Tópicos:

- Introdução Sistemas de Computação de Uso Geral
- A arquitetura MIPS

Questões:

1. Quais são os 3 blocos fundamentais de um sistema computacional?
2. Quais são os 3 principais blocos funcionais que integram um CPU?
3. Qual a função do registo *Program Counter*?
4. Quais os passos mais importantes em que se decompõe a execução de uma instrução no CPU?
5. Descreva de forma sucinta a função de um compilador.
6. Descreva de forma sucinta a função de um assembler.
7. Quantos registos internos de uso geral tem o MIPS?
8. Qual a dimensão, em bits, que cada um dos registos internos do MIPS pode armazenar?
9. Qual a sintaxe, em *Assembly*, de uma instrução aritmética no MIPS?
10. O que distingue a instrução SRL da instrução SRA do MIPS?
11. Se **`$5=0x81354AB3`**, qual o resultado, expresso em hexadecimal, das instruções:
    a. **`srl $3,$5,1`**
    a. **`sra $4,$5,1`**
12. *System calls*:
    a. O que é uma *system call*?
    b. No MIPS, qual o registo usado para identificar a *system call* a executar?
    c. Qual o registo ou registos usados para passar argumentos para as *systems calls*?
    d. Qual o registo usado para obter o resultado devolvido por uma *system call* nos casos em que isso se aplica?
13. Em Arquitetura de Computadores, como definiria o conceito de endereço?
14. O que é o espaço de endereçamento de um processador?
15. Como se organiza internamente um processador? Quais são os blocos fundamentais da secção de dados? Para que serve a unidade de controlo?
16. Qual é o conceito fundamental por detrás do modelo de arquitetura "*stored-program*"?
17. Como se codifica uma instrução? Que informação fundamental deverá ter o código de uma instrução?
18. Descreva pelas suas próprias palavras o conceito de **ISA**.

19. Quantas e quais são as classes de instruções que agrupam as diferentes instruções de uma dada arquitetura?

20. O que carateriza e distingue as arquiteturas do tipo "*register-memory*" e "*load-store*"? De que tipo é a arquitetura MIPS?

21. O ciclo de execução de uma instrução é composto por uma sequência ordenada de operações. Quantas e quais são essas operações (passos de execução)?

22. Como se designa o barramento que permite identificar, na memória, a origem/destino da informação transferida?

23. Qual a finalidade do barramento normalmente designado por *Data Bus*?

24. Os processadores da arquitetura hipotética ZWYZ possuem 4 registos internos e todas as instruções são codificadas em 24 bits. Num dos formatos de codificação existem 5 campos: um *OpCode* com 5 bits, três campos para identificar registos internos em operações aritméticas e lógicas e um campo para codificar valores constantes imediatos em complemento para dois. Qual a gama de representação destas constantes?

25. A arquitetura hipotética ZPTZ tem um barramento de endereços de 32 bits e um barramento de dados de 16 bits. Se a memória desta arquitetura for ***bit_addressable***:
    a. Qual a dimensão do espaço de endereçamento desta arquitetura?
    b. Qual a dimensão máxima da memória suportada por esta arquitetura expressa em *bytes*?

26. Considere agora 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. Tendo o espaço de endereçamento do processador 24 bits, qual a dimensão máxima de memória que este sistema pode acomodar expresso em *bytes*?

27. Relativamente à arquitetura MIPS:
    c. Com quantos bits são codificadas as instruções no MIPS?
    d. O que diferencia o registo **`$0`** dos restantes registos de uso geral?
    e. Qual o endereço do registo interno do MIPS a que corresponde a designação lógica **`$ra`**?

28. No MIPS, um dos formatos de codificação de instruções é designado por R:
    a. Quais os campos em que se divide este formato de codificação?
    b. Qual o significado de cada um desses campos?
    c. Qual o valor do campo *opCode* nesse formato?
    d. O que faz a instrução cujo código máquina é: **`0x00000000`**?

29. O símbolo ” >> “ da linguagem C significa deslocamento à direita e é traduzido por SRL ou SRA (no caso do MIPS). Em que casos é que o compilador gera um SRL e quando é que gera um SRA?

30. Qual a instrução nativa do MIPS em que é traduzida a instrução virtual "**`move $4,$15`**"?

31. Determine o código máquina das seguintes instruções (verifique a tabela na última página):

    a. xor $5,$13,$24

    b. sub $25,$14,$8

    c. sll $3,$9,7

    d. sra $18,$9,8

32. Traduza para instruções *Assembly* do MIPS a seguinte expressão aritmética, supondo **x** e **y** são inteiros e residentes em **$t2** e **$t5**, respetivamente (apenas pode usar instruções nativas e não deverá usar a instrução de multiplicação):

```
y = -3 * x + 5;
```

33. Traduza para instruções *assembly* do MIPS o seguinte trecho de código:

```
int a, b, c;                //a:$t0, b:$t1, c:$t2
unsigned int x, y, z;       //x:$a0, y:$a1, z:$a2
z = x >> 2 + y;
c = a >> 5 - 2 * b;
```

34. Considere que as variáveis **g**, **h**, **i** e **j** são conhecidas e podem ser representadas por uma variável de 32 bits num programa em C. Qual a correspondência em linguagem C às seguintes instruções:

```
a. add  h, i, j          #

b. addi j, j, 1          #
   add  h, g, j          #
```

35. Assumindo que **g=1**, **h=2**, **i=3** e **j= 4** qual o valor destas variáveis no final das sequências das alíneas da questão anterior?

36. Qual a operação realizada pela instrução "**slt**" e quais os resultados possíveis?

37. Qual o valor armazenado no registo **$1** na execução da instrução "**slt $1, $3, $7**", admitindo que:

    a. $3=5 e $7=23

    b. $3=0xFE e $7=0x913D45FC

38. Com que registo implícito comparam as instruções "**bltz**", "**blez**", "**bgtz**" e "**bgez**"?

39. Decomponha em instruções nativas do MIPS as seguintes instruções virtuais:

    a. blt $15,$3,exit

    b. ble $6,$9,exit

    c. bgt $5,0xA3,exit

    d. bge $10,0x57,exit

    e. blt $19,0x39,exit

    f. ble $23,0x16,exit

40. Na tradução e C para *assembly*, quais as principais diferenças entre um ciclo "**while(…){…}**" e um ciclo "**do{…}while(…);**" ?

41. Traduza para *assembly* do MIPS os seguintes trechos de código de linguagem C (admita que **a**, **b** e **c** residem nos registos **$4**, **$7** e **$13**, respetivamente):

```
a. if(a > b && b != 0)
     c = b << 2;
   else
     c = (a & b) ^ (a | b);


b. if(a > 3 || b <= c)
     c = c - (a + b);
   else
     c = c + (a - 5);
```

42. Qual o modo de endereçamento usado pelo MIPS para ter acesso a palavras residentes na memória externa?

43. Na instrução "**lw $3,0x24($5)**" qual a função dos registos **$3** e **$5** e da constante **0x24**?

44. Qual é o formato de codificação das instruções de acesso à memória no MIPS e qual o significado de cada um dos seus campos?

45. Qual a diferença entre as instruções "**sw**" e "**sb**"?

46. O que distingue as instruções "**lb**" e "**lbu**"?

47. O que acontece quando uma instrução **lw/sw** acede a um endereço que não é múltiplo de 4?

48. Traduza para *assembly* do MIPS os seguintes trechos de código de linguagem C (atribua registos internos para o armazenamento das variáveis **i** e **k**) :

```
a. int i, k;
   for(i=5, k=0; i < 20; i++, k+=5);

b. int i=100, k=0;
   for( ; i >= 0; )
   {
     i--;
     k -= 2;
    }

c. unsigned int k=0;
   for( ; ; )
   {
     k += 10;
   }

d. int k=0, i=100;
   do
   {
     k += 5;
   } while(--i >= 0);
```

49. Sabendo que o *OpCode* da instrução "**`lw`**" é **`0x23`**, determine o código máquina, expresso em hexadecimal, da instrução "**`lw $3,0x24($5`**)".

50. Suponha que a memória externa foi inicializada, a partir do endereço **`0x10010000`**, com os valores **`0x01,0x02,0x03,0x04,0x05`** e assim sucessivamente. Suponha ainda que **`$3=0x1001`** e **`$5=0x10010000`**. Qual o valor armazenado no registo destino após a execução da instrução "**`lw $3,0x24($5)`**" admitindo uma organização de memória *little endian*?

51. Considere as mesmas condições da questão anterior. Qual o valor armazenado no registo destino pelas instruções:

    a. **`lbu $3,0xA3($5)`**

    b. **`lb  $4,0xA3($5)`**

52. Quantos *bytes* são reservados em memória por cada uma das diretivas:

    a. **`L1: .asciiz "Aulas5&6T"`**

    b. **`L2: .byte 5,8,23`**

    c. **`L3: .word 5,8,23`**

    d. **`L4: .space 5`**

53. Desenhe esquematicamente a memória e preencha-a com o resultado das diretivas anteriores admitindo que são interpretadas sequencialmente pelo *Assembler*.

54. Supondo que "**`L1:`**" corresponde ao endereço inicial do segmento de dados, e que esse endereço é **`0x10010000`**, determine os endereços a que correspondem os *labels* "**`L2:`**", "**`L3:`**" e "**`L4:`**".

55. Suponha que "**`b`**" é um *array* declarado como "**`int b[25];`**":

    a. Como é obtido o endereço inicial do *array*, i.e., o endereço a partir do qual está armazenado o seu primeiro elemento?

    b. Supondo uma memória "*byte-addressable*", como é obtido o endereço do elemento "**`b[6]`**"?

56. O que é codificado no campo offset do código máquina das instruções "**`beq/bne`**" ?

57. A partir do código máquina de uma instrução "**`beq/bne`**", como é formado o endereço-alvo (*Branch Target Address*)?

58. Qual o formato de codificação de cada uma das seguintes instruções: "**`beq/bne`**", "**`j`**", "**`jr`**"?

59. A partir do código máquina de uma instrução "**`j`**", como é formado o endereço-alvo (*Jump Target Address*)?

60. Dada a seguinte sequência de declarações:

```
int b[25];
int a;
int *p = b;
```

Identifique qual ou quais das seguintes atribuições permitem aceder ao elemento de índice 5 do *array* "**`b`**":

`a = b[5];`	`a = *p + 5;`	`a = *(p + 5);`	`a = *(p + 20);`

61. Assuma que as variáveis **`f, g, h, i`** e **`j`** correspondem aos registos **`$t0`**, **`$t1`**, **`$t2`**, **`$t3`** e **`$t4`** respetivamente. Considere que o endereço base dos *arrays* **A** e **B** está contido nos registos **`$s0`** e **`$s1`**. Considere ainda as seguintes expressões:

```
f = g + h + B[2]

j = g – A[B[2]]
```

a. Qual a tradução para *assembly* de cada uma das instruções C indicadas?

b. Quantas instruções *assembly* são necessárias para cada uma das instruções C indicadas? E quantos registos auxiliares são necessários?

c. Considerando a tabela seguinte que representa o conteúdo byte-a-byte da memória, nos endereços correspondentes aos *arrays* A e B, indique o valor de cada elemento dos *arrays* assumindo uma organização *little endian*.

Endereço	Valor
A+12	…
A+11	0x00
A+10	0x00
A+9	0x00
A+8	0x01
A+7	0x22
A+6	0xED
A+5	0x34
A+4	0x00
A+3	0x00
A+2	0x00
A+1	0x00
A+0	0x12

Endereço	Valor
B+12	…
B+11	0x00
B+10	0x00
B+9	0x00
B+8	0x02
B+7	0x00
B+6	0x00
B+5	0x50
B+4	0x02
B+3	0xFF
B+2	0xFF
B+1	0xFF
B+0	0xFE

A[0]=
A[1]=
A[2]=

B[0]=
B[1]=
B[2]=

d. Assumindo que **g = -3** e **h = 2**, qual o valor final das variáveis **f** e **j**?

62. Pretende-se escrever uma função para a troca do conteúdo de duas variáveis (troca(a, b);).
Isto é, se, antes da chamada à função, **`a=2`** e **`b=5`**, então, após a chamada à função, os valores de **`a`** e **`b`** devem ser: **`a=5`** e **`b=2`**

Uma solução incorreta para o problema é a seguinte:

```
void troca(int x, int y)
{
     int aux;
     aux = x;
     x = y;
     y = aux;
}
```

Identifique o erro presente no trecho de código e faça as necessárias correções para que a função tenha o comportamento pretendido

63. Na instrução "**`jr $ra`**", como é obtido o endereço-alvo?

64. Qual é o menor e o maior endereço para onde uma instrução "**`j`**", residente no endereço de memória **`0x5A18F34C`**, pode saltar?

65. Qual é o menor e o maior endereço para onde uma instrução "**`beq`**", residente no endereço de memória **`0x5A18F34C`**, pode saltar?

66. Qual é o menor e o maior endereço para onde uma instrução "**`jr`**", residente no endereço de memória **`0x5A18F34C`** pode saltar?

67. Qual a gama de representação da constante nas instruções aritméticas imediatas?

68. Qual a gama de representação da constante nas instruções lógicas imediatas?

69. Por que razão não existe, no ISA do MIPS, uma instrução que permita manipular diretamente uma constante de 32 bits?

70. Como é que, no *assembly* do MIPS, se podem manipular constantes de 32 bits?

71. Apresente a decomposição em instruções nativas das seguintes instruções virtuais:

```
a. li      $6,0x8B47BE0F
b. xori    $3,$4,0x12345678
c. addi    $5,$2,0xF345AB17
d. beq     $7,100,L1
e. blt     $3,0x123456,L2
```

72. O que é uma sub-rotina?

73. Qual a instrução do MIPS usada para saltar para uma sub-rotina?

74. Por que razão não pode ser usada a instrução "j" para saltar para uma sub-rotina?

75. Quais as operações que são sequencialmente realizadas na execução de uma instrução "**`jal`**"?

76. Qual o nome virtual e o número do registo associado à execução dessa instrução?

77. No caso de uma sub-rotina ser simultaneamente chamada e chamadora (sub-rotina intermédia) que operações é obrigatório realizar nessa sub-rotina?

78. Qual a instrução usada para retornar de uma sub-rotina?

79. Que operação fundamental é realizada na execução dessa instrução?

80. O que é uma *stack* e qual a finalidade do *stack pointer*?

81. Como funcionam as operações de ***push*** e ***pop***?

82. Por que razão as *stacks* crescem normalmente no sentido dos endereços mais baixos?

83. Quais as regras para a implementação em software de uma *stack* no MIPS?

84. Qual o registo usado, no MIPS, como *stack pointer*?

85. De acordo com a convenção de utilização de registos no MIPS:
    a. Que registos são usados para passar parâmetros e para devolver resultados de uma sub-rotina?
    b. Quais os registos que uma sub-rotina pode livremente usar e alterar sem necessidade de prévia salvaguarda?
    c. Quais os registos que uma sub-rotina não pode alterar?
    d. Quais os registos que uma sub-rotina chamadora tem a garantia que a sub-rotina chamada não altera?
    e. Em que situação devem ser usados registos "**$sn**"?
    f. Em que situação devem ser usados os restantes registos: **$tn**, **$an** e **$vn**?

86. De acordo com a convenção de utilização de registos do MIPS:
    a. Que registos podem ter que ser copiados para a stack numa sub-rotina intermédia?
    b. Que registos podem ter que ser copiados para a stack numa sub-rotina terminal?

87. Para a função com o protótipo seguinte indique, para cada um dos parâmetros de entrada e para o valor devolvido, qual o registo do MIPS usado para a passagem dos respetivos valores:

```
char fun(int a, unsigned char b, char *c, int *d);
```

88. Para uma codificação em complemento para 2, apresente a gama de representação que é possível obter com **3**, **4**, **5**, **8** e **16** bits (indique os valores-limite da representação em binário, hexadecimal e em decimal com sinal e módulo).

89. Traduza para *assembly* do MIPS a seguinte função "**fun1()**", aplicando a convenção de passagem de parâmetros e salvaguarda de registos:

```
char *fun2(char *, char);

char *fun1(int n, char *a1, char *a2)
{
      int j = 0;
      char *p = a1;

      do
      {
            if((j % 2) == 0)
            fun2(a1++, *a2++);
      } while(++j < n);
      *a1='\0';
      return p;
}
```

90. Determine a representação em complemento para 2 com 16 bits das seguintes quantidades:

    5, -3, -128, -32768, 31, -8, 256, -32

91. Determine o valor em decimal representado por cada uma das quantidades seguintes, supondo que estão codificadas em complemento para 2 com 8 bits:

    0b00101011, 0xA5, 0b10101101, 0x6B, 0xFA, 0x80

92. Determine a representação das quantidades do exercício anterior em hexadecimal com 16 bits (também codificadas em complemento para 2).

93. Como é realizada a deteção de *overflow* em operações de adição com quantidades sem sinal?

94. Como é realizada a deteção de *overflow* em operações de adição com quantidades com sinal (codificadas em complemento para 2)?

95. Considere os seguintes pares de valores em **$s0** e **$s1**:

    i. $s0 = 0x70000000 $s1 = 0x0FFFFFFF
    ii. $s0 = 0x40000000 $s1 = 0x40000000

    a. Qual o resultado produzido pela instrução **add $t0, $s0, $s1**?

    b. Para a alínea anterior os resultados são os esperados ou ocorreu *overflow*?

    c. Qual o resultado produzido pela instrução **sub $t0, $s0, $s1**?

    d. Para a alínea anterior os resultados são os esperados ou ocorreu *overflow*?

    e. Qual o resultado produzido pelas instruções:

    ```
    add $t0, $s0,$s1
    add $t0, $t0,$t1 ?
    ```

    f. Para a alínea anterior os resultados são os esperados ou ocorreu *overflow*?

96. Para a multiplicação de dois operandos de "**m**" e "**n**" bits, respetivamente, qual o número de bits necessário para o armazenamento do resultado?

97. Apresente a decomposição em instruções nativas das seguintes instruções virtuais:

    a. **mul $5,$6,$7**

    b. **la $t0,label c/ label = 0x00400058**

    c. **div $2,$1,$2**

    d. **rem $5,$6,$7**

    e. **ble $8,0x16,target**

    f. **bgt $4,0x3F,target**

98. Determine o resultado da instrução **mul $5,$6,$7**, quando

    $6=0xFFFFFFFE e $7=0x00000005.

99. Determine o resultado da execução das instruções nativas das instruções virtuais **div $5,$6,$7 e rem $5,$6,$7** quando **$6=0xFFFFFFF0 e $7=0x00000003**

100. Admita que pretendemos executar, em *Assembly* do MIPS, as operações:

     $t0 = $t2/$t3 e **$t1 = $t2 % $t3**.

     Escreva a sequência de instruções em *Assembly* que permitem realizar estas duas operações. Use apenas instruções nativas

101. Descreva as regras que são usadas, na ALU do MIPS, para realizar uma divisão inteira entre duas quantidades com sinal.

102. Considerando que **$t0=-7** e **$t1=2**, determine o resultado da instrução **div $t0,$t1** e o valor armazenada respetivamente nos registos **HI** e **LO**.

103. Repita o exercício anterior admitindo agora que **$t0=0xFFFFFFF9** e **$t1=0x00000002.**

104. Considerando que **$5=-9** e **$10=2**, determine o valor que ficará armazenado no registo destino pela instrução virtual **rem $6, $5, $10**.

105. Para a implementação de uma arquitetura de multiplicação de 32 bits são necessários, entre outros, registos para o multiplicador e multiplicando, e ainda uma ALU. Determine a dimensão exata, em bits, de cada um destes três elementos funcionais.

106. As duas sub-rotinas seguintes permitem detetar *overflow* nas operações de adição com e sem sinal, no MIPS. Analise o código apresentado e determine o resultado produzido, pelas duas sub-rotinas, nas seguintes situações:

a. $a0=0x7FFFFFF1, $a1=0x0000000E;

b. $a0=0x7FFFFFF1, $a1=0x0000000F;

c. $a0=0xFFFFFFF1, $a1=0xFFFFFFFF;

d. $a0=0x80000000, $a1=0x80000000;

```
# Overflow detection, signed
# int isovf_signed(int a, int b);
isovf_signed: ori $v0,$0,0
              xor $1,$a0,$a1
              slt $1,$1,$0
              bne $1,$0,notovf_s
              addu $1,$a0,$a1
              xor $1,$1,$a0
              slt $1,$1,$0
              beq $1,$0,notovf_s
              ori $v0,$0,1
notovf_s:     jr $ra

# Overflow detection, unsigned
# int isovf_unsigned(unsigned int a, unsigned int b);
isovf_unsig: ori $v0,$0,0
             nor $1,$a1,$0
             sltu $1,$1,$a0
             beq $1,$0,notovf_u
             ori $v0,$0,1
notovf_u:    jr $ra
```

107. As duas sub-rotinas anteriores podem ser também escritas alternativamente com o código abaixo. A abordagem á ligeiramente diferente. No caso de operações sem sinal, o *overflow* pode ser detetado para as operações de soma e subtração. Analise o código apresentado e determine o resultado produzido, pelas duas sub-rotinas, nas condições indicadas nas alíneas da questão anterior:

```
# Overflow detection in addition, unsigned
# int isovf_unsigned_plus(unsigned int a, unsigned int b);
isovf_unsig_plus:
            ori $v0,$0,0
            addu $t2, $a0, $a1    # temp = A + B;
            bge  $t2, $t0, notovf_uadd
            bge  $t2, $t1, notovf_uadd
            ori  $v0, $0,1
notovf_uadd: jr $ra


# Overflow detection in subtraction, unsigned
# int isovf_unsigned_sub(unsigned int a, unsigned int b);
isovf_unsig_sub:
            ori $v0,$0,0
            slt $1, $a0, $a1
            beq $1, $0, notovf_usub
            ori  $v0, $0,1
notovf_usub: jr $ra

# Overflow detection, signed
# int isovf_signed(int a, int b);
isovf_signed:
            add  $1, $a0, $a1       # res = a + b;
            xor  $a1, $a0, $a1      # tmp = a ^ b;
            bltz $a1, notovf_s      # if (tmp < 0) no_ovf();
            xor  $a1, $1, $a0       # tmp = res ^ a;
            bgez $t3, notovf_s      # if (tmp >= 0) no_ovf();
            ori $v0,$0,1
notovf_s:   jr $ra
```

108. Ainda no código das sub-rotinas das questões anteriores, qual a razão para não haver salvaguarda de qualquer registo na stack?

109. Na conversão de uma quantidade codificada em formato IEEE 754, precisão simples, para decimal, qual o número máximo de casas decimais com que o resultado deve ser apresentado?

110. Responda à questão anterior admitindo que o valor original se encontra agora representado com precisão dupla no formato IEEE 754.

111. Determine a representação em formato IEEE 754, precisão simples, da quantidade real $19{,}1875_{10}$. Determine a representação da mesma quantidade em precisão dupla.

112. Determine, em decimal (vírgula fixa), o valor das quantidades representadas em formato IEEE 754, precisão simples, como:

a. 0xC19AB000.

b. 0x80580000.

113. Considere que o conteúdo dos dois seguintes registos da FPU representam a codificação de duas quantidades reais no formato IEEE754 precisão simples:

```
$f0 = 0x416A0000
$f2 = 0xC0C00000
```

Calcule o resultado das instruções seguintes, apresentando o seu resultado em hexadecimal:

```
a. abs.s   $f4,$f2          # $f4 = abs($f2)
b. neg.s   $f6,$f0          # $f6 = neg($f0)
c. sub.s   $f8, $f0,$f2     # $f8 = $f0 – $f2
d. sub.s   $f10,$f2,$f0     # $f10 = $f2 – $f0
e. add.s   $f12,$f0,$f2     # $f12 = $f0 + $f2
f. mul.s   $f14,$f0,$f2     # $f14 = $f0 * $f2
g. div.s   $f16,$f0,$f2     # $f16 = $f0 / $f2
h. div.s   $f18,$f2,$f0     # $f18 = $f2 / $f0
i. cvt.d.s $f20,$f2         # Convert single to double
j. cvt.w.s $f22,$f0         # Convert single to integer
```

114. Considere a sequência de duas instruções Assembly:

```
lui  $t0,0xC0A8
mtc1 $t0,$f8
```

qual o valor que ficará armazenado no registo **$f8,** expresso em base dez e vírgula fixa, admitindo uma interpretação em IEEE 754 precisão simples?

115. Considerando que **$f2=0x3A600000** e **$f4=0xBA600000**, determine o resultado armazenado em **$f0** pela instrução **sub.s $f0,$f2,$f4**.

116. Repita o exercício anterior admitindo agora as seguintes condições:

$f4=0x3F100000 e **$f6=0x408C0000** e a instrução **add.s $f8,$f4,$f6.**

$f2=0x3F900000 e **$f4=0xBEA00000** e a instrução **mul.s $f0,$f2,$f4**

$f2=0x258c0000 e **$f4=0x41600000** e a instrução **div.s $f0,$f2,$f4**

117. Numa norma hipotética KPT de codificação em vírgula flutuante, a mantissa normalizada após a realização de uma operação aritmética tem o valor **1.1111 1111 1111 1110 1000 0000**. Qual será o valor final da mantissa (com 16 bits na parte fracionária) após arredondamento para o ímpar mais próximo?

118. Assuma que **x** é uma variável do tipo **float** residente em **$f8** e que o *label* **endWhile** corresponde ao endereço da primeira instrução imediatamente após um ciclo *while()*. Se a avaliação da condição para executar o *loop* for *while (x > 1.5){..}* escreva, em Assembly do MIPS, a sequencia de instruções necessárias para determinar esta condição.

119. Determine, de acordo com o formato IEEE 754 precisão simples, a representação normalizada, e arredondada para o par mais próximo, do número $\mathbf{100,1101100000000000000101}\mathbf{10}_2$.

120. Numa implementação *single cycle* da arquitetura MIPS, a frequência máxima de operação é de 2GHz (para os atrasos de propagação a seguir indicados). Determine o atraso máximo que pode ocorrer nas operações da ALU.

Memórias externas: leitura – 175ps, escrita – 150ps; *File register*: leitura – 25ps, Escrita – 35ps;
Unidade de Controlo: 10ps; Somadores: 50ps; Outros: 0ns; Escrita noutros elementos de estado: 20ps.
Setup time dos elementos de estado: 5ps

121. Determine, numa implementação *single-cycle* da arquitetura MIPS, a frequência máxima de operação imposta pela instrução "sw", assumindo os atrasos a seguir indicados:

Memórias externas: leitura – 8ns, escrita – 10ns; File register: leitura – 2ns, Escrita – 3ns;
Unidade de Controlo: 1ns; ALU (qualquer operação): 6ns; Somadores: 4ns; Outros: 0ns.
Escrita noutros elementos de estado: 2ns; Setup time dos elementos de estado: 1ns

122. Determine, numa implementação *single-cycle* da arquitetura MIPS, a frequência máxima de operação imposta pela instrução "beq", assumindo os atrasos a seguir indicados, é:

Memórias externas: leitura – 6ns, escrita – 8ns; File register: leitura – 2ns, Escrita – 3ns;
Unidade de Controlo: 1ns; ALU (qualquer operação): 6ns; Somadores: 4ns; Outros: 0ns.
Escrita noutros elementos de estado: 1ns; Setup time dos elementos de estado: 0.5ns

123. Determine, numa implementação *single cycle* da arquitetura MIPS, o período mínimo do sinal de relógio imposto pelas instruções tipo R, assumindo os atrasos a seguir indicados, é:

Memórias externas: leitura – 9ns, escrita – 11ns; File register: leitura – 3ns, Escrita – 4ns;
Unidade de Controlo: 2ns; ALU (qualquer operação): 7ns; Somadores: 4ns; Outros: 0ns.
Escrita noutros elementos de estado: 2ns; Setup time dos elementos de estado: 1ns

124. Identifique os principais aspetos que caracterizem uma arquitetura *single cycle*, quer do ponto de vista do modelo da arquitetura, como das características da sua unidade de controlo.

125. Numa implementação *single cycle* da arquitetura MIPS, no decurso da execução de uma qualquer instrução, a que corresponde o valor presente na saída do registo PC?

126. Preencha a tabela seguinte, para as instruções indicadas, com os valores presentes à saída da unidade de controlo principal da arquitetura *single cycle* dada nas aulas.

Instrução	Opcode	ALUOp[1..0]	Branch	RegDst	ALUSrc	Memto Reg	Reg Write	Mem Read	Mem WRite
lw	100011								
sw	101011								
addi	001000								
slti	001010								
beq	'000100								
R - Format	000000								

127. Admita que na versão *single cycle* do CPU MIPS dado nas aulas, pretendíamos acrescentar o suporte das instruções **`jal address`** e **`jr $reg`**. Esquematize as alterações que teria de introduzir no *datapath* para permitir a execução destas instruções (use o esquema da próxima página.

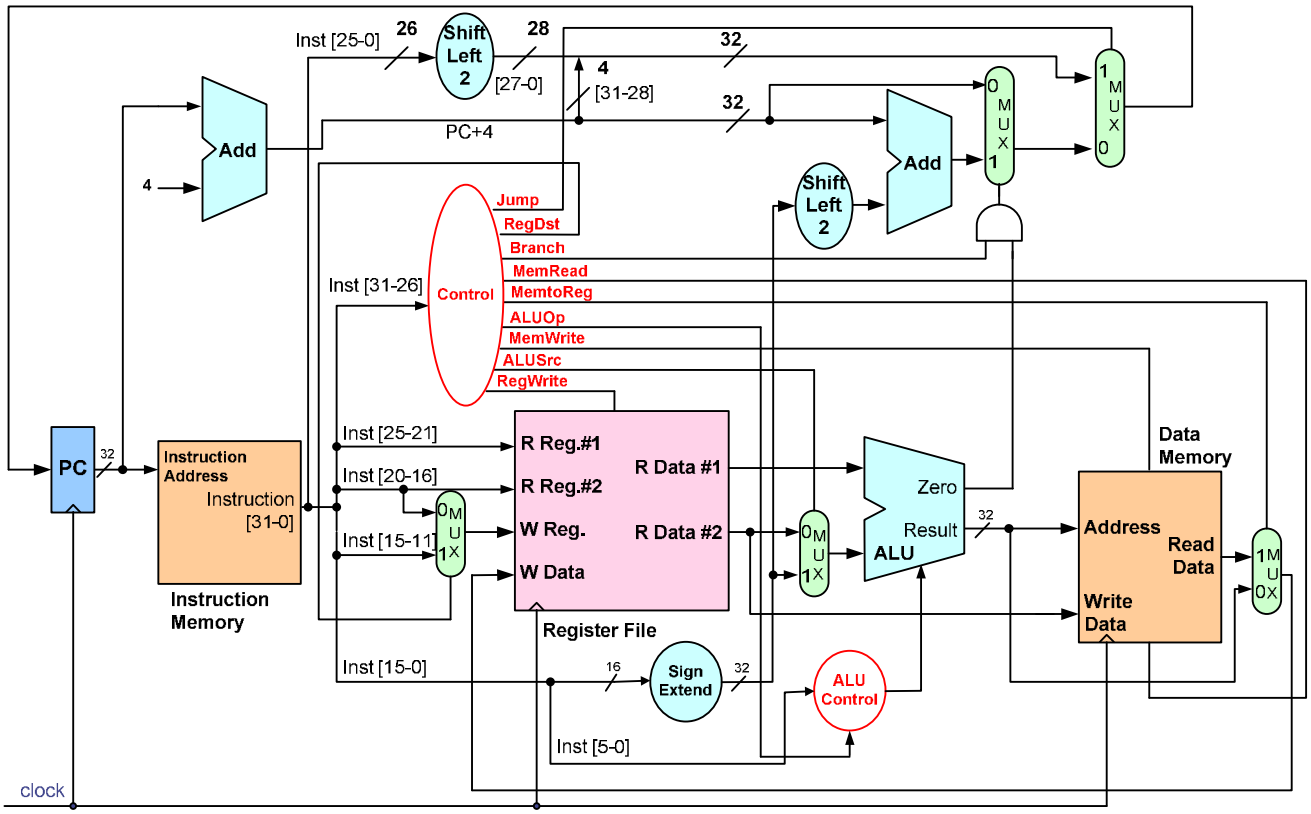


Fig. 1 - Datapath single-cycle

128. Admita que na versão *single cycle* do CPU MIPS, pretendíamos executar a instrução **`slt $3,$5,$9`**. Descreva por palavras suas como é esta instrução realizada ao nível da ALU, e qual o conteúdo final no registo **`$3`**, admitindo que **`$5=0xFF120008`** e **`$9=0x00C00FFF`**.

129. Suponha que os tempos de atraso introduzidos pelos vários elementos funcionais de um *datapath single-cycle* são os seguintes:

Acesso à memória para leitura (tRM):	6ns	Acesso à memória para preparar escrita (tWM):	4ns
Acesso ao register file para leitura (tRRF):	4ns	Acesso ao register file para preparar escrita (tWRF):	2ns
Operação da ALU (tALU):	5ns	Operação de um somador (tADD):	2ns
Multiplexers e restantes elementos funcionais:	0ns	Unidade de controlo (tCNTL):	2ns
Tempo de setup do PC (tstPC):	1ns		

a. Determine o tempo mínimo para execução das instruções tipo **`R`**, **`LW`**, **`SW`**, **`BEQ`** e **`J`**.

b. Calcule a máxima frequência do relógio que garanta uma correta execução de todas as instruções.

130. Suponha agora que dispunha de uma tecnologia que que o período de relógio podia ser adaptado instrução a instrução, em função da instrução em curso. Determine qual o ganho de eficiência que poderia obter com esta tecnologia face a uma tecnologia em que a frequência do relógio é a que obteve na questão anterior (admita os mesmos atrasos de propagação). Para isso, assuma que o programa de *benchmarking* tem a seguinte distribuição de ocorrência de instruções:

15% de **`lw`**, 15% de **`sw`**, 40% de tipo **`R`**, 20% de **`branches`** e 10% de **`jumps`**

131. Ainda para os tempos utilizados nas duas questões anteriores, determine qual a máxima frequência de trabalho no caso de o *datapath* ser do tipo *multi-cycle*.

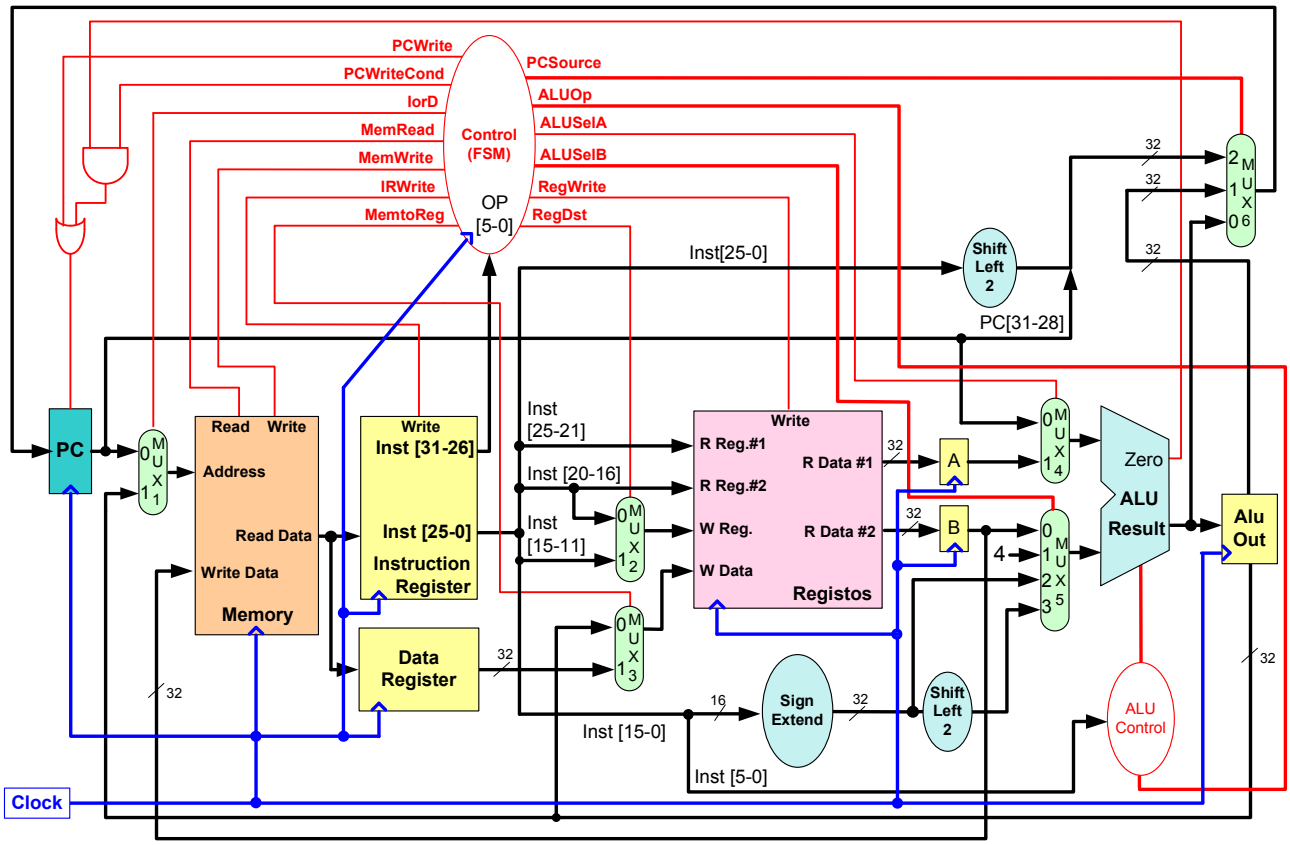
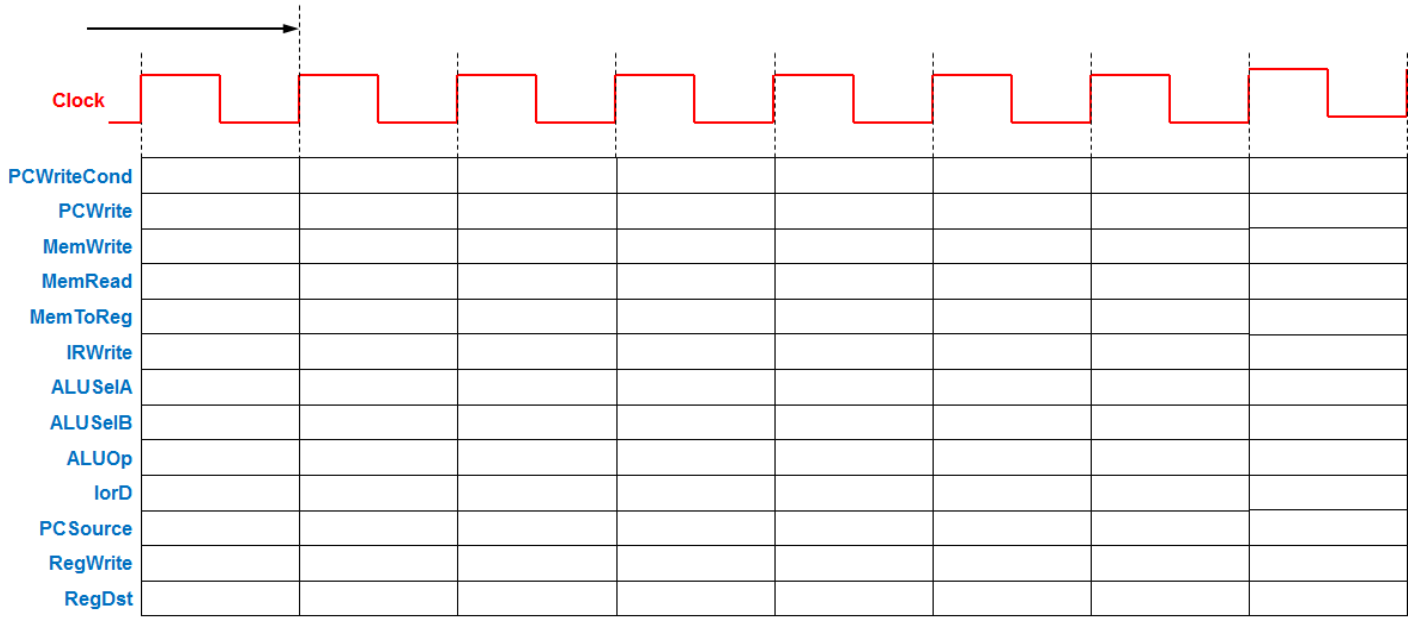


Fig. 2 - Datapath multi-cycle

132. Considere o *datapath multi-cycle* presente na figura anterior e a respetiva unidade de controlo. Preencha a tabela abaixo considerando que a coluna da esquerda corresponde ao último ciclo de execução de uma instrução, e que a sequencia em causa é a seguinte:

```
a.      add   $t0, $t2, $t1
        sw    $t0, 0($t3)
        beq   $t0, $t1, next
```

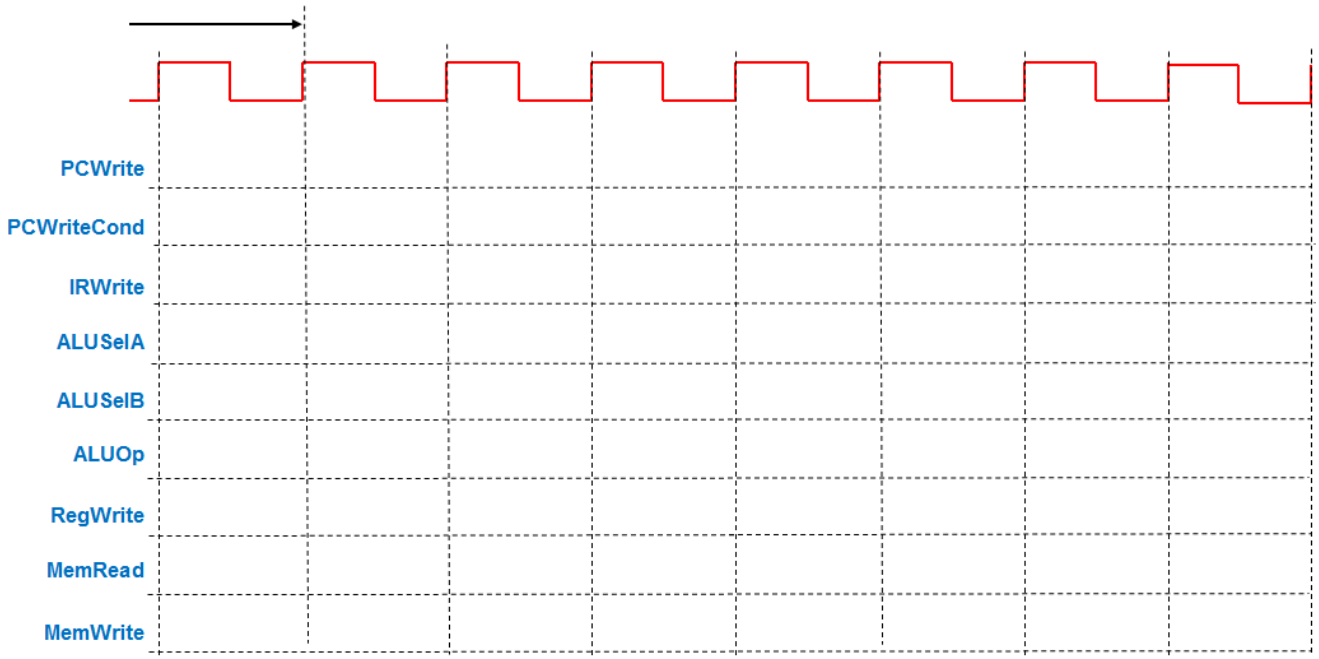


133. Repita o exercício anterior para as seguintes sequências de instrução:

a.	b.	c.

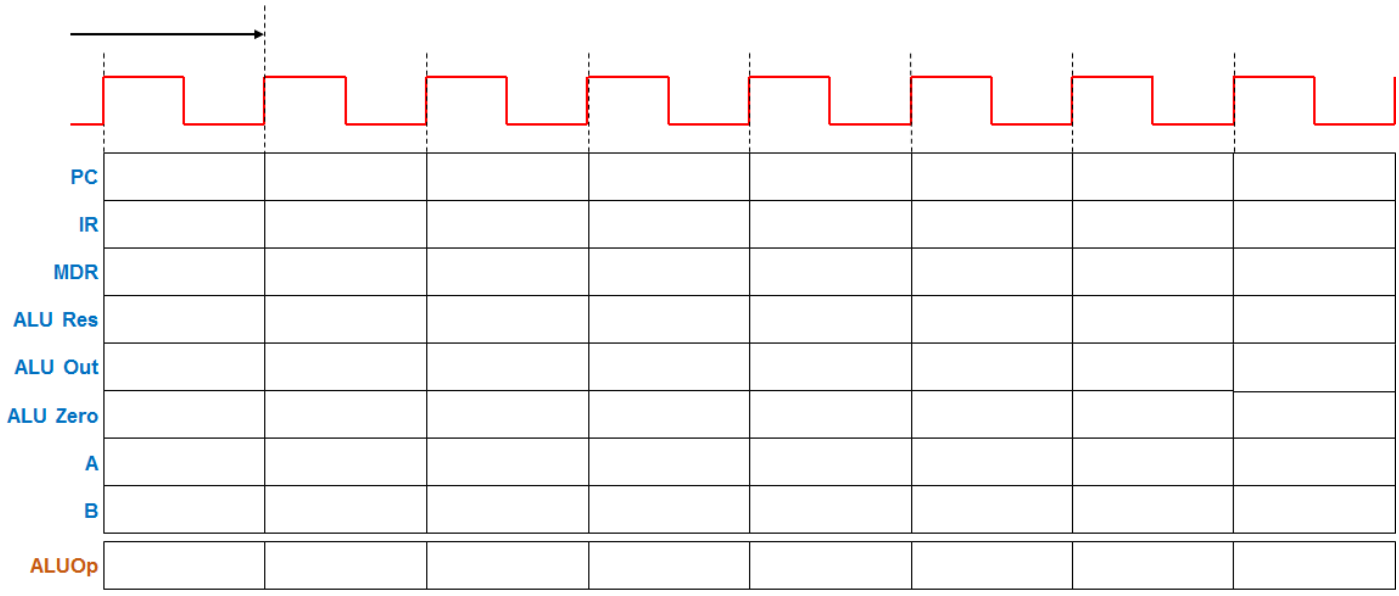
`or $t0, $0, $t1`
`addi $t0, $t1, 0x20`
`j label`
	`lw $s0, 0($t1)`
	`lw $s1, 4($t1)`
	`add $t2, $s1, $s2`
		`sw $t0, 0($t1)`
		`sub $t0, $t3, $t2`
		`slt $t1, $t0, $t2`

134. Para as mesmas sequências de instruções apresentadas nos dois exercícios anteriores, preencha, na forma de um diagrama temporal, a tabela seguinte.



135. Ainda para as mesmas sequências de instruções apresentadas nos três exercícios anteriores, preencha a tabela abaixo com os valores presentes à saída da ALU e dos elementos de estado indicados. Consulte a tabela da última página se necessário. Admita que, no início de cada sequência, o conteúdo dos registos relevantes é o seguinte:

`[$t0=0x000013FC],[$t1=0x10010000],[$t2=0x90FFFF64],[$t3=0x00000028] e que na memória [(0x10010000)=0x00000020] e [(0x10010004)=0x00000038]`



136. Calcule o número de ciclos de relógio que o programa seguinte demora a executar, desde o *Instruction fetch* da 1ª instrução até à conclusão da última instrução:

    a. num *datapath single-cycle*

    b. num *datapath multi-cycle*

Memória de dados	
Address	**Value**
0x0000000	0x10
0x0000004	0x20

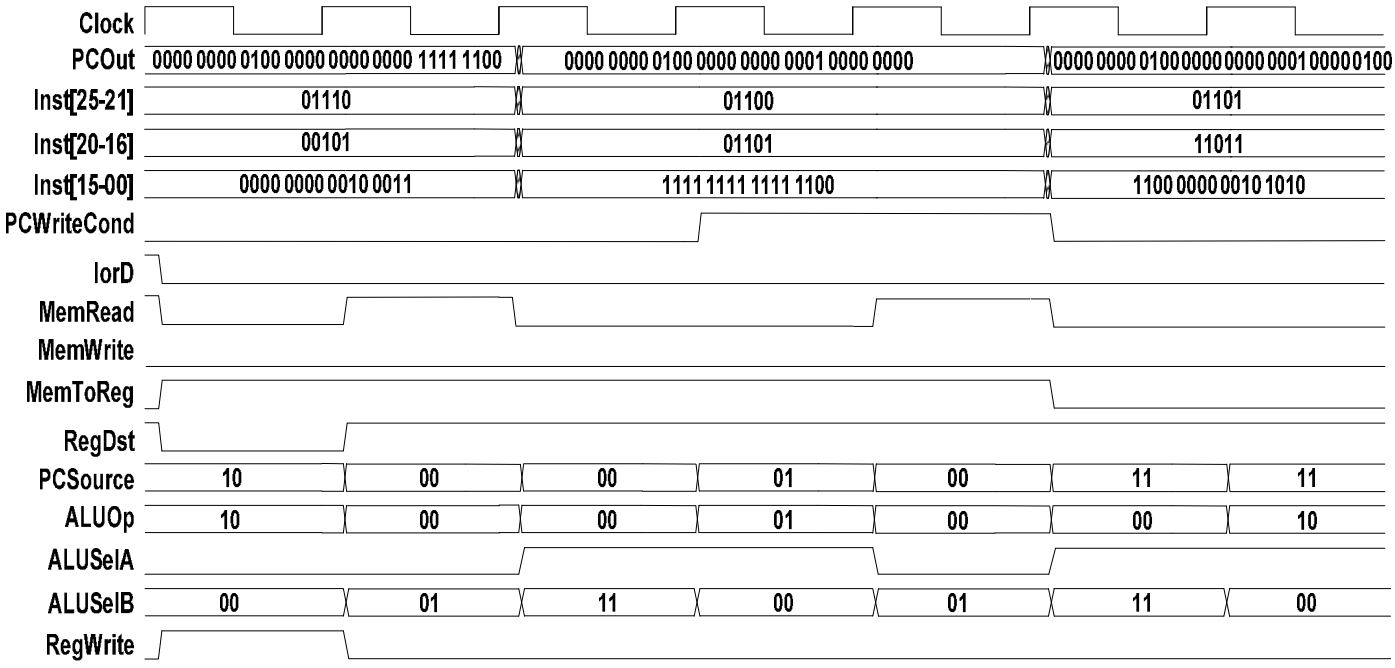
```
main:                   # p0 = 0;
        lw      $1,0($0)    # p1 = *p0 = 0x10;
        add     $4,$0,$0    # v = 0;
        lw      $2,4($0)    # p2=*(p0+1)=0x20;
loop:                   # do {
        lw      $3,0($1)    #      aux1 = *p1;
        add     $4,$4,$3    #      v = v + *p1;
        sw      $4,36($1)   #      *(p1 + 9) = v;
        addiu   $1,$1,4             #      p1++;
        sltu    $5,$1,$2    #
        bne     $5,$0,loop  # } while(p1 < p2);
        sw      $4,8($0)    # *(p0 + 2) = v;
        lw      $1,12($0)   # aux2 = *(p0 + 3);
```

137. Repita o exercício anterior assumindo que o valor armazenado no endereço de memória **0x00000004** é **0x2C**.

138. Descreva, sucintamente, as principais diferenças, ao nível estrutural, entre os *datapath single-cycle* e *multi-cycle*.

139. Indique, para o caso de um *datapath multi-cycle,* quais as operações realizadas pela ALU no decurso dos dois primeiros ciclos de relógio de qualquer instrução.

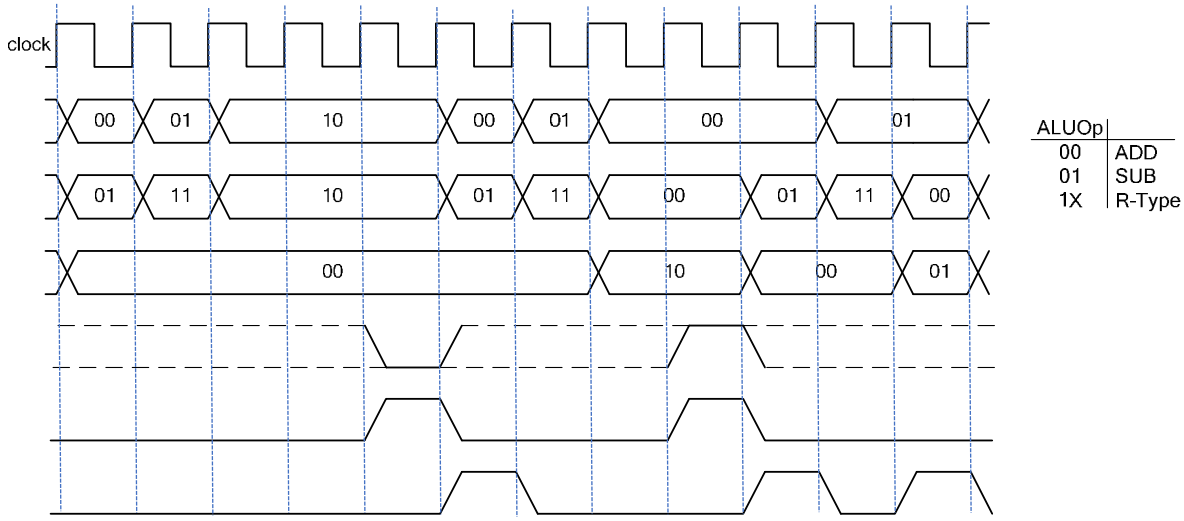
140. Considere o diagrama temporal seguinte relativo à execução de uma sequência de três instruções, das quais apenas a segunda está completamente representada. Obtenha o código assembly desta sequência de três instruções.

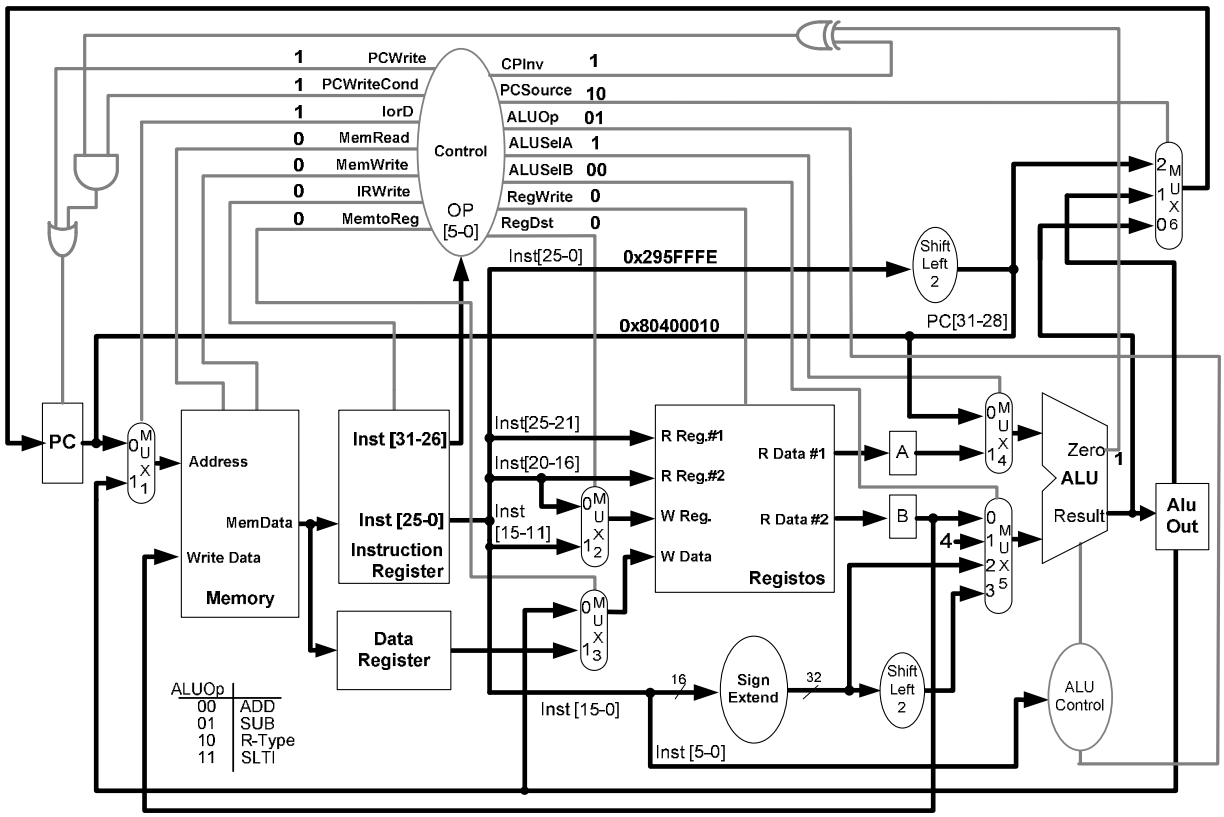


141. Considere a seguinte sequência de três instruções a serem executadas num *datapath muti-cycle*:

```
lw  $6, 0($7)
and $8, $6, $5
beq $8, $0, L1
```

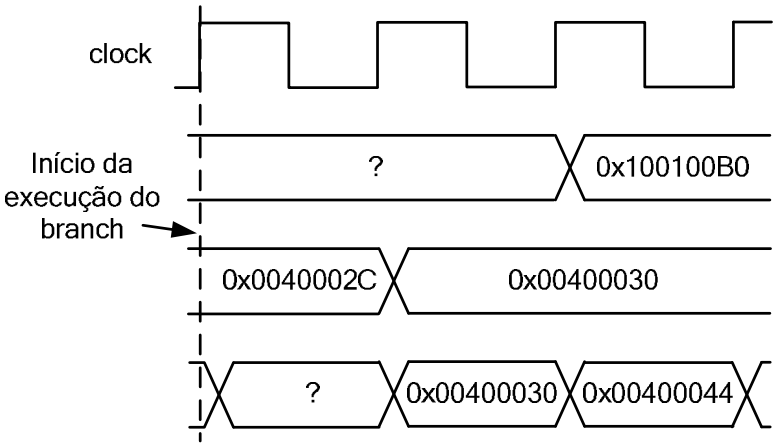
No diagrama temporal seguinte, relativo à execução desta sequência, identifique o nome dos sinais de controlo representados. (Note: o IorD não faz parte destes sinais)

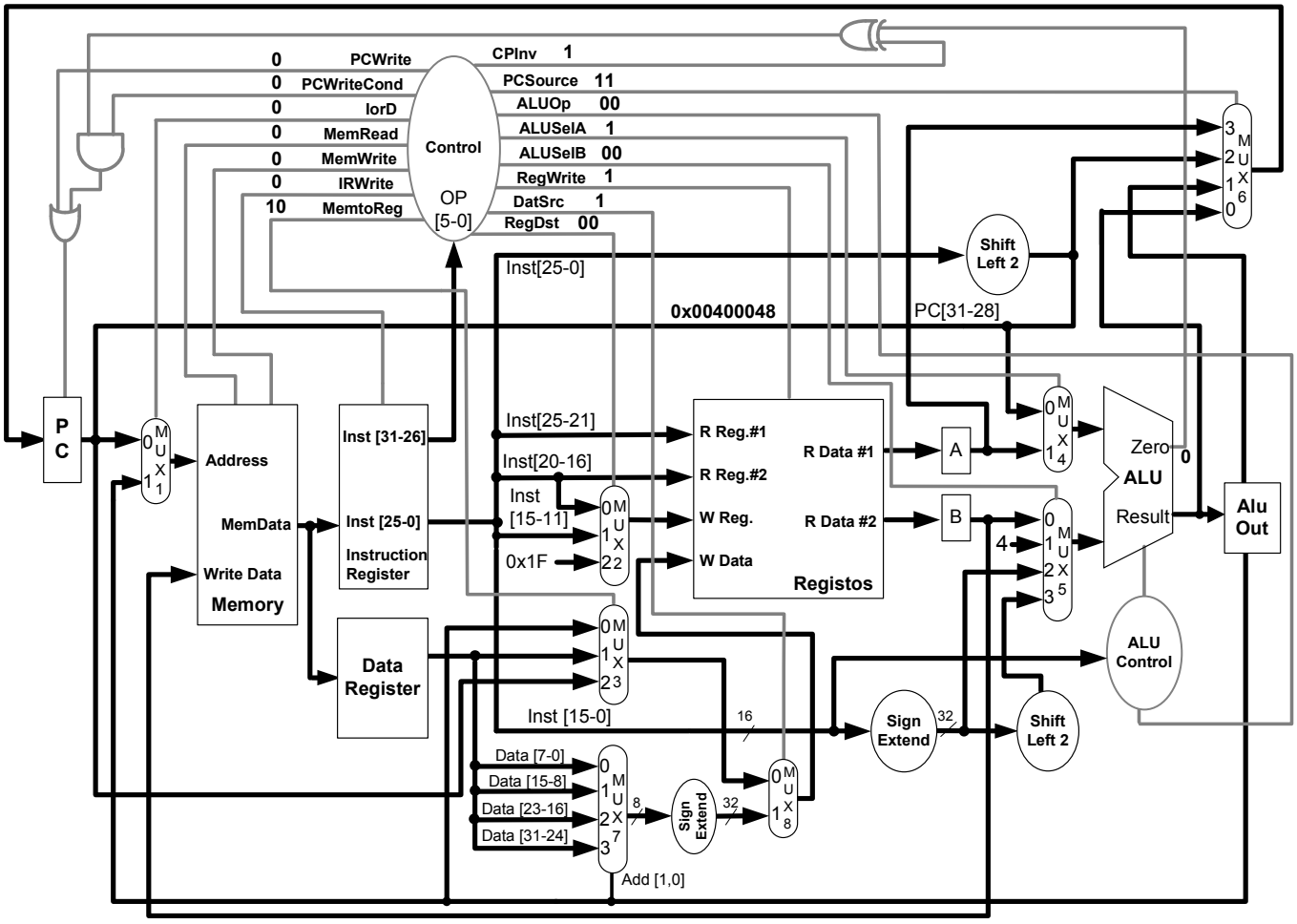




142. Considere o *datapath multi-cycle* e a unidade de controlo fornecidos na figura acima. Admita que os valores indicados no *datapath* fornecido correspondem à "fotografia" tirada no decurso da execução de uma instrução armazenada no endereço **0x8040000C**. Tendo em conta todos os sinais, identifique, em *assembly*, a instrução que está em execução e a respetiva fase.

143. Considere a instrução **`beq $5 $6,L2`** armazenada no endereço **`0x0040002C`**. Admita que **`$5=0x1001009C`** e **`$6=0x100100B0`**. Identifique os registos representados na figura seguinte e obtenha o código máquina, em hexadecimal, da instrução indicada.





144. Considere o *datapath* e a unidade de controlo fornecidos na figura acima (com ligeiras alterações relativamente à versão das aulas teórico-práticas). Analise cuidadosamente as alterações introduzidas e identifique quais são as novas instruções que este *datapath* permite executar quando comparado com a versão fornecida nas aulas TP.

145. Descreva, justificando, as principais características da unidade de controlo numa implementação pipelined da arquitetura MIPS, incluindo a sua natureza (combinatória ou síncrona) os sinais que constituem as variáveis independentes de entrada e as suas saídas.

146. Indique o que determina a máxima frequência de relógio de uma implementação *pipelined* da arquitetura MIPS com base nos principais elementos operativos que a constituem.

147. Calcule, numa implementação *pipelined* da arquitetura MIPS em que a operação de *Write Back* é executada a meio do ciclo de relógio, a frequência máxima de operação, assumindo que os elementos operativos apresentam os seguintes atrasos de propagação:

    a. Memórias externas: Leitura: 10 ns, Escrita: – 8ns; File register: Leitura – 2ns, Escrita – 2ns; Unidade de Controlo: 2ns; ALU (qualquer operação): 6ns; Somadores: 4ns; Outros: 0ns.

    b. Memórias externas: Leitura: 5 ns, Escrita: – 7ns; File register: Leitura – 1ns, Escrita – 1ns; Unidade de Controlo: 1ns; ALU (qualquer operação): 8ns; Somadores: 1ns; Outros: 0ns.

    c. Memórias externas: Leitura: 8 ns, Escrita: – 10ns; File register: Leitura – 2ns, Escrita – 4ns; Unidade de Controlo: 2ns; ALU (qualquer operação): 6ns; Somadores: 2ns; Outros: 0ns.

148. Identifique os principais tipos de *hazard* que podem existir numa implementação *pipelined* de um processador.

149. Numa arquitetura *pipelined*, como se designa a técnica que permite utilizar como operando de uma instrução um resultado produzido por outra instrução que se encontra numa etapa mais avançada do mesmo.

150. Explique por palavras suas em que circunstâncias pode ocorrer um *hazard* de dados numa implementação *pipelined* de um processador

151. A existência de *hazards* de controlo pode ser resolvida por diferentes técnicas dependendo da arquitetura em causa. Identifique a técnica usada para o efeito numa arquitetura MIPS com *datapath pipelined*, como se designa essa técnica e em que consiste.

152. Em certas circunstâncias relacionadas com *hazards* de dados, não é possível resolver o problema sem recorrer a uma paragem parcial do *pipeline,* através do atraso de um ou mais ciclos de relógio no início da execução de uma instrução. Indique como se designa essa técnica e em que consiste ao nível do controlo do *pipeline*

153. Determine o número de ciclos de relógio que o trecho de código seguinte demora a executar num pipeline de 5 fases, desde o instante em que é feito o *Instruction Fetch* da 1ª instrução, até à conclusão da última.

```
add  $1,$2,$3
lw   $2,0($4)
sub  $3,$4,$3
addi $4,$4,4
and  $5,$1,$5      #"and" em ID, "add" já terminou
sw   $2,0($1)      #"sw" em ID, "add" e "lw" já terminaram
```

154. Num datapath *single-cycle* o código da pergunta anterior demoraria 6 ciclos de relógio a executar. Por que razão é a execução no *datapath pipelined* mais rápida?

155. Quantos ciclos de relógio demora a execução do mesmo código num *datapath multi-cycle*?

156. Admita uma implementação *pipelined* da arquitetura MIPS com unidade de *forwarding* para EX e ID. Identifique, para as seguintes sequências de instruções, de onde e para onde deve ser executado o *forwarding* para que não seja necessário realizar qualquer *stall* ao pipeline:

a.
```
add  $t0,$t1,$t2
lw   $t1,0($t3)
beq  $t3,$t0,LABEL
```
b.
```
sub  $t0,$t1,$t2
addi $t3,$t0, 0x20
```
c.
```
lw   $t0,0($t2)
sw   $t3,0($t0)
```
d.
```
lw   $t3,0($t6)
xori $t0,$t4,0x20
sw   $t3,($t0)
```

157. Descreva, por palavras suas, a função da unidade de *forwarding* de uma implementação *pipelined* da arquitetura MIPS.

158. Admita o seguinte trecho de código, a executar sobre uma implementação *pipelined* da arquitetura MIPS com *delayed branches*, e unidade de *forwarding* de MEM e WB para o estágio EX.

```
LABEL:  lw   $t3, 0($t4)      # 1
        sub  $t7, $t5, $t6    # 2
        ori  $t2, $0, $0      # 3
        beq  $t2, $0, LABEL   # 4
        add  $t4, $t7, $t7    # 5
```

a. Identifique os vários *hazards* neste código e determine se os mesmos podem ser resolvidos por *forwarding*.

b. Identifique as situações em que é necessário executar *stalling* do pipeline e o respetivo número de *stalls*

c. Resolva o problema anterior supondo que a arquitetura suporta *forwarding* de MEM para ID.

159. Para o trecho de código seguinte identifique todas as situações de *hazard* de dados e de controlo que ocorrem na execução num pipeline de 5 fases, com *branches* resolvidos em ID.

```
main:   lw      $1,0($0)
        add     $4,$0,$0
        lw      $2,4($0)
loop:   lw      $3,0($1)
        add     $4,$4,$3
        sw      $4,36($1)
        addiu   $1,$1,4
        sltu    $5,$1,$2
        bne     $5,$0,loop
        sw      $4,8($0)
        lw      $1,12($0)
```

Memória de dados

Addr	Value
0x0000000	0x10
0x0000004	0x20

160. Apresente o modo de resolução das situações de *hazard* de dados do código da questão 159, admitindo que o pipeline não implementa *forwarding*.

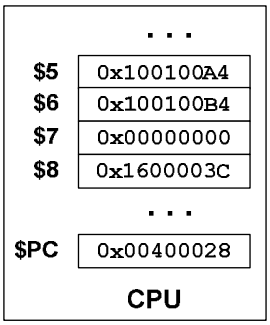
161. Calcule o número de ciclos de relógio que o programa anterior demora a executar num pipeline de 5 fases, sem *forwarding*, com *branches* resolvidos em ID e *delayed branch*, desde o IF da 1ª instrução até à conclusão da última instrução.

162. Resolva o problema anterior, considerando agora que o pipeline implementa *forwarding* para EX e para ID.

163. Calcule finalmente o número de ciclos de relógio que o programa do problema 159 demora a executar num pipeline de 5 fases, com *forwarding* para EX e para ID, com *branches* resolvidos em ID e *delayed branch*, desde o IF da 1ª instrução até à conclusão da última instrução.

164. Considere o trecho de código apresentado na figura seguinte, bem como as tabelas e 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, para já, o *datapath* e a unidade de controlo fornecidos na pergunta 132 (Fig. 2), correspondentes a uma implementação *multi-cycle* simplificada da arquitetura MIPS.

Endereço	Dados
...	...
0x1001009C	0x01234567
0x100100A0	0x7CABCDEF
0x100100A4	0xF9FC3CF3
0x100100A8	0xDF11347E
0x100100AC	0x377933FD
0x100100B0	0x5EFF00BC
...	...



Figura

```
L1: beq  $5,$6,L2
    lw   $7,-4($5)
    and  $7,$7,$8
    addi $5,$5,4
    bne  $7,$8,L1
L2: nop
```

165. Determine o valor presente à saída do registo **ALUOut** durante a terceira fase de execução da segunda instrução (**`lw $7,-4($5)`**).

166. Face aos valores presentes no segmento de dados (tabela da esquerda) e nos registos, calcule o número total de ciclos de relógio que demora a execução completa do trecho de código apresentado, numa implementação *multi-cycle* do MIPS (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:**").

167. Suponha agora que o mesmo código é executado numa versão *pipelined* do *datapath* do MIPS semelhante à abordada nas aulas teórico-práticas de AC1. Admita que este *datapath* suporta apenas *forwarding* para EX. Determine o número total de ciclos de relógio que demora a execução completa do trecho de código apresentado, até ao instante inicial do *instruction fetch* da instrução imediatamente a seguir ao **`nop`**.

168. Continue a considerar a execução do código numa versão *pipelined* do *datapath* do MIPS. Admita que no instante zero, correspondente a uma transição ativa do sinal de relógio, vai iniciar-se o *instruction fetch* da primeira instrução. Determine o valor à saída da ALU na conclusão do sexto ciclo de relógio contando a partir do instante zero.

169. Repita as questões 165 a 168 para os dados da figura seguinte:

Endereço	Dados
...	...
0x10010028	0x31434120
0x1001002C	0x31303220
0x10010030	0x00000032
0x10010034	0xE0DE0AC1
0x10010038	0x0000FFFF
0x1001003C	0x00000000
...	...

...	
$6	0x00000000
$7	0x10010038
$8	0x10010028
$9	0xE0DE0AC1
...	
$PC	0x004000C4

CPU

Figura

```
L1: lb   $9,0($8)
    lw   $2,4($7)
    addi $2,$2,1
    xor  $5,$6,$9
    addi $8,$8,1
    sw   $2,4($7)
    bne  $5,$0,L1
    xor  $0,$0,$0
L2: nop
```

170. Considere a versão com *pipeline* do *datapath* apresentado na Fig. 3. Identifique todas as combinações de *forwarding* disponíveis neste *datapath* e, <u>para cada uma delas</u>, escreva uma curta sequência de instruções que desencadeie esse tipo específico de *forwarding*. Nos casos em que tal se aplique, identifique igualmente os casos em que é preciso gerar *stalling* e o número de ciclos de *stalling* necessários.

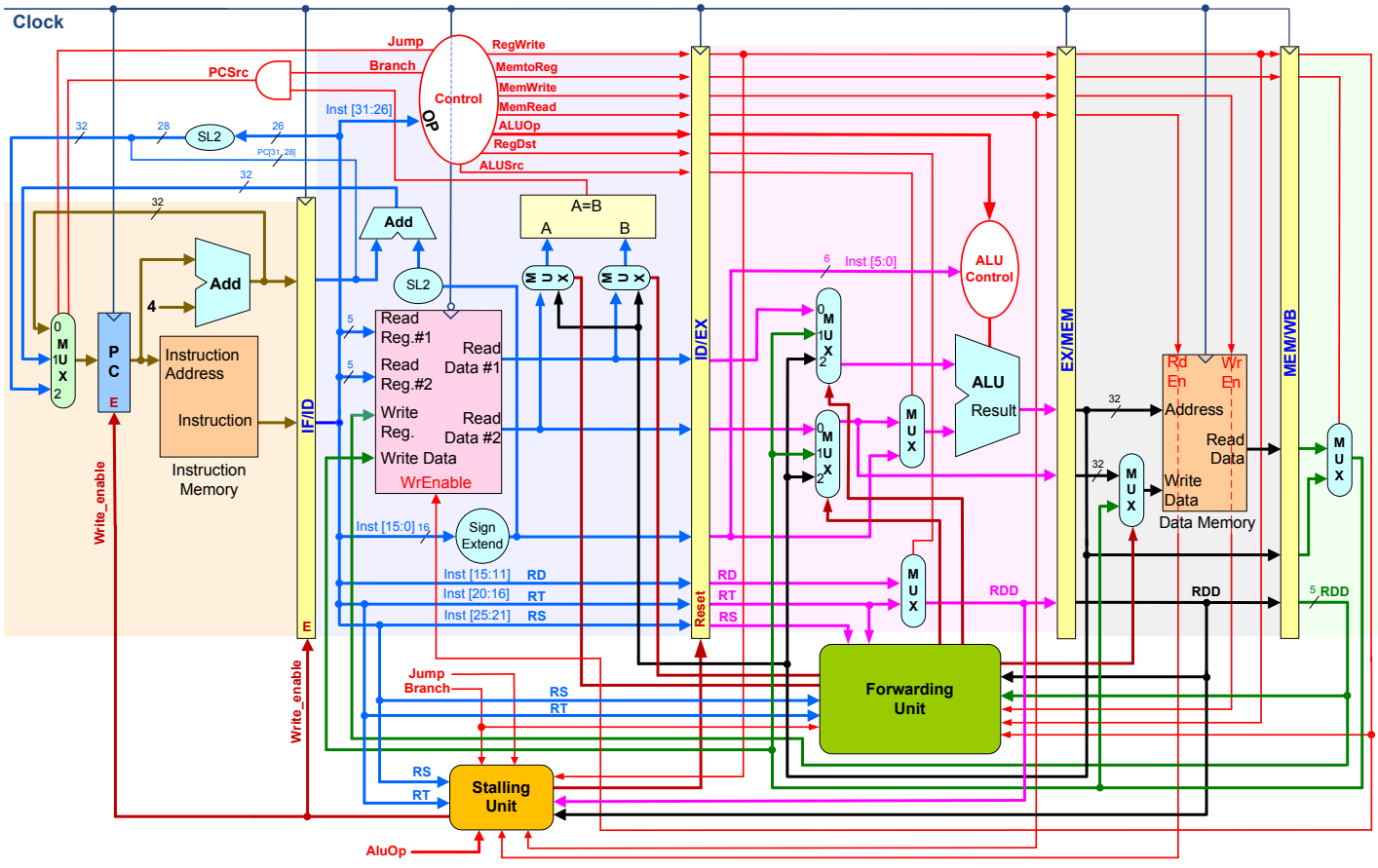


Fig. 3 – Piplined Datapath

Tabela de códigos de função (funct) e códigos de operação (OpCode) das principais instruções do MIPS

Arithm / Logical Instructions			Comparison Instructions	
Instruction	(funct)		Instruction	(OpCode)
add	100000 (0x20)		slt	101010 (0x2A)
addu	100001 (0x21)		sltu	101001 (0x29)
and	100100 (0x24)		slti	001010 (0x0A)
div	011010 (0x1A)		sltiu	001001 (0x09)
divu	011011 (0x1B)			
mult	011000 (0x18)		Branch Instructions	
multu	011001 (0x19)		beq	000100 (0x04)
nor	100111 (0x27)		bne	000101 (0x05)
or	100101 (0x25)		bgtz	000111 (0x07)
sll	000000 (0x00)		bgez	000001 (0x01)[1]
sra	000011 (0x03)		bltz	000001 (0x01)
srl	000010 (0x02)		blez	000110 (0x06)
sub	100010 (0x22)			
subu	100011 (0x23)		Jump Instructions	
xor	100110 (0x26)		j	000010 (0x02)
			jal	000011 (0x03)
Arithm / Logical Imm			jalr	001001 (0x09)
Instruction	(OpCode)		jr	001000 (0x08)
addi	001000 (0x08)			
addiu	001001 (0x09)		Load/Store Instructions	
andi	001100 (0x0C)		lb	100000 (0x20)
ori	001101 (0x0D)		lbu	100100 (0x24)
xori	001110 (0x0E)		lw	100011 (0x23)
			sb	101000 (0x28)
			sw	101011 (0x2B)
			Data Movement Instructions	
			mfhi	010000 (0x10)
			mflo	010010 (0x12)
			mthi	010001 (0x11)
			mtlo	010011 (0x13)

[1] O OpCode é igual ao da instrução **bltz** mas o valor de **rt** é igual a 00001_b