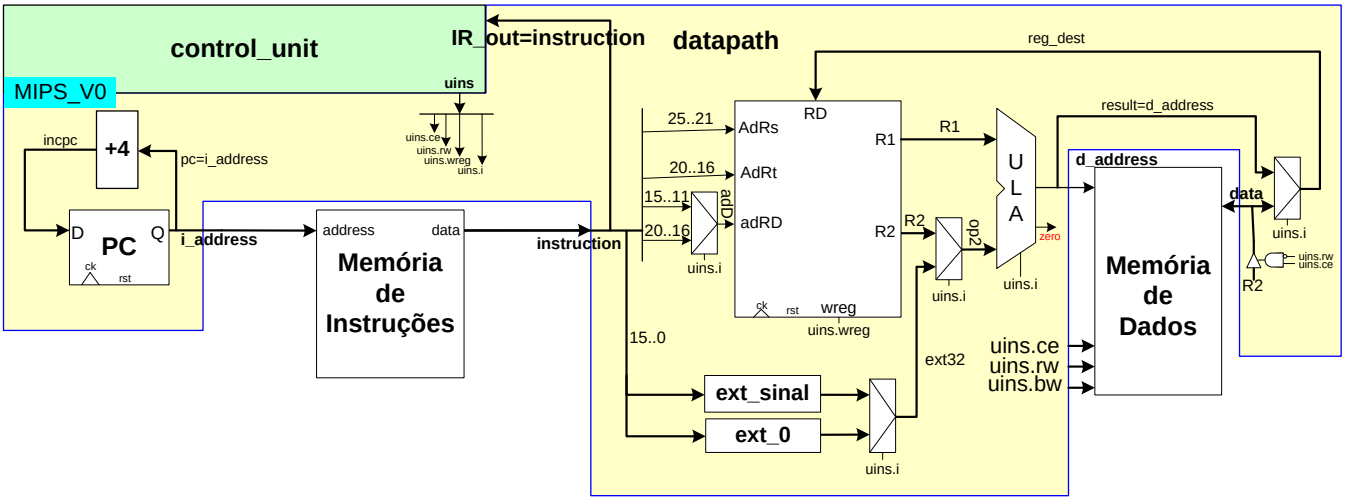
Prova P2 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 30/maio/2014

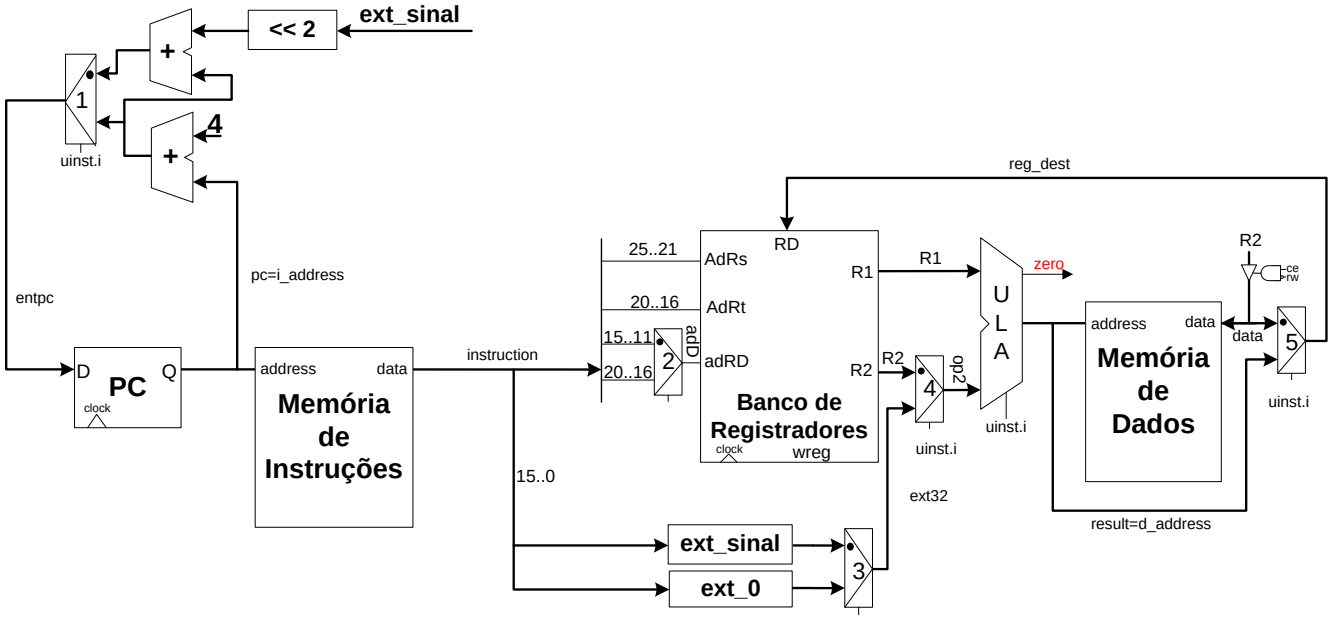
Para realizar a prova, refira-se à proposta de organização MIPS monociclo vista em aula. O desenho da versão monociclo aparece abaixo, com detalhamento do Bloco de Dados. Assuma que as instruções às quais esta organização monociclo dá suporte de execução são apenas as seguintes (exceto quando a questão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



1. [3,5 pontos] Assuma uma frequência de operação de 300 MHz para o processador **MIPS monociclo** e que a organização original foi alterada para dar suporte a executar todas as instruções do programa abaixo, mantendo a característica monociclo. Calcule:
   a) (1,0 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?
   d) (0,5 pontos) Este programa possui subrotinas? Se sim, onde esta(s) se encontra(m)? Defina o intervalo de linhas.

```
1.          .data
2.  str:    .asciiz       "Alo Mamae!"
3.  uca:    .word  0x0
4.  lca:    .word  0x0
5.
6.          .text
7.          .globl main
8.  main:   la     $t3,uca
9.          lw     $t4,0($t3)
10.         la     $t5,lca
11.         lw     $t6,0($t3)
12.         la     $t0,str
13. loop:   lb     $t1,0($t0)
14.         beq    $t1,$zero,end
15.         slti   $t2,$t1,0x41
16.         bne    $t2,$zero,floop
17.         slti   $t2,$t1,0x5b
18.         bne    $t2,$zero,guc
19.         slti   $t2,$t1,0x61
20.         bne    $t2,$zero,floop
21.         slti   $t2,$t1,0x7b
22.         beq    $t2,$zero,floop
23.         addiu  $t6,$t6,1
24.         j      floop
25. guc:    addiu  $t4,$t4,1
26. floop:  addiu  $t0,$t0,1
27.         j      loop
28. end:    sw     $t4,0($t3)
29.         sw     $t6,0($t5)
30.         li     $v0,10
31.         syscall
```

2. [3 pontos] Considere o bloco de dados monociclo apresentado abaixo. Ele resultou da transformação do bloco de dados visto em aula para dar suporte às novas instruções **BEQ** e **LUI**. O ponto nos multiplexadores indica condição verdadeira (ou, equivalentemente, sinal de controle do mux em 1). Assuma que as instruções às quais este processador dá suporte de execução são apenas as seguintes: **ADDU, SUBU, AND, OR, XOR, NOR, LW, SW, ORI, BEQ e LUI.**



a) Suponha que existe uma única falha, que afeta apenas o multiplexador que escolhe na entrada op2 da ULA (Mux 4). A falha é que o sinal de controle deste multiplexador está grudado eternamente no valor 1. Diga quais instruções ainda podem ser executadas corretamente, justificando para cada uma delas.

b) Observe na documentação do MIPS a funcionalidade da instrução XORI e diga se e como o hardware acima deveria ser alterado para ser capaz de executar esta instrução. Que operação deve ser realizada na ULA? Quais condições deveriam ser fixadas nos 5 multiplexadores para executar esta instrução? Diga se alguma condição de multiplexador é irrelevante.

3. (3,5 pontos) Considere as linhas de código VHDL abaixo, que são parte da descrição do MIPS monociclo (MIPS_V0). Este trecho contém a definição do sinal **adD** do bloco de dados, que é a saída do multiplexador cuja saída é o endereço do registrador de destino para instruções que escrevem no banco de registradores. Assuma como verdadeiro o seguinte fato:

- As instruções da MIPS monociclo são codificadas em 4 bits com valores na faixa de 0000 a 1000, na seguinte ordem crescente de valores: (**ADDU, SUBU, AND, OR, XOR, NOR, LW, SW, ORI**).

Pede-se:

a) Defina os códigos associados às instruções mencionadas no VHDL.

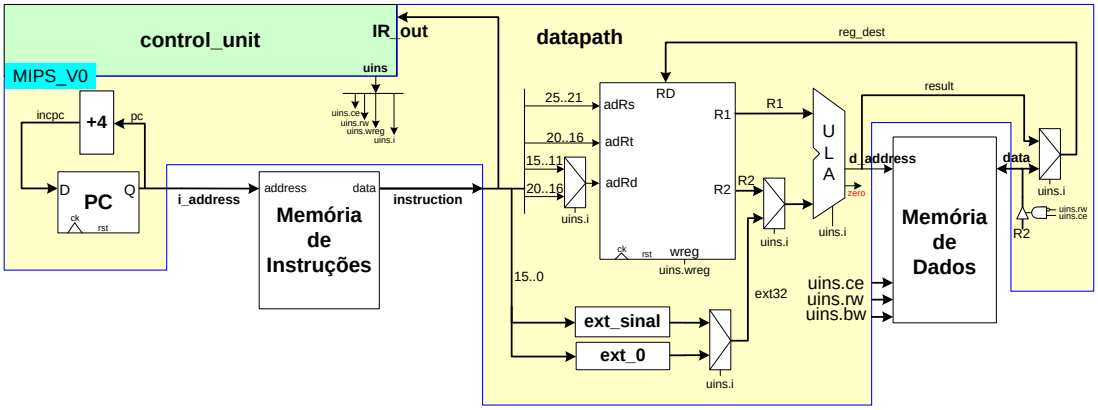
b) Usando os códigos definidos no item a), desenhe um diagrama esquemático que use apenas portas lógicas e componentes tais como multiplexadores, demultiplexadores, decodificadores, e que corresponda a um circuito correto e completo que pode resultar da conversão do código VHDL dado para hardware.

Dicas: (*i*) Assuma que as entradas do esquemático são todos os sinais não atribuídos no trecho VHDL dado; (*ii*) O sinal adD é a única saída do esquemático, e instR é um sinal interno do mesmo.

```
1.    instR <= '1' when uins.i=ADDU or uins.i=SUBU or
2.                uins.i=AAND or uins.i=OOR or uins.i=XXOR or uins.i=NNOR else
3.             '0';
4.    adD<= instruction(15 downto 11) when instR='1' else
5.          instruction(20 downto 16);
```

Gabarito

Para realizar a prova, refiram-se as propostas de organização MIPS monociclo e multiciclo vistas em aula. O desenho da versão monociclo aparece abaixo, com detalhamento do Bloco de Dados. O Bloco de Dados da versão multiciclo encontra-se no verso. Assuma que as instruções às quais a 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. [3,5 pontos] Assuma uma frequência de operação de 300 MHz para o processador **MIPS monociclo** e que a organização original foi alterada para dar suporte a executar todas as instruções do programa abaixo, mantendo a característica monociclo. Calcule:
   a) (1,0 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?
   d) (0,5 pontos) Este programa possui subrotinas? Se sim, onde esta(s) se encontra(m)? Defina o intervalo de linhas.

```
1.          .data                       # 0 ciclos - diretiva
2.  str:    .asciiz       "Alo Mamae!"  # 0 ciclos – definição de dados
3.  uca:    .word  0x0                  # 0 ciclos – definição de dados
4.  lca:    .word  0x0                  # 0 ciclos – definição de dados
5.
6.          .text                       # 0 ciclos - diretiva
7.          .globl main                 # 0 ciclos - diretiva
8.  main:   la     $t3,uca              # 2 ciclos
9.          lw     $t4,0($t3)           # 1 ciclo
10.         la     $t5,lca              # 2 ciclos
11.         lw     $t6,0($t3)           # 1 ciclo
12.         la     $t0,str              # 2 ciclos
13. loop:   lb     $t1,0($t0)           # 1 ciclo
14.         beq    $t1,$zero,end        # 1 ciclo – Salta só quando acabou o vetor
15.         slti   $t2,$t1,0x41         # 1 ciclo
16.         bne    $t2,$zero,floop      # 1 ciclo – Salta se código do caracter menor que 'A',
    ou seja para o caractere ' ' e para o '!'. 2*6 ciclos
17.         slti   $t2,$t1,0x5b         # 1 ciclo
18.         bne    $t2,$zero,guc        # 1 ciclo – Salta se código do caracter entre 'A' e
    'Z', ou seja para 'A' e 'M'. 2*9 ciclos
19.         slti   $t2,$t1,0x61         # 1 ciclo
20.         bne    $t2,$zero,floop      # 1 ciclo – Salta se código do caracter maior que 'Z' e
    menor que 'a'. 0*10 ciclos
21.         slti   $t2,$t1,0x7b         # 1 ciclo
22.         beq    $t2,$zero,floop      # 1 ciclo – Salta se código do caracter maior que 'z'.
    0*12 ciclos
23.         addiu  $t6,$t6,1            # 1 ciclo – Aqui só se letra minúscula, ou seja para
    "loamae". 6*14 ciclos.
24.         j      floop                # 1 ciclo
25. guc:    addiu  $t4,$t4,1            # 1 ciclo
26. floop:  addiu  $t0,$t0,1            # 1 ciclo
27.         j      loop                 # 1 ciclo
28. end:    sw     $t4,0($t3)           # 1 ciclo
29.         sw     $t6,0($t5)           # 1 ciclo
30.         li     $v0,10               # 1 ciclo
31.         syscall                     # 1 ciclo
```

Solução:

a) Cada instrução entre as linhas 8-12 é executada somente uma vez, o que corresponde a 8 ciclos de relógio gastos para executá-las. Não existem sub-rotinas neste programa, mas existe um laço entre as linhas 13-27, com várias partes executadas de forma condicional. As linhas 28-31, são também executadas apenas uma vez, gastando 4 ciclos. Vamos agora à análise do laço:
   - (i) a última vez que o laço for executado, passa-se apenas pelas linhas 13-14, o que consome 2 ciclos;
   - (ii) em seguida existem 5 maneiras de passar pelo rótulo floop:
     i. Saltando da linha 16, o que leva a uma execução que gasta 4+2=6 ciclos de relógio. Ocorre para todos os caracteres com código menor que o de uma letra;
     ii. Saltando da linha 18 para o rótulo guc, o que leva a uma execução que gasta 6+3=9 ciclos de relógio. Ocorre para todas as letras maiúsculas;
     iii. Saltando da linha 20, uma execução que gasta 8+2=10 ciclos de relógio. Ocorre para todos os caracteres com código maior que o de uma letra maiúscula e menos que uma letra minúscula;
     iv. Saltando da linha 22, uma execução que gasta 10+2=12 ciclos de relógio. Ocorre para todos os caracteres com código maior que o de qualquer letra;
     v. Finalmente, saltando da linha 24, execução que gasta 12+2=14 ciclos de relógio. Cada execução que não seja a final do laço pode gastar um destes valores (6, 9, 10, 12 ou 14), dependendo do resultado dos testes. Ocorre para todas as letras minúsculas.

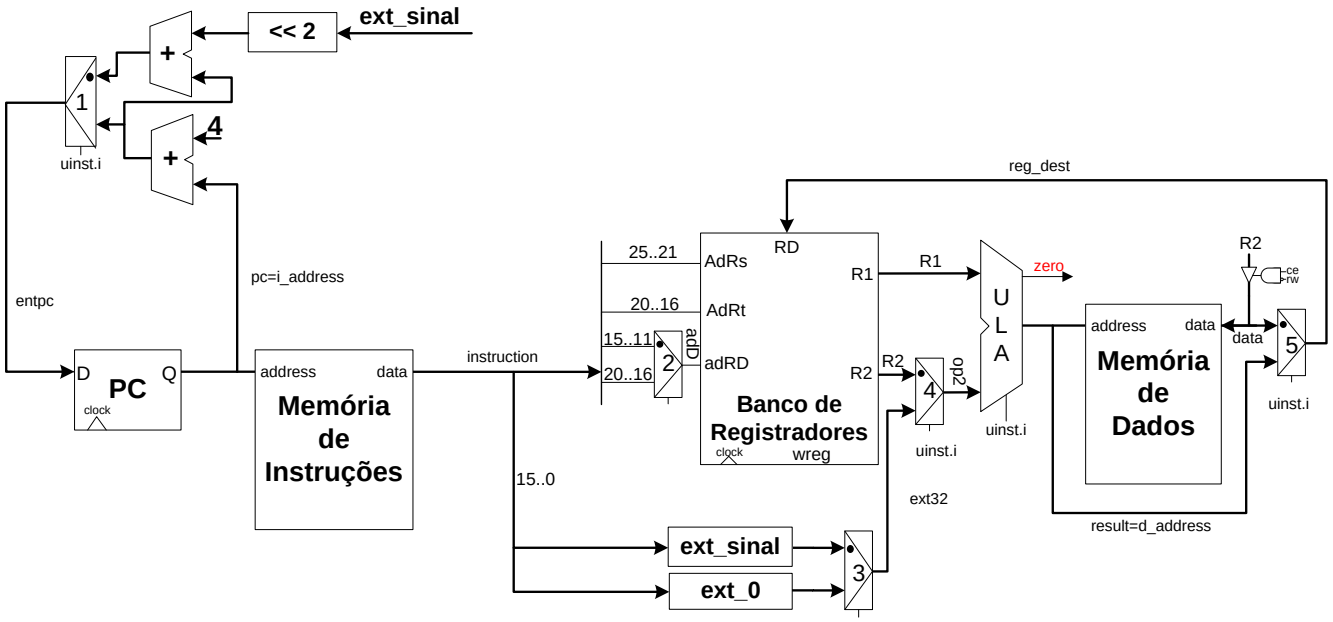
   Estes testes dependem do caractere processado a cada volta do laço (ver detalhamento disto na resposta ao item c) abaixo). Com os dados fornecidos, o laço será executado 10 vezes, mais a execução final que sai fora dele. Das 10 execuções, o caso (i) acontece 2 vezes, para o caractere branco e para o caractere !; o caso (ii) acontece também 2 vezes, para as letras maiúsculas; o caso (iii) não ocorre (não há caractere com código maior que 'Z' e menor que 'a'); o caso (iv) não ocorre (pois não há caractere com código maior que 'z'); finalmente o caso (v) ocorre 6 vezes, para todas as letras minúsculas. Assim, o número total de ciclos gastos para executar o programa é 8+(2*6+2*9+0*10+0*12+6*14+2)+4=**128 ciclos**.

b) Como o relógio é de 300MHz, o período deste é $1/300*10^6=3{,}33x10^{-9}$s. Logo, o tempo de execução é $128+3{,}33x10^{-9}s=4{,}27x10^{-7}$s, ou quase 0,43 microssegundos.

c) Este programa computa o número de letras maiúsculas e letras minúsculas em uma cadeia ASCII dada, armazenada na memória a partir do endereço associado ao rótulo **str**. Ao final da contagem, estes valores são escritos nas variáveis **uca** (*upper case letters*) e **lca** (*lower case letters*) para letras maiúsculas e minúsculas, respectivamente.

d) Não, este programa não possui qualquer subrotina, pois não se usa em nenhuma parte dele um dos pares de instruções que define tais estruturas na linguagem de montagem do MIPS (**jal** ou **jalr** seguido de **jr $ra**).

2. [3 pontos] Considere o bloco de dados monociclo apresentado abaixo. Ele resultou da transformação do bloco de dados visto em aula para dar suporte às novas instruções BEQ e LUI. O ponto nos multiplexadores indica condição verdadeira (ou, equivalentemente, sinal de controle do mux em 1). Assuma que as instruções às quais este processador dá suporte de execução são apenas as seguintes: **ADDU, SUBU, AND, OR, XOR, NOR, LW, SW, ORI, BEQ** e **LUI**.



a) Suponha que existe uma única falha, que afeta apenas o multiplexador que escolhe na entrada op2 da ULA (Mux 4). A falha é que o sinal de controle deste multiplexador está grudado eternamente no valor 1. Diga quais instruções ainda podem ser executadas corretamente, justificando para cada uma delas.

b) Observe na documentação do MIPS a funcionalidade da instrução XORI e diga se e como o hardware acima deveria ser alterado para ser capaz de executar esta instrução. Que operação deve ser realizada na ULA? Quais condições deveriam ser fixadas nos 5 multiplexadores para executar esta instrução? Diga se alguma condição de multiplexador é irrelevante.

Solução:

a) O problema afeta todas as instruções que necessitam um valor estendido a partir dos 16 bits inferiores da palavra de instrução, com exceção da instrução **BEQ**, que não usa este mux. Ora, as instruções tipo R não usam nem extensão de 0 nem extensão de sinal. Logo, elas não são afetadas pela faha do mux 4 que deixa sempre passar o que elas escolheriam passar para a saída do mux. Logo as instruções **ADDU, SUBU, AND, OR, XOR** e **NOR** ainda são executadas corretamente. LW e SW, por outro lado, necessitam da extensão de sinal que não passa mais para a entrada da ULA. Logo elas não funcionam mais. LUI e ORI necessitam da extensão, logo estas não funcionam mais. **BEQ**, por sua vez continua a funcionar, pois não usa de forma alguma a saída do mux 4.

b) A instrução **XORI** usa no bloco de dados exatamente os mesmos caminhos que a instrução ORI. Sua única diferença em relação a esta última é que a ULA deve executar uma operação XOR bit a bit de suas entradas op1 e op2. No bloco de controle deve-se acrescentar a lógica de decodificação do XORI e acrescentar o nome desta no tipo inst_type. No bloco de dados absolutamente nada muda, mas a ULA é alterada para executar a operação XOR bit a bit não apenas no caso da instrução **XOR**, mas também no caso de **uins.i=XORI**. As condições dos muxs são as seguintes:

- Controle do Mux 1 = 0 (incrementa o PC);
- Controle do Mux 2 = 0 (instrução(20 downto 16) define o registrador destino);
- Controle do Mux 3 = 0 (deixa passar extensão de 0);
- Controle do Mux 4 = 0 (deixa passar extensão de 0 para entrada da ULA);
- Controle do Mux 5 = 0 (saída da ULA, result, passa para reg_dest).

Não existe sinal de controle irrelevante em nenhum mux.

3. (3,5 pontos) Considere as linhas de código VHDL abaixo, que são parte da descrição do MIPS monociclo (MIPS_V0). Este trecho contém a definição do sinal adD do bloco de dados, que é a saída do multiplexador cuja saída é o endereço do registrador de destino para instruções que escrevem no banco de registradores. Assuma como verdadeiro o seguinte fato:

- As instruções da MIPS monociclo são codificadas em 4 bits com valores na faixa de 0000 a 1000, na seguinte ordem crescente de valores: (**ADDU, SUBU, AND, OR, XOR, NOR, LW, SW, ORI**).

Pede-se:

a) Defina os códigos associados às instruções mencionadas no VHDL.

b) Usando os códigos definidos no item a), desenhe um diagrama esquemático que use apenas portas lógicas e componentes tais como multiplexadores, demultiplexadores, decodificadores, e que corresponda a um circuito correto e completo que pode resultar da conversão do código VHDL dado para hardware.

Dicas: (*i*) Assuma que as entradas do esquemático são todos os sinais não atribuídos no trecho VHDL dado; (*ii*) O sinal adD é a única saída do esquemático, e instR é um sinal interno do mesmo.

```
1.    instR <= '1' when uins.i=ADDU or uins.i=SUBU or
2.              uins.i=AAND or uins.i=OOR or uins.i=XXOR or uins.i=NNOR else
3.             '0';
4.    adD<=     instruction(15 downto 11) when instR='1' else
5.         instruction(20 downto 16);
```

Solução:

a) A solução desta questão é trivial. Dado o pressuposto, a codificação é óbvia: ADDU=0000, SUBU=0001, AND=0010, OR=0011, XOR=0100, NOR=0101, LW=0110, SW=0111 e ORI=1000.

b) Usando as dicas fornecidas, a solução desta questão é o diagrama de esquemáticos abaixo: