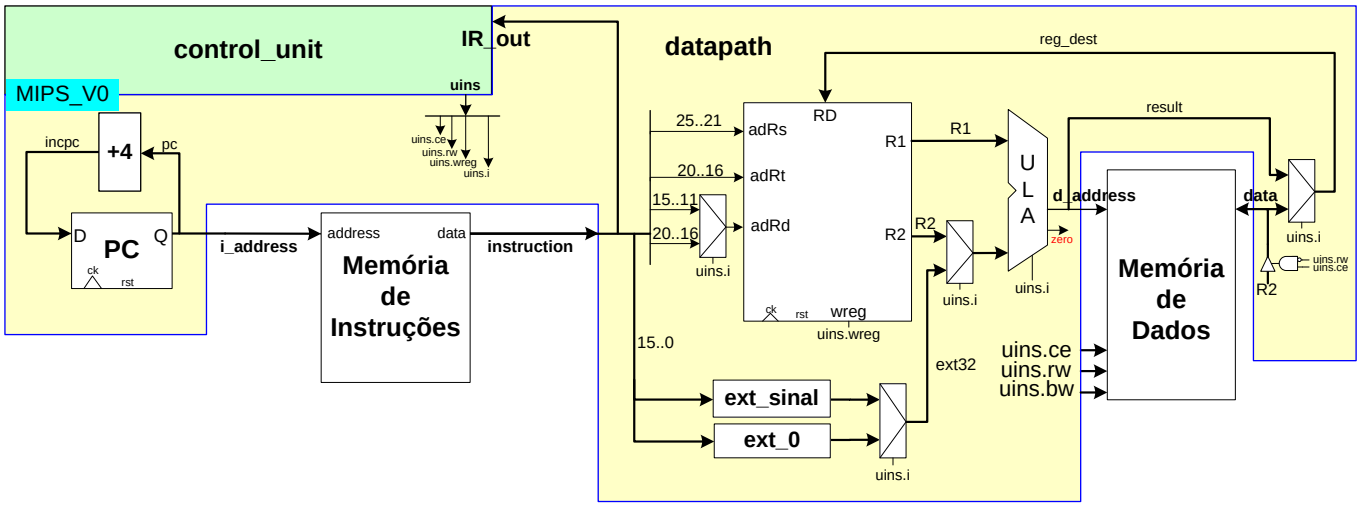
Prova P2 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 06/junho/2012

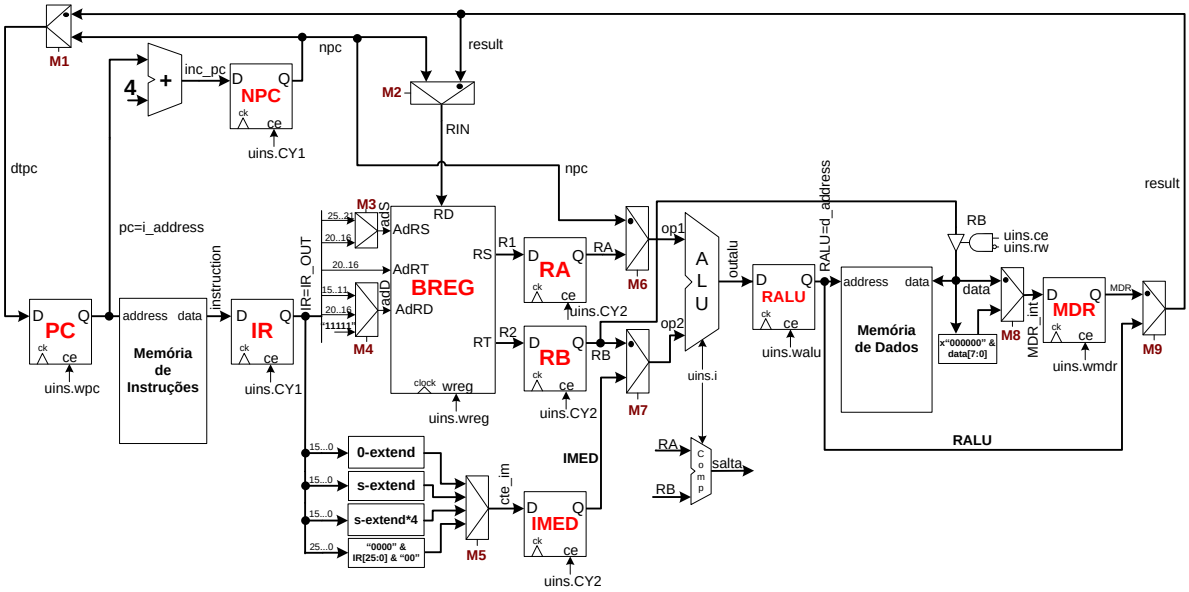
Para realizar a prova, refiram-se as propostas de organização MIPS monociclo e multiciclo vistas em aula. O desenho da versão monociclo aparece abaixo, com detalhamento do Bloco de Dados. O Bloco de Dados da versão multiciclo encontra-se no verso. Assuma que as instruções às quais o processador monociclo 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] Seja uma frequência de operação de 400 MHz para o processador MIPS monociclo e assuma que a organização original foi alterada para dar suporte à execução de todas as instruções do programa abaixo, mantendo sua característica monociclo. Calcule para o programa abaixo:
   a) O número de ciclos que leva a execução do programa, com a área de dados fornecida;
   b) O tempo de execução do programa em μs (1μs=10^{-6} segundos);
   c) Diga o que faz este programa, do ponto de vista semântico;
   d) Este programa possui subrotinas? Se sim, onde esta se encontra (em que linhas)?

```
1.          .data
2.   num:   .word 3
3.   result:.word 0
4.          .text
5.   main:  la      $t0,num
6.          lw      $a0,0($t0)
7.          addiu   $sp,$sp,-4
8.          sw      $ra,0($sp)
9.          jal     func
10.         lw      $ra,0($sp)
11.         addiu   $sp,$sp, 4
12.         la      $t0,result
13.         sw      $v0,0($t0)
14.         li      $v0,10
15.         syscall
16.  func:  addiu   $sp,$sp,-8
17.         sw      $ra,0($sp)
18.         sw      $a0,4($sp)
19.         sltiu   $t0,$a0,1
20.         beq     $t0,$zero,rec
21.         addiu   $v0,$zero,1
22.         lw      $ra,0($sp)
23.         addiu   $sp,$sp,8
24.         jr      $ra
25.  rec:   addiu   $a0,$a0,-1
26.         jal     func
27.         lw      $a0,4($sp)
28.         lw      $ra,0($sp)
29.         addiu   $sp,$sp,8
30.         multu   $v0,$a0
31.         mflo    $v0
32.         jr      $ra
```

Organização Multiciclo. Instruções a que esta organização dá suporte: **ADDU, SUBU, AND, OR, XOR, NOR, SLL, SLLV, SRA, SRAV, SRL, SRLV, ADDIU, ANDI, ORI, XORI, LUI, LBU, LW, SB, SW, SLT, SLTU, SLTI, SLTIU, BEQ, BGEZ, BLEZ, BNE, J, JAL, JALR, JR.**



2. [2,0 pontos] O código VHDL dado abaixo representa um dos sinais auxiliares descritos no início do Bloco de Dados multiciclo do MIPS, usado para detectar se uma dada instrução pertence à classe de instruções de saltos condicionais. Considere que o sinal uins.i é codificado com o menor número possível de bits, e que as instruções são codificadas com valores binários crescentes a partir de 0, na sequência dada acima do desenho do Bloco de Dados monociclo (ou seja, ADDU recebe código 0 , SUBU recebe código 1, etc.). A partir destes pressupostos, desenhe um diagrama de portas lógicas que poderia resultar da síntese desta linha do VHDL do processador. Use apenas portas lógicas quaisquer (inversores, portas E, OU, Não-E, Não-OU, etc.) e fios. Qual(is) é(são) a(s) entrada(s) do circuito e qual(is) sua(s) saída(s)?

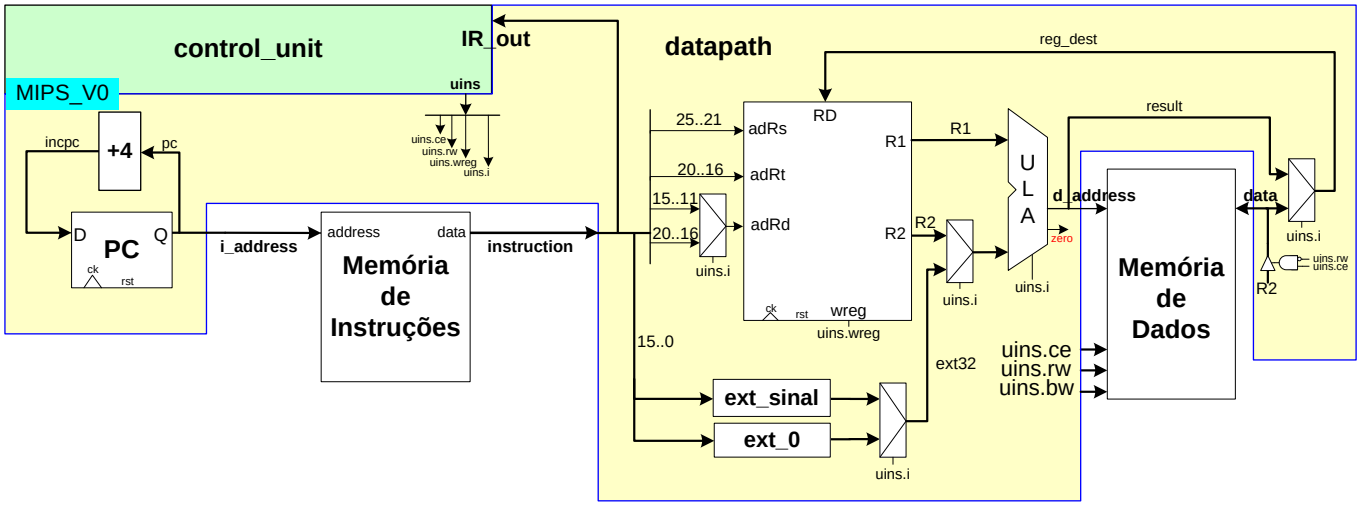
```
inst_branch <= '1' when uins.i=BEQ or uins.i=BGEZ or uins.i=BLEZ or uins.i=BNE else '0';
```

3. [2 pontos] Considere o bloco de dados monociclo apresentado na página anterior. Suponha que existe uma falha, que afeta apenas o multiplexador mais à direita no desenho, que gera a entrada de dados do Banco de Registradores (RD). A falha é que este componente sempre deixa passar a sua entrada superior para a saída, independente 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?

4. [2 pontos] Use a documentação do MIPS (Apêndice A) e o VHDL da versão multiciclo para esta questão. Defina a partir da funcionalidade da instrução **SRL**, do Bloco de Dados multiciclo e de sua implementação em VHDL quais os caminhos do Bloco de Dados multiciclo realmente transportam informação relevante para executar esta instrução. Marque este caminho no desenho do Bloco de Dados acima e marque os blocos de hardware necessários a sua execução. Que operação é realizada na ULA para esta instrução? Quais condições são fixadas nos multiplexadores para executar esta instrução (use a numeração dos multiplexadores para identificá-los, se achar interessante)? Diga se alguma condição de algum multiplexador é irrelevante.

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 o processador 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] Seja uma frequência de operação de 400 MHz para o processador MIPS monociclo e assuma que a organização original foi alterada para dar suporte à execução de todas as instruções do programa abaixo, mantendo sua característica monociclo. Calcule para o programa abaixo:
   a) O número de ciclos que leva a execução do programa, com a área de dados fornecida;
   b) O tempo de execução do programa em μs (1μs=10^{-6} segundos);
   c) Diga o que faz este programa, do ponto de vista semântico;
   d) Este programa possui subrotinas? Se sim, onde esta se encontra (em que linhas)?

```
1.          .data
2.  num:    .word 3                 # número cujo fatorial será calculado
3.  result:.word 0                  # local onde armazenar resultado do cálculo do fatorial
4.          .text
5.  main:   la      $t0,num         # 2 ciclos - Obtém endereço do número
6.          lw      $a0,0($t0)      # 1 ciclo  - valor armazenado em num em $a0, parâmetro de entrada
7.          addiu   $sp,$sp,-4      # 1 ciclo  - aloca 1 palavra na pilha
8.          sw      $ra,0($sp)      # 1 ciclo  - guarda end retorno p/ SO na pilha
9.          jal     func            # 1 ciclo  - chama subrotina
10.         lw      $ra,0($sp)      # 1 ciclo  - na volta, recupera end retorno da pilha
11.         addiu   $sp,$sp, 4      # 1 ciclo  - desaloca pilha
12.         la      $t0,result      # 2 ciclos - obtém endereço onde guardar fatorial
13.         sw      $v0,0($t0)      # 1 ciclo  - escreve resultado de fatorial em result
14.         li      $v0,10          # 1 ciclo  - prepara-se para sair do programa → SO
15.         syscall                 # 1 ciclo  - volta ao SO
16. func:   addiu   $sp,$sp,-8      # 1 ciclo  - Aloca pilha para eventual recursão
17.         sw      $ra,0($sp)      # 1 ciclo  - guarda end retorno no topo
18.         sw      $a0,4($sp)      # 1 ciclo  - guarda argumento de entrada abaixo do topo
19.         sltiu   $t0,$a0,1       # 1 ciclo  - $t0 recebe 1 se argumento < 1, senão 0
20.         beq     $t0,$zero,rec   # 1 ciclo  - vai para chamada recursiva se arg >=1
21.         addiu   $v0,$zero,1     # 1 ciclo  - senão, gera cte 1 em $v0, valor de retorno
22.         lw      $ra,0($sp)      # 1 ciclo  - recupera end retorno da pilha
23.         addiu   $sp,$sp,8       # 1 ciclo  - desaloca pilha alocado no inicio da rotina
24.         jr      $ra             # 1 ciclo  - retorna ao programa principal
25. rec:    addiu   $a0,$a0,-1      # 1 ciclo  - Quando chega aqui, recursão. arg <= arg-1
26.         jal     func            # 1 ciclo  - e recorre
27.         lw      $a0,4($sp)      # 1 ciclo  - Na volta da recursão, recupera arg da pilha
28.         lw      $ra,0($sp)      # 1 ciclo  - recupera endereço de retorno da pilha
29.         addiu   $sp,$sp,8       # 1 ciclo  - desaloca pilha
30.         multu   $v0,$a0         # 1 ciclo  - gera resultado =res recursão*arg original da chamada
31.         mflo    $v0             # 1 ciclo  - recupera valor para $v0 da parte baixa do resultado
32.         jr      $ra             # 1 ciclo  - e retorna ao programa principal
```

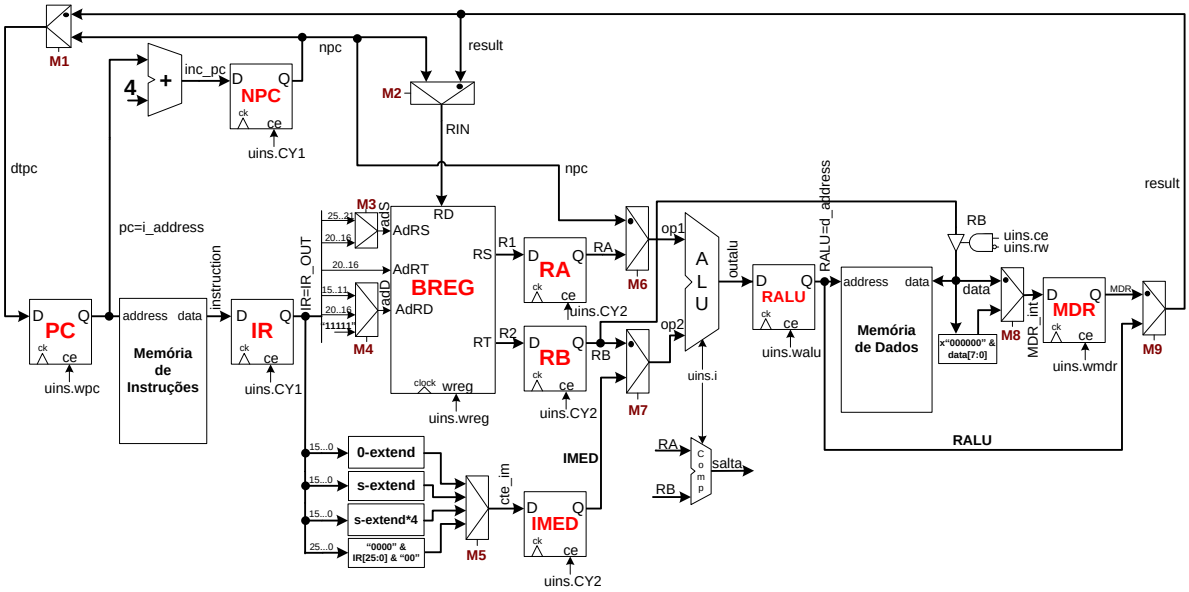
Solução:

a) As linhas 5-15 são executadas exatamente uma vez, gastando 13 ciclos de relógio. A subrotina recursiva **func** tem dois tipos de execução: não-folha, quando a execução implica uma chamada a ela mesma e folha, caso contrário. Cada chamada de **func** não-folha executa uma vez as linhas 16-20 e 25-32, gastando 13 ciclos de relógio. A chamada folha executa as linhas 16-24, gastando 9 ciclos. Com o dado inicial 3, se executa **func** com este valor do dado e se realiza uma chamada recursiva passando o dado decrementado de uma unidade. Como a folha da recursão só ocorre quando o dado for menor que 1, isto implica 3 execuções não-folha de func. Logo o número total de ciclos do programa com esta área de dados é 13+3*13+9=61 ciclos de relógio.

b) Um relógio de 400MHz implica um período de $(1/(400*10^6))$s ou 2,5ns=$2,5*10^{-3}$μs. Logo o tempo de execução do programa é $61*2,5*10^{-3}$μs=0,1525 μs.

c e d) Este programa calcula o fatorial do número armazenado na posição de memória **num**, usando a subrotina recursiva **func**, localizada entre as linhas 16 a 32 do programa. O resultado é armazenado na posição de memória **result**.

Organização Multiciclo. Instruções a que esta organização dá suporte: **ADDU, SUBU, AND, OR, XOR, NOR, SLL, SLLV, SRA, SRAV, SRL, SRLV, ADDIU, ANDI, ORI, XORI, LUI, LBU, LW, SB, SW, SLT, SLTU, SLTI, SLTIU, BEQ, BGEZ, BLEZ, BNE, J, JAL, JALR, JR.**

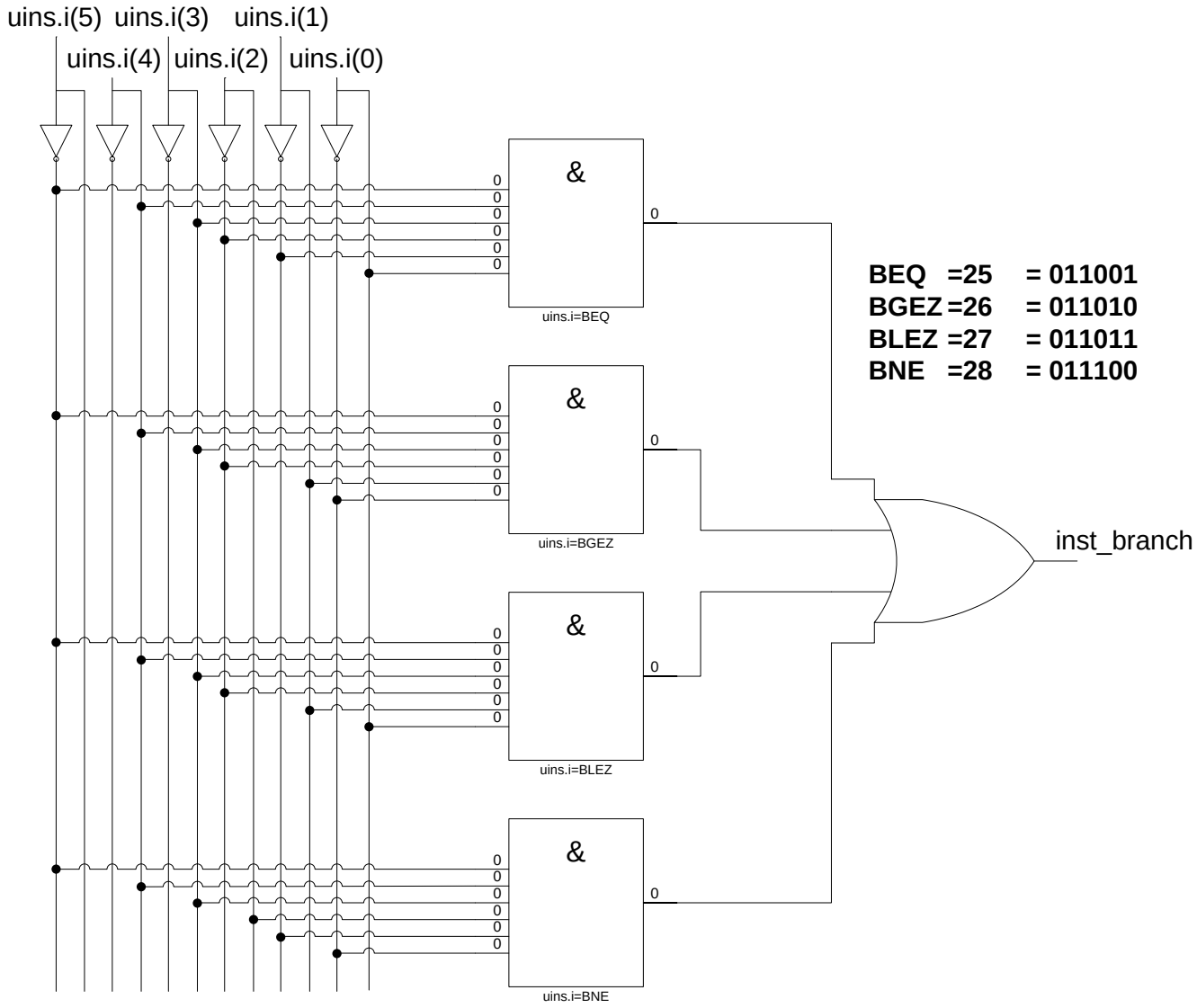


2. [2,0 pontos] O código VHDL dado abaixo representa um dos sinais auxiliares descritos no início do Bloco de Dados multiciclo do MIPS, usado para detectar se uma dada instrução pertence à classe de instruções de saltos condicionais. Considere que o sinal uins.i é codificado com o menor número possível de bits, e que as instruções são codificadas com valores binários crescentes a partir de 0, na sequência dada acima do desenho do Bloco de Dados monociclo (ou seja, ADDU recebe código 0 , SUBU recebe código 1, etc.). A partir destes pressupostos, desenhe um diagrama de portas lógicas que poderia resultar da síntese desta linha do VHDL do processador. Use apenas portas lógicas quaisquer (inversores, portas E, OU, Não-E, Não-OU, etc.) e fios. Qual(is) é(são) a(s) entrada(s) do circuito e qual(is) sua(s) saída(s)?

```
inst_branch <= '1' when uins.i=BEQ or uins.i=BGEZ or uins.i=BLEZ or uins.i=BNE else '0';
```

Solução:

Usando os pressupostos da questão, e notando que existem 33 instruções que devem ser identificadas pelo sinal uins.i, percebe-se que um mínimo de 6 bits são necessários para codificar este sinal em formato binário. Ainda, dos pressupostos de codificação pode-se notar que às instruções BEQ, BGEZ, BLEZ e BNE devem ser associados os códigos 25, 26, 27 e 28, respectivamente, o que em binário corresponde a 011001, 011010, 011011 e 011100. A partir daí, pode-se conceber a implementação da linha VHDL como um circuito onde portas AND de 6 entradas detectam cada uma das instruções, e as saídas destas podem ser combinadas através de uma porta OU de 4 entradas para gerar a saída inst_branch. O desenho abaixo mostra o circuito resultante. Do desenho nota-se que o circuito possui como entrada os seis bits do sinal uins.i(5 downto 0) e como única saída o sinal inst_branch. Note-se que o circuito mostrado ainda poderia ser simplificado. Usem um mapa de Karnaugh, por exemplo.



3. [2 pontos] Considere o bloco de dados monociclo apresentado na página anterior. Suponha que existe uma falha, que afeta apenas o multiplexador mais à direita no desenho, que gera a entrada de dados do Banco de Registradores (RD). A falha é que este componente sempre deixa passar a sua entrada superior para a saída, independente 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?

Solução:

As únicas instruções que podem ser afetadas são as que fazem acesso à memória. Mais ainda apenas as que leem dado da memória, pois a escrita não passa pelo multiplexador com falha. Logo, todas as instruções podem ser executadas sem nenhuma falha, exceto a instrução LW. A única possibilidade de a falha ser mascarada seria no caso em que o endereço da memória ao qual se quer fazer acesso conter um dado exatamente igual ao endereço onde ele está armazenada, um fato bastante incomum.

4. [2 pontos] Use a documentação do MIPS (Apêndice A) e o VHDL da versão multiciclo para esta questão. Defina a partir da funcionalidade da instrução **SRL**, do Bloco de Dados multiciclo e de sua implementação em VHDL quais os caminhos do Bloco de Dados multiciclo realmente transportam informação relevante para executar esta instrução. Marque este caminho no desenho do Bloco de Dados acima e marque os blocos de hardware necessários a sua execução. Que operação é realizada na ULA para esta instrução? Quais condições são fixadas nos multiplexadores para executar esta instrução (use a numeração dos multiplexadores para identificá-los, se achar interessante)? Diga se alguma condição de algum multiplexador é irrelevante.

Solução:

A primeira parte da questão está resolvida na figura abaixo que coloca em relevância (usando linhas vermelhas) os caminhos que realmente transportam informação relevante para executar a instrução. No desenho, se nota que o único multiplexador irrelevante para esta instrução e o M8, pois por ele não passa nenhuma informação relevante. Todos os demais são usados para transportar informação útil. A

ULA neste caso recebe dois operandos: em **op1** vem o conteúdo do registrador RT do formato da instrução (aquele endereçado pelos bits 20-16 do IR, quando este contém o código objeto de uma instrução **SRL**); em **op2** vem o valor resultante da extensão de sinal dos bits 15-0 do IR. Destes 32 bits, os únicos relevantes são os bits 6-10, que contém o valor do campo **shamt** da instrução **SRL** (shamt significa *shift amount* ou quantidade de deslocamento, um valor entre 0 e 31). A ULA usa estes 5 bits e desloca o valor na entrada **op1** de tantos bits para a direita quanto seja o valor nos bits 6-10 de **op2**, inserindo zeros à esquerda. Por exemplo, se op1=1001 1100 1111 0000 1010 0101 1110 1001 e op2=0000 0000 0000 0000 0000 1**000 11**00 0010, o resultado na saída da ULA será opula=0001 0011 1001 1110 0001 0100 1011 1101. Ou seja, o valor de op1 foi deslocado 3 bits para a direita, entrando 3 zeros à esquerda e os TRE bits menos significativos de op1 sendo perdidos. O valor do deslocamento deriva do fato de os bits 10-6 de op2 serem 00011, ou seja 3 em decimal.