

1. Verifique, para todas as questões, qual a resposta correcta e assinale com uma cruz a sua escolha na tabela ao lado. Por cada resposta incorrecta será descontada, à cotação global, 1/3 da cotação da respectiva pergunta.
2. Durante a realização do teste não é permitida a permanência na sala de calculadoras, telemóveis ou outros dispositivos electrónicos.

Notas Importantes!

	a	b	c	d
1	X			
2		X		
3		X		
4	X			
5		X		
6		X		
7	X			
8		X		
9		X		
10			X	
11		X		
12		X		
13	X			
14			X	
15			X	
16	X			
	a	b	c	d
17				X
18			X	
19			X	
20				
21	X			
22		X		
23			X	
24			X	
	a	b	c	d
25				X
26			X	
27			X	
28				

11/5

3/4

2/1

Grupo I

- ✓ 1. Uma arquitectura do tipo *Harvard* é caracterizada por:
- a. ter segmentos de memória independentes para dados e para código.
 - b. ter dois barramentos de dados e um barramento de endereços.
 - c. partilhar a mesma memória entre dados e instruções.
 - d. permitir o acesso a instruções e dados no mesmo ciclo de relógio.
- ✓ 2. Quando um endereço se obtém da adição do conteúdo de um registo com um *offset* constante:
- a. diz-se que estamos perante um endereçamento imediato.
 - b. diz-se que estamos perante um endereçamento directo a registo com *offset*.
 - c. diz-se que estamos perante um endereçamento indireto a registo com deslocamento.
 - d. diz-se que estamos perante um endereçamento indireto relativo a PC.
- ✓ 3. Na convenção adoptada pela arquitectura MIPS, a realização de uma operação de *pop* da *stack* do valor do registo **\$ra** é realizada pela seguinte sequência de instruções:
- a. **addu \$sp,\$sp,4** seguida de **lw \$ra,0(\$sp)**.
 - b. **lw \$ra,0(\$sp)** seguida de **addu \$sp,\$sp,4**.
 - c. **lw \$ra,0(\$sp)** seguida de **subu \$sp,\$sp,4**.
 - d. **subu \$sp,\$sp,4** seguida de **lw \$ra,0(\$sp)**.
- ✓ 4. A detecção de *overflow* numa operação de adição de números com sinal faz-se através:
- a. do “ou” exclusivo entre o *carry in* e o *carry out* da célula de 1 bit mais significativa.
 - b. da avaliação do bit mais significativo do resultado.
 - c. do “ou” exclusivo entre os 2 bits mais significativos do resultado.
 - d. da avaliação do *carry out* do bit mais significativo do resultado.
- ✓ 5. Considerando que o código ASCII do carácter '0' é 0x30 e que os valores das três *words* armazenadas em memória a partir do endereço 0x10010000 são 0x30313200, 0x33343536 e 0x37380039, num computador MIPS *little endian* a string ASCII armazenada a partir do endereço 0x10010001 é:
- a. "21065439".
 - b. "65439".
 - c. "12".
 - d. "345678".
- ✓ 6. Considerando que **\$t0=-4** e **\$t1=5**, o resultado da instrução **mult \$t0,\$t1** é:
- a. HI=0x80000000, LO=0x000000EC.
 - b. HI=0xFFFFFFFF, LO=0xFFFFFEC.
 - c. HI=0xFFFFFFFEC, LO=0xFFFFFFFF.
 - d. HI=0x00000000, LO=0xFFFFFFFEC.
- ✓ 7. A decomposição numa sequência de adições e subtrações, de acordo com o algoritmo de *Booth*, da quantidade binária $010110_{(2)}$: $\begin{array}{r} S \ 4 \ 3 \ 2 \ 1 \\ + - + \emptyset - \\ \hline 1 \ -2^1 + 2^3 - 2^4 + 2^5 + - 2^6 \end{array}$ $\begin{array}{r} + 2^1 - 2^3 + 2^4 - 2^5 \\ + 2^0 + 2^2 \end{array}$
- a. $-2^1 + 2^3 - 2^4 + 2^5 + - 2^6$.
 - b. $+ 2^0 + 2^2$.
 - c. $+ 2^1 - 2^3 + 2^4 - 2^5$.
 - d. $-2^0 + 2^2$.
- ✓ 8. Os endereços mínimo e máximo para os quais uma instrução de salto incondicional (“j”) da arquitectura MIPS, presente no endereço 0x0043FFFC pode saltar são:
- a. 0x0041FFFC, 0x0045FFF8.
 - b. 0x00420000, 0x0045FFFC.
 - c. 0x00000000, 0xFFFFFFFF.
 - d. 0x00000000, 0x0FFFFFFC.
- ? ✓ 9. No MIPS, as instruções do tipo “I” incluem um campo imediato de 16 bits que:
- a. permite armazenar um *offset* de endereçamento de ± 32 KBytes para as instruções “sw”.
 - b. permite armazenar um *offset* de endereçamento $\pm (32*4)$ KBytes para as instruções “sw”.
 - c. permite armazenar um *offset* de endereçamento de ± 32 KBytes para as instruções “beq”.
 - d. permite armazenar um *offset* de endereçamento de $\pm (32*4)$ Kilo instruções para as instruções “beq”.

Cotações: Grupo I: cada 0.5 valores; Grupo II: cada 1.0 valor; Grupo III: cada 1.0 valor.

10. Numa implementação single-cycle da arquitectura MIPS:
- existe uma única ALU para realizar todas as operações aritméticas e lógicas necessárias para executar num único ciclo de relógio qualquer uma das instruções suportadas.
 - existem registos à saída dos elementos operativos fundamentais para guardar valores a utilizar no ciclo de relógio seguinte.
 - todas as operações de leitura e escrita são síncronas com o sinal de relógio.
 - existem memórias específicas para código e dados para possibilitar o acesso a ambos os tipos de informação num único ciclo de relógio.
11. A frequência de relógio de uma implementação single cycle da arquitectura MIPS:
- é limitada pelo maior dos tempos de atraso dos elementos operativos Memória, ALU e File Register.
 - varia em função da instrução que está a ser executada.
 - é limitada pelo maior dos atrasos cumulativos dos elementos operativos envolvidos na execução da instrução mais longa.
 - é limitada pelo menor dos tempos de atraso dos elementos operativos Memória, ALU e File Register.
12. A unidade de controlo de uma implementação multi-cycle da arquitectura MIPS:
- é um elemento combinatório que gera os sinais de controlo em função do campo *opcode* do código máquina da instrução.
 - é uma máquina de estados em que o primeiro e o segundo estados são comuns à execução de todas as instruções.
 - é uma máquina de estados com um número de estados igual ao número de fases da instrução mais longa.
 - é um elemento combinatório que gera os sinais de controlo em função do campo *funct* do código máquina da instrução.
13. Numa implementação multi-cycle da arquitectura MIPS, na segunda e terceira fases de execução de uma instrução de salto condicional ("**beq/bne**"), a ALU é usada, pela ordem indicada, para:
- calcular o valor do Branch Target Address e comparar os registos (operandos da instrução).
 - calcular o valor de PC+4 e comparar os registos (operandos da instrução).
 - comparar os registos (operandos da instrução) e calcular o valor do Branch Target Address.
 - calcular o valor de PC+4 e o valor do Branch Target Address.
14. Uma implementação pipelined de uma arquitectura possui, relativamente a uma implementação single-cycle da mesma, a vantagem de:
- diminuir o tempo de execução de cada uma das instruções.
 - permitir a execução de uma nova instrução a cada novo ciclo de relógio.
 - aumentar o débito de execução das instruções.
 - todas as anteriores.
15. A frequência de relógio de uma implementação pipelined da arquitectura MIPS:
- é limitada pelo maior dos atrasos cumulativos dos elementos operativos envolvidos na execução da instrução mais longa.
 - é definida de forma a evitar *stalls*, assim como *delay slots*.
 - é limitada pelo menor dos tempos de atraso dos elementos operativos Memória, ALU e File Register.
 - é limitada pelo maior dos tempos de atraso dos elementos operativos Memória, ALU e File Register.
16. A técnica de *forwarding/bypassing* num processador MIPS pipelined permite:
- utilizar como operando de uma instrução um resultado produzido por outra instrução que se encontra numa etapa mais recuada do pipeline.
 - trocar a ordem de execução das instruções de forma a resolver um *hazard* de dados.
 - utilizar como operando de uma instrução um resultado produzido por outra instrução que se encontra numa etapa mais avançada do pipeline.
 - escrever o resultado de uma instrução no File Register antes de ela chegar à etapa WB.

Zona de rascunho:

Grupo II

17. O código máquina da instrução **sw \$3, -128(\$4)**, representado em hexadecimal, é (considerando que para esta instrução **opcode=0x2B**):
- a. **0xAC838080.**
 - b. **0xAC83FF80.**
 - c. **0xAC64FF80.**
 - d. **0xAC648080.**
18. Considere que **a=0xC0D00000** representa uma quantidade codificada em hexadecimal segundo a norma IEEE 754 precisão simples. O valor representado em "a" é, em notação decimal:
- a. $-0,1625 \times 2^1$. $\text{exp} = 2$
 - b. $-0,1625 \times 2^3$. -1.101×2^2
 - c. $-3,25 \times 2^1$.
 - d. $-16,25 \times 2^1$.
19. Considerando que **\$f2=0x3A600000** e **\$f4=0xBA600000**, o resultado da instrução **sub.s \$f0,\$f2,\$f4** é:
- a. **\$f0=0x39E00000.**
 - b. **\$f0=0x3AE00000.**
 - c. **\$f0=0x00000000.**
 - d. **\$f0=0x80000000.**
20. Numa implementação single cycle da arquitectura MIPS, a frequência máxima de operação imposta pela instrução de leitura da memória de dados é, assumindo os atrasos a seguir indicados:
Memórias externas: leitura – 9ns, escrita – 11ns; File register: leitura – 3ns, Escrita – 4ns;
Unidade de Controlo: 2ns; ALU (qualquer operação): 7ns; Somadores: 4ns; Outros: 0ns.
- a. 32,25 MHz (T=31ns).
 - b. 25,00 MHz (T=40ns).
 - c. 29,41 MHz (T=34ns).
 - d. 31,25 MHz (T=32ns).
21. Considerando as seguintes frequências relativas de instruções de um programa a executar num processador MIPS: **lw** - 20%; **sw** - 10%; **tipo R** - 50%; **beq/bne** - 15%; **j** - 5%, a melhoria de desempenho proporcionada por uma implementação multi-cycle a operar a 100 MHz relativamente a uma single-cycle a operar a 20 MHz é de:
- a. 1,25.
 - b. 1.
 - c. 5.
 - d. 0,8.

Zona de rascunho:

1	0	1	0	1	1	0	0	1	0	0	1	1
A	C	8	3									

1707

$$9 \Rightarrow \text{Exp} = 129 - 127 = 2$$

$$0100 \cdot 2^2 = 110.100_2 = -6.5_{10}$$

00 ... 10000000 $\times 2$
 7-1111,1000,0000
 F F 8 0

22. Um *hazard* de controlo numa implementação *pipelined* de um processador ocorre quando:
- um dado recurso de hardware é necessário para realizar no mesmo ciclo de relógio duas ou mais operações relativas a instruções em diferentes etapas do *pipeline*.
 - é necessário fazer o *instruction fetch* de uma nova instrução e existe numa etapa mais avançada do *pipeline* uma instrução que ainda não terminou e que pode alterar o fluxo de execução.
 - existe uma dependência entre o resultado calculado por uma instrução e o operando usado por outra que segue mais atrás no *pipeline*.
 - por azar, a unidade de controlo desconhece o *opcode* da instrução que se encontra na etapa ID.
23. O seguinte trecho de código, a executar sobre uma implementação *pipelined* da arquitectura MIPS, apresenta os seguintes *hazards*:
- um *hazard* de controlo na quarta instrução e um *hazard* de dados na segunda instrução que pode ser resolvido por *forwarding*.
 - um hazard* estrutural na primeira instrução e um *hazard* de controlo na quarta instrução.
 - um *hazard* de controlo na quarta instrução e *hazards* de dados na segunda, terceira e na quarta instruções que podem ser resolvidos por *forwarding*.
 - um hazard* de controlo na quarta instrução e *hazards* de dados na terceira e na quarta instruções que podem ser resolvidos por *forwarding*.
24. Considere o *datapath* e a unidade de controlo fornecidos na figura da última página (com ligeiras alterações relativamente à versão das aulas teórico-práticas) correspondendo a uma implementação *multi-cycle* simplificada da arquitectura MIPS. Admita que os valores indicados no *datapath* fornecido correspondem à "fotografia" tirada no decurso da execução de uma instrução. Tendo em conta todos os sinais, pode-se concluir que está em execução a instrução:
- lw \$6,0x2020(\$5)* na terceira fase.
 - add \$4,\$5,\$6* na quarta fase.
 - add \$4,\$5,\$6* na terceira fase.
 - lw \$6,0x2020(\$5)* na quinta fase.

Grupo III

Considere o trecho de código apresentado na Figura 1, bem como as tabelas os valores dos registos que aí se apresentam. Admita que o valor presente no registo **\$PC** corresponde ao endereço da primeira instrução, que nesse instante o conteúdo dos registos é o indicado, e que vai iniciar-se o *instruction fetch* dessa instrução. Considere ainda o *datapath* e a unidade de controlo fornecidos na Figura 2 (última página).

Endereço	Dados	Opcode	Funct	Operação
...	...	0	0x20	add
0x1001009C	0xFFFF0000	0	0x22	sub
0x100100A0	0x021B581A	0	0x24	and
0x100100A4	0x00008000	0	0x25	or
0x100100A8	0x1B54E790	0x02		j
0x100100AC	0x00FE7F00	0x04		beq
0x100100B0	0x5FF38C29	0x05		bne
...	...	0x08		addi
		0x0C		andi
		0x23		lw
		0x2B		sw

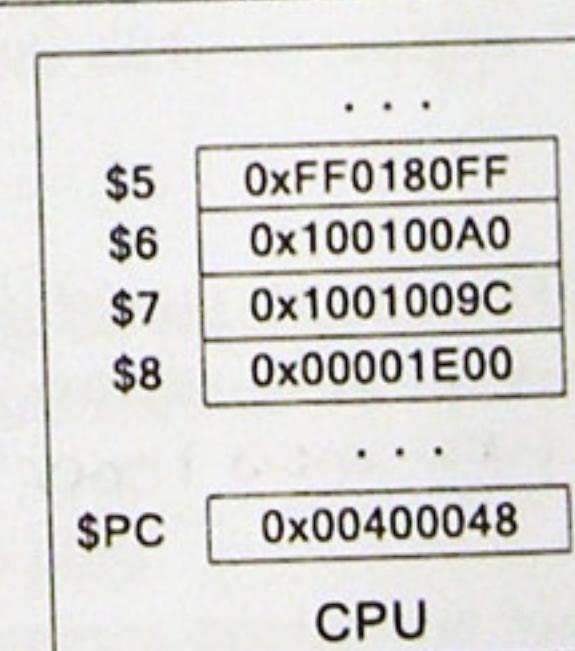
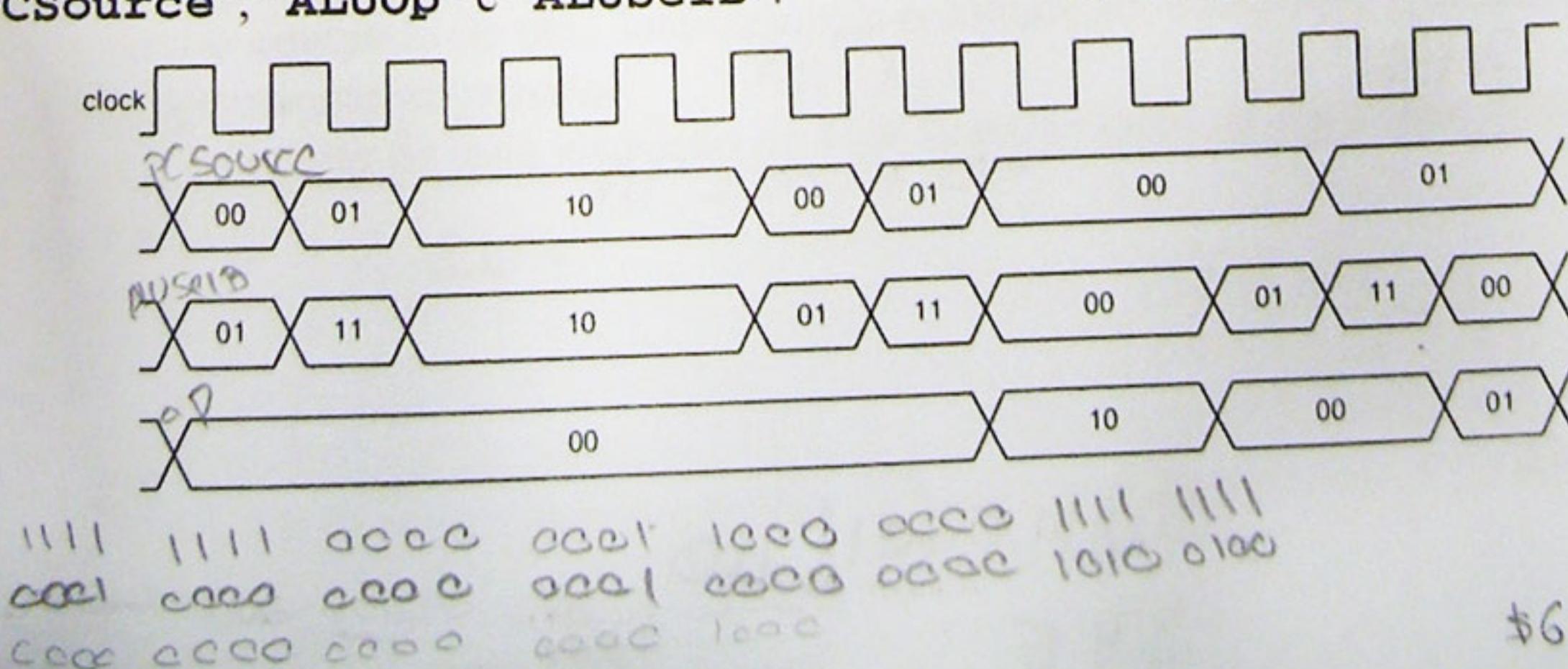


Figura 1

L1:	lw	\$6,0(\$7)	5
	and	\$8,\$6,\$5	4
	beq	\$8,\$0,L2	3
	sw	\$8,4(\$7)	4
	addi	\$7,\$7,8	4
	j	L1	3
L2:	...		23

AA
23
46

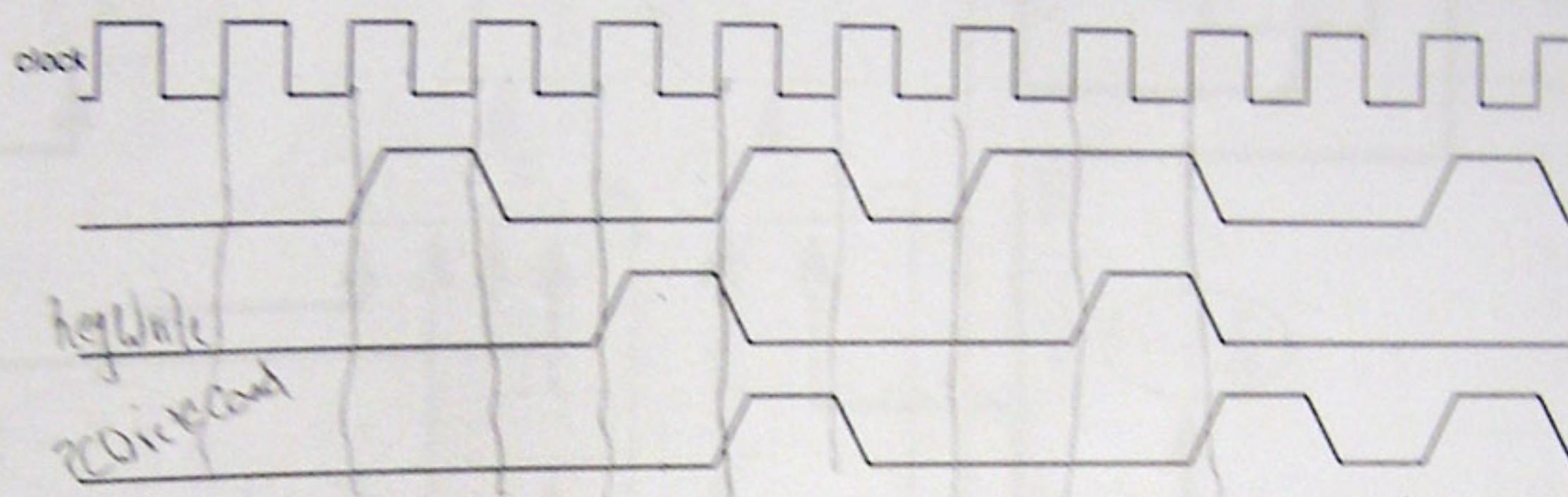
25. Para as 3 primeiras instruções do trecho de código apresentado na Figura 1, os sinais de controlo representados no seguinte diagrama temporal correspondem, pela ordem indicada, a:
- "ALUSelB", "ALUOp" e "PCSource".
 - "PCSource", "ALUOp" e "ALUSelB".
 - "PCSource", "ALUOp" e "ALUSelB" e "ALUOp".
 - "ALUSelB", "PCSource" e "ALUOp".



ALUOp	00	ADD
	01	SUB
	1X	R-Type

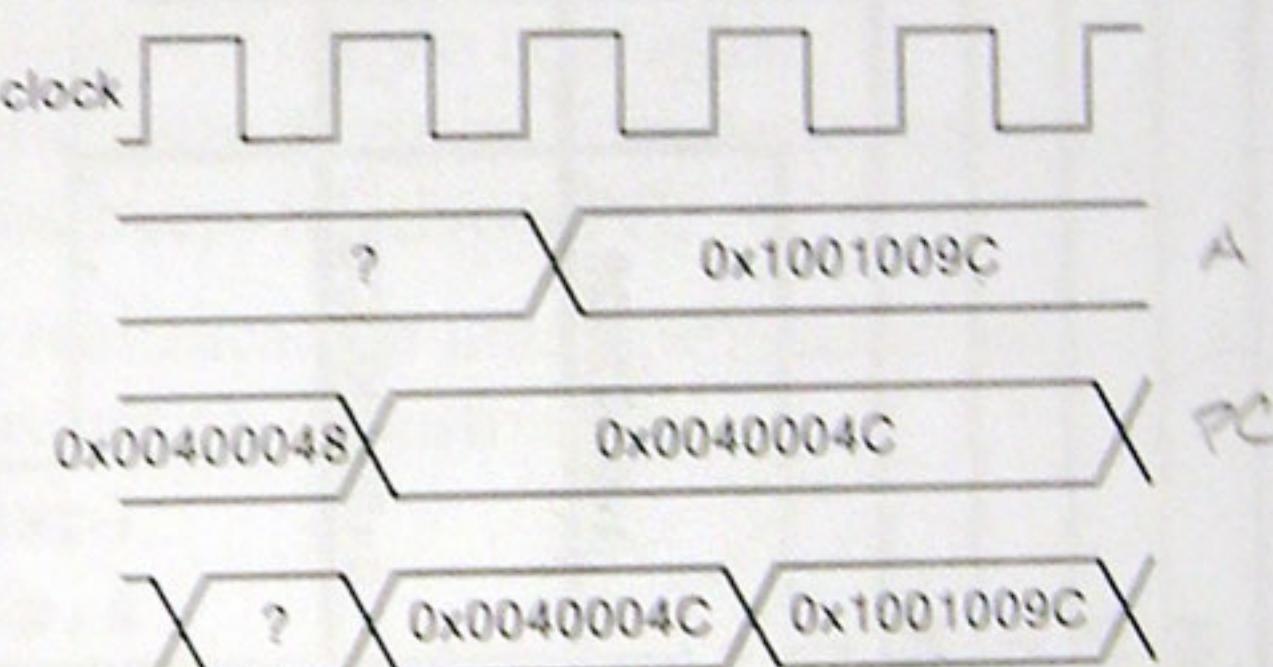
26. Também para as 3 primeiras instruções do trecho de código apresentado na Figura 1, os sinais de controlo representados no seguinte diagrama temporal correspondem, pela ordem indicada, a:

 - a. "RegWrite", "PCWriteCond" e "RegDst".
 - b. "RegDst", "RegWrite" e "PCWriteCond".
 - c. "PCWriteCond", "RegWrite" e "RegDst".
 - d. "RegDst", "PCWriteCond" e "RegWrite".



27. Para a primeira instrução do trecho de código apresentado na Figura 1, e supondo que os valores dos registos do CPU são os que se indicam na mesma figura, os sinais do *datapath* representados no seguinte diagrama temporal correspondem, pela ordem indicada, a:

- a. "A", "InstRegister" e "PC".
b. "B", "PC" e "ALUOut".
c. "A", "PC" e "ALUOut".
d. Nenhuma das anteriores.



28. Face aos valores presentes no segmento de dados (tabela da esquerda) e nos registos, o número total de ciclos de relógio que demora a execução completa do trecho de código apresentado, numa implementação multi-cycle do MIPS, é (desde o instante inicial do *instruction fetch* da primeira instrução até ao momento em que vai iniciar-se o *instruction fetch* da instrução presente em "L2:");

 - a. 58 ciclos de relógio.
 - b. 12 ciclos de relógio.
 - c. 6 ciclos de relógio.
 - d. 35 ciclos de relógio.

Zona de rascunho:

0x004000C8 - 0x0040004C
? ? . 1001008C
? ? 100100AC

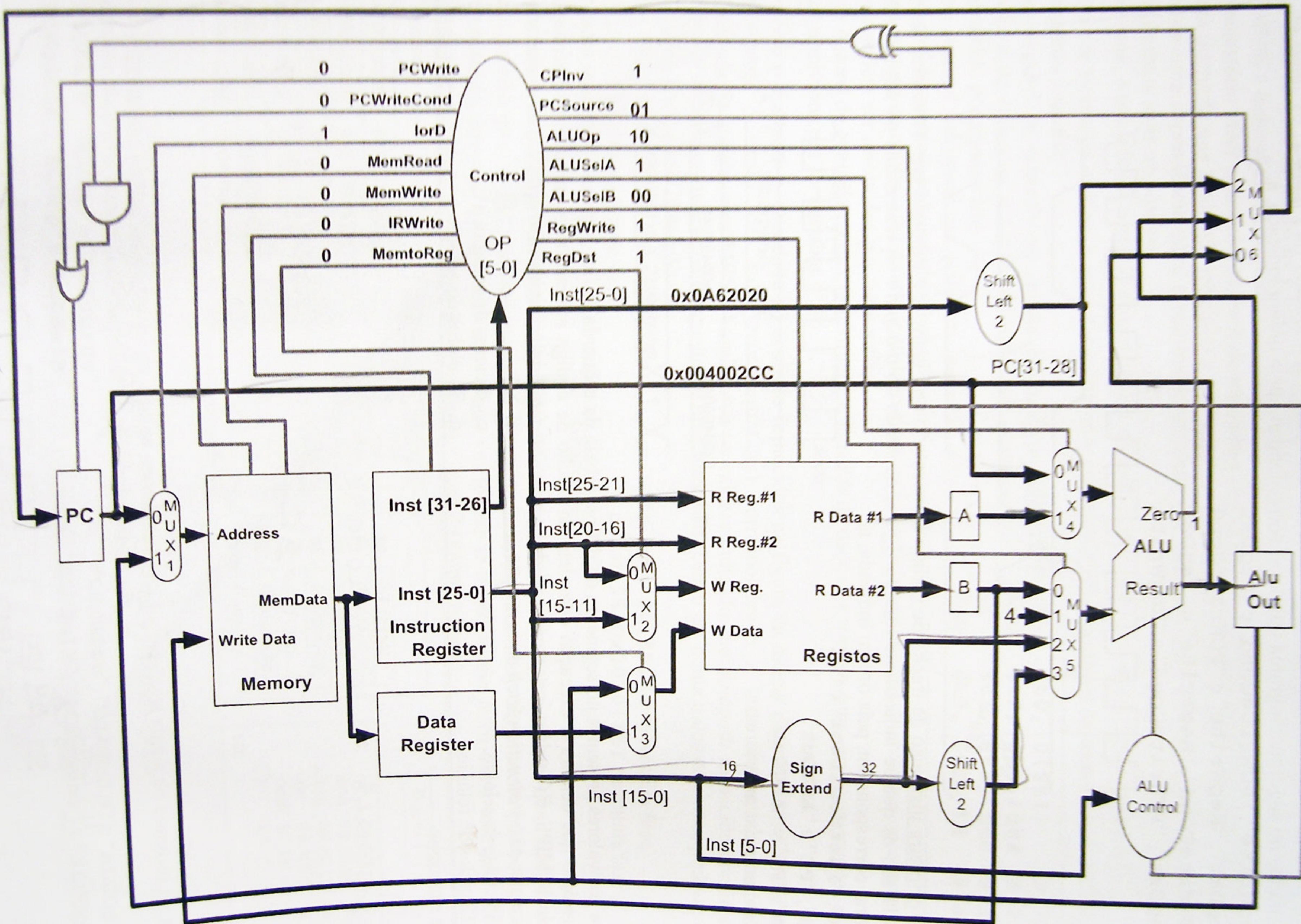
$$\begin{array}{r} \underline{} \\ \underline{} \\ 1100 \\ 0101 \\ \hline 0100 \end{array}$$

0x0040004C 0x0040004C

$$\begin{array}{r}
 117 \longdiv{12} \\
 117 \quad \text{R} \\
 \hline
 0 \quad 58 \longdiv{12} \\
 0 \quad 58 \quad \text{R} \\
 \hline
 0 \quad 29 \longdiv{12} \\
 0 \quad 24 \quad \text{R} \\
 \hline
 0 \quad 5 \longdiv{12} \\
 0 \quad 3 \quad \text{R} \\
 \hline
 2 \quad 12
 \end{array}$$

ALU Sel B
 ALUOp
 PC Source

Figura 2



1. Uma arquitectura do tipo *Harvard* é caracterizada por:

- Teste 1**
b. permitir o acesso a instruções e dados no mesmo ciclo de relógio

	a	b	c	d
1		X		
2	X			
3				X
4		X		
5		X		
6			X	
7		X		
8	X			
9			X	
10	X			
11		X		
12			X	
13			X	
14		X		
15	X			
16	X			
17				
18			X	
19		X		
20				X
21	X			
22			X	
23		X		
24				X
25				
26				
27			X	
28				X

Teste 2

- d. permitir o acesso a instruções e dados no mesmo ciclo de relógio.

	a	b	c	d
1				X
2				X
3			X	
4	X			
5	X			
6			X	
7	X			
8				X
9	X			
10				X
11			X	
12		X		
13	X			
14			X	
15				X
16			X	
17				X
18	X			
19				X
20		X		
21			X	
22			X	
23				X
24	X			
25				
26			X	
27		X		
28	X			

I

Nome: _____ N°. Mec. _____

Notas Importantes!

1. Verifique, para as questões de escolha múltipla, qual a resposta correcta e assinale com uma cruz a sua escolha na tabela ao lado. Nestas questões, por cada resposta incorrecta será descontada, à cotação global, 1/3 da cotação da respectiva pergunta.
2. Durante a realização do teste não é permitida a permanência na sala de calculadoras, telemóveis ou outros dispositivos electrónicos.

	a	b	c	d
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				

I

1. Um endereço de memória externa num sistema computacional é:
 - a. a gama de posições de memória que a CPU pode referenciar.
 - b. um número único que identifica cada posição de memória.
 - c. a informação armazenada em cada posição.
 - d. um índice de um registo de uso geral.
2. A arquitectura MIPS é caracterizada por:
 - a. possuir 32 registos de uso geral de 32 bits cada.
 - b. ser do tipo *load-store*.
 - c. possuir poucos formatos de instrução.
 - d. todas as anteriores.
3. Na arquitectura MIPS, os campos de uma instrução tipo "R" designam-se por:
 - a. "opcode", "rs", "rt" e "imm".
 - b. "opcode" e "address".
 - c. "opcode", "rs", "rt", "rd", "shamt" e "imm".
 - d. nenhuma das anteriores.
4. A instrução virtual "li \$t0,0x10012345" da arquitectura MIPS decompõe-se na seguinte sequência de instruções nativas:
 - a. "lui \$1,0x2345" seguida de "ori \$t0,\$1,0x1001".
 - b. "ori \$t0,\$1,0x1001" seguida de "lui \$1,0x2345".
 - c. "lui \$1,0x1001" seguida de "ori \$t0,\$1,0x2345".
 - d. "ori \$t0,\$1,0x2345" seguida de "lui \$1,0x1001".
5. Nas instruções de acesso à memória da arquitectura MIPS é utilizado o modo de endereçamento:
 - a. indirecto por registo.
 - b. registo.
 - c. imediato.
 - d. directo.
6. O formato de instruções tipo "I" da arquitectura MIPS é usado nas instruções de:
 - a. salto condicional.
 - b. aritméticas em que somente um dos operandos está armazenado num registo.
 - c. acesso à memória de dados externa.
 - d. todas as anteriores.

7. Considerando que no endereço de memória acedido pela instrução “lb \$t0, 0xFF(\$t1)” está armazenado o valor 0x82, o valor armazenado no registo destino no final da execução dessa instrução é:
- 0xFF.
 - 0x82.
 - 0xFFFFF82.
 - 0xFF82.
8. A instrução virtual “bgt \$t8,\$t9,target” da arquitectura MIPS decompõe-se na seguinte sequência de instruções nativas:
- “slt \$1,\$t8,\$t9” seguida de “bne \$1,\$0,target”.
 - “slt \$1,\$t9,\$t8” seguida de “bne \$1,\$0,target”.
 - “slt \$1,\$t8,\$t9” seguida de “beq \$1,\$0,target”.
 - “slt \$1,\$t9,\$t8” seguida de “beq \$1,\$0,target”.
9. Os endereços mínimo e máximo para os quais uma instrução “bne” presente no endereço 0x00430210 pode saltar são:
- 0x00000000, 0xFFFFFFFF.
 - 0x00000000, 0x0FFFFFFC.
 - 0x00428214, 0x00438213.
 - 0x00410214, 0x00450210.
10. A instrução “jal funct” executa sequencialmente as seguintes operações:
- \$PC = \$PC + 4, \$ra = \$PC, \$PC = funct.
 - \$PC = \$PC + 4, \$PC = funct, \$ra = \$PC.
 - \$ra = \$PC, \$PC = funct.
 - Nenhuma das anteriores.
11. Os endereços mínimo e máximo para os quais uma instrução “j” presente no endereço 0x00430210 pode saltar são:
- 0x00428214, 0x00438213.
 - 0x00000000, 0xFFFFFFFF.
 - 0x00410214, 0x00450210.
 - 0x00000000, 0x0FFFFFFC.
12. Segundo a convenção de utilização de registos da arquitectura MIPS, uma subrotina não necessita de salvaguardar os registos com os prefixos:
- \$a, \$v, \$s.
 - \$s, \$v, \$t.
 - \$a, \$v, \$t.
 - \$a, \$s, \$t.
13. Na arquitectura MIPS a *stack* é gerida de acordo com os seguintes princípios:
- cresce no sentido dos endereços mais altos, apontando o registo \$sp para a última posição ocupada.
 - cresce no sentido dos endereços mais baixos, apontando o registo \$sp para a última posição ocupada.
 - cresce no sentido dos endereços mais altos, apontando o registo \$sp para a primeira posição livre.
 - cresce no sentido dos endereços mais baixos, apontando o registo \$sp para a primeira posição livre.

14. Numa ALU, a detecção de *overflow* nas operações de adição algébrica é efectuada através:
- do “ou” exclusivo entre o *carry in* e o *carry out* da célula de 1 bit mais significativa.
 - da avaliação do bit mais significativo do resultado.
 - do “ou” exclusivo entre o bit mais significativo e o menos significativo do resultado.
 - do “ou” exclusivo entre os 2 bits mais significativos do resultado.
15. A decomposição numa sequência de adições e subtrações, de acordo com o algoritmo de Booth, da quantidade binária $101101_{(2)}$ é:
- $-2^0 + 2^1 - 2^2 + 2^4 - 2^5$
 - $2^0 - 2^1 + 2^2 - 2^4 + 2^5$
 - $2^0 - 2^1 + 2^2 + 2^3 - 2^4 + 2^5$
 - $-2^0 + 2^1 - 2^2 - 2^3 + 2^4 - 2^5$
16. A quantidade real binária $1011,11000000_2$ quando representada em decimal é igual a:
- 12,6
 - 11,75
 - 3008,0
 - 1504,0

II

Apresente o diagrama de blocos de um multiplicador de números de 16 bits sem sinal, iterativo e optimizado em termos da dimensão dos seus elementos funcionais. Indique os registos onde devem ser carregados os operandos e lido o resultado, a função de cada um dos restantes blocos, bem como as respectivas dimensões.

III

Considere as tabelas a seguir apresentadas. Admita que o valor presente no registo \$PC corresponde ao endereço da primeira instrução, que nesse instante o conteúdo dos registos é o indicado, e que vai iniciar-se o *instruction fetch* dessa instrução.

Endereço ...	Dados ...
0x10010040	0xFEC81248
0x10010044	0x00410312
0x10010048	0xC630F731
0x1001004C	0x3A509DB0
0x10010050	0x8421C630
0x10010054	0x5FF38C29
...	...

Opcode	Funct	Operação
0	0x20	add
0	0x22	sub
0	0x24	and
0	0x25	or
0x02		j
0x04		beq
0x05		bne
0x08		addi
0x0C		andi
0x23		lw
0x2B		sw

\$4	...	0x00000F03
\$5	...	0x10010050
\$6	...	0x10010040
\$7	...	0x0040003C
\$8	...	0x10010054
\$9	...	0x00000004
SPC	...	0x0040002C

CPU

```
L1: beq $5,$6,L2
    lw $7,0($5)
    and $8,$7,$4
    sw $8,4($5)
    addi $5,$5,-8
    j L1
L2: ...
```

- Traduza para código máquina do MIPS o trecho de código correspondente às seis instruções da tabela da direita (expressando o resultado em hexadecimal) e indique o endereço de memória em que cada uma se encontra. Justifique todos os passos da sua resposta.
- Indique o conteúdo dos registos do CPU e das posições do segmento de dados apresentadas, após a execução do trecho de código fornecido, isto é, imediatamente antes de ser iniciado o *instruction fetch* da instrução armazenada no endereço correspondente à etiqueta L2. Justifique adequadamente a sua resposta.

1. NOTA: Use **no máximo 40 palavras** para responder a cada uma das 4 questões seguintes:

- Apresente, justificando, a decomposição em instruções nativas da instrução “**bge \$4, \$5, else**”.
 - Apresente o formato de codificação da instrução “**J**”, referindo quais os campos em que é decomposta a dimensão de cada um e o seu significado. Explique ainda como essa instrução é executada.
 - Indique o modo de codificação do expoente na norma IEEE-754, precisão simples, e apresente a razão pela qual se utiliza esse método.
 - Apresente a razão pela qual, num datapath *pipelined*, as instruções do tipo R passam pela fase “memory access”.
- 2. Considere o datapath single-cycle** que foi apresentado nas aulas teóricas. Admita os seguintes atrasos de propagação envolvidos nos vários elementos operativos e de estado:

Memória externa (dados e código): Leitura – 3ns; Escrita – 9ns

File Register: Leitura – 3ns; Escrita – 4ns *Sign Extend*: 4ns *Shifter*: 3ns

ALU (qualquer operação): 5ns *Somadores*: 5ns *Outros dispositivos*: 0ns

- Determine o tempo mínimo necessário à execução de cada uma das três instruções seguintes (consideradas independentemente) e calcule a máxima frequência de relógio desta arquitectura. **Justifique a sua resposta.**

sw \$a0, 0(\$s0) [45%] **add \$t1, \$t2, \$a0 [35%]** **beq \$s12, \$0, loop [20%]**

- Considerando apenas as três instruções da alínea anterior, que a frequência de relógio é de 25MHz, e que a taxa média de ocorrência é a indicada entre parêntesis rectos, determine o valor médio do tempo durante o qual o *datapath* não sofre alterações em cada ciclo.

3. Considere que o conteúdo dos registo \$f6 e \$f4 é, respectivamente:

$$\$f6 = 0.90625_{10} \times 2^{-58} \quad \$f4 = -16.3125_{10} \times 2^{65}$$

- Obtenha a representação em binário das quantidades armazenadas naqueles registo (codificadas segundo a norma IEEE 754, precisão simples).
- Determine o resultado da instrução **div.s \$f2, \$f4, \$f6**, realizando, **em binário e/ou hexadecimal**, os passos necessários. Indique, **em binário** (de acordo com a norma IEEE 754, precisão simples), qual o conteúdo do registo **\$f2** após a execução da instrução.

4. Considere o datapath e a unidade de controlo fornecido na Figura 1 (ligeiramente alterado em relação à versão das aulas teóricas), sabendo que corresponde a uma implementação multi-ciclo simplificada do MIPS, sem *pipelining*.

- Preencha a tabela fornecida em anexo com o nome de cada uma das fases de execução da instrução “**sub \$16, \$15, \$11**” e com o valor que tomam, em cada uma delas, os sinais e valores do *datapath* e os sinais de controlo ali indicados. Admita que o valor lógico “1” corresponde ao estado activo (considere que os registo, antes da execução da instrução, têm os seguintes valores: **\$15=0xA0F4, \$11=0x100F4F0A, \$PC=0x4000C0**).
- Admita que os valores indicados no *datapath* fornecido correspondem à “fotografia” tirada no decurso da execução de uma dada instrução. Identifique, observando com atenção todos os sinais: **1)** qual a instrução em causa (apresente a instrução completa); **2)** qual a fase de execução em que se encontra; **3)** qual o valor do PC após a execução da instrução. **Justifique todos os passos da sua resposta.**

5. Considere, na Figura 2, as tabelas ali apresentadas. Admita que o valor presente em **\$PC** corresponde ao endereço da primeira instrução, que nesse instante o conteúdo dos registo é o indicado, e que vai iniciar-se o “*instruction fetch*” dessa instrução. Considere ainda o *datapath* e a unidade de controlo fornecido na Figura 1.

- Escreva, em *Assembly* do MIPS, o trecho de código correspondente às seis instruções presentes na tabela da direita e indique, para cada uma delas, o endereço de memória em que se encontra. Crie *labels* sempre que tal seja adequado. **Justifique todos os passos da sua resposta.**
- Face aos valores presentes no segmento de dados (tabela da esquerda) e ao código que obteve na alínea anterior, determine, **justificando**, o número total de ciclos de relógio que demora a execução completa desse trecho de código (desde o instante inicial do *instruction fetch* da primeira instrução até ao momento em que vai iniciar-se o *instruction fetch* da instrução presente em “**next:**”), bem como o valor final do registo **\$6**.
- Represente, sob a forma de um diagrama temporal, a evolução do sinal de relógio e dos sinais de controlo “**PCWrite**”, “**RegWrite**”, “**RegDst**”, “**IRWrite**” e “**ALUOp**” durante a execução (em sequência e pela ordem apresentada) das 3 primeiras instruções do trecho de código apresentado na Figura 2 (Nota: represente “don’t care” por **////**). **Justifique adequadamente a sua resposta.**

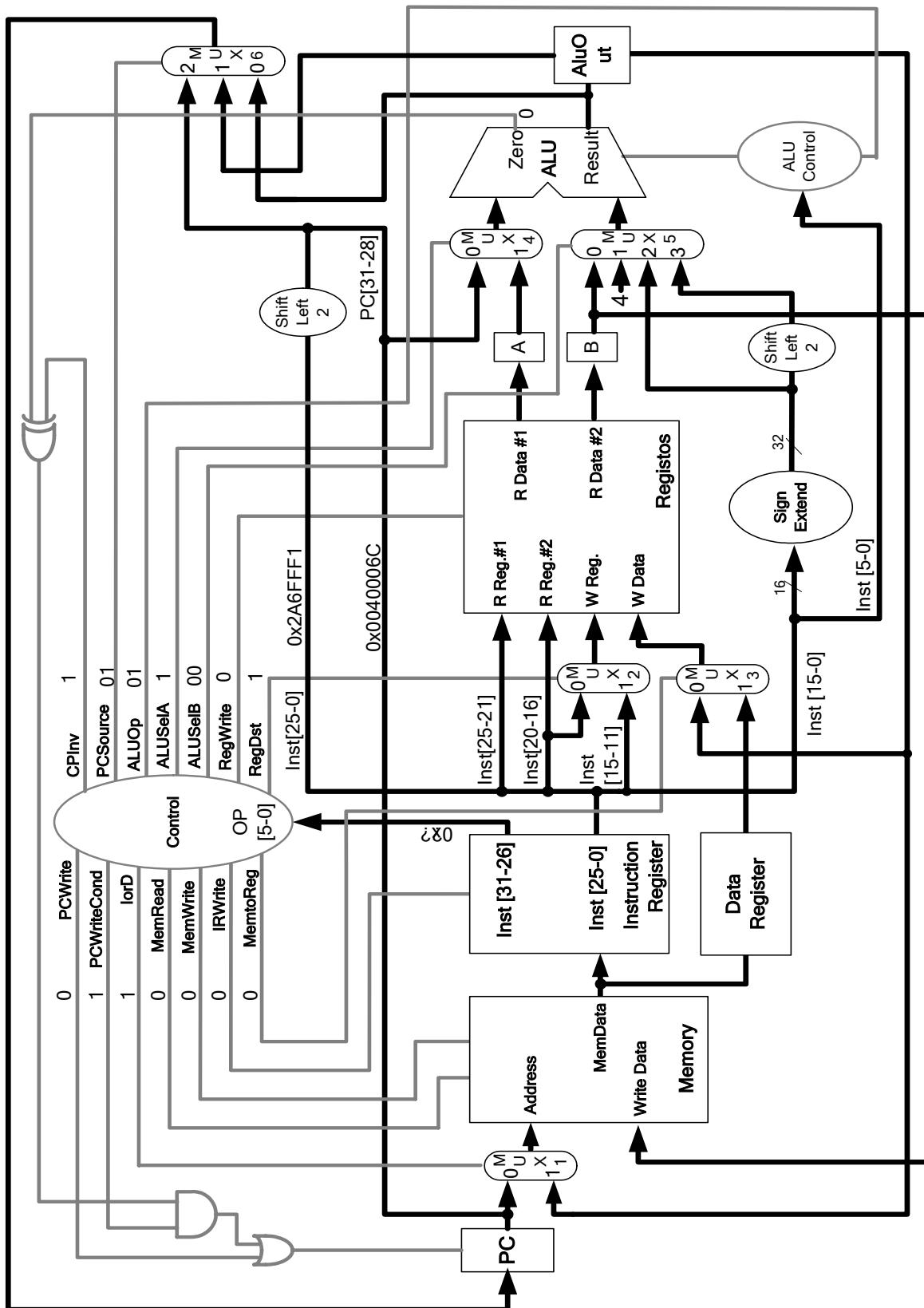


Figura 1

1.

- a) As instruções nativas utilizadas para saltos condicionais são beq, bne e slt. O salto é efectuado de $\$4 \geq \5 . Mas $\$4 \geq \$5 \Leftrightarrow \sim(\$4 < \$5)$. Decompondo em instruções nativas tem-se

slt	$\$1, \$4, \$5$
beq	$\$1, \$0, \text{else}$

b)

opcode (6 bits)	address (26 bits)
--------------------	----------------------

O campo address é multiplicado por 4 (o resultado da multiplicação tem 28 bits) e é de seguida concatenado aos 4 bits mais significativos do PC (Program Counter).

c)

S (1 bit)	E (8 bits)	F (23 bits)
--------------	---------------	----------------

A representação é feita em sinal e módulo.

S representa o sinal (0 para positivos e 1 para negativos)

E é o expoente codificado em excesso 127 (Expoente real = E – 127)

F é a mantissa

Se este método não fosse utilizado, não seria possível saber quantos dígitos reservar para a parte inteira e para a parte fraccionária, sabendo que o espaço de armazenamento é limitado.

- d) O datapath pipelined está dividido em 5 fases. Como a fase “Memory Access” (MEM) aparece antes da fase de “Write Back” (WB), as instruções do tipo R têm que passar por essa fase para poderem chegar à fase WB (fase necessária à sua conclusão). Na figura seguinte mostra-se um problema que poderia ocorrer caso as instruções do tipo R não passassem pela fase MEM. Trata-se de uma instrução de load seguida de uma instrução do tipo R. Do lado esquerdo mostra-se o que é correcto (todas as instruções passam pelas 5 fases) e no lado direito mostra-se o problema. As duas instruções estão a tentar escrever no banco de registo ao mesmo tempo.

IF	ID	EX	MEM	WB	
IF	ID	EX	MEM	WB	

IF	ID	EX	MEM	WB
IF	ID	EX		WB

2.

a) SW

$$T = \text{Somador} + (\text{Sign Extend} + \text{Shifter} + \text{Somador}) + \text{ALU} + \text{Escrever memória} = \\ = 5 + (4 + 3 + 5) + 5 + 9 = 31 \text{ ns}$$

TIPO-R

$$T = \text{Somador} + \text{Ler File Register} + \text{ALU} + \text{Escrever File Register} = \\ = 5 + 3 + 5 + 4 = 17 \text{ ns}$$

Branch

$$T = \text{Somador} + (\text{Sign Extend} + \text{Shifter} + \text{Somador}) + \text{ALU} = \\ = 5 + (4 + 3 + 5) + 5 = 22 \text{ ns}$$

Considerando o set de instruções apenas com estas três instruções, a máxima frequência do relógio é o inverso do maior atraso, ou seja

$$f = 1/31\text{ns} = 32 \text{ MHz}$$

b) $f = 25 \text{ MHz} \Rightarrow T = 1/25 \mu\text{s} = 0.04 \mu\text{s} = 40 \text{ ns}$ (tempo para cada instrução)

$$T' = 0.45*31 + 0.35*17 + 0.20*22 = 24.3 \text{ ns}$$

$T' / T = 61\%$ (aprox.) é a utilização do datapath.

3.

a) $0.90625 \text{ (decimal)} = 0.11101 = 1.1101*2^{-1} \text{ (binário)}$
 $-16.3125 \text{ (decimal)} = -10000.0101 = -1.00000101*2^4$

Os valores a codificar são $1.1101*2^{-59}$ e $-1.00000101*2^{69}$

1º valor:

$$\begin{aligned} S &= 0 \\ E &= -59 + 127 = 68 \text{ (decimal)} = 01000100 \text{ (binário)} \\ F &= 11010000000000000000000000000000 \end{aligned}$$

2º valor

$$\begin{aligned} S &= 1 \\ E &= 69 + 127 = 196 \text{ (decimal)} = 11000100 \text{ (binário)} \\ F &= 00000101000000000000000000000000 \end{aligned}$$

Os valores normalizados para IEEE-754 são portanto

$$\begin{aligned} 0 & 01000100 11010000000000000000000000000000 \Leftrightarrow 0x22680000 \\ 1 & 11000100 000001010000000000000000 \Leftrightarrow 0xE2028000 \end{aligned}$$

- b) Expoente do resultado: $69 - (-59) = 128$
 Sinal do resultado : 1 xor 0 = 1 (o resultado é negativo)

$$\begin{array}{r}
 1.00000101 \quad | 1.1101 \\
 111 \quad .1001 \\
 1110 \\
 11101 \\
 0
 \end{array}$$

O resultado é: $-0.1001 * 2^{128} = -1.001 * 2^{127}$

No formato IEEE-754 fica:

1 11111110 00010000000000000000000000000000 \Leftrightarrow 0xFF100000

4. A tabela vem preenchida na última página.
 a) sub \$16, \$15, \$11

Código máquina

000000	01111	01011	10000	00000	100010
opcode	rs	rt	rd	shamt	funct

Branch Target

$$0x8022*4 + PC = 0x20088 + 0x4000C8 = 0x0042014C$$

Operação \$15 - \$11

$$0x000A0F4 - 0x100F4F0A = 0xEFF151EA$$

b)

1)

PCWrite=0
 PCWriteCond=1

Conclui-se que é uma instrução do tipo branch

ALUOp=01 (subtração)
 CPInv=1 (se a ALU devolver 0 o salto é efectuado – beq)

Trata-se da instrução beq (opcode = 0x04)

Inst[25-0] = 0x2A6FFF1

000100 10101 00110 1111111111110001

O offset está representado em complemento para 2, o seu valor é -0xF.
Para calcular o branch target address efectua-se a seguinte operação

$$(PC+4) + 4*offset \rightarrow 0x0040006C - 0x3C = 0x00400030$$

A instrução é finalmente,

beq \$21, \$6, 0x00400030

2) Está na fase Branch Conclusion (3^a e última fase) já que PCWriteCond=1 e ALUZero = 0 (a ALU já fez a subtração e já tem o resultado disponível).

3) Como o salto não é efectuado (PCWriteCond=1 e CPIInv=1 e ALUZero=0) então o valor do PC após a execução da instrução será o Branch Target, ou seja,

0x00400030

5.

a)

100011 00111 00101 0000000000000000100 \Leftrightarrow lw \$5, 4(\$7)
lw

000000 00101 01001 00101 00000 100100 \Leftrightarrow and \$5, \$5, \$9
tipoR and

000100 00101 01000 000000000000000011 \Leftrightarrow beq \$5, \$8, next
beq offset=3
 $(PC+4)+4*offset = 0x00400038 + 0xC = 0x00400044$ (label next)

000000 00111 01010 00111 00000 100000 \Leftrightarrow add \$7, \$7, \$10
tipoR add

001000 00110 00110 1111111111111111 \Leftrightarrow addi \$6, \$6, -1
addi offset = -1

000101 00110 00000 111111111111010
bne offset = -6
 $(PC+4)+4*offset = 0x00400044 - 0x18 = 0x0040002C$ (label start)

Endereço	Código
0x0045002C	start: lw \$5, 4(\$7)
0x00400030	and \$5, \$5, \$9
0x00400034	beq \$5, \$8, next
0x00400038	add \$7, \$7, \$10
0x0040003C	addi \$6, \$6, -1
0x00400040	bne \$6, \$0, start
0x00400044	next: ...

b)

$\$5 = 0x0021B5C3$
 $\$5 = 0x000000C3$
 $\$5 \neq \$8 \rightarrow$ o salto não é efectuado
 $\$7 = 0x10010098$
 $\$6 = 0x0000001B$
 $\$6 \neq 0 \rightarrow$ o salto é efectuado para “start”

$\$5 = 0x0021B54E$
 $\$5 = 0x0000004E$
 $\$5 \neq \$8 \rightarrow$ o salto não é efectuado
 $\$7 = 0x10010094$
 $\$6 = 0x0000001A$
 $\$6 \neq 0 \rightarrow$ o salto é efectuado para “start”

$\$5 = 0x0021B52F$
 $\$5 = 0x0000002F$
 $\$5 \neq \$8 \rightarrow$ o salto não é efectuado
 $\$7 = 0x10010090$
 $\$6 = 0x00000019$
 $\$6 \neq 0 \rightarrow$ o salto é efectuado para “start”

$\$5 = 0x0021B581$
 $\$5 = 0x00000081$
 $\$5 == \$8 \rightarrow$ o salto é efectuado para “next”

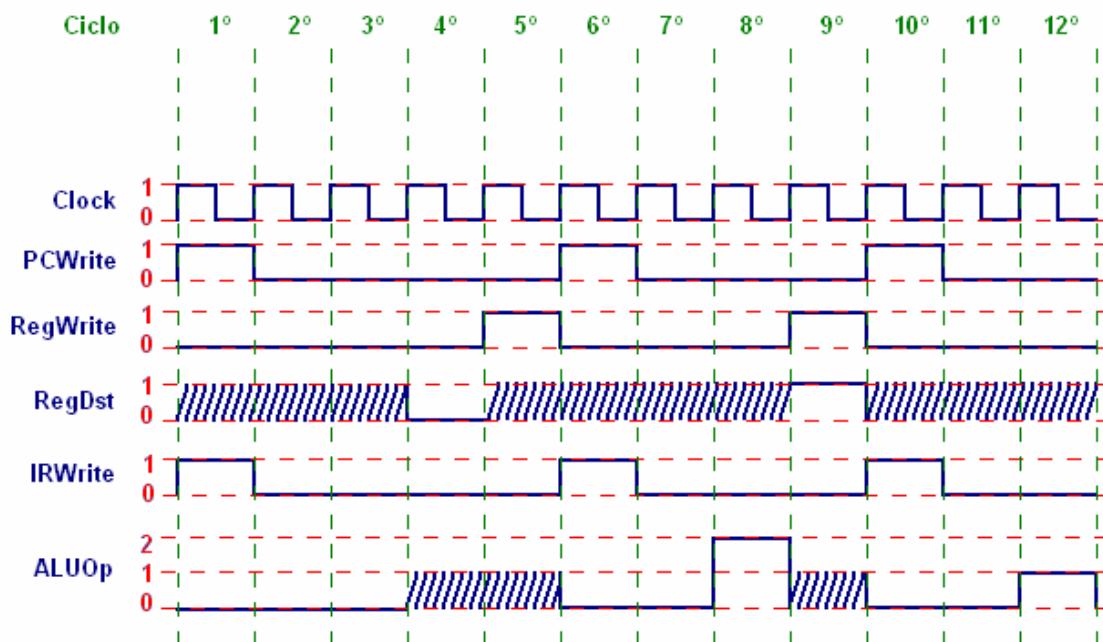
Note-se que:

Instrução	Nº de ciclos necessários	Nº de vezes que é utilizada
lw	5	4
and	4	4
beq	3	4
add	4	3
addi	4	3
bne	3	3

O nº de ciclos total é, portanto: $5*4 + 4*4 + 3*4 + 4*3 + 4*3 = 81$ ciclos

O valor final em \$6 é 0x00000019

c) Para executar as 3 instruções são necessários no total $5+4+3 = 12$ ciclos.



- O PCWrite só toma o valor 1 no 1º ciclo de cada instrução (Instruction Fetch)
- O RegWrite é 1 no último ciclo do lw e no último ciclo do and para que se possa escrever no registo destino. O branch não escreve no banco de registos.
- O RegDst pode tomar qualquer valor (don't care) quando o RegWrite é 0. Quando este é 1, o RegDst toma o valor 0 para o lw (escreve no registo rt) e toma o valor 1 para o and (escreve no registo rd).
- O IRWrite apenas toma o valor 1 durante o Instruction Fetch.
- O ALUOp é sempre 00 (soma) no 1º e no 2º ciclos para poder calcular o PC+4 e o branch target, respectivamente. No 3º ciclo é 00 para o lw para calcular rs+offset (endereço de memória a aceder) e para o and é 10 para que a operação da ALU dependa do campo funct (instruções tipo R). No caso do branch é 01 (subtração) para efectuar a subtração que permite comparar os registos rs e rt através da saída "Zero" da ALU.
- A partir do 3º ciclo, o ALUOp é "don't care" porque a ALU deixa de ser necessária.

Anexo: Tabela do Exercício 4

	Fase 1	Fase 2	Fase 3	Fase 4	Fase 5
--	--------	--------	--------	--------	--------

Nome da fase	Instruction Fetch	Instruction Decode	Operation Execute	Write Back	
--------------	-------------------	--------------------	-------------------	------------	--

Datapath					
PC	0x004000C0	0x004000C4	0x004000C4	0x004000C4	
Instr. Register	?	0x01EB8022	0x01EB8022	0x01EB8022	
Data Register	?	0x01EB8022	?	?	
A	?	?	0x0000A0F4	0x0000A0F4	
B	?	?	0x100F4F0A	0x100F4F0A	
ALU Result	0x004000C4	0x0042014C	0x1FF151EA	?	
ALU Out	?	0x004000C4	0x0042014C	0x1FF151EA	
ALU Zero	0	0	0	?	

Controlo					
PCSource	00	XX	XX	XX	
lrd	0	X	X	X	
PCWriteCond	X	0	0	0	
PCWrite	1	0	0	0	
RegWrite	0	0	0	1	
RegDst	X	X	X	1	
MemWrite	0	0	0	0	
MemtoReg	X	X	X	0	
MemRead	1	0	0	0	
IRWrite	1	0	0	0	
ALUOp	00	00	10	XX	
ALUSelB	01	11	00	XX	
ALUSelA	0	0	1	X	

Endereço ...	Dados ...	OpCode	Funct	Operação	... reg \$5 0x0045007C reg \$6 0x0000001C reg \$7 0x1001009C reg \$8 0x00000081 reg \$9 0x000000FF reg \$10 0xFFFFFFF ... \$PC 0x0040002C CPU	Endereço ...	Código ...
0x10010090	0x0021B500	0	0x20	add		0x0040002C	0x8CE50004
0x10010094	0x0021B581	0	0x22	sub		0x00400030	0x00A92824
0x10010098	0x0021B52F	0	0x24	and		0x00400034	0x10A80003
0x1001009C	0x0021B54E	0x02	0x25	or		0x00400038	0x00EA3820
0x100100A0	0x0021B5C3	0x04		j		0x0040003C	0x20C6FFFF
0x100100A4	0x0021B5FF	0x05		beq		0x00400040	0x14C0FFFA
		0x08		bne			
		0x0C		addi			
		0x23		andi			
		0x2b		lw			
				sw			
...	...					next:	...

1. **NOTA:** Use no máximo 40 palavras para responder a cada uma das 4 alíneas seguintes:
 - a) Descreva sucintamente as duas operações básicas que devem ser levadas a cabo pelo *datapath* sempre que ocorre uma excepção.
 - b) Na norma IEEE 754 existem valores reservados para o expoente que configuram casos particulares de codificação. Indique quais são e o seu significado.
 - c) Diga o que entende por “*forwarding*” e qual a sua utilidade (dê um exemplo ilustrativo).
 - d) Identifique o tipo a que pertence a instrução ‘**Iw \$2, 0(\$3)**’, os respectivos campos que a compõem e os respectivos valores.
2. Considere o *datapath single-cycle* que foi apresentado nas aulas teóricas. Admita os seguintes atrasos de propagação envolvidos nos vários elementos operativos e de estado:

Memória externa (dados e código): Leitura – 5ns; Escrita – 8ns

File Register: Leitura – 2ns; Escrita – 4ns *Sign Extend*: 1ns

ALU (qualquer operação): 3ns *Somadores*: 1ns *Outros dispositivos*: 0ns

 - a) Determine o tempo mínimo necessário à execução das seguintes três instruções:
add \$s0, \$s1, \$s2 lw \$t0, 20(\$a0) sw \$t9, 0(\$s0)
 - b) Determine, justificando adequadamente, a máxima frequência de relógio desta arquitectura.
3. Considere que o conteúdo dos registos **\$f6** e **\$f4** é respectivamente:

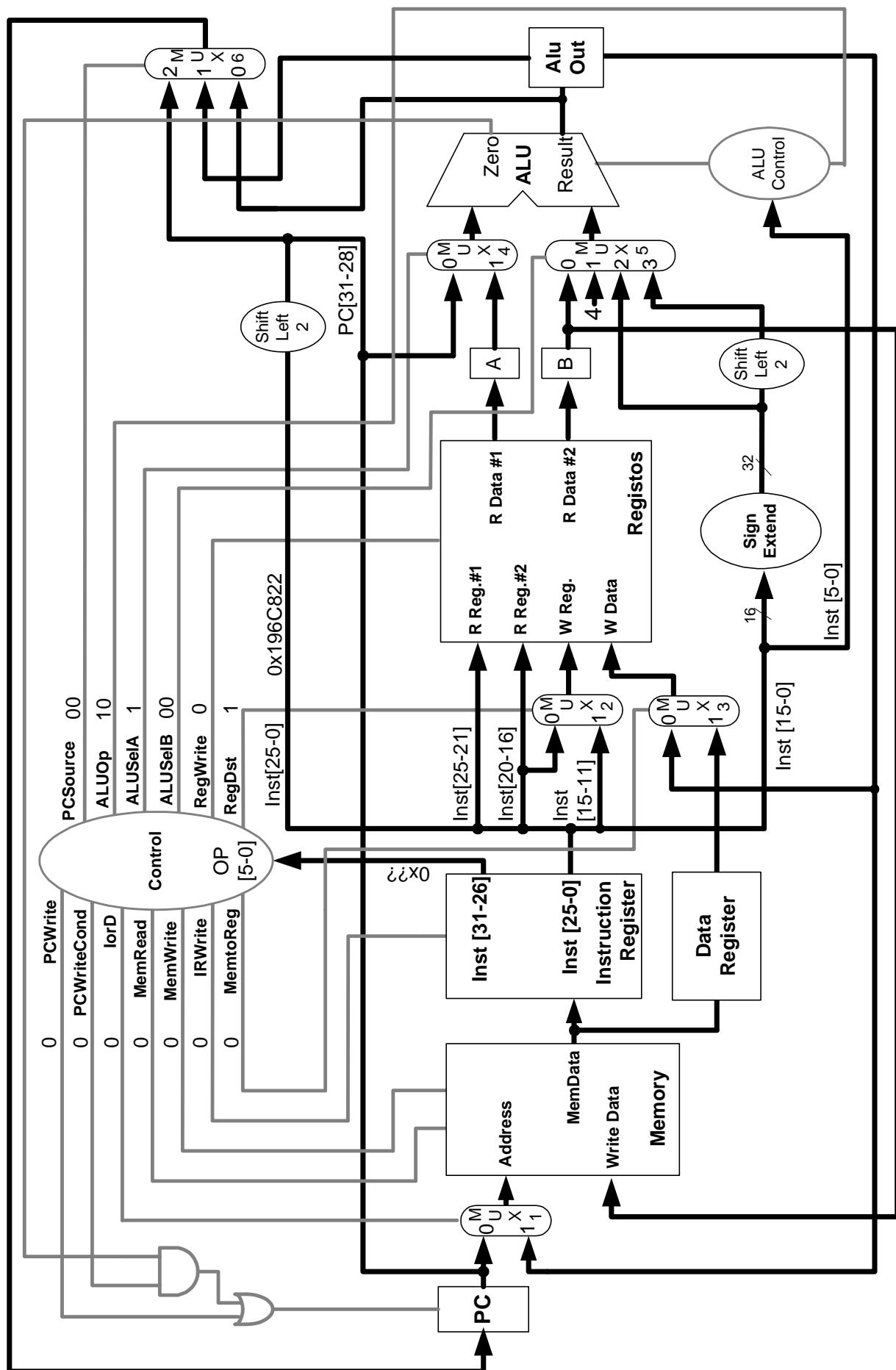
\$f6 = 0 00010010 01010000000000000000000000000000

\$f4 = 1 01101111 11000000000000000000000000000000

 - a) Obtenha a representação das quantidades armazenadas naqueles registos (codificadas segundo a norma IEEE 754) na forma $\pm m \times 2^{\exp}$, em que “m” e “exp” estão em decimal.
 - b) Determine o resultado da instrução **div.s \$f0, \$f6, \$f4**, realizando, em binário e/ou hexadecimal, os passos necessários à sua obtenção. Indique, em binário, qual o conteúdo do registo **\$f0** após a execução da instrução.
4. Considere, na **figura 2**, o trecho de código *Assembly* ali apresentado. Admita que o valor presente em **\$PC** corresponde ao *label* “**Ip1:**”, que nesse instante o conteúdo dos registos é o indicado, e que vai iniciar-se o “*instruction fetch*” da próxima instrução. Considere ainda o *datapath* e a unidade de controlo fornecido na próxima página, no pressuposto de que corresponde a uma implementação simplificada do MIPS de execução multi-ciclo sem *pipelining*.
 - a) Escreva em hexadecimal, nas duas colunas do lado direito da fig.2, o endereço em que se encontra armazenada cada uma das instruções do código fornecido, e o respectivo conteúdo, sabendo que a primeira linha corresponde à instrução presente no *label* “**Ip1:**”.
 - b) Considere que os valores indicados no *datapath* fornecido correspondem à “fotografia” tirada no decurso da execução de uma dada instrução. Identifique qual a instrução em causa, escreva a linha de código correspondente em *Assembly* do MIPS e determine em que fase de execução esta se encontra. **Justifique adequadamente a sua resposta.**
 - c) Preencha a tabela fornecida em anexo com o nome de cada uma das fases de execução da instrução “**addi \$10, \$10, -1**” e com o valor que tomam, em cada uma delas, os sinais e valores do *datapath* e os sinais de controlo ali indicados. Admita que o valor lógico “1” corresponde ao estado activo. (considere que \$10 contém o valor indicado na tabela da figura 2).

Não se esqueça de preencher o cabeçalho
 - d) Sabendo que a frequência do relógio do CPU é de 500 MHz, determine o tempo total que demora a executar o código fornecido, desde o instante inicial do *instruction fetch* da instrução presente em “**Ip1:**” até ao momento em que vai iniciar-se o *instruction fetch* da instrução presente em “**next:**”. **Justifique adequadamente a sua resposta.**

Cotações: 1a) a 1d) – 0,5; 2a) – 1,0; 2b) – 0,5; 3a) – 1,0; 3b) – 1,5; 4a) – 2,0; 4b) – 1,0; 4c) – 2,0; 4d) – 1,0



Nome: _____

Curso: _____ N° Mecanográfico: _____

				Endereço ...	Código ...
OpCode	Funct	Operação			
lp1: beq \$10, \$0, next	0	0x20	add	reg \$6	...
lw \$8, -0x3(\$9)	0	0x22	sub	reg \$7	0x000000FF
sub \$6, \$6, \$8	0x02		j	reg \$8	0xFFFFFFF0
addi \$10, \$10, -1	0x04		beq	reg \$9	0x80020C05
add \$9, \$9, \$7	0x08		addi	reg \$10	0x1001000B
j lp1	0x0D		ori	\$PC	0x00000003
next: ...	0x0F		lui		...
	0x23		lw		...
	0x2b		sw	CPU	...

Figura 2 (Problema 4)

Fase 1	Fase 2	Fase 3	Fase 4	Fase 5
--------	--------	--------	--------	--------

Nome da fase					

Datapath					
A					
B					
Data Register					
ALU Out					
ALU Result					
ALU Zero					

Controlo					
ALUSelA					
ALUSelB					
ALUOp					
IorD					
IRWrite					
MemRead					
MemtoReg					
MemWrite					
PCSource					
RegDst					
RegWrite					
PCWrite					
PCWriteCond					

PARTE II

NOTE BEM: Leia atentamente todas as questões, comente o código usando a linguagem C e respeite a convenção de passagem de parâmetros e salvaguarda de registos que estudou ☺. Respeite rigorosamente os aspectos estruturais e a sequência de instruções indicadas no código original fornecido, bem como as indicações, quando existirem, sobre quais os registos a usar para cada variável.

Considere o seguinte trecho de código em linguagem C:

```
int main (int argc, char* argv[])
{
    static int lista[10];
    int totalmin;      /* deve residir no registo $s2 */
    float media;       /* deve residir no registo $f20 */
    float varia;

    if ((argc > 0) && (argc < 11))
    {
        totalmin = analise (argc, argv, lista);

        (...)

        varia = variancia (argc, lista, media);

        print_str ("Variância: "); /* system call */
        print_float (varia);      /* system call */
        return 0;
    }
    else
    {
        print_str ("Erro");
        return 1;
    }
}
```

1. Codifique em *Assembly* do MIPS, a função **main**, sabendo que os protótipos das funções **analise** e **variancia** são os seguintes:

```
int analise (int num, char* palavras[], int* lista);

float variancia (int num, int* lista, float media);
```

Defina, no segmento de dados, as strings e as variáveis declaradas como static.

(v.s.f.f.)

2. Considerando que a função **contamin** tem o seguinte protótipo:

```
int contamin (char* str);
```

traduza para *Assembly* do MIPS, a função **análise**.

```
int analise (int num, char* palavras[], int* lista)
{
    int i;
    int min;
    int total = 0;

    for (i = 0; i < num; i++)
    {
        min = contamin (palavras[i]);
        lista[i] = min;
        total = total + min;
    }

    return total;
}
```

3. Traduza para *Assembly* do MIPS, a função **contamin**.

```
int contamin (char* str)
{
    int cont = 0;

    while (*str != '\0')
    {
        if ((*str >= 'a') && (*str <= 'z'))
        {
            cont++;
        }
        str++;
    }

    return cont;
}
```

4. Traduza para *Assembly* do MIPS, a função **variância**.

```
float variancia (int num, int* lista, float media)
{
    int i;
    float dif;
    float varia = 0.0;

    for (i = 0; i < num; i++)
    {
        dif = (float)lista[i] - media;
        varia = varia + (dif * dif);
    }

    return varia / (float)num;
}
```