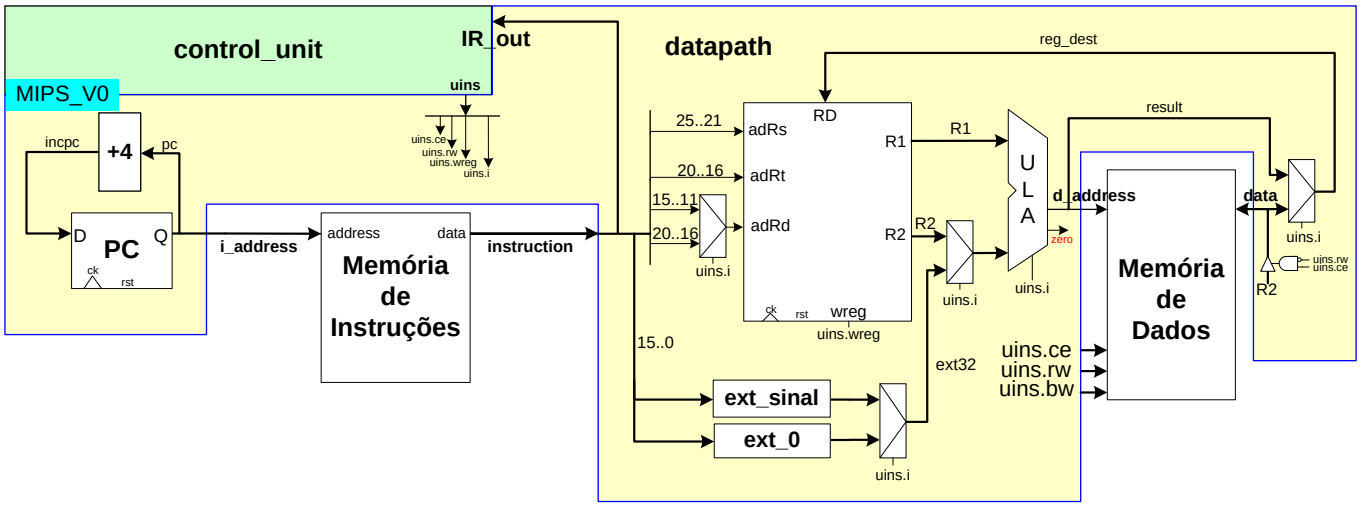
Prova P2 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 09/novembro/2012

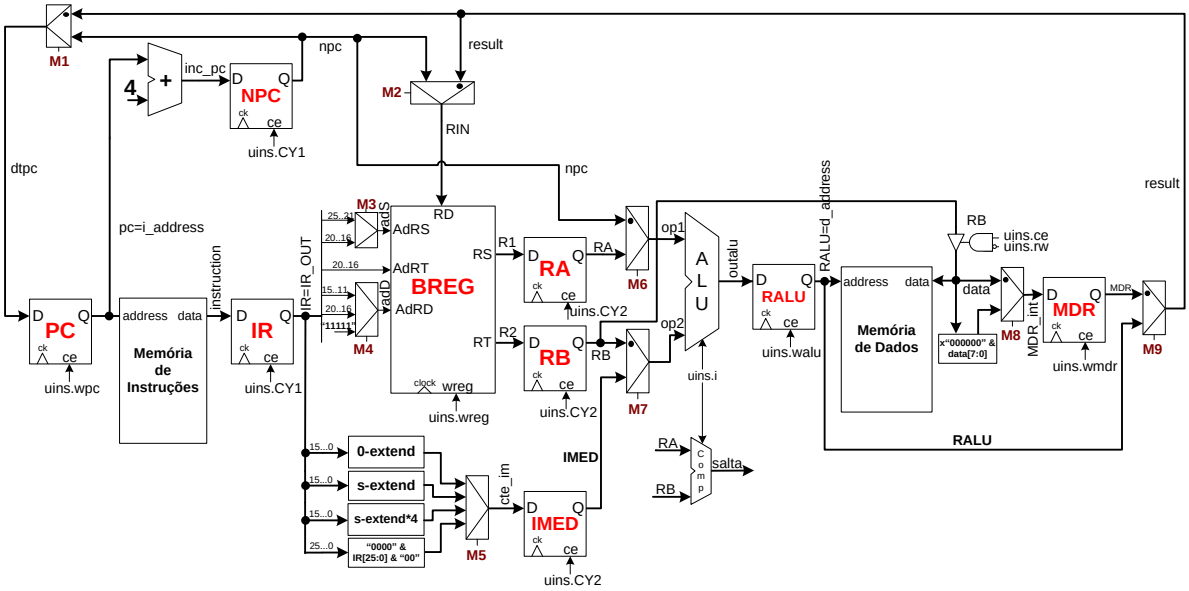
Para realizar a prova, refiram-se as propostas de organização MIPS monociclo e multiciclo vistas em aula. O desenho da versão monociclo aparece abaixo, com detalhamento do Bloco de Dados. O Bloco de Dados da versão multiciclo encontra-se no verso. Assuma que as instruções às quais o processador monociclo 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 500 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:
   1.1. O número de ciclos que leva a execução do programa, com a área de dados fornecida;
   1.2. O tempo de execução do programa em segundos (1ns=10^{-9} segundos);
   1.3. Diga o que faz este programa, do ponto de vista semântico;
   1.4. Este programa possui subrotinas? Se sim, onde esta se encontra (em que linhas)?

```
1.          .text
2.          .globl main
3.  main:   la      $t0,dor
4.          lw      $t0,0($t0)
5.          la      $t1,dendo
6.          lw      $t1,0($t1)
7.          li      $t2,0
8.  loop:   subu    $t1,$t1,$t0
9.          bltz    $t1,end
10.         addiu   $t2,$t2,1
11.         j       loop
12. end:    addu    $t1,$t1,$t0
13.         la      $t0,q
14.         sw      $t2,0($t0)
15.         la      $t0,r
16.         sw      $t1,0($t0)
17.         li      $v0,10
18.         syscall
19.         .data
20. dendo:  .word   122
21. dor:    .word   20
22. q:      .word   0
23. r:      .word   0
```

Bloco de Dados da organização MIPS 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**.



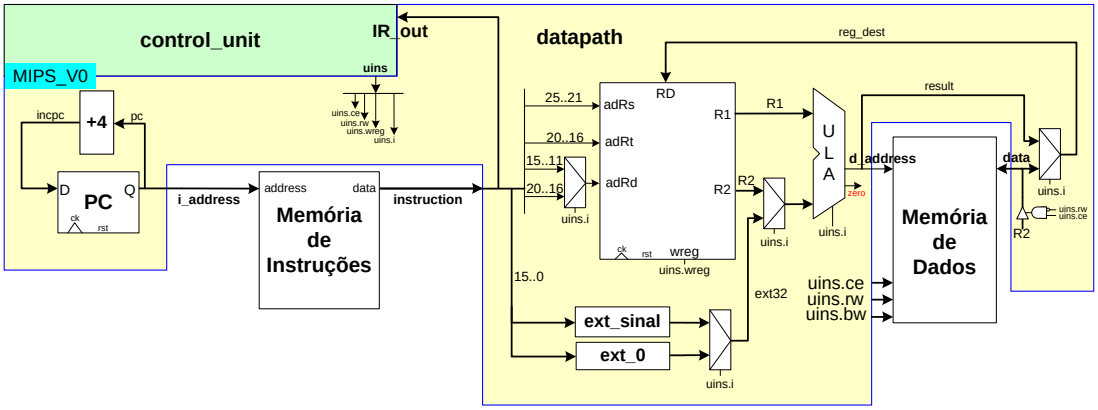
2. [3,0 pontos] Considere o bloco de dados multiciclo apresentado acima. Suponha que existe uma falha, que afeta apenas o multiplexador que gera a entrada do registrador IMED. A falha é que este componente sempre deixa passar a sua entrada superior (saída do bloco **`0-extend`**) para a saída, independente do valor do sinal de controle **M5** (que de fato corresponde ao sinal uins.i). Diga:
   2.1. Quais instruções (ou grupos de instruções) ainda podem ser executadas corretamente, justificando sua resposta.
   2.2. 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?

3. [3,0 pontos] O código VHDL dado abaixo representa um dos sinais importantes para a execução de saltos condicionais na organização **MIPS multiciclo** (instruções BEQ, BGEZ, BLEZ e BNE). Pede-se:
   3.1. Desenhe a interface externa do hardware gerado por este código. Mostre quem são as entradas e saídas e mencione o tamanho de cada sinal de interface, em número de fios. Se você não souber o tamanho exato de algum sinal, calcule o valor mínimo para o mesmo, a partir das informações disponíveis. Isto é possível para todos os sinais.
   3.2. Este código VHDL gera um comparador com quatro funcionalidades distintas. Escolha uma das quatro funcionalidades e dê um exemplo de implementação do hardware desta em portas lógicas. Se o hardware escolhido for muito grande, apenas esboce sua estrutura e explique-a.
   3.3. O programa da primeira questão desta prova usa a instrução BLTZ. Como você alteraria o comparador que gera o sinal salta tratado aqui para dar suporte à instrução BLTZ? Modifique o VHDL abaixo para realizar isto.

```
salta <=  '1' when ( (RA=RB  and uins.i=BEQ)  or (RA>=0  and uins.i=BGEZ) or
                     (RA<=0  and uins.i=BLEZ) or (RA/=RB and uins.i=BNE) )  else
          '0';
```

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 500 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:
   1.1. O número de ciclos que leva a execução do programa, com a área de dados fornecida;
   1.2. O tempo de execução do programa em segundos (1ns=10^{-9} segundos);
   1.3. Diga o que faz este programa, do ponto de vista semântico;
   1.4. Este programa possui subrotinas? Se sim, onde esta se encontra (em que linhas)?

```
1.          .text
2.          .globl main             #
3.  main:   la      $t0,dor         # 2 ciclos - Gera endereço do divisor
4.          lw      $t0,0($t0)      # 1 ciclo  - Carrega valor do divisor em $t0
5.          la      $t1,dendo       # 2 ciclos - Gera endereço do dividendo
6.          lw      $t1,0($t1)      # 1 ciclo  - Carrega valor do dividendo em $t1 (vira resto depois)
7.          li      $t2,0           # 1 ciclo  - Inicializa $t2 com 0 (vira quociente ao final)
8.  loop:   subu    $t1,$t1,$t0     # 1 ciclo  - Subtrai divisor do dividendo
9.          bltz    $t1,end         # 1 ciclo  - Se chegou a negativo, fim da divisão
10.         addiu   $t2,$t2,1       # 1 ciclo  - Senão, incrementa quociente
11.         j       loop            # 1 ciclo  - E volta para nova subtração
12. end:    addu    $t1,$t1,$t0     # 1 ciclo  - Ao fim, desfaz subtração que gerou negativo
13.         la      $t0,q           # 2 ciclos - Obtém endereço do quociente
14.         sw      $t2,0($t0)      # 1 ciclo  - Escreve quociente na memória
15.         la      $t0,r           # 2 ciclos - Obtém endereço do resto
16.         sw      $t1,0($t0)      # 1 ciclo  - Escreve resto na memória
17.         li      $v0,10          # 1 ciclo  - Prepara saída do programa
18.         syscall                 # 1 ciclo  - E sai
19.         .data                   #
20. dendo:  .word   122             # Valor do dividendo
21. dor:    .word   20              # Valor do divisor
22. q:      .word   0               # Lugar para guardar o quociente da divisão
23. r:      .word   0               # Lugar para guardar o resto da divisão
```

Solução:

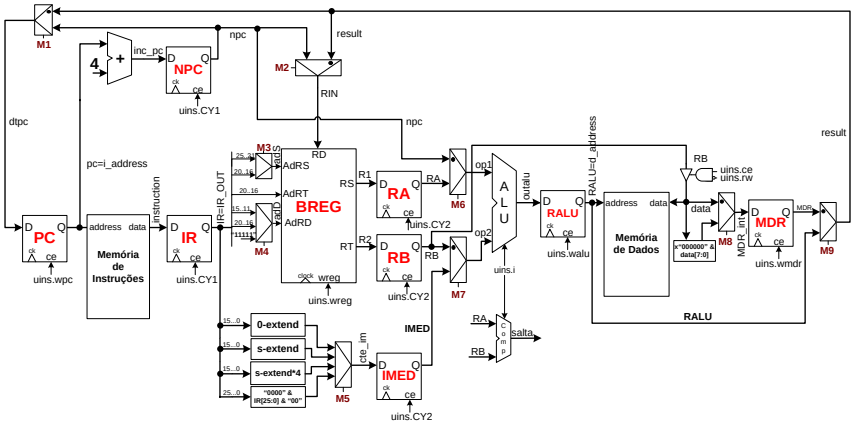
1.1. As linhas 1-2 não possuem instruções. As linhas 3-7(7 ciclos) e 12-18 (9 ciclos) são executadas exatamente uma vez, gastando um total de 16 ciclos de relógio. Uma execução intermediária do laço nas linhas 8-11 gasta 4 ciclos de relógio. A última vez que este é executado gasta 2 ciclos apenas. O trecho antes de entra no laço carrega o divisor (20) no registrador $t0 e o dividendo (122) no registrador $t1, além de inicializar o quociente em $t2 com 0. O laço subtrai o divisor do dividendo e atualiza este último a cada execução do laço com o resultado da subtração. Após cada subtração, testa-se o resultado deste para ver se ele é negativo, o que constitui a condição para deixar o laço. Assim, com os valores dados, o laço é executado exatamente 7 vezes, testando os valores <102, 82, 62, 42, 22, 2, -18(condição de saída)>. Logo o número total de ciclos do programa com esta área de dados é 7+9+6*4+2=42 ciclos de relógio.

1.2. Um relógio de 500MHz implica um período de $(1/(500*10^6))$s ou $2ns=2*10^{-9}$s. Logo o tempo de execução do programa é $42*2*10^{-9}s= 84*10^{-9}$s.

1.3. Este programa calcula o resultado inteiro da divisão de dois números naturais armazenados nas posições de memória **dendo** (o dividendo) e **dor** (o divisor), armazenando o quociente na posição de memória **q** e o resto na posição de memória **r**. O algoritmo usado é divisão por subtrações sucessivas.

1.4. O programa não possui subrotinas, pois em nenhum ponto do mesmo se encontra uma das instruções **jal** ou **jalr**.

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. [3,0 pontos] Considere o bloco de dados multiciclo apresentado acima. Suponha que existe uma falha, que afeta apenas o multiplexador que gera a entrada do registrador IMED. A falha é que este componente sempre deixa passar a sua entrada superior (saída do bloco 0-extend) para a saída, independente do valor do sinal de controle M5 (que de fato corresponde ao sinal uins.i). Diga:
   2.1. Quais instruções (ou grupos de instruções) ainda podem ser executadas corretamente, justificando sua resposta.
   2.2. 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:

2.1. Note-se que a MIPS multiciclo dá suporte a 33 instruções divididas nos grupos R, I e J. As instruções que não usam o registrador IMED de forma alguma são todas as instruções R que não usam deslocamento como dado imediato (ou seja, **ADDU, SUBU, AND, OR, XOR, NOR, SLLV, SRAV, SRLV, SLT, SLTU**) e as instruções de salto sem dado imediato, **JR, JALR.** Todas estas continuam a executar (obviamente) sem problema, pois não dependem do registrador afetado pela falha. Outras instruções que dependem deste registrador mas continuarão a funcionar são as que usam extensão de 0 (**ANDI, ORI, XORI**), ou as que não dependem do resultado da extensão nos 16 bits superiores, ou seja **LUI, SLL, SRA** e **SRL**. As demais instruções não funcionarão mais corretamente (quais sejam: **ADDIU, LBU, LW, SB, SW, SLTI, SLTIU, BEQ, BGEZ, BLEZ, BNE, J, JAL**), pois dependem de IMED receber um valor diferente de extensão de 0.

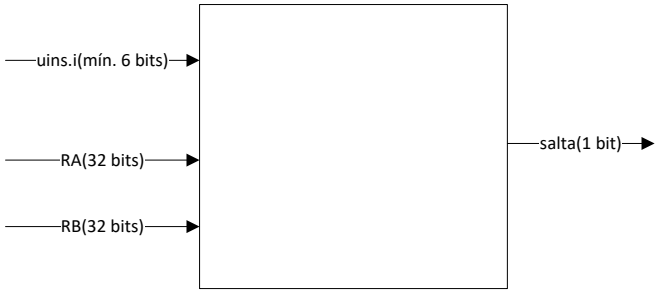
2.2. Instruções que podem eventualmente funcionar e eventualmente não funcionar são aquelas que dependem de IMED receber o resultado do bloco de extensão de sinal (**ADDIU, LBU, LW, SB, SW, SLTI, SLTIU**) Elas funcionarão corretamente sempre que o dado imediato for um número não-negativo em 16 bits e funcionarão incorretamente sempre que o dado imediato for um número negativo.

3. [3,0 pontos] O código VHDL dado abaixo representa um dos sinais importantes para a execução de saltos condicionais na organização MIPS multiciclo (instruções BEQ, BGEZ, BLEZ e BNE). Pede-se:
   3.1. Desenhe a interface externa do hardware gerado por este código. Mostre quem são as entradas e saídas e mencione o tamanho de cada sinal de interface, em número de fios. Se você não souber o tamanho exato de algum sinal, calcule o valor mínimo para o mesmo, a partir das informações disponíveis. Isto é possível para todos os sinais.
   3.2. Este código VHDL gera um comparador com quatro funcionalidades distintas. Escolha uma das quatro funcionalidades e dê um exemplo de implementação do hardware desta em portas lógicas. Se o hardware escolhido for muito grande, apenas esboce sua estrutura e explique-a.
   3.3. O programa da primeira questão desta prova usa a instrução BLTZ. Como você alteraria o comparador que gera o sinal **salta** tratado aqui para dar suporte à instrução BLTZ? Modifique o VHDL abaixo para realizar isto.

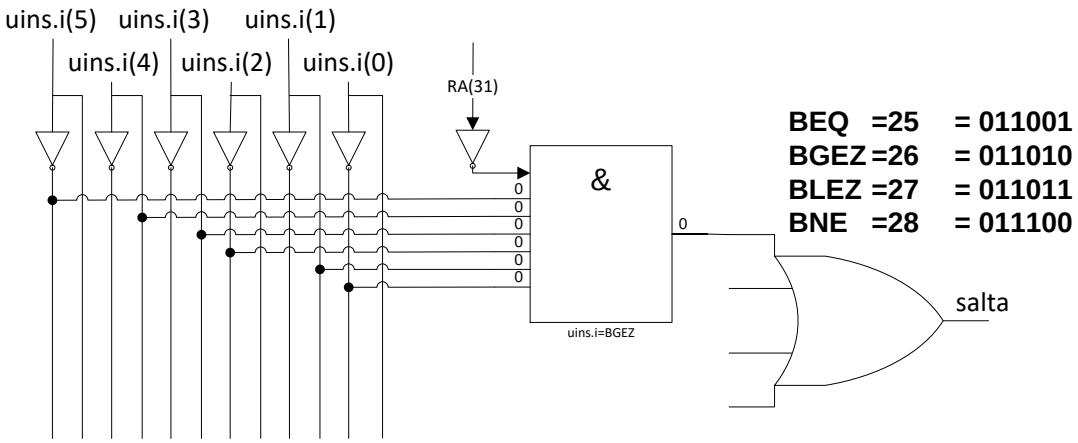
```
salta <=  '1' when ( (RA=RB  and uins.i=BEQ)  or (RA>=0  and uins.i=BGEZ) or
                     (RA<=0  and uins.i=BLEZ) or (RA/=RB and uins.i=BNE) )  else
          '0';
```

Solução:

3.1. Ver desenho abaixo. São duas entradas de 32 bits (RA e RB) e uma de no mínimo 6 bits (uins.i) A saída é o sinal salta de 1 bit.



3.2. Note-se que a saída salta pode ser gerada por uma porta lógica OU de 4 entradas (Já que cada comparação é exclusiva em relação às demais. Ou seja, quando uma delas gera resultado verdadeiro na comparação, todas as demais dão resultado falso). Para simplificar o problema escolhe-se aqui a comparação (RA>=0 and uins.i=BGEZ), pois o primeiro teste resume-se a verificar apenas o bit mais significativo de RA, RA(31) (Se este bit é 0, o número em RA certamente é positivo). Assume-se que uins.i será representado com o número mínimo de bits (6), e que os códigos binários de cada instrução serão atribuídos a partir de 000000 em ordem crescente, na ordem citada no desenho da MIPS multiciclo acima. Dados estes pressupostos, o código da instrução BGEZ será 26 ou em binário 011010. A partir desta proposta de codificação, fica fácil construir o circuito que gera 1 na sua saída quando a instrução em execução é BGEZ e RA é um número positivo. O circuito detecta esta situação com apenas uma porta AND de 7 entradas, onde algumas destas estão invertidas (precisamente RA(31), uins.i(5), uins.i(2) e uins.i(0)), conforme mostra o desenho.



3.3. A instrução BLTZ salta se um registrador passado como operando contiver um dado menor que zero. A modificação do VHDL para dar suporte a esta nova instrução é simples, basta adicionar um teste específico, como feito abaixo:

```
salta <=  '1' when ((RA=RB  and uins.i=BEQ)  or (RA>=0  and uins.i=BGEZ) or
                    (RA<=0  and uins.i=BLEZ) or (RA/=RB and uins.i=BNE)  or
                    (RA<0  and uins.i=BLTZ))  else
          '0';
```