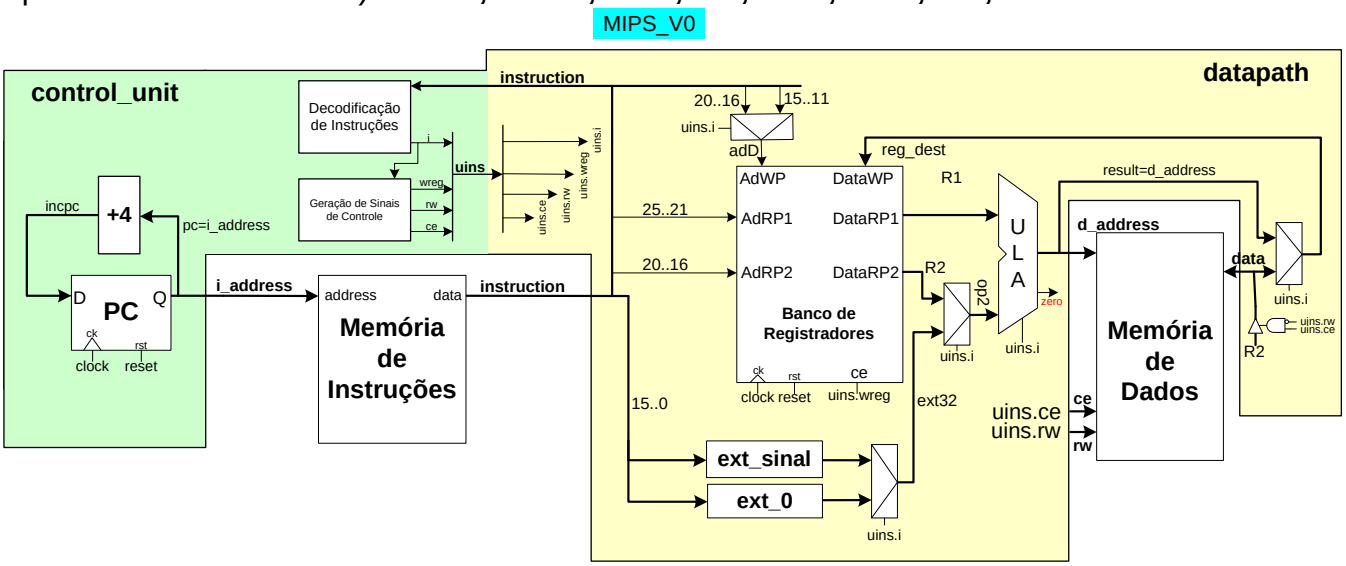
Prova P2 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 16/junho/2017

Para realizar a prova, refira-se à proposta de organização MIPS monociclo vista em aula. O diagrama de blocos deste processador aparece abaixo, e detalha o Bloco de Dados (**datapath**) e o Bloco de Controle (**control_unit**). Assuma que as instruções às quais esta organização monociclo dá suporte de execução são apenas as seguintes (exceto se a questão particular especificar de outra forma): **ADDU, SUBU, AND, OR, XOR, NOR, LW, SW e ORI**.



1. [3,0 pontos] Quando aprendemos sobre circuitos digitais, um dos componentes mais comumente usados na construção de tais circuitos são os multiplexadores (muxes). Na organização MIPS_V0 existem quatro destes componentes. Três destes são descritos nas linhas de código VHDL abaixo deste parágrafo. Estes três muxes são todos do tipo 2:1, no sentido de que recebem exatamente duas entradas de dados e uma destas é selecionada para ser passada para a saída do componente. Ora, um mux 2:1 precisa de apenas 1 bit de controle para escolher, entre suas 2 entradas, qual destas direcionar à saída. Mas como vimos em aula, o sinal de controle destes muxes, **uins.i** não pode ser de apenas um bit. Logo, o que chamamos de mux na MIPS_V0 é mais que um mux, é um mux associado a uma lógica de controle específica. Considere que **uins.i** é um sinal de 4 bits. O valor instantâneo deste sinal caracteriza uma das 9 instruções do MIPS_V0. Assuma que os códigos atribuídos a cada instrução vão de **0000** para a instrução ADDU até **1000** para a instrução ORI, com os valores binários sucessivos atribuídos às demais instruções na ordem listada acima. A partir daí, com relação a estes três multiplexadores, resolva as questões abaixo.

```
I.    adD <= instruction(15 downto 11) when instR='1' else instruction(20 downto 16);
II.   op2 <= R2 when instR='1' else ext32;
III.  reg_dest <= data when uins.i=LW else result;
```

   a) [1,0 ponto] Caracterize cada um dos três multiplexadores em termos de entradas e saídas. Para isto, diga quantos fios de entrada e quantos fios de saída cada componente possui, ou desenhe um bloco com estas características de entradas e saídas. Diga quais dos fios são de controle e quais são de dados.

   b) [2,0 pontos] Escolha um dos três muxes e desenhe um diagrama de esquemáticos que o implementa. Assuma que se trata de um mux 2:1 com 2 entradas e 1 saída de dados de mesmas dimensões e implemente com portas lógicas tudo que existir em volta deste mux para implementar de forma completa o VHDL do componente que você escolheu.

2. [3,0 pontos] Considere o bloco de dados monociclo apresentado acima.

   a) [1,0 ponto] Suponha que existe uma falha, que afeta apenas o multiplexador que gera o sinal **ext32**. A falha é que este componente sempre deixa passar a sua entrada superior para a saída, independentemente do valor do sinal de controle uins.i. Diga quais instruções ainda podem ser executadas corretamente, justificando sua resposta. Alguma instrução pode ser executada de forma correta às vezes e de forma incorreta outras vezes? Se sim, qual instrução é esta e em que condições tais situações ocorrem?

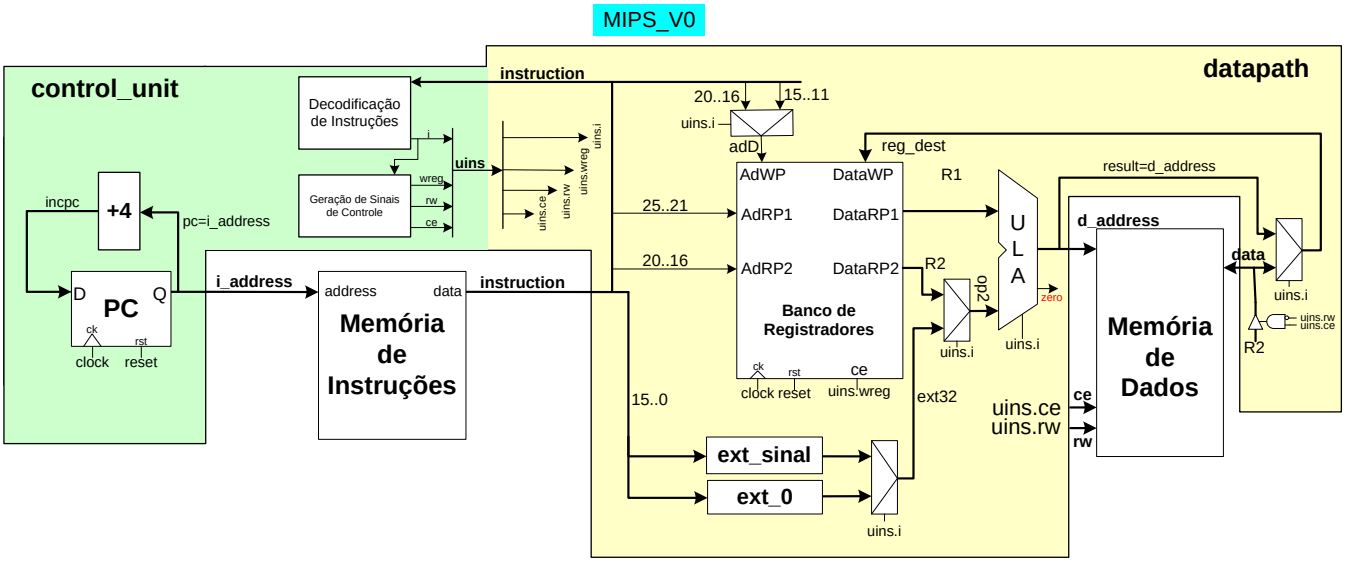
b) [2 pontos] Estude na documentação do MIPS a funcionalidade da instrução **XOR** e marque no desenho acima ou cite os blocos de hardware efetivamente usados para sua execução. Que operação é realizada na ULA para esta instrução? Quais condições são fixadas nos multiplexadores para executar esta instrução (numere os multiplexadores para identificá-los, se achar interessante)? Diga se alguma condição de algum multiplexador é irrelevante.

3. [4,0 pontos] Dada uma frequência de operação de 0,8GHz para o processador **MIPS monociclo**, assuma que a organização original foi alterada para dar suporte a todas as instruções do programa abaixo, mantendo a característica monociclo. Com estes pressupostos, calcule e diga:
   a) (1,5 pontos) O número de ciclos de relógio que leva p/ executar o programa, com a área de dados fornecida;
   b) (1 ponto) O tempo de execução do programa em segundos (1ns=10^{-9} segundos);
   c) (1 ponto) O que faz este programa, do ponto de vista semântico. Diga o que ele gera como saída;
   d) (0,5 pontos) Se este programa possui subrotina(s). Se sim, diga onde esta(s) se encontra(m), definindo o intervalo de linhas que ela(s) ocupa(m).

```
1.          .data
2.  txt:    .asciiz      "Mamae Me AmA!"
3.          .text
4.          .globl       main
5.  main:   la           $t0,txt
6.  ltl:    lbu          $t1,0($t0)
7.          beq          $t1,$zero,f
8.          slti         $t2,$t1,0x5B
9.          beq          $t2,$zero,nlM
10.         slti         $t2,$t1,0x41
11.         bne          $t2,$zero,nlM
12.         addiu        $t1,$t1,0x20
13.         sb           $t1,0($t0)
14. nlM:    addiu        $t0,$t0,1
15.         j            ltl
16. f:      li           $v0,10
17.         syscall
```

Gabarito

Para realizar a prova, refira-se à proposta de organização MIPS monociclo vista em aula. O desenho da versão monociclo aparece abaixo, com detalhamento do Bloco de Dados. Assuma que as instruções às quais esta organização monociclo dá suporte de execução são apenas as seguintes (exceto quando a questão particular especificar de outra forma): **ADDU, SUBU, AND, OR, XOR, NOR, LW, SW e ORI**.



1. [3,0 pontos] Quando aprendemos sobre circuitos digitais, um dos componentes mais comumente usados na construção de tais circuitos são os multiplexadores (muxes). Na organização MIPS_V0 existem quatro destes componentes. Três destes são descritos nas linhas de código VHDL abaixo deste parágrafo. Estes três muxes são todos do tipo 2:1, no sentido de que recebem exatamente duas entradas de dados e uma destas é selecionada para ser passada para a saída do componente. Ora, um mux 2:1 precisa de apenas 1 bit de controle para escolher, entre suas 2 entradas, qual destas direcionar à saída. Mas como vimos em aula, o sinal de controle destes muxes, **uins.i** não pode ser de apenas um bit. Logo, o que chamamos de mux na MIPS_V0 é mais que um mux, é um mux associado a uma lógica de controle específica. Considere que **uins.i** é um sinal de 4 bits. O valor instantâneo deste sinal caracteriza uma das 9 instruções do MIPS_V0. Assuma que os códigos atribuídos a cada instrução vão de **0000** para a instrução ADDU até **1000** para a instrução ORI, com os valores binários sucessivos atribuídos às demais instruções na ordem listada acima. A partir daí, com relação a estes três multiplexadores, resolva as questões abaixo.

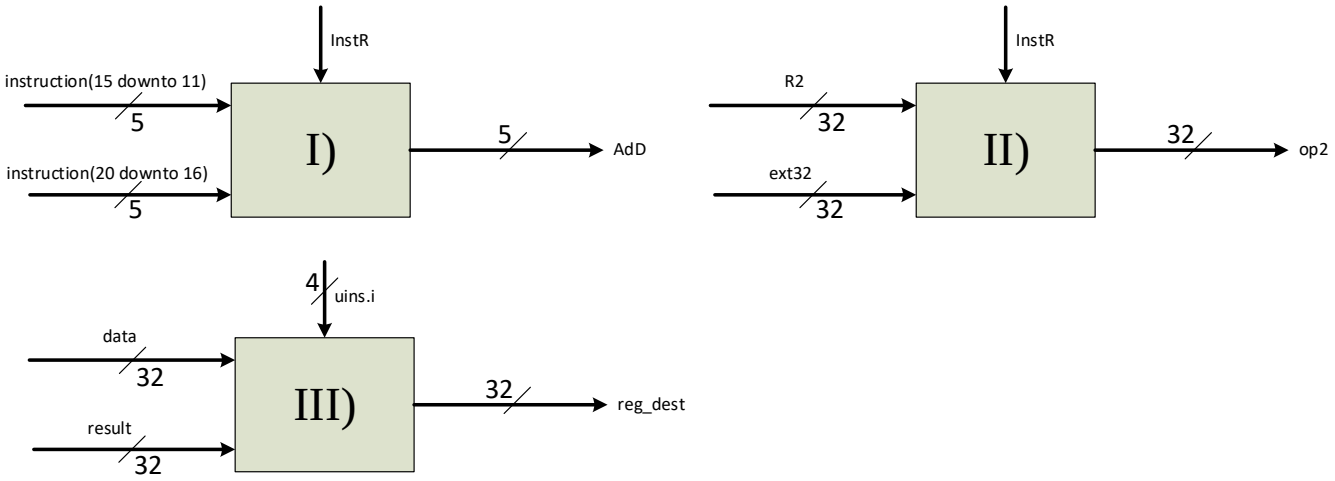
```
I.    adD <= instruction(15 downto 11) when instR='1' else instruction(20 downto 16);
II.   op2 <= R2 when instR='1' else ext32;
III.  reg_dest <= data when uins.i=LW else result;
```

   a) [1,0 ponto] Caracterize cada um dos três multiplexadores em termos de entradas e saídas. Para isto, diga quantos fios de entrada e quantos fios de saída cada componente possui, ou desenhe um bloco com estas características de entradas e saídas. Diga quais dos fios são de controle e quais são de dados.

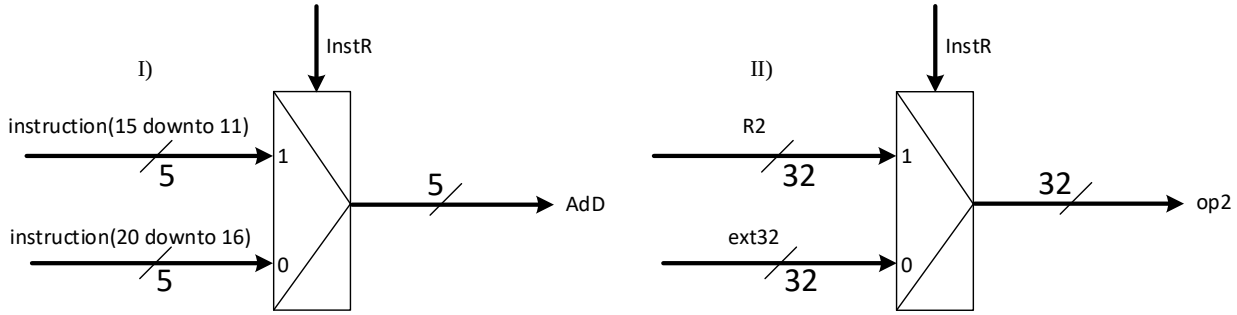
   b) [2,0 pontos] Escolha um dos três muxes e desenhe um diagrama de esquemáticos que o implementa. Assuma que se trata de um mux 2:1 com 2 entradas e 1 saída de dados de mesmas dimensões e implemente com portas lógicas tudo que existir em volta deste mux para implementar de forma completa o VHDL do componente que você escolheu.

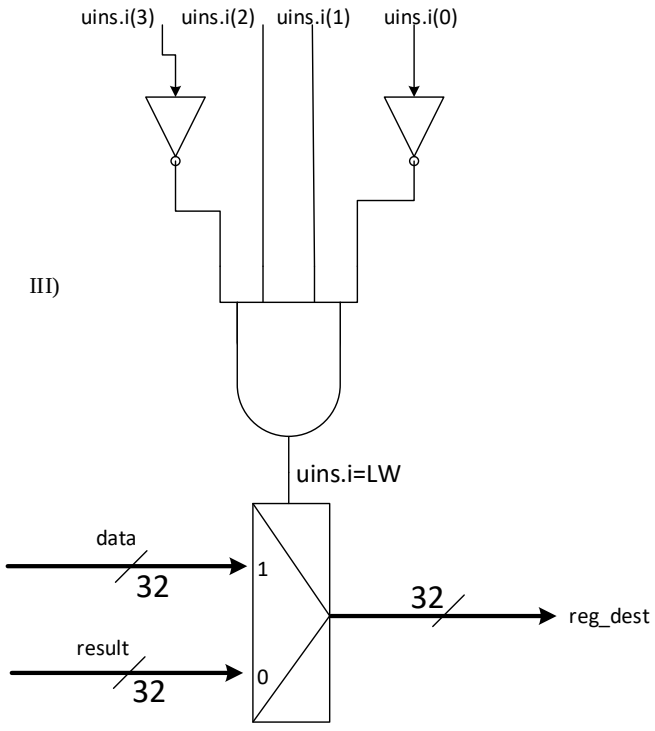
Solução:

a) O desenho abaixo mostra a interface de cada um dos multiplexadores. Entradas e saídas estão claramente indicadas por setas. Cada retângulo representa o multiplexador que corresponde à descrição da linha de VHDL cujo numeral romano se encontra em seu interior. Sinais de dados correspondem a linhas horizontais e sinais de controle a linhas verticais. Quando um sinal é composto por múltiplos fios isto está claramente indicado no desenho, a falta de indicação indica que o sinal é representado em um fio.



b) Como os multiplexadores descritos nas linhas I) e II) possuem apenas um fio de controle (instR), eles não necessitam de nenhuma lógica adicional. Apenas o mux da linha III) usa o sinal uins.i de 4 bits, e este necessita ser transformado em um bit de controle. Como a expressão que designa o controle deste mux é uins.i=LW precisamos de uma lógica que responda (por exemplo) '1' quando uins.i contiver o código associado a LW (que pelos pressupostos da questão é o código "0110") e que responda '0' em caso contrário. Logo, basta empregar uma porta lógica AND de quatro entradas e dois inversores. Esta porta AND recebe uins.i(3) negado, uins.i(2), uins.i(1) e uins.i(0) negado. O desenho abaixo ilustra as implementações dos três multiplexadores. Note-se que a questão pede apenas um dos desenhos abaixo.



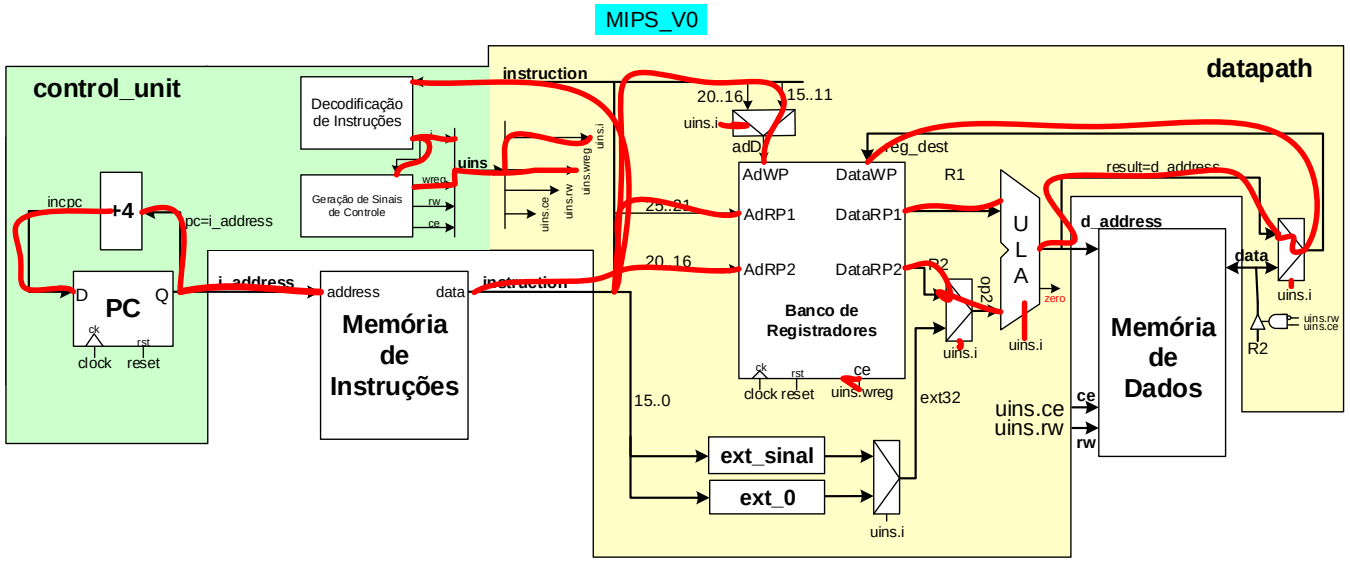


2. [3,0 pontos] Considere o bloco de dados monociclo apresentado acima.
   a) [1,0 ponto] Suponha que existe uma falha, que afeta apenas o multiplexador que gera o sinal **ext32**. A falha é que este componente sempre deixa passar a sua entrada superior para a saída, independentemente do valor do sinal de controle uins.i. Diga quais instruções ainda podem ser executadas corretamente, justificando sua resposta. Alguma instrução pode ser executada de forma correta às vezes e de forma incorreta outras vezes? Se sim, qual instrução é esta e em que condições tais situações ocorrem?
   b) [2 pontos] Estude na documentação do MIPS a funcionalidade da instrução **XOR** e marque no desenho acima ou cite os blocos de hardware efetivamente usados para sua execução. Que operação é realizada na ULA para esta instrução? Quais condições são fixadas nos multiplexadores para executar esta instrução (numere os multiplexadores para identificá-los, se achar interessante)? Diga se alguma condição de algum multiplexador é irrelevante.

Solução:

a) A saída ext32 somente é relevante para executar instruções que usam algum dado imediato. Logo, nenhuma das instruções tipo R (**ADDU, SUBU, AND, OR, XOR, NOR**) pode ser afetada por esta falha. Logo, estas são executadas corretamente. Além disto, a falha garante que a extensão de sinal sempre estará disponível em ext32. Logo, as instruções que usam extensão de sinal (**LW e SW**) continuam a executar corretamente. A única instrução que pode não executar corretamente é então **ORI**, que precisa usar a extensão de 0. Ainda assim, se o valor a ser estendido for positivo, extensão de sinal e extensão de 0 operam exatamente da mesma forma, e mesmo **ORI** funcionará corretamente.

b) No desenho abaixo as linhas vermelhas marcam os caminhos do processador efetivamente usados para executar a instrução XOR. Trata-se de uma instrução tipo R que lê o conteúdo de dois registradores do banco (endereçados por instruction(25 downto 21) e instruction(20 downto 16), respectivamente nas portas AdRP1 e AdRP2). Os valores lidos vão para a ULA, que executa uma operação ou-exclusivo bit a bit entre os dois valores de 32 bits e coloca o resultado em sua saída. Este valor é encaminhado através do mux mais à direita no desenho para a entrada da porta de escrita do banco de registradores (DataWP). Os bits instruction(15 downto 11) chegam à entrada de endereço de escrita do banco (AdWP) e a ativação de uins.wreg=1 permite que o resultado da ULA seja escrito no registrador destino. As condições fixadas nos multiplexadores do MIPS_V0 estão explicitadas no desenho. O único multiplexador irrelevante para esta instrução é aquele que gera o sinal ext32.



3. [4,0 pontos] Dada uma frequência de operação de 0,8GHz para o processador MIPS monociclo, assuma que a organização original foi alterada para dar suporte a todas as instruções do programa abaixo, mantendo a característica monociclo. Com estes pressupostos, calcule e diga:
   a) (1,5 pontos) O número de ciclos de relógio que leva p/ executar o programa, com a área de dados fornecida;
   b) (1 ponto) O tempo de execução do programa em segundos (1ns=10^{-9} segundos);

c) (1 ponto) O que faz este programa, do ponto de vista semântico. Diga o que ele gera como saída;

d) (0,5 pontos) Se este programa possui subrotina(s). Se sim, diga onde esta(s) se encontra(m), definindo o intervalo de linhas que ela(s) ocupa(m).

```
1.          .data
2.  txt:    .asciiz     "Mamae Me AmA!"     # Cadeia a ser processada
3.          .text                           #
4.          .globl      main                # Ciclos
5.  main:   la          $t0,txt             # (2) $t0 contém ponteiro p/ cadeia a processar
6.  ltl:    lbu         $t1,0($t0)          # (1) toma próximo char
7.          beq         $t1,$zero,f         # (1) Se for char NULL, finaliza programa
8.          slti        $t2,$t1,0x5B        # (1) Se char < que prim char > 'Z', $t2<=1
9.          beq         $t2,$zero,nlM       # (1) salta se já sabe não ser letra maiúscula
10.         slti        $t2,$t1,0x41        # (1) Se char < que 'A', $t2<=1
11.         bne         $t2,$zero,nlM       # (1) salta se não é letra maiúscula
12.         addiu       $t1,$t1,0x20        # (1) gera novo nibble mais signif
13.         sb          $t1,0($t0)          # (1) Qdo aqui, achou maiúscula, transf. minúsc.
14. nlM:    addiu       $t0,$t0,1           # (1) avança ponteiro para próximo char
15.         j           ltl                 # (1) e repete o laço
16. f:      li          $v0,10              # (1) Depois do laço, cai fora
17.         syscall                         # (1) do programa
```

Solução:

a) O programa principal tem uma parte com instruções que só executam uma vez (linha 5 e linhas 16-17). Estas linhas tomam um total de **4 ciclos**. Além destas, existe um laço entre as linhas 6-15, que vai executar exatamente 14 vezes, uma para cada caractere da cadeia **txt**. Existem quatro tipos execução distintos para o laço: (1) um tipo para caracteres com código ASCII maior que o da letra 'Z' (maiúscula), (2) um tipo para caracteres menores que a letra 'A' (maiúscula), exceto o caractere NULL, (3) um tipo para letras maiúsculas e finalmente (4) um tipo para o caractere NULL. Os tipos diferem no número de ciclos que cada um leva para executar. O primeiro tipo executa em 6 ciclos (linhas 6-9 e linhas 14-15). A cadeia **txt** possui 6 caracteres deste tipo, o que gasta 6*6=**36 ciclos** para executar. O segundo tipo executa em 8 ciclos (linhas 6-11 e linhas 14-15), e a cadeia **txt** possui 3 caracteres que serão assim tratados (os 2 espaços e o ponto de exclamação). Logo, gasta-se nestas execuções 3*8=**24 ciclos**. O terceiro tipo executa todas as instruções do laço, ou seja, toma 10 ciclos. Como existem 4 letras maiúsculas em **txt**, este tipo gasta para executar 4*10=**40 ciclos**. O último tipo apenas passa pelas linhas 6-7 e é executado apenas uma vez, tomando **2 ciclos**. Dados os fatos anteriores a execução completa do programa com os dados fornecidos tomará 4+36+24+40+2=106 ciclos de relógio.

b) Como a frequência é de 0,8GHz=800MHz, o período (ou o tempo de um ciclo) é $(1/(800*10^6))$ segundos, ou 1,25 $x10^{-9}$s. Logo, o tempo de execução do programa será de

$$106*1{,}25\ x10^{-9}s = \sim 1{,}325x10^{-7}s.$$

c) O programa converte todas a letras maiúsculas da cadeia de caracteres **txt** para letras minúsculas, mantendo todos os demais caracteres desta inalterados.

d) O programa não possui subrotinas (não usa a instrução jal seguida de jr $ra).