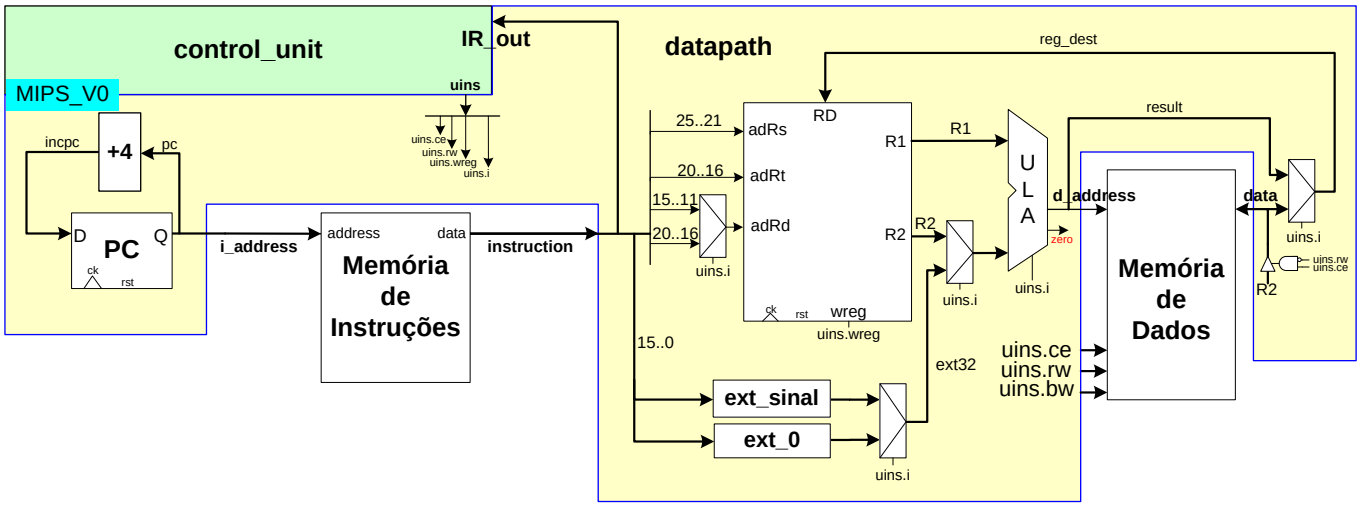
Prova P2 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 12/junho/2013

Para realizar a prova, refiram-se às 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**.



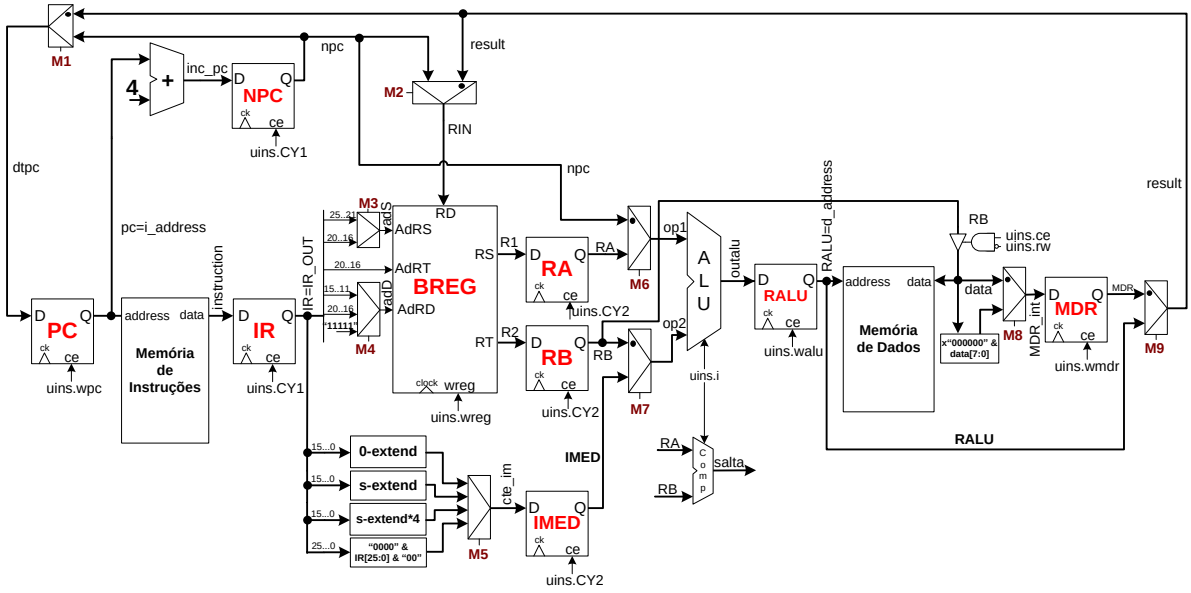
1. [3,0 pontos] Assuma que a organização MIPS monociclo estudada em aula na versão expandida, com suporte para as instruções **LUI** e **BEQ**, é usada aqui, mas que esta foi adicionalmente modificada para permitir executar também as instruções **SH** e **SB** (ignore a instrução **syscall**). O programa dado abaixo é um teste de execução, idealizado para comparar o efeito das instruções SW, SH e SB. As posições de memória **position1** e **position2** correspondem respectivamente às duas primeiras posições da área de dados mostrada (iniciando no endereço 0x10010000) em uma máquina com organização *little endian*. Diga exatamente quais endereços da memória de dados serão alterados na execução do programa e diga que valor é escrito em cada endereço específico (não se esqueça que no MIPS a memória de dados é organizada a byte, ou seja, em cada endereço de memória reside apenas 1 byte de informação, onde 1 byte=8 bits).

```
1.                      .text
2.          main:       li    $t0,0xbfae9d8c
3.                      li    $t1,0x7c6b5a49
4.                      li    $t2,0x38271605
5.                      la    $t3,position1
6.                      la    $t4,position2
7.                      sw    $t0,0($t3)
8.                      sb    $t1,0($t4)
9.                      sh    $t2,2($t4)
10.                     li    $v0,10
11.                     syscall
12.
13.                     .data
14.         position1:  .word 0
15.         position2:  .word 0
```

2. [4 pontos] Assuna uma frequência de operação de 400 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,lt
4.          lb      $a0,0($a0)
5.          jal     cl
6.          addu    $a0,$zero,$v0
7.          li      $v0,10
8.          syscall
9.  cl:     addu    $v0,$zero,$zero
10.         la      $s0,d
11. w:      lb      $t1,0($s0)
12.         beq     $t1,$zero,fc
13.         bne     $t1,$a0,na
14.         addiu   $v0,$v0,1
15. na:     addiu   $s0,$s0,1
16.         j       w
17. fc:     jr      $ra
18.         .data
19. d:      .asciiz     "Minha mãe mandou!!!"
20. lt:     .ascii      "m"
```

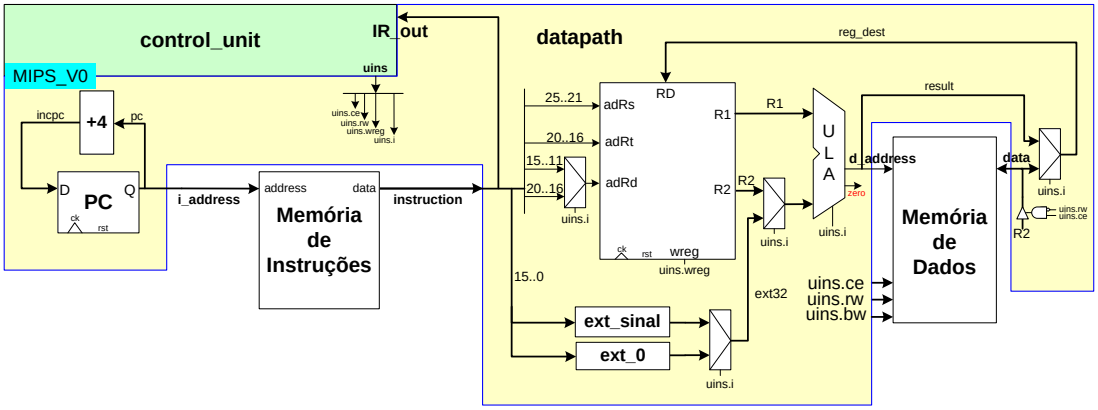
Bloco de Dados da organização MIPS Multiciclo. Instruções a que esta organização dá suporte: **ADDU, SUBU, AND, OR, XOR, NOR, SLL, SLLV, SRA, SRAV, SRL, SRLV, ADDIU, ANDI, ORI, XORI, LUI, LBU, LW, SB, SW, SLT, SLTU, SLTI, SLTIU, BEQ, BGEZ, BLEZ, BNE, J, JAL, JALR, JR**.



3. [3,0 pontos] Considere o bloco de dados multiciclo apresentado acima, e considere o módulo que gera a entrada do registrador denominado IMED na figura. Diga sobre este bloco:
   a) (1 ponto) Ele é similar a módulo em posição análoga no bloco de dados monociclo, mas possui quatro partes distintas (0-extend, s-extend, s-extend*4 e "0000"&IR[25:0]&"00") que geram valores possíveis de entrada para o registrador IMED. Explique para que serve cada uma das quatro partes, diferenciando-as.
   b) (2 pontos) Para cada uma das 33 instruções que a organização MIPS é capaz de executar, diga quais usam cada uma das quatro partes e quais (se existirem) não usam nenhuma das partes.

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. [3,0 pontos] Assuma que a organização MIPS monociclo estudada em aula na versão expandida, com suporte para as instruções **LUI** e **BEQ**, é usada aqui, mas que esta foi adicionalmente modificada para permitir executar também as instruções **SH** e **SB** (ignore a instrução **syscall**). O programa dado abaixo é um teste de execução, idealizado para comparar o efeito das instruções SW, SH e SB. As posições de memória **position1** e **position2** correspondem respectivamente às duas primeiras posições da área de dados mostrada (iniciando no endereço 0x10010000) em uma máquina com organização *little endian*. Diga exatamente quais endereços da memória de dados serão alterados na execução do programa e diga que valor é escrito em cada endereço específico (não se esqueça que no MIPS a memória de dados é organizada a byte, ou seja, em cada endereço de memória reside apenas 1 byte de informação, onde 1 byte=8 bits).

```
1.                      .text
2.        main:         li     $t0,0xbfae9d8c
3.                      li     $t1,0x7c6b5a49
4.                      li     $t2,0x38271605
5.                      la     $t3,position1
6.                      la     $t4,position2
7.                      sw     $t0,0($t3)
8.                      sb     $t1,0($t4)
9.                      sh     $t2,2($t4)
10.               li     $v0,10
11.               syscall
12.
13.               .data
14. position1:    .word  0
15. position2:    .word  0
```

Solução: Como existe no trecho de programa apenas uma instrução SW (Store Word), uma instrução SH (Store Half Word) e uma instrução SB (Store Byte), e não há laços no programa, serão escritos exatamente sete bytes na memória. A SW escreverá uma palavra (4 bytes) nos endereços 0x10010000 a 10010003, a SH escreverá nos endereços 10010006 a 1001007, e a SB escreverá somente um byte, no endereço 10010004, perfazendo um total de 7 bytes escritos na memória. Os conteúdos escritos serão (lembrando que se assume uma organização *little endian*:

10010000 – 0x8C (sw LSB)
10010001 – 0x9D (sw)
10010002 – 0xAE (sw)
10010003 – 0xBF (sw MSB)
10010004 – 0x49 (sb)
10010006 – 0x05 (sh LSB)
10010007 – 0x16 (sh MSB)

2. [4 pontos] Assuna uma frequência de operação de 400 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,lt          # 2 ciclos – Gera endereço do parâmetro (letra a procurar)
4.          lb      $a0,0($a0)      # 1 ciclo - Carrega parâmetro em $a0
5.          jal     cl              # 1 ciclo - Chama subrotina conta_letras
6.          addu    $a0,$zero,$v0   # 1 ciclo  - Na volta, move número de letras retornado para $a0
7.          li      $v0,10          # 1 ciclo  - Prepara saída do programa
8.          syscall                 # 1 ciclo  - Sai do programa

9.  cl:     addu    $v0,$zero,$zero # 1 ciclo - Início da subrotina, zera contador de letras
10.         la      $s0,d           # 2 ciclos - Gera endereço da cadeia de caracteres
11. w:      lb      $t1,0($s0)      # 1 ciclo - Busca próximo caracter
12.         beq     $t1,$zero,fc    # 1 ciclo – Testa fim da cadeia
13.         bne     $t1,$a0,na      # 1 ciclo – Se não chegou ao fim da cadeia, testa caracter
14.         addiu   $v0,$v0,1       # 1 ciclo - Se é caracter procurado, incrementa contador
15. na:     addiu   $s0,$s0,1       # 1 ciclo - Avança ponteiro para próximo caracter
16.         j       w               # 1 ciclo  - Volta a testar fim da cadeia
17. fc:     jr      $ra             # 1 ciclo  - Ao final do tratamento da cadeia, apenas retorna

18.         .data                   #
19. d:      .asciiz       "Minha mãe mandou!!!" # cadeia exemplo, possui dois m´s
20. lt:     .ascii        "m"       # Caracter a procurar
```

Solução:

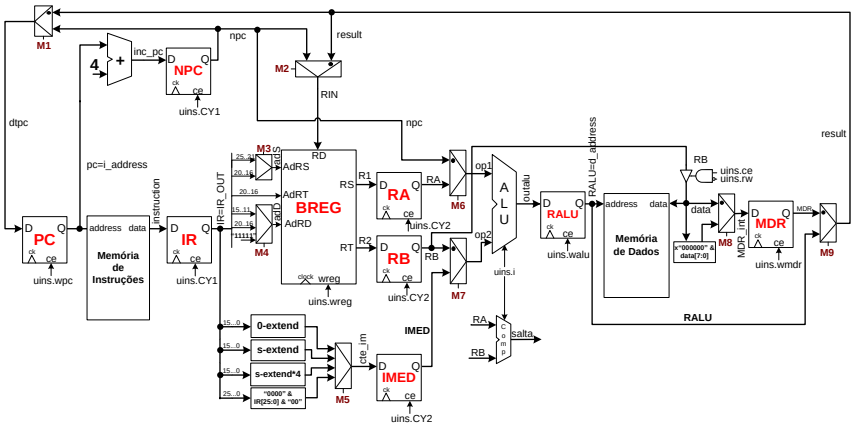
2.1. (1,5 pontos) As linhas 1-2 não possuem instruções. As linhas 3-8(7 ciclos), 9-10 (3 ciclos) e 17 (1 ciclo) são executadas exatamente uma vez cada, gastando um total de 11 ciclos de relógio. Existe um único laço, nas linhas 11-16, e cada execução deste pode gastar 5 ciclos (quando o caracter analisado é diferente do parâmetro passado), 6 ciclos (quando o caracter analisado é igual ao parâmetro passado), ou 2 ciclos (quando se chega ao caracter de fim de cadeia). O exemplo de cadeia possui exatamente 20 caracteres (incluindo o NULL ao final da cadeia), sendo que destes, apenas dois são iguais ao parâmetro passado em $a0 para a subrotina (a letra m). Assim, com os valores dados, o laço é executado exatamente 20 vezes, sendo que em 17 vezes gasta-se 5 ciclos para executá-lo, 2 vezes gasta-se 6 ciclos e uma vez se gasta 2 ciclos. Logo o número total de ciclos do programa com esta área de dados é 11+17*5+2*6+2=110 ciclos de relógio.

1.2. (1 ponto) Um relógio de 400MHz implica um período de (1/(400*10^6))s ou 2,5ns=2,5*10^{-9}s. Logo o tempo de execução do programa é 110*2,5*10^{-9}s= 275*10^{-9}s.

1.3. (1 ponto) Este programa computa o número de vezes que o caracter armazenado na posição de memória **lt** ocorre na cadeia de caracteres armazenada a partir da posição de memória **d**, retornando em $a0 o valor computado. O caracter é passado via registrador $a0 para uma subrotina e **d** é uma variável global, acessível à subrotina **cl**.

1.4. (0,5 pontos) O programa possui uma subrotina denominada **cl**, que ocupa as linhas 9-17, e que é chamada uma vez pelo programa principal.

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.**



3. [3,0 pontos] Considere o bloco de dados multiciclo apresentado acima, e considere o módulo que gera a entrada do registrador denominado IMED na figura. Diga sobre este bloco:
   a) (1 ponto) Ele é similar a módulo em posição análoga no bloco de dados monociclo, mas possui quatro partes distintas (0-extend, s-extend, s-extend*4 e "0000"&IR[25:0]&"00") que geram valores possíveis de entrada para o registrador IMED. Explique para que serve cada uma das quatro partes, diferenciando-as.
   b) (2 pontos) Para cada uma das 33 instruções que a organização MIPS é capaz de executar, diga quais usam cada uma das quatro partes e quais (se existirem) não usam nenhuma das partes.

Solução:

a) As quatro partes se chamam **0-extend**, **s-extend**, **s-extend*4** e **"0000"&IR[25:0]&"00"**. A função geral destas partes é gerar um valor imediato de 32 bits, a partir de uma das várias formas em que dados imediatos existem (quando existem) em uma instrução. As funções de cada um destes é:

- **0-extend** - Realiza a extensão de 0 dos bits 15-0 do IR, ou seja acrescenta 16 bits 0 à esquerda dos bits 15-0 do IR para produzir um valor de 32 bits a ser carregado em IMED. É usado sobretudo para instruções lógicas com dado imediato, tais como ORI, ANDI e XORI.

- **s-extend** - Realiza a extensão de sinal dos bits 15-0 do IR, ou seja acrescenta 16 bits à esquerda dos bits 15-0 do IR para produzir um valor de 32 bits a ser carregado em IMED Cada um dos 16 bits acrescentados é uma cópia do bit 15 do IR. É usado nas instruções aritméticas com dado imediato, tais como ADDI e ADDIU, de comparação com dado imediato como SLTIU e para cálculo do deslocamento em instruções que usam o modo de endereçamento base-deslocamento, tal como as instruções de leitura e escrita na memória de dados (LW, SW, LB, LBU, SB)

- **s-extend*4** - Similar a **s-extend**, mas depois de realizar a extensão de sinal, acrescenta dois bits em 0 à direita dos 32 bits gerados e descarta os dois bits mais significativos, produzindo um valor em 32 bits que é o resultado de multiplicar o resultado da extensão de sinal pela constante 4. É usado nas instruções de desvio condicional tal como BEQ e BNE) para computar o valor a ser somado ao PC para obter o endereço para onde saltar quando a condição for válida.

- **"0000"&IR[25:0]&"00"** - O próprio nome desta parte explica o que ela faz. Gera o valor de 32 bits a partir da concatenação dos bits 25-0 do IR multiplicado por 4 (ou seja, com dois bits adicionais colocados à direita dos bits menos significativos) com 4 bits em 0. Esta parte é usada para gerar parte do endereço de salto para as instruções tipo J (J e JAL). O endereço em si é obtido na ULA, substituindo os 4 bits mais significativos de IMED pelos 4 bits mais significativos do PC.

b) Pode-se classificar as 33 instruções da MIPS multiciclo de acordo com qual parte do hardware de geração da entrada do IMED usam:

- Não usam IMED - Qualquer instrução que não tenha nenhum dado imediato no seu código objeto. Isto inclui todas as instruções tipo R, exceto as de deslocamento, onde o campo **shamt** é um dado imediato. As instruções desta classe são então: **ADDU, SUBU, AND, OR, XOR, NOR, SLLV, SRAV, SRLV, SLT, SLTU, JALR e JR**.

- Usam **0-extend** - Algumas instruções **tem** de usar esta parte, as instruções lógicas com dado imediato: **ANDI, ORI, XORI.** Algumas instruções podem usar esta parte ou a parte **s-extend** de forma indiferente, pois nelas os bits 31-16 do imed não são usados, mas os bits 15-0 sim, mesmo que de forma parcial: **SLL, SRA, SRL e LUI**;
- Usam **s-extend** - As instruções que precisam usar extensão de sinal são as aritméticas com dado imediato, as de comparação com dado imediato e as de escrita e leitura na/da memória de dados, ou seja: **ADDIU, SLTI, SLTIU, LBU, LW, SB, SW.** Algumas instruções podem usar esta parte ou a parte **0-extend** de forma indiferente, pois nelas os bits 31-16 do IMED não são usados, mas os bits 15-0 sim, mesmo que de forma parcial: **SLL, SRA, SRL e LUI**;
- Usam **s-extend*4** - Somente usam esta parte as instruções de salto condicional, ou seja: **BEQ, BGEZ, BLEZ, BNE**;
- Usam **"0000"&IR[25:0]&"00"** - Esta classe incui apenas as instruções tipo J que especificam nos bits 25-0 um endereço pseudo-absoluto, ou seja **J** e **JAL**.