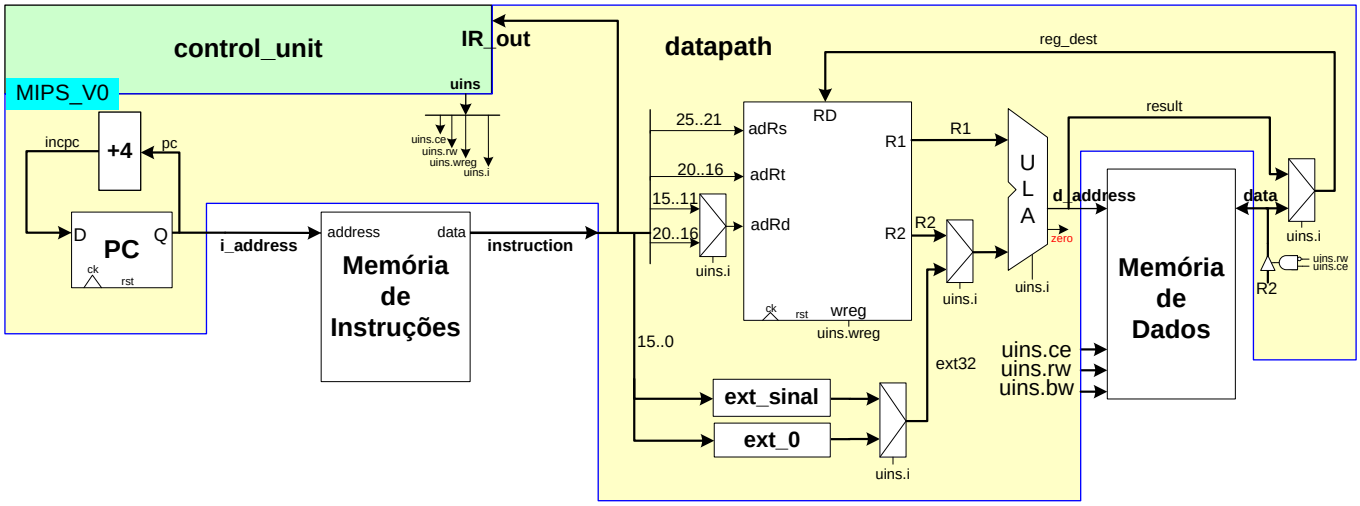
Prova P2 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 11/novembro/2013

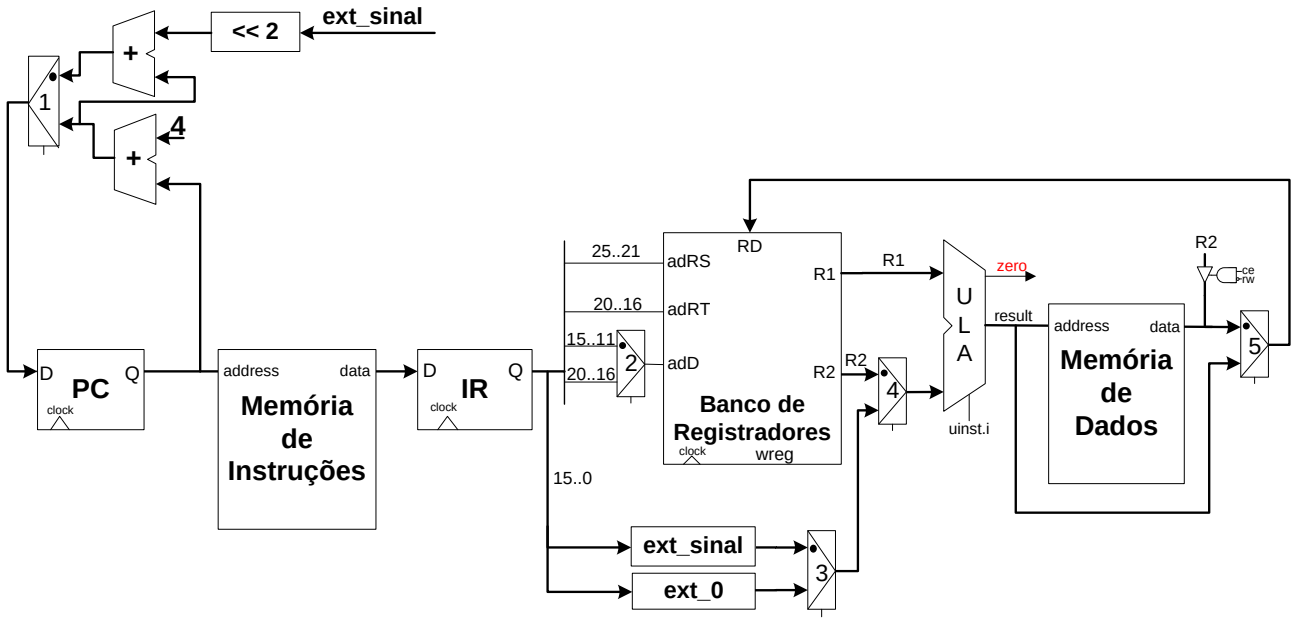
Para realizar a prova, refiram-se à proposta de organização MIPS monociclo vista em aula. O desenho da versão monociclo aparece abaixo, com detalhamento do Bloco de Dados. Assuma que as instruções às quais a organização monociclo dá suporte de execução são apenas as seguintes (exceto se a questão particular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



1. [4 pontos] Assuma uma frequência de operação de 500 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) (1,5 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;
   d) (0,5 pontos) Este programa possui subrotinas? Se sim, onde esta se encontra (defina linhas)?

```
1.          .text
2.          .globl main
3.  main:   la      $a0,num
4.          lw      $a0,0($a0)
5.          jal     cbts
6.          la      $t0,nb
7.          sb      $v0,0($t0)
8.          li      $v0,10
9.          syscall
10. cbts:   li      $v0,32
11.         beq     $a0,$zero,ez
12.         lui     $t0,0x8000
13. loop:   and     $t1,$a0,$t0
14.         bne     $t1,$zero,fcb
15.         addiu   $v0,$v0,-1
16.         srl     $t0,$t0,1
17.         j       loop
18. fcb:    jr      $ra
19. ez:     li      $v0,1
20.         jr      $ra
21.         .data
22. num:    .word   62
23. nb:     .byte   0
```

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 entre a extensão de sinal e a extensão de zero (Mux 3). A falha é que o sinal de controle deste multiplexador está grudado eternamente no valor 0. Diga quais instruções ainda podem ser executadas corretamente, justificando para cada uma delas.

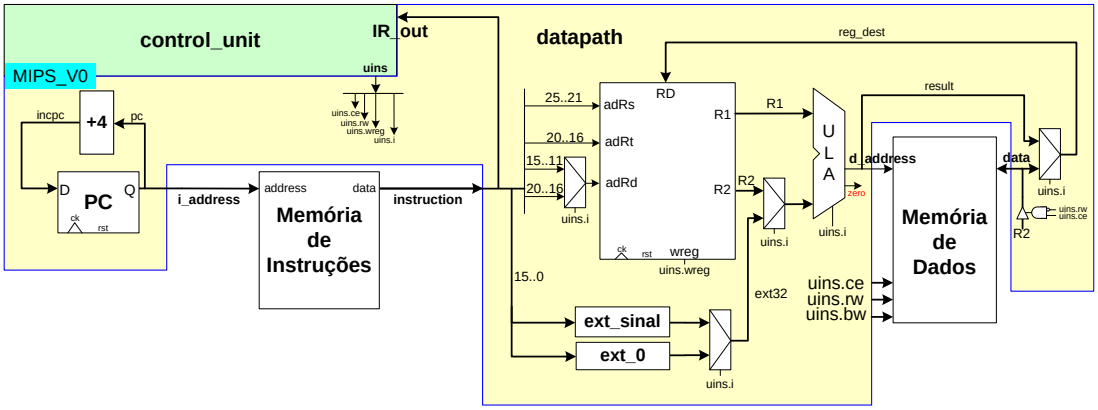
b) Agora suponha que existe outra falha única, que afeta apenas o multiplexador que escolhe que valor a ser escrito no banco de registradores por alguma instrução (o Mux 5). A falha é que o sinal de controle deste multiplexador está grudado eternamente no valor 0. Diga quais instruções não podem mais ser executadas corretamente, justificando para cada uma delas.

c) Estude na documentação do MIPS a funcionalidade da instrução SRL 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 pontos] Assuma que se quer dotar a implementação monociclo da questão anterior (2) da capacidade de executar as instruções BNE e BGTZ. Existe uma lógica não mostrada na Figura que gera o controle do multiplexador responsável por selecionar o valor a ser escrito no PC (o Mux 1). Diga como esta lógica é construída (1 ponto). Em seguida descreva como ela deveria ser alterada para dar suporte às 2 novas instruções de salto (2 pontos). Você pode usar portas lógicas ou VHDL na resposta, mas seja preciso no nível da estrutura de hardware necessária.

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. [4 pontos] Assuma uma frequência de operação de 500 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) (1,5 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;
   d) (0,5 pontos) Este programa possui subrotinas? Se sim, onde esta se encontra (defina linhas)?

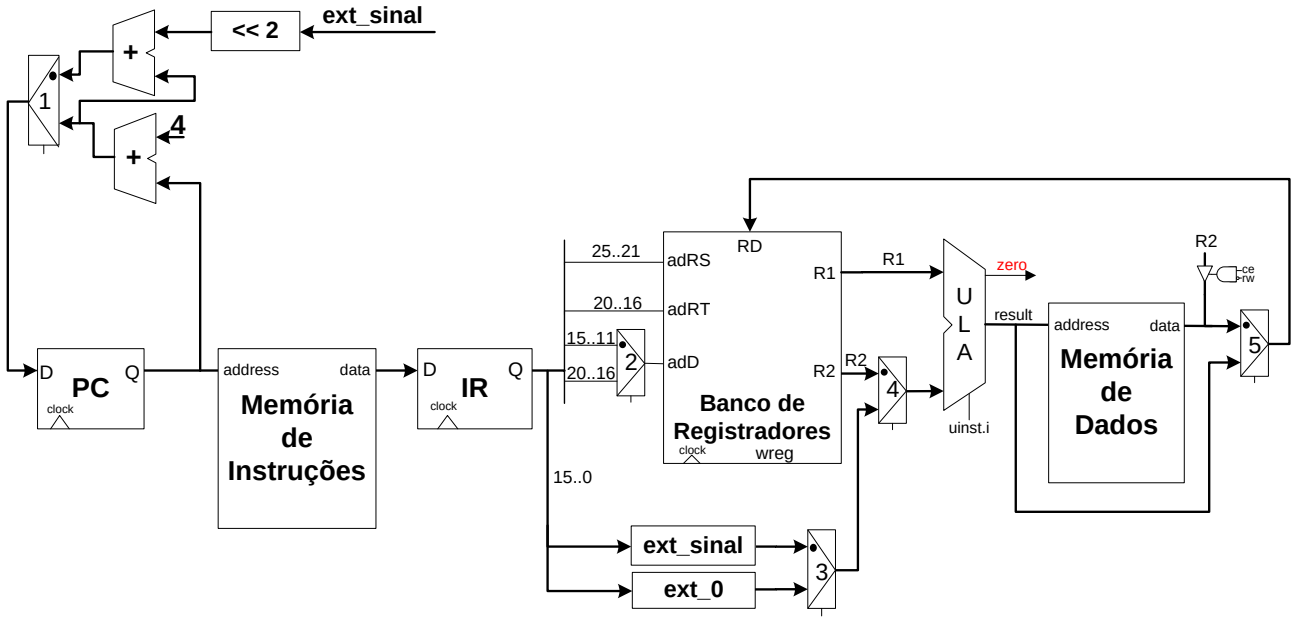
```
1.          .text                   # 0 ciclos - diretiva
2.          .globl main             # 0 ciclos - diretiva
3.  main:   la      $a0,num         # 2 ciclos - pseudo que gera lui+ori
4.          lw      $a0,0($a0)      # 1 ciclo -
5.          jal     cbts            # 1 ciclo -
6.          la      $t0,nb          # 2 ciclos - pseudo que gera lui+ori
7.          sb      $v0,0($t0)      # 1 ciclo -
8.          li      $v0,10          # 1 ciclo - pseudo que gera addiu
9.          syscall                 # 1 ciclo -

10. cbts:   li      $v0,32          # 1 ciclo - pseudo que gera addiu
11.         beq     $a0,$zero,ez    # 1 ciclo -
12.         lui     $t0,0x8000      # 1 ciclo -
13. loop:   and     $t1,$a0,$t0     # 1 ciclo -
14.         bne     $t1,$zero,fcb   # 1 ciclo -
15.         addiu   $v0,$v0,-1      # 1 ciclo -
16.         srl     $t0,$t0,1       # 1 ciclo -
17.         j       loop            # 1 ciclo -
18. fcb:    jr      $ra             # 1 ciclo -
19. ez:     li      $v0,1           # 1 ciclo - pseudo que gera addiu
20.         jr      $ra             # 1 ciclo -

21.         .data                   # 0 ciclos - diretiva
22. num:    .word   62              # 0 ciclos - diretiva
23. nb:     .byte   0               # 0 ciclos - diretiva
```

Solução:

a) Cada instrução do programa principal (linhas 3-9) somente é executada uma vez, gastando **9 ciclos** de relógio para executar. A subrotina **cbts**, por outro lado (linhas 10 a 20) possui uma parte que é executada apenas uma vez (com a área de dados fornecida), formada pelas linhas 10 a 12 e a linha 18, perfazendo um total de **4 ciclos**. Note-se que o **BEQ** da linha 11 não vai saltar (pois 62 é diferente de 0). Logo, as linhas 19 a 20 não serão executadas. Resta apenas definir o número de ciclos para executar o laço entre as linhas 13 e 17. Para isto, parte-se da codificação binário do parâmetro passado para a subrotina (62): 0000 0000 0000 0000 0000 0000 0011 1110. Note-se então que o número de voltas do laço é o número de 0s à esquerda deste valor mais 1, sendo que a última vez (quando achar o primeiro 1) o laço executa apenas as instruções das linhas 13 e 14 (**2 ciclos**). Nas outras vezes o laço

é percorrido inteiro levando 5 ciclos. Assim o número total de ciclos gastos para executar o programa é 9+4+26*5+2=**145 ciclos**. Note-se que 26 é o número de zeros à esquerda na representação em binário do valor 62, ou seja, o número de vezes que o laço é completamente executado.
b) Como o relógio é de 500MHz, o período deste é $1/500*10^{6}=2x10^{-9}$s. Logo, o tempo de execução é $145+2x10^{-9}s=290x10^{-9}$s.
c) Este programa calcula o número mínimo de bits necessário para armazenar o valor contido na posição **num** de memória, chamando a subrotina **cbts**, que faz o cálculo. Ela faz isto a partir do tamanho máximo de bits que se pode usar para armazenar um valor em uma palavra no MIPS (32, valor este armazenado inicialmente em $v0) e observa os bits do valor passado como parâmetro da esquerda para a direita, até achar o primeiro bit em 1. Enquanto não achar um bit em 1, a rotina decrementa de 1 o número de bits necessário para representar o parâmetro. O teste no início da rotina serve para detectar o caso especial do número 0, que mesmo sem ter nenhum bit em 1 necessita de 1 bit para ser armazenado.
d) Sim, e esta ocupa as linhas 10 a 20 do código fonte. É uma subrotina do tipo folha, e ela é chamada pelo programa principal uma vez, na linha 5. Recebe um parâmetro via registrador de argumento $a0 e retorna um valor via o registrador de retorno, $v0.

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 entre a extensão de sinal e a extensão de zero (Mux 3). A falha é que o sinal de controle deste multiplexador está grudado eternamente no valor 0. Diga quais instruções ainda podem ser executadas corretamente, justificando para cada uma delas.
b) Agora suponha que existe outra falha única, que afeta apenas o multiplexador que escolhe que valor a ser escrito no banco de registradores por alguma instrução (o Mux 5). A falha é que o sinal de controle deste multiplexador está grudado eternamente no valor 0. Diga quais instruções não podem mais ser executadas corretamente, justificando para cada uma delas.
c) Estude na documentação do MIPS a funcionalidade da instrução SRL 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) Primeiro, as instruções tipo R não usam nem extensão de 0 nem extensão de sinal. Logo, elas prescindem deste mux ,pois o valor por ele gerado é bloqueado pelo mux 4 que deixa passar para a entrada da ULA o sinal R2. 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. Para LUI a extensão é irrelevante, desde que o mux passa uma das duas. Logo **LUI** ainda funciona. O mesmo ocorre com o ORI, pois a falha deixa passar sempre a extensão de 0, que é o que o ORI necessita na ULA. **BEQ**, por sua vez continua a funcionar, pois não usa de forma alguma a saída do mux 3. Em resumo, todas as instruções continuam a funcionar, exceto LW e SW.

b) O mux 5 com a falha descrita vai afetar apenas a leitura da memória, logo a única instrução que para de funcionar neste caso é **LW**, que precisa que o dado proveniente da memória passe para a saída do mux 5. Mesmo SW continua funcionando, pois ela não usa o mux 5 (ver o tristate que conduz R2 para a entrada da memória).

c) Consultando o Apêndice A, nota-se que a instrução SRL realiza o deslocamento de **rt** para a direita por shamt bits, um valor imediato entre 0 e 31, escolhido pelo programador, e coloca o resultado no registrado **rd**. O formato da SRL é:

```
                          srl   rd,rt,shamt
                           0     0  rt  rd  shamt    2
Número de bits/campo:      6     5   5   5    5      6
```

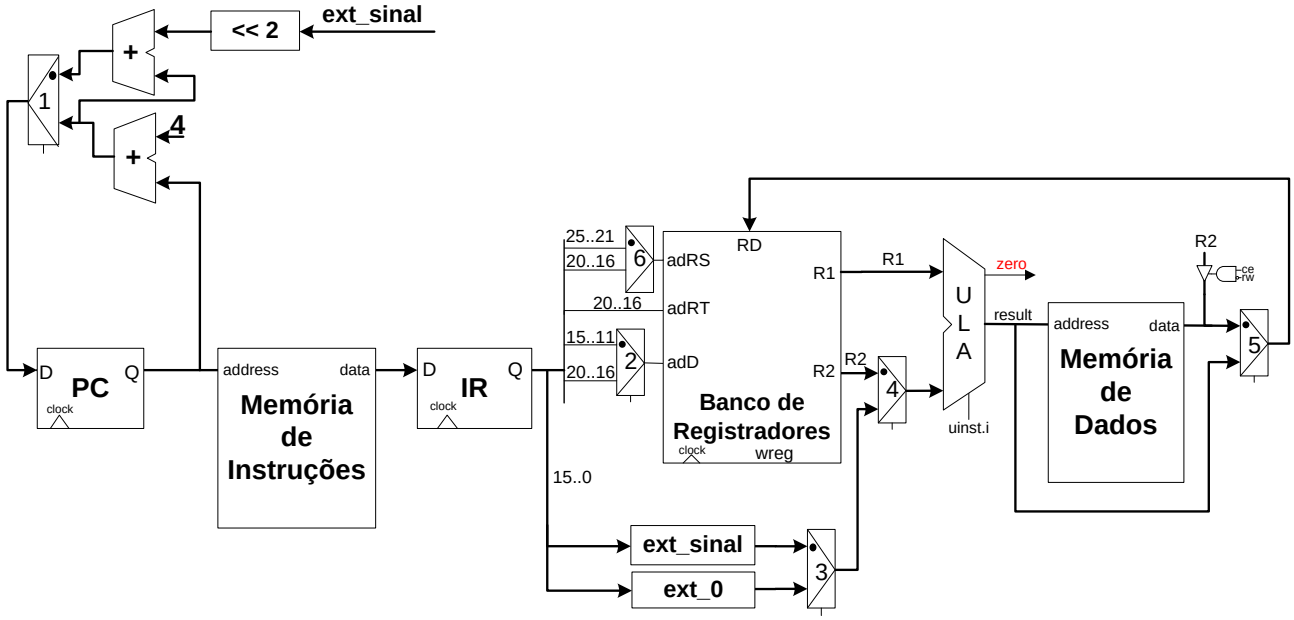
Note-se que o campo **rs** não é usado pela SRL. Como precisamos de um registrador e um valor imediato (**shamt**) entrando na ULA, é natural e mais simples alterar o hardware de forma que o registrador vá para a entrada superior da ULA e que o **shamt** vá pela entrada inferior desta. A ULA vai realizar o deslocamento dos bits na entrada **op1** pelo número de bits especificado pelo **shamt**, presente na entrada **op2**. Note-se que o campo **shamt** ocupa os bits 10-6 da instrução, que passa inalterado pelos blocos de extensão e deve aparecer nos bits 10-6 do **op2**. Assim, a ULA deve ser alterada para conter uma nova operação. Basta acrescentar a seguinte linha VHDL na ULA, usando a palavra reservada **srl** de VHDL:

```
...
op1 srl  CONV_INTEGER(op2(10 downto 6)  when  op_alu=SRL   else
...
```

Outra alteração importante é garantir que o campo **rt** seja usado para produzir **op1** (entrada superior da ULA), pois hoje as linhas que entram no banco para gerar a saída **R1** são apenas as que correspondem ao campo **rs** (bits 25-21). Logo, deve-se inserir um mux na entrada **adRs** (digamos o mux 6) do banco para permitir escolher entre o campo **rs** e o campo **rt** (bits 20-16) da palavra de instrução como endereço da porta de leitura 1 do banco. Estas são as principais alterações necessárias para a MIPS-V0 dar suporte à instrução SRL. Os muxes devem ser configurados como segue:

mux 1 - deixa passar entrada inferior (incrementa PC de 4)
mux 2 - deixa passar bits 15-11 para permitir que o campo rd enderece o registrador destino
mux 3 - pode passar qualquer coisa, ext_sinal ou ext_0
mux 4 - deixa passar entrada inferior para sua saída (saída do bloco de extensão para entrada op2 da ULA))
mux 5 - deixa passar entrada inferior para sua saída (saída da ULA para entrada do banco)

O novo Bloco de Dados ficaria como no desenho abaixo



3. [3 pontos] Assuma que se quer dotar a implementação monociclo da questão anterior (2) da capacidade de executar as instruções BNE e BGTZ. Existe uma lógica não mostrada na Figura que gera o controle do multiplexador responsável por selecionar o valor a ser escrito no PC (o Mux 1). Diga como esta lógica é construída (1 ponto). Em seguida descreva como ela deveria ser alterada para dar suporte às 2 novas instruções de salto (2 pontos). Você pode usar portas lógicas ou VHDL na resposta, mas seja preciso no nível da estrutura de hardware necessária.

Solução:

a) Como visto em aula, a lógica VHDL pode ser descrita assim:

```
uins.i=BEQ and zero='1'
```

b) Agora vamos supor que a ULA realiza a mesma operação que é feita para o BEQ para o BNE. Assim, o valor do sinal zero vai para '0' quando indicar que os registradores comparados são diferentes (ou seja, a condição para saltar).

Adicionalmente, para a instrução BGTZ, assuma-se que a ULA foi mudada para fazer comparação de magnitude (BGTZ) e que a condição maior que 0 é indicada colocando o sinal zero='1'. Dados estes pressupostos e sabendo que o cálculo de endereço de salto é idêntico para as três instruções (BEQ, BNE e BGTZ), fica simples determinar a nova expressão de controle do mux 1, que é, em VHDL: (uins.i=BEQ and zero='1') or (uins.i=BNE and zero='0') or (uins.i=BGTZ and zero='1').

Se o pressuposto de BGTZ acima não for assumido, outra forma seria assumir que a operação realizada na ULA é uma subtração de 0 do operando da BGTZ (o que passa este operando para a saída da ULA). Com isto, é possível testar se zero=0 (indicando que o número testado não é 0) ao mesmo tempo em que o bit 31 da saída da ALU é 0, ou seja outalu(31)='0'. Estas duas condições só podem ser simultaneamente verdadeiras se o número testado é maior que 0. O código de controle do mux neste caso ficaria (uins.i=BEQ and zero='1') or (uins.i=BNE and zero='0') or (uins.i=BGTZ and zero='0' and outalu(15)='0').