

1º Parte

1- A quantidade de memória em bytes, reservada pelo conjunto das directivas da figura do lado correspondendo a (La-L1), é:

- a. 13
- b. 27
- c. 20
- d. 18

```
.align 4
L1: .ascii "AC1-2010"
    .align 4
L2: .word 0x2901, 0x10
L3: .space 3
L4:
```

2- 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.

3- Um endereço de memória externa num sistema computacional é:

- a. a gama de posições de memória que o 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.

4- Espaço de endereçamento de memória num sistema computacional é:

- a. Um numero único que identifica cada posição de memória.
- b. A gama total de posições de memória que o CPU pode referenciar.
- c. A informação armazenada em cada posição de memória.
- d. A gama de posições de memória efectivamente disponíveis no sistema

5- Numa memória com uma organização do tipo *byte-addressable*:

- a. A cada endereço está associado um dispositivo de armazenamento de 1 byte.
- b. Cada posição de memória é identificada com um endereço com a dimensão de 1 byte.
- c. O acesso apenas pode ser efectuado por instruções que transferem 1 byte de informação.
- d. Não é possível o armazenamento de quantidades com dimensão superior a 1 byte.

6- A arquitectura MIPS é caracterizada por:

- a. possuir 16 registos de uso geral de 32 bits cada.
- b. possuir um cpu capaz de realizar directamente operações aritméticas cujos operadores residem na memória externa.
- c. ser do tipo load-store.
- d. ter instruções de tamanho variável.

7- 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.

- 8- A arquitectura MIPS é do tipo “Load-Store”. Isso significa que:
- Os operandos das operações aritméticas e lógicas podem residir na memória externa.
 - Os operandos das operações aritméticas e lógicas apenas podem residir em registos internos.
 - As instruções de Load e Store apenas podem ser usadas imediatamente antes de operações aritméticas e lógicas.
 - Nesta arquitectura foi dada especial importância à implementação das instruções Load e Store, de forma a não comprometer o desempenho global.
- 9- Na arquitectura MIPS, os campos de uma instrução do tipo “R” designam-se por:
- “opcode”, “rs”, “rt” e “imm”.
 - “opcode” e “address”.
 - “opcode”, “rs”, “rt”, “rd”, “shamt” e “imm”.
 - nenhuma das anteriores.
- 10- Na arquitectura MIPS os campos de uma instrução tipo “I” designam-se por:
- opcode, rs, rt e offset/imm
 - opcode, rs, rt, shamt e funct
 - ☒ opcode, rs, rt, rd e offset/imm
 - opcode, rs, rt, rd, shamt e funct
- 11- Nas instruções de acesso à memória da arquitectura MIPS é utilizado o modo de endereçamento:
- ☒ indirecto por registo.
 - registo.
 - imediato.
 - directo.
- 12- No MIPS, a instrução de salto incondicional indirecto através de registo:
- ☒ É codificada usando o formato de codificação R.
 - É codificada usando o formato de codificação L.
 - É codificada usando o formato de codificação J.
 - A instrução em causa não existe.
- 13- Quando um endereço se obtém da adição do conteúdo de um registo com um offset constante:
- diz-se que estamos perante um endereçamento imediato.
 - diz-se que estamos perante um endereçamento directo a registo com offset.
 - ☒ diz-se que estamos perante um endereçamento indirecto a registo com deslocamento.
 - diz-se que estamos perante um endereçamento indirecto relativo a PC
- 14- O formato de instruções tipo “I” da arquitectura MIPS é usado nas instruções de:
- salto condicional.
 - aritméticas em que somente um dos operandos está armazenado num registo.
 - acesso à memória de dados externa.
 - ☒ todas as anteriores.
- 15- Na instrução **lb** da arquitectura MIPS, o operando é obtido através de endereçamento:
- relativo ao PC com deslocamento.
 - imediato
 - directo
 - ☒ indirecto a registo com deslocamento.

16- Nas instruções tipo R da arquitectura MIPS é utilizado o modo de endereçamento,

- a. indirecto por registo
- b. registo.**

17 – O modo de endereçamento utilizado na instrução `sb $8,-8($s8)` é:

- a. Relativo.
- b. Imediato.
- c. Indirecto por registo com deslocamento.**
- d. Absoluto por registo com deslocamento.

18- 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, $t1, 0x1001"`.
- b. `"ori $t0, $1, 0x1001"` seguida de `"ori $t0, $s1, 0x2345"`.
- c. `"lui $1, 0x1001"` seguida de `"ori $t0, $t1, 0x2345"`**
- d. `"ori $t0, $1, 0x2345"` seguida de `"lui $1, 0x1001"`.

19- A instrução virtual `"bgt $t8, $t9, target"` da arquitectura MIPS decompõe-se na seguinte sequência de instruções nativas:

- a. `"slt $1, $t8, $t9"` seguida de `"bne $1, $0, target"`.
- b. `"slt $1, $t9, $t8"` seguida de `"bne $1, $0, target"`**
- c. `"slt $1, $t8, $t9"` seguida de `"beq $1, $0, target"`
- d. `"slt $1, $t9, $t8"` seguida de `"beq $1, $0, target"`

20- A instrução virtual `bgt $8,0x16,target` da arquitectura MIPS decompõe-se na seguinte sequência de instruções nativas.

- a. `slti $1,$8,0x17` seguida de `bne $1,$0,target`.
- b. `slti $1,$8,0x16` seguida de `bne $1,$0,target`.**
- c. `slti $1,$8,0x17` seguida de `beq $1,$0,target`.
- d. `stli $1,$8,0x16` seguida de `beq $1,$0,target`.

21- A instrução virtual `bge $t8,$t9,target` da arquitectura MIPS decompõem-se na seguinte sequência de instruções nativas:

- a. `slt $1,$t8,$t9` seguida de `bne $1,$0,target`.
- b. `slt $1,$t9,$t8` seguida de `bne $1,$0,target`.
- c. `slt $1,$t8,$t9` seguida de `beq $1,$0,target`.**
- d. `slt $1,$t9,$t8` seguida de `beq $1,$0,target`.

22- A instrução virtual `ble $8,0x16, target` da arquitectura MIPS decompõe-se na seguinte sequência de instruções nativas:

- a. `Slti $1, $8, 0x17` seguida de `bne $1, $0, target`.
- b. `Slti $1, $8, 0x16` seguida de `bne $1, $0, target`.
- c. `Slti $1, $8, 0x17` seguida de `beq $1, $0, target`.
- d. `Slti $1, $8, 0x16` seguida de `beq $1, $0, target`.

23- A instrução virtual `div $20,$21,$22` da arquitectura MIPS decompõe-se na seguinte sequência de instruções nativas:

- a. `div $21,$22` seguida de `mfhi $20`.
- b. `div $21,$22` seguida de `mflo $20`.**
- c. `div $20,$21` seguida de `mfhi $22`.
- d. `div $20,$21` seguida de `mflo $22`.

24- A instrução `div $2,$1,$2` da arquitectura MIPS decompõe-se na seguinte sequência de instruções nativas:

- a. `Div $1,$2` seguida de `mflo $2`.
- b. `Div $1,$2` seguida de `mfhi $2`.
- c. `Div $1,$2` seguida de `mtlo $2`.
- d. A instrução referida já é nativa.

25- A instrução virtual `la $t0,lable` da arquitectura MIPS, em que `lable` corresponde ao segundo endereço do segmento de dados do PCSPIM, decompõem-se na seguinte sequência de instruções nativas

- a. `lui $1,0x1001` seguida de `ori $t0,$1,0x0001`.
- b. `ori $t0,$1,0x001` seguida de `lui $1,0x1001`.
- c. `lui $1,0x0040` seguida de `ori $t0,$1,0x001`.
- d. `ori $t0,$1,0x0001` seguida de `lui $1,0x0040`.

26- Considerando que `$5=0xFFFFFFFF7` e `$10=0x00000002`, o valor armazenado no registo destino pela instrução virtual `rem $6, $5, $10` é:

- a. `$6=0x00000004`.
- b. `$6=0xFFFFFFFF4`.
- c. `$6=0x00000001`.
- d. `$6=0xFFFFFFFF`.

27- 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:

- a. `0xFF`.
- b. `0x82`.
- c. `0xFFFFFFFF82`.
- d. `0xFF82`.

28- Considere que no endereço de memória acedido pela instrução `lb $9,0xC7 ($9)` está armazenado o valor `0x83`, e que no registo `$9` está armazenado, antes da sua execução o valor `0x1001FF00`. O valor que ficará no registo `$9`, no final da execução da instrução é:

- a. `0x83`.
- b. `0x1001FFC7`.
- c. `0xC7`.
- d. `0xFFFFFFFF83`.

29- Na arquitectura MIPS o endereço-alvo de uma instrução de salto condicional ("`beq/bne`") armazenada no endereço `0x00400032` cujo código original é `0x13ABFFFD` é:

- a. `0x00400034`
- b. `0x0040FFF4`
- c. `0x00400018`
- d. `0x0040001C`

30- Considere uma instrução de salto condicional residente no endereço `0x004038AC`, cujo código máquina é `0x1185FFF0`. O endereço-alvo dessa instrução é:

- a. `0x0040386C`.
- b. `0x004038A0`.
- c. `0x00403870`.
- d. `0x0041389C`.

31- Os endereços mínimo e máximo para os quais uma instrução “bne” presente no endereço 0x00430210 pode saltar são:

- a. 0x00000000, 0xFFFFFFFF.
- b. 0x00000000, 0x0FFFFFFC.
- c.** 0x00428214, 0x00438213.
- d. 0x00410214, 0x00450210.

32- Os endereços mínimo e máximo para os quais uma instrução de salto condicional (beq ou bne) da arquitectura MIPS, presente no endereço 0x0043FFFC pode saltar são:

- a. 0x0041FFFC, 0x0045FFFB.
- b. 0x00437FFC, 0x00447FFB.
- c.** 0x00420000, 0x0045FFFC.
- d. 0x00438000, 0x00447FFF.

33- Os endereços mínimo e máximo para os quais uma instrução “j” presente no endereço 0x00430210 pode saltar são:

- a. 0x00428214, 0x00438213.
- b. 0x00000000, 0xFFFFFFFF
- c. 0x00410214, 0x00450210.
- d.** 0x00000000, 0x0FFFFFFC.

34- 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.

35- A instrução jal label executa sequencialmente as seguintes operações:

- a.** PC=PC+4, \$ra=PC, PC=label.
- b. PC=PC+4, PC=label, \$ra=PC.
- c. \$ra=PC, PC=PC+4, PC=label.
- d. \$ra=PC, PC=label, PC=PC+4

36- A instrução “jal funct” executa sequencialmente as seguintes operações:

- a.** \$PC = \$PC + 4, \$ra = \$PC, \$PC = funct.
- b. \$PC = \$pc + 4, \$PC = funct, \$ra = \$PC.
- c. \$ra = \$PC, \$PC = funct.
- d. Nenhuma das anteriores.

37- A instrução jalr \$5 (jump and link on register) executa sequencialmente as seguintes operações:

- a. PC=PC+4, \$ra=PC, PC=\$5.
- b. PC=PC+4, \$5=PC, PC=\$ra.
- c. \$5=PC, PC=PC+4, \$ra=PC.
- d. \$ra=PC, PC=PC+4, PC=\$5.

38- 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 +- (32*4) KBytes para as instruções “sw”
- c. permite armazenar um offset de endereçamento de +- 32KBytes para as instruções “beq”
- d. permite armazenar um offset de endereçamento de +-(32*4) Kilo instruções para as instruções “beq”

39- Segundo a convenção de utilização de registos na arquitectura MIPS, uma subrotina tem de preservar os registos

- a. \$s0... \$s7, \$v0, \$v1.
- b. \$s0... \$s7, \$a0... \$a3
- c. \$a0... \$a3, \$ra
- d. \$s0... \$s7, \$ra

40- Segundo a convenção de utilização de registos da arquitectura MIPS, uma subrotina não necessita de salvaguardar os registos com os prefixos:

- a. \$a, \$v, \$s.
- b. \$s, \$v, \$t.
- c. \$a, \$v, \$t.
- d. \$a, \$s, \$t.

41- O trecho de código que permite atribuir o valor 0xFF à variável “i” indirectamente através do ponteiro “p” é:

<p>a.</p> <pre>int i; int *p; i = &p; *i = 0xFF;</pre>	<p>b.</p> <pre>int i; int *p; p = *i; *p = 0xFF;</pre>	<p>c.</p> <pre>int i; int *p; p = &i; *p = 0xFF;</pre>	<p>d.</p> <pre>int i; int *p=0xFF; p = &i; I = *p;</pre>
---	---	--	---

42- Na arquitectura MIPS a stack é gerida de acordo com os seguintes princípios:

- a. cresce no sentido dos endereços mais altos, apontando o registo \$sp para a última posição ocupada.
- b.** cresce no sentido dos endereços mais baixos, apontando o registo \$sp para a última posição ocupada.
- c. cresce no sentido dos endereços mais altos, apontando o registo \$sp para a primeira posição livre
- d. cresce no sentido dos endereços mais baixos, apontando o registo \$sp para a primeira posição livre.

43- Numa arquitectura MIPS a stack é gerida de acordo com os seguintes princípios:

- a. cresce no sentido dos endereços mais altos, apontando o registo \$sp para a última posição ocupada.
- b. cresce no sentido dos endereços mais altos, apontando o registo \$sp para a primeira posição livre.
- c. cresce no sentido dos endereços mais baixos, apontando o registo \$sp para a primeira posição livre.
- d. cresce no sentido dos endereços mais baixos, apontando o registo \$sp para a última posição ocupada.

44- Na convenção adoptada pela arquitectura MIPS, a realização de uma operação 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).

45 – Na convenção adoptada pela arquitectura MIPS, a realização de uma operação de pop do valor do registo \$ra é realizado 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. addu \$sp,\$sp,4 seguida de sw \$ra,0(\$sp).
- d. sw \$ra, 0(\$sp) seguida de addu \$sp,\$sp,4

46- A detecção de overflow numa operação de adição de números sem sinal faz-se através:

- a. da avaliação do bit mais significativo do resultado
- b. do “ou” exclusivo entre o carry in e o carry out da célula de 1 bit mais significativa.
- c. do “ou” exclusivo entre os 2 bits mais significativos do resultado.
- d. da avaliação do carry out do bit mais significativo do resultado

47- 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.

48- Numa ALU, a detecção de overflow nas operações de adição algébrica é efectuada 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 o bit mais significativo e o menos significativo do resultado.
- d. do “ou” exclusivo entre os 2 bits mais significativos do resultado.

49- Para a implementação de uma arquitectura de multiplicação de 32 bits são necessários, entre outros, registos para o multiplicador e multiplicando e uma ALU. A dimensão exacta, em bits, de cada um destes elementos deve ser:

- a. Multiplicando: 32 bits; Multiplicador: 32 bits; ALU: 64 bits
- b. Multiplicando: 32 bits; Multiplicador: 64 bits; ALU: 32 bits.
- c. Multiplicando: 64 bits; Multiplicador: 32 bits; ALU: 32 bits.
- d. Multiplicando: 64 bits; Multiplicador: 32 bits; ALU: 64 bits

50- Numa implementação de uma arquitectura de divisão de 32 bits pode recorrer-se a um algoritmo que, sem para alterar o registo que armazena o divisor:

- a. Faça deslocamentos sucessivos do quociente à esquerda, mantendo o dividendo.
- b. Faça deslocamentos sucessivos do quociente à esquerda e do dividendo à direita.
- c. Faça deslocamentos sucessivos do quociente à esquerda e do dividendo à esquerda
- d. Faça deslocamentos sucessivos do quociente à direita e do dividendo à esquerda.

51- Aplicando o algoritmo de Booth, o produto das quantidades 101010×100011 pode ser obtido através da seguinte soma algébrica:

- a. $-(101010 \times 2^1) + (101010 \times 2^1)$.
- b. $+(101010 \times 2^0) - (101010 \times 2^2) + (101010 \times 2^5)$.
- c. $+(101010 \times 2^1) - (101010 \times 2^4)$.
- d. $-(101010 \times 2^0) + (101010 \times 2^2) - (101010 \times 2^5)$

52- A decomposição numa sequência de adições e subtracções, de acordo com o algoritmo de Booth, da quantidade binária $101101_{(2)}$ é:

- a. $-2^0 + 2^1 - 2^2 + 2^4 - 2^5$
- b. $2^0 - 2^1 + 2^2 - 2^4 + 2^5$
- c. $2^0 - 2^1 + 2^2 + 2^3 - 2^4 + 2^5$
- d. $-2^0 + 2^1 - 2^2 - 2^3 + 2^4 - 2^5$

53- A decomposição numa sequência de adições e subtracções, de acordo com o algoritmo de Booth, da

Quantidade binária $010110_{(2)}$:

- a. $-2^1 + 2^3 - 2^4 + 2^5$.
- b. $+2^0 - 2^2$.
- c. $+2^1 - 2^3 + 2^4 - 2^5$
- d. $-2^0 + 2^2$.

54- A quantidade real binária $1011,11000000_{(2)}$ quando representada em decimal é igual a:

- a. 12,6
- b. 11,75
- c. 3008,0
- d. 1504,0

55- Imagina que se pretende inicializar o conteúdo do registo \$f4 com a quantidade real 2.0. A sequência de instruções que efectua esta operação é:

a.	b.	c.	d.
li \$t2, 2	lui \$t0, 0x4000	li.s \$f4, 2.0	li \$t0, 2
mtc1 \$t0, \$f4	mtc1 \$t0, \$f4	cvt.s.w \$f4, \$f0	mtc1 \$t0, \$f0
			mov.s \$f4, \$f0

56 – O resultado da instrução multu \$t0,\$t1 é representável em 32 bits se:

- a. HI for uma extensão com sinal de LO
- b. HI=0x00000000.
- c. HI for diferente de zero.
- d. HI=0xFFFFFFFF.

57- 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 = 0xFFFFFEC, LO = 0xFFFFFFFF.
- d. HI = 0x00000000, LO = 0xFFFFFEC.

58- Considerando que \$t0=-7 e \$t1=2, o resultado da instrução div \$t0,\$t1 é:

- a. LO=-3, HI=-1.
- b. LO=-3, HI= 1.
- c. LO= 1, HI=-4.
- d. LO=-1, HI=-3.

59- Considerando que \$t0=0x00000007 e \$t1=0xFFFFFFFF, o resultado da instrução div \$t0,\$t1 é:

- a. Hi= 0x00000001, LO=0xFFFFFFFF0.
- b. HI=0xFFFFFFFF0, LO=0xFFFFFFFF.
- c. HI=0xFFFFFEC, LO=0x00000001.
- d. Nenhuma das anteriores.

60- 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".

61- 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

62- Considerando que \$f2=0x3A600000 e \$fa=0xBA600000, o resultado da instrução sub.s \$f0,\$f2,\$f4 é:

- a. \$f0=0x39E00000.
- b. \$f0=0x3AE00000.
- c. \$f0=0x00000000.
- d. \$f0=0x80000000.

63- Considere que no endereço de memória acessido pelas instruções lb \$t0,0xFF(\$t0) e lb \$t1,0xFF(\$t0) está armazenado o valor 0x82, o valor armazenado nos respectivos registos destino, no final da execução dessas instruções é: **0x82**

- a. \$t0=0x000000FF, \$t1=0xFFFFFFFF
- b. \$t0=0x00000082, \$t1=0xFFFFFFFF82
- c. \$t0=0xFFFFFFFF82, \$t1=0x00000082
- d. \$t0=0xFFFFFFFF, \$t1=0x000000FF

64- Assumindo que o registo \$f8 possui o valor (representado em hexadecimal) 0x3FE00000, após a execução da instrução cvt.d.s \$f10,\$f8, os registos \$f10 e \$f11 terão, respectivamente, os valores:

- a. 0x00000000, 0x3FFC0000.
- b. 0x00000000, 0x3FE00000.
- c. 0x00000000, 0x07FC0000.
- d. 0x07FC0000, 0x00000000.

extender 32 bits → 64 bits

65- Admitindo que \$f8=0x00000000 e \$f9=0x618A0000, após a execução da instrução cvt.s.d \$f10,\$f8, o registo \$f10 terá o valor (assuma que \$f9 contém a parte mais significativa do operando):

- a. \$f10=0x7F800000.
- b. \$f10=0x61D00000.
- c. \$f10=0x0C500000.
- d. Nenhuma das anteriores.

66- Considere que os valores reais representados nos registos \$f4 e \$f6 são (em base 2) $\$f4=1,00011010 \times 2^2$ e $\$f6=-1,10101000 \times 2^{-2}$. O valor armazenado no registo \$f0, em hexadecimal, após a execução da instrução add.s \$f0,\$f4,\$f6 é:

- a. 0x40708000.
- b. 0x40F28000.
- c. 0x40650000.
- d. 0xBE920000.

67- 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. $-0,1625 \times 2^1$.
- b. $-0,1625 \times 2^3$.
- c. $-3,25 \times 2^1$.
- d. $-16,25 \times 2^1$.

68- A codificação do número $+1,13125 \times 10^1$ no formato IEEE 754, precisão simples, representado em hexadecimal é:

- a. 0x415A8000.
- b. 0x41350000.**
- c. 0x3E350000.
- d. 0x01DA8000.

69- A representação normalizada e arredondada para o par mais próximo de acordo com o formato IEEE 754 precisão simples do número $100,110110000000000000010110_2$ é:

- a. $1,001101100000000000000110 = 2^3$.
- b. $1,001101100000000000000110 = 2^{-3}$.
- c. $1,001101100000000000000101 = 2^3$.
- d. $1,000000000000000000000000 = 2^3$.



70- Considere duas máquinas (A e B) com implementações distintas da mesma arquitectura do conjunto de instruções (ISA). A máquina A possui uma duração de sinal de relógio de 0,5ns e a máquina B de 0,4ns. Para um dado programa a máquina A apresenta um CPI de 2,0 e a B de 3,0.

- a. A máquina A é mais rápida do que a máquina B por um factor de 1,25.
- b. A máquina A é mais rápida do que a máquina B por um factor de 1,2.
- c. A máquina B é mais rápida do que a máquina A por um factor de 1,25.
- d. A máquina B é mais rápida do que a máquina A por um factor de 1,2.

2º Parte - Datapath

71- Numa implementação single-cycle da arquitectura MIPS:

- a. 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.
- b. Existem registos à saída dos elementos operativos fundamentais para guardar valores a utilizar no ciclo de relógio seguinte.
- c. Todas as operações de leitura e escrita são síncronas com o sinal de relógio.
- d. Existem memórias específicas para código e dados para possibilitar o acesso a ambos os tipos de informação num único ciclo de relógio.

72- A frequência de relógio de uma implementação single cycle da arquitectura MIPS:

- a. É limitada pelo maior dos tempos de atraso dos elementos operativos Memória, ALU e *File Register*.
- b. Varia em função da instrução que está a ser executada.
- c. É limitada pelo maior dos atrasos cumulativos dos elementos operativos envolvidos na execução da instrução mais longa.
- d. É limitada pelo menor dos tempos de atraso dos elementos operativos Memória, ALU e *File Register*.

73 – Numa implementação single-cycle da arquitectura MIPS:

- a. Existem memórias independentes para código e dados para possibilitar o acesso a ambos os tipos de informação num único ciclo relógio.
- b. Existe uma única ALU para realizar todas as operações aritméticas e lógicas (incluído o cálculo do valor do PC, BTA, endereços de acesso á memória e comparação de registos) necessários para (executar) num único ciclo de relógio qualquer uma das instruções suportadas
- c. Existe uma única memória acedida para código e e para dados em ciclos de relógio distintos.
- d. Todas as operações de leitura e escrita são síncronas com o sinal de relógio

74- A unidade de controlo de uma implementação multi-cycle da arquitectura MIPS:

- a. é um elemento combinatório que gera os sinais de controlo em função do campo opcode do código máquina da instrução.
- b. é uma máquina de estados em que o primeiro e o segundo estados são comuns à execução de todas as instruções.
- c. é uma máquina de estados com um número de estados igual ao número de fases da instrução mais longa.
- d. é um elemento combinatório que gera os sinais de controlo em função do campo *funct* do código máquina da instrução.

75- 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, pelo ordem indicada, para:

- a. calcular o valor do *Branch Target Address* e comparar os registos (operandos da instrução)
- b. calcular o valor de PC+4 e comparar os registos (operandos da instrução).
- c. comparar os registos (operandos da instrução) e calcular o valor do *Branch Target Address*.
- d. calcular o valor de PC+4 e o valor do *Branch Target Address*.

76- Uma implementação *pipelined* de uma arquitectura possui, relativamente a uma implementação *single-cycle* da mesma, a vantagem de:

- a. Diminuir o tempo de execução de cada uma das instruções.
- b. Permitir a execução de uma nova instrução a cada novo ciclo de relógio
- c. Aumentar o débito de execução das instruções.
- d. Todas as anteriores.

77- A frequência de relógio de uma implementação *pipelined* da arquitectura MIPS:

- a. É limitada pelo maior dos atrasos cumulativos dos elementos operativos envolvidos na execução da instrução mais longa.
- b. É definida de forma a evitar *stalls*, assim como *delay slots*.
- c. É limitada pelo menor dos tempos de atraso dos elementos operativos Memória, ALU e *File Register*.
- d. É limitada pelo maior dos tempos de atraso dos elementos operativos Memória, ALU e *File Register*.

78- A técnica de *forwarding/bypassing* num processador MIPS *pipelined* permite:

- a. Utilizar como operando de uma instrução um resultado produzido por outra instrução que se encontra numa etapa mais recuada do *pipeline*.
- b. Trocar a ordem de execução das instruções de forma a resolver um *hazard* de dados.
- c. Utilizar como operando de uma instrução um resultado produzido por outra instrução que se encontra numa etapa mais avançada do *pipeline*.
- d. Escrever o resultado de uma instrução no *File Register* antes de ela chegar à etapa WB

79- 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:

- a. 32,25 MHz (T=31ns).
- b. 25,00 MHz (T=40ns).
- c. 29,41 MHz (T=34ns).
- d. 31,25 MHz (T=32ns).

Memórias externas: leitura - 9ns, escrita - 11ns; File register: leitura - 3ns, escrita - 4ns; Unidade de controlo: 2ns; ALU (qualquer operação): 7ns; Somadores: 4ns; Outros: 0ns
--

$T_{lw} = T_{ReadMem} + T_{ReadFileReg} + T_{ALU} + T_{ReadMem} + T_{WriteFileReg}$

80 - 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 100MHz relativamente a uma *single-cycle* a operar a 20 MHz é de:

- a. 1,25.
- b. 1.
- c. 5.
- d. 0,8.

81 - Um hazard de controlo numa implementação *pipelined* de um processador ocorre quando:

- a. 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*.
- b. 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.
- c. Existe uma dependência entre o resultado calculado por uma instrução e o operando usado por outra que segue mais atrás do *pipeline*.
- d. Por azar, a unidade de controlo desconhece o *opcode* da instrução que se encontra na etapa ID.

82 – O seguinte trecho de código, a executar sobre uma implementação *pipelined* da arquitectura MIPS, apresenta os seguintes *hazards*:

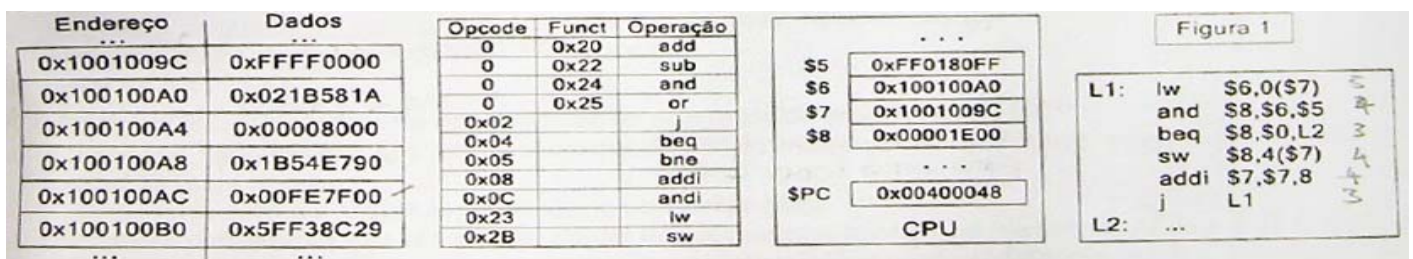
- Um *hazard* de controlo na quarta instrução e um *hazard* de dados na segunda instrução que pode ser resolvido por *forwarding*.
- Um *hazard* estrutural na primeira instrução e um *hazard* de controlo na quarta instrução.
- Um *hazard* de controlo na quarta instrução e *hazards* de dados na segunda, 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*.

```
L1: lw    $t0, 0($t1)    # 1
      Add  $t2, $t3, $t4  # 2
      Or   $t1, $t2, $t0  # 3
      Beq  $t5, $t1, L1   # 4
```

83 – 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:

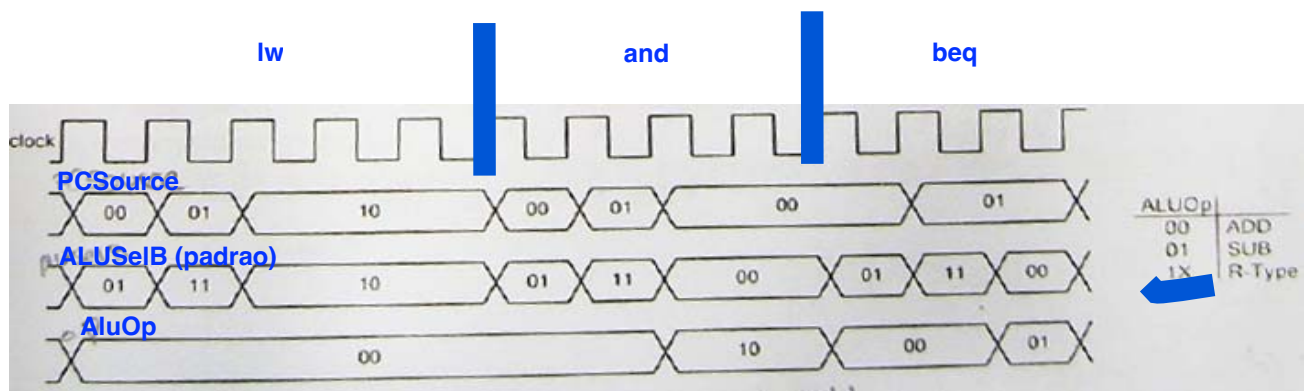
- lw** \$6,0x2020(\$5) na terceira fase.
- add** \$4,\$5,\$6 na quarta fase.
- add** \$4,\$5,\$6 na terceira fase.
- lw** \$6,0x2020(\$5) na quinta fase.

Considere o trecho de código apresentado na Figura 1, bem como as tabelas os valores dos registos que ai 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).



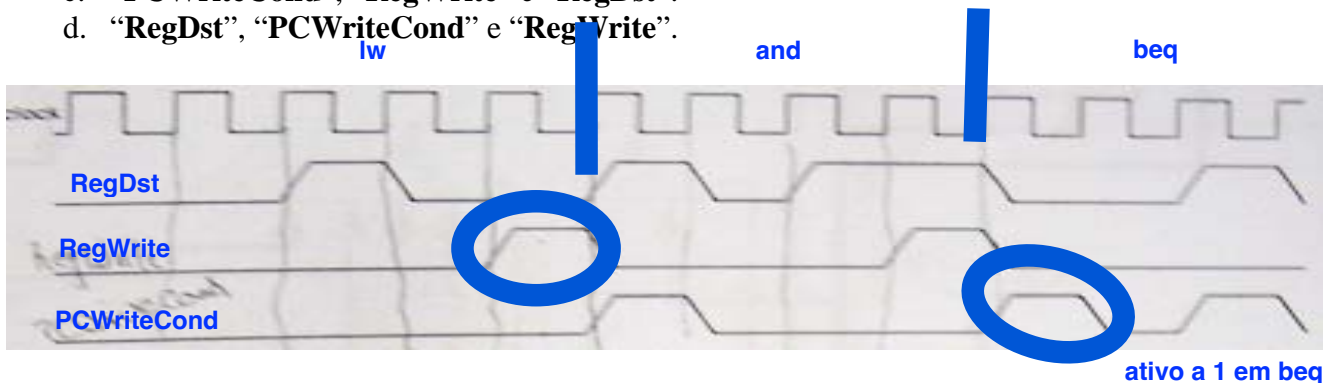
84 – Para as 3 primeiras instruções do trecho de código apresentado na Figura 1, os sinais de controlo representados no seguinte diagrama temporal correspondem, pela ordem indicada, a:

- “ALUSelB”, “ALUOp” e “PCSource”.
- “PCSource”, “ALUOp” e “ALUSelB”.
- “PCSource”, “ALUSelB” e “ALUOp”.**
- “ALUSelB”, “PCSource” e “ALUOp”.



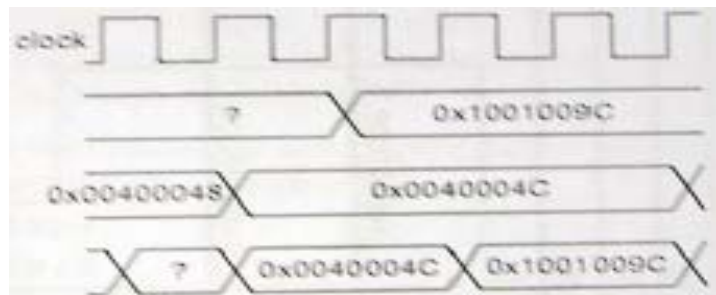
85- Também para as 3 primeiras instruções do trecho de código apresentado na Figura 1, os sinais de controlo representados no seguinte diagrama temporal correspondem, pela ordem indicada, a:

- “RegWrite”, “PCWriteCond” e “RegDst”.
- “RegDst”, “RegWrite” e “PCWriteCond”.
- “PCWriteCond”, “RegWrite” e “RegDst”.
- “RegDst”, “PCWriteCond” e “RegWrite”.



86 – 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”, “InstRegister” e “PC”.
- “B”, “PC” e “ALUOut”.
- “A”, “PC” e “ALUOut”.
- Nenhuma das anteriores.



87 – 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 *multicycle* 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.”);

- 58 ciclos de relógio.
- 12 ciclos de relógio.
- 6 ciclos de relógio.
- 35 ciclos de relógio.

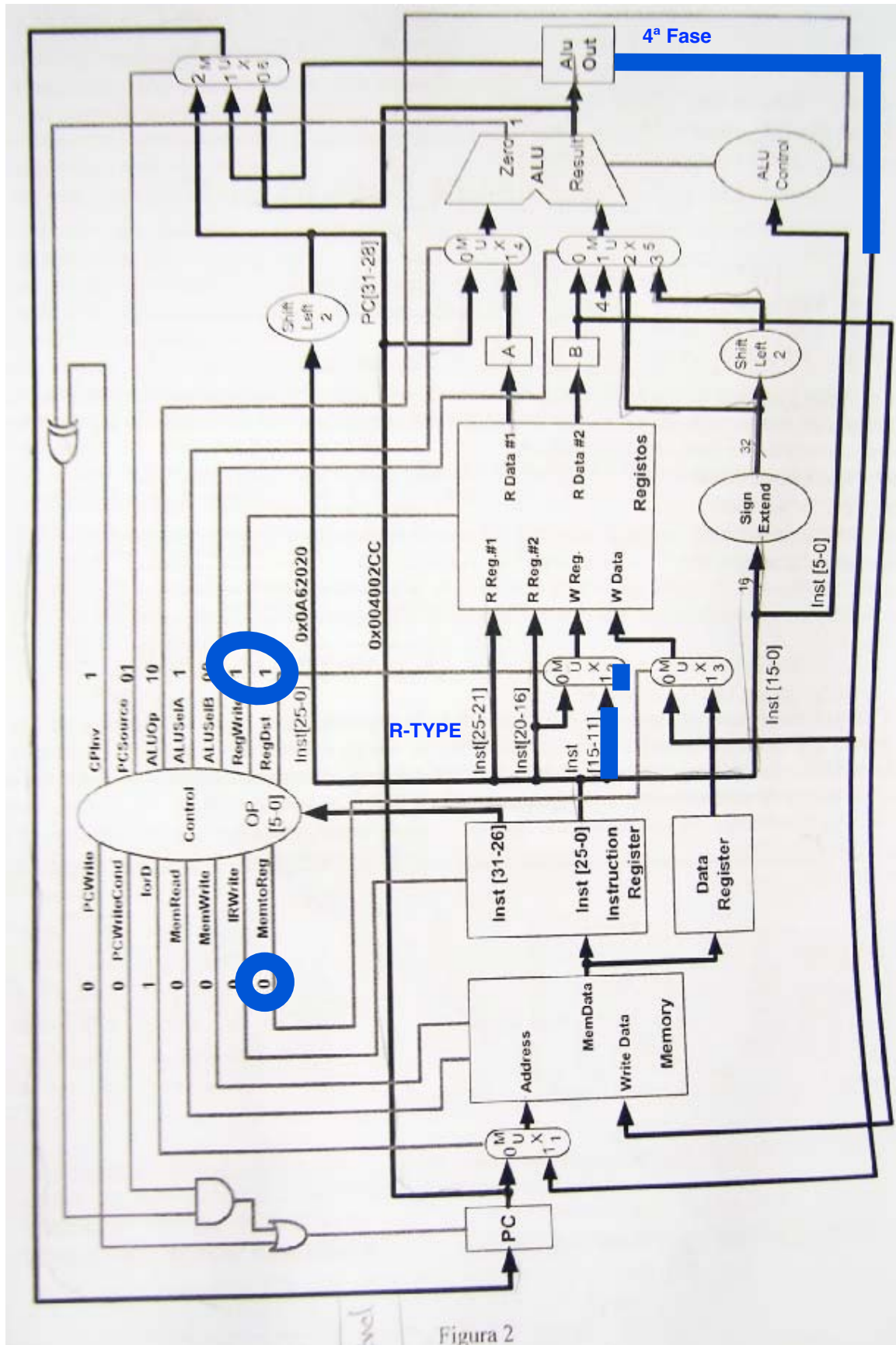


Figura 2