ciclo de relógio da execução de uma instrução **LW**, numa arquitetura MIPS *multi-cycle* é (**offset32** é o *offset* da instrução estendido com sinal para 32 bits):
- $PC + 4$
  - $[rs] + \text{offset32}$ , em que **[rs]** é o conteúdo do registo codificado no campo **rs** da instrução
  - $[rt] + \text{offset32}$ , em que **[rt]** é o conteúdo do registo codificado no campo **rt** da instrução
  - $PC + (\text{offset32} \ll 2)$
23. Numa implementação *pipeline* da arquitetura MIPS, os sinais de controlo fundamentais são gerados:
- no estágio **IF** por um circuito lógico combinatório e propagados para os estágios seguintes a cada transição ativa do sinal de relógio
  - no estágio **ID** por um circuito lógico sequencial, com um número de estados igual ao número de estágios do *pipeline*; os sinais de controlo são função do estado atual da máquina de estados
  - no estágio **EX** por um circuito lógico combinatório e propagados para os estágios seguintes e anteriores, a cada transição ativa do sinal de relógio
  - no estágio **ID** por um circuito lógico combinatório e propagados para os estágios seguintes a cada transição ativa do sinal de relógio
24. Numa arquitetura MIPS *pipelined* a técnica de *forwarding/bypassing* consiste:
- na resolução de *hazards* estruturais
  - na utilização, como operando, de um resultado produzido por outra instrução que se encontra numa etapa mais avançada do *pipeline*
  - na resolução de *hazards* de controlo
  - na escrita antecipada do resultado de uma instrução no *Register File*

25. A técnica *delayed-branch* num processador *pipelined* consiste:

- na execução da instrução que se encontra a seguir a um *branch* independentemente do resultado *taken / not taken* do *branch*
- na paragem parcial do *pipeline* atrasando a execução da instrução a seguir a um *branch* até ser conhecido o resultado da instrução de *branch*
- no atraso de 1 ciclo de relógio na escrita do *Register File*, nas instruções de *branch*
- no atraso de 1 ciclo de relógio na execução da instrução de *branch*

**Pergunta bónus (não pode usar dupla)**

26. Suponha que se pretende implementar um sistema baseado na arquitetura MIPS *multi-cycle*, dotado de uma memória de 1 kByte organizada em *words* de 32 bits. Os bits do barramento de endereços do CPU que devem ser usados na ligação à memória são:

- A<sub>4</sub> a A<sub>0</sub>
- A<sub>7</sub> a A<sub>2</sub>
- A<sub>9</sub> a A<sub>2</sub>
- A<sub>9</sub> a A<sub>0</sub>

$$\text{bonus} = \left(1 - \frac{\text{respostas erradas}}{\text{respostas certas}}\right) \cdot 0.5 \text{ (totais, incluindo duplas)}$$

$$= \min(20, \text{nota}_{\text{exame}} + \text{bonus}) \text{ (se bonus} > 0)$$

### Grupo II

27. Numa implementação *single cycle* da arquitetura MIPS, a frequência máxima de operação, assumindo que os elementos operativos apresentam os atrasos de propagação a seguir indicados, é, aproximadamente:

Memórias externas: Leitura - 3 ns, Escrita - 2 ns; Register File: Leitura - 2 ns, Escrita - 1 ns;

Unidade de Controlo: 1 ns; ALU (qualquer operação): 1 ns; Somadores: 1 ns; Outros: 0 ns.

- 90 MHz
- 100 MHz
- 125 MHz
- 333 MHz

$$3 + 2 + 1 + 3 + 1 = 10$$

$$\frac{1}{10} = 0,1 \text{ GHz} = 100 \text{ MHz}$$

28. Suponha que, no *datapath* da Figura 2, imediatamente antes da ocorrência da transição ativa do relógio, os sinais de controlo *RegDst*, *Load* e *MemtoReg* apresentam os valores lógicos '0', '1' e '1', respetivamente. Pode então concluir-se que:

- está em execução uma instrução *SW* na fase *MEM*
- está em execução uma instrução *LW* na fase *MEM*
- não é possível identificar a instrução que está em execução
- está em execução uma instrução *LW* na fase *WB*

29. Pretende-se incluir no *datapath* da Figura 2 suporte para a execução da instrução *LUI \$reg, imm*. Para além de adaptar a unidade de controlo, é necessário:

- acrescentar uma entrada ao *multiplexer* M3 e ligar os 16 bits mais significativos aos 16 bits menos significativos do código máquina da instrução; os restantes 16 bits são colocados a zero
- acrescentar uma entrada ao *multiplexer* M3 e ligar os 16 bits mais significativos aos 16 bits mais significativos do código máquina da instrução; os restantes 16 bits são colocados a zero
- acrescentar uma entrada ao *multiplexer* M1 e ligar os 16 bits mais significativos aos 16 bits menos significativos do código máquina da instrução; os restantes 16 bits são colocados a zero
- acrescentar uma entrada ao *multiplexer* M3 e ligá-la à saída do bloco *sign extend*



30. O trecho de código ao lado, a executar sobre uma implementação pipelined da arquitetura MIPS, apresenta os seguintes hazards:

- um hazard estrutural na primeira instrução e hazards de dados na primeira, terceira e quinta instruções que podem ser resolvidos com forwarding para EX
- um hazard de controlo na quinta instrução e hazards de dados na terceira, quarta e quinta instruções que podem ser resolvidos com forwarding para EX
- um hazard de controlo na quinta instrução, um hazard de dados na terceira instrução que pode ser resolvido com forwarding para EX e um hazard de dados na quinta instrução que não pode ser resolvido com forwarding para EX
- um hazard de controlo na quinta instrução e hazards de dados na terceira e quarta instruções que podem ser resolvidos com forwarding para EX e um hazard de dados na quinta instrução que pode ser resolvido com forwarding para ID

```

Z: lw $6, 0($2) #1
   addi $2, $2, 4 #2
   add $5, $5, $6 #3
   slti $1, $6, 23 #4
   bne $1, $5, Z #5
  
```

lw	addi	add
IF		
ID	IF	
EX	ID	IF
MEM	EX	ID
WB	MEM	EX
	WB	M
		WB

$$\begin{array}{r}
 0.40625 \\
 \times 2 \\
 \hline
 0.81250 \\
 \times 2 \\
 \hline
 1.62500 \\
 \times 2 \\
 \hline
 1.25000
 \end{array}$$

$$-0.40625$$

31. A representação da quantidade  $-40.625_{10} \times 10^{-2}$  no formato IEEE 754 precisão simples, é:

- 0xBEE80000
- 0xBED00000
- 0xFF500000
- 0xC0D00000

$$\begin{aligned}
 & b = 0,01101 \times 2^0 \quad \text{int} = 0 \\
 & 1.101 \times 2^{-2} \quad R: \underbrace{1}_{B} \underbrace{01111101}_{E} \underbrace{101000}_{D} \underbrace{000}_{0} \\
 & E = 127 - 2 = 125 = 01111101
 \end{aligned}$$

$$\begin{array}{r}
 \times 2 \\
 \hline
 0.50000 \\
 \times 2 \\
 \hline
 1.00000
 \end{array}$$

32. Com \$f2=0x40DA0000 e \$f6=0x3E000000, o resultado da instrução "div.s \$f4,\$f2,\$f6" é:

- \$f4=0x426D0000
- \$f4=0x3F5A0000
- \$f4=0x02DA0000
- \$f4=0x425A0000

$$\begin{aligned}
 f2: & 0100 \ 0000 \ 1101 \ 1010 \ 0000 \\
 f6: & 0011 \ 1110 \ 0000 \dots\dots\dots
 \end{aligned}$$

33. Para implementação num microcontrolador, pretende-se definir um novo formato de vírgula flutuante de 20 bits (baseado no formato IEEE 754), em que se requer uma precisão de 4 casas decimais, maximizando a gama de representação (note que  $\log_{10}(2) \approx 0.30$ ). Nesse formato, o expoente será codificado em:

- excesso de 15
- excesso de 31
- excesso de 63
- excesso de 127

LD: \$5 = 7A34 → 38  
 L1: \$3 = 0B506C9A  
 add: \$6 += \$3

and: \$3 = \$3 + 81  
 → \$3 = 81? → L1  
 → add: \$5 += 4

and: \$6 = \$6 and \$2

and  
 00 | 0  
 01 | 0  
 10 | 0  
 11 | 1

Nas questões deste grupo, tome como referência as tabelas a seguir apresentadas. Admita que o valor presente no registro PC corresponde ao endereço da primeira instrução do programa. Considere ainda a implementação pipelined da arquitetura MIPS que estudou nas aulas, com *delayed-branch* e *forwarding* para EX e para ID.

Address	Value	CPU	Figure 1
0x7A44	0x09ABC869	\$2: 0x00000081	L0: addi \$5, \$0, 0x7A34
0x7A40	0x012347CD	\$6: 0x00000000	L1: lw \$3, 0(\$5)
0x7A3C	0x03D51EAE	...	add \$6, \$6, \$3
0x7A38	0x0F193AB8	PC: 0x00006024	and \$3, \$3, \$2
0x7A34	0x0B506C9A		beq \$3, \$2, L1
0x7A30	0x03C12975		addi \$5, \$5, 4
			and \$6, \$6, \$2
			L2: sw \$6, 0(\$5)

34. A execução completa do trecho de código fornecido, desde o *instruction fetch* da instrução referenciada pelo label L0 até à conclusão da instrução referenciada pelo label L2, demora:
- 28 ciclos de relógio
  - 32 ciclos de relógio
  - 35 ciclos de relógio
  - 40 ciclos de relógio

35. Quando a instrução "add \$6, \$6, \$3" está na sua fase WB de execução, o valor do registro PC é:
- 0x0000603C
  - 0x00006038
  - 0x00006034
  - 0x00006030

36. Admita que no instante zero, correspondente a uma transição ativa do sinal de relógio, vai iniciar-se o *instruction fetch* da primeira instrução. O valor à saída da ALU na conclusão do décimo ciclo de relógio, contado a partir do instante zero, é:
- 0x0B506C9A
  - 0x00000080
  - 0xFFFFFFFF
  - 0x00007A38