Prova de Arquitetura de Computador

Aluno:

1a Questão) O fragmento de código abaixo processa um array de palavras e retorna 2 valores importantes em $v0 e $v1. Assuma que o array tem armazenado 5.000 palavras indexadas de 0 a 4.999, sabendo que o endereço base do array está armazenado em $a0 e o tamanho do array (5.000) em $a1. Descreva em uma frase o que faz o programa e o que é retornado em $v0 e $v1.

```
[1]                 addu    $a1, $a1, $a1
[2]                 addu    $a1, $a1, $a1
[3]                 addu    $v0, $zero, $zero
[4]                 addu    $t0, $zero, $zero
[5]     outer:      addu    $t4, $a0, $t0
[6]                 lw      $t4, 0($t4)
[7]                 addu    $t5, $zero, $zero
[8]                 addu    $t1, $zero, $zero
[9]     inner:      addu    $t3, $a0, $t1
[10]                lw      $t3, 0($t3)
[11]                bne     $t3, $t4, skip
[12]                addiu   $t5, $t5, 1
[13]    skip:       addiu   $t1, $t1, 4
[14]                bne     $t1, $a1, inner
[15]                slt     $t2, $t5, $v0
[16]                bne     $t2, $zero, next
[17]                addu    $v0, $t5, $zero
[18]                addu    $v1, $t4, $zero
[19]    next:       addiu   $t0, $t0, 4
[20]                bne     $t0, $a1, outer
```

Resposta: O código determina qual palavra aparece no array com maior freqüência (1,0 ponto), retornando a palavra em $v1 (0,5 ponto) e o número de vezes que ela aparece em $v0 (0,5 ponto).

Descreve o código mas não fala o que ele faz (0.5 ponto)

2a Questão) Considerando o modo de endereçamento relativo ao PC no MIPS, considere o trecho de código abaixo:

1. here: beq $t1,$t2,there

2. ... muitas linhas de código

3. there: add $t1,$t1,$t1

Pede-se sobre este código:

a) (1,0 ponto) Em que condições um montador poderia ter problemas para montar este• código corretamente?

b) (1,0 ponto) Se "muitas linhas de código" são muitas mesmo (mais do que quantas• linhas? Calcule. Linha não dá problema, 2 linhas também não, mas e 1000? E 1 bilhão?

c) (1,0 ponto) Como o montador poderia resolver o problema de gerar código correto para o trecho?

Solução

a) Poderá ter dificuldade em montar o código quando tiver muitas linhas códigos

b) A instrução beq ocupa 32 bits, e trabalha com modo de endereçamento relativo ao PC para definir a posição para onde saltar. O formato da instrução beq é:

	beq	**rs**	**rt**	**label**
	4	**rs**	**rt**	**offset**
Número de bits/campo:	**6**	**5**	**5**	**16**

Ou seja, a partir do rótulo (label) determina-se o deslocamento (offset) necessário para atingir o rótulo a partir do valor do registrador PC durante a execução do beq. Este deslocamento é expresso em instruções à frente ou para trás da instrução seguinte ao beq (pois o PC aponta para esta ao se executar o beq). Como offset é um campo de 16 bits, e como valores positivos saltam para linhas à frente do beq (incluindo offset=0), o que é o caso do rótulo there, nota-se que a distância máxima (número máximo de instruções entre here e there) é especificada quando o offset for o maior número número positivo que se pode representar em 16 bits, ou seja, 0111 1111 1111 1111 em binário, equivalente a 0x7FFF em hexa, e equivalente a 32767 em decimal. Ou seja, se houver mais de 32767 instruções depois do beq para chegar a there, um único beq não pode ser usado para traduzir a funcionalidade da linha 1.

c) Uma forma que o montador teria para resolver o problema seria partir do endereço correspondente ao rótulo there, colocar este valor no registrador $at (por convenção reservado para uso do montador), e executar uma instrução jr para este endereço se a condição de teste for verdadeira. Assim, supondo que there esteja a mais de 32767 instruções de distância de here, o código abaixo poderia ser gerado pelo montador para substituir a linha 1:

```
la $at,  there          # note a pseudo la, que deve ser transformada
                        # na sequência de instruções lui-ori
bne $t1, $t2,    np     # np (não pula) é rótulo criado pelo montador
jr $at                  # se beq deve saltar, é aqui que se salta
np: ....                # lugar onde vai a primeira instrução depois do beq
```

3a Questão) Escreva um programa em linguagem de montagem (assembly language) do processador MIPS que processa um vetor de números inteiros (VET), contando quantos elementos pares e ímpares contém o vetor. Os resultados devem ser armazenados nas variáveis PAR e IMPAR em memória. Utilize a área de dados abaixo para escrever o seu programa. Implemente sua funcionalidade com instruções reais da arquitetura

Solução:

```
                .text
                .globl      main
main:           la          $t0,VET             # Gera ponteiro para vetor em $t0
                la          $t1,TAM
                lw          $t1,0($t1)          # Inicializa $t1 com o tamanho do vetor
                add         $t2,$zero,$zero     # $t2=contador de pares, no início 0
                add         $t3,$zero,$zero     # $t3=contador de ímpares, no início 0
loop:           beq         $t1,$zer0,end        # Qdo cont. de elementos chega a 0, fim
                lw          $t4,0($t0)          # Lê elemento do vetor
                andi        $t4,$t4,1           # Este AND escreve 0 em $t4 se número
                                                # é par, senão e escreve 1 em $t4
                beq         $t4,$zero, ehpar    # Salta se for par
                addi        $t3,$t3,1           # Senão, incrementa contador de ímpares
                j           loopend             # E vai finalizar laço
ehpar:                      addi $t2,$t2,1      # Incrementa contador de pares
loopend:                    addi $t0,$t0,4      # Avança ponteiro para próximo elemento
                            addi $t1,$t1,-1     # Decr contador de elementos a tratar
                j           loop                # Executa novo teste-iteração
end:            la          $t0,PAR             # Obtém ponteiro para PAR
                sw          $t2,0($t0)          # Escr. no. de pares na variável PAR
                la          $t0,IMPAR           # Obtém ponteiro para IMPAR
                sw           $t3,0($t0)         # Escr. no. de ímpares na variável IMPAR
                li $v0,10
                syscall                         # E cai fora do programa
.data
VET: .word 23 -43 55 -9 -7 21 -76 12 -45 -10
TAM: .word 10
PAR: .word 0
IMPAR: .word 0
```

3a Questão) Escreva um programa em linguagem de montagem (assembly language) do processador MIPS que processa um vetor de números inteiros (VET), transformando cada elemento do vetor em seu valor absoluto. Por exemplo, o valor absoluto do número -43 é +43 e do número +55 é +55. O programa deve obrigatoriamente chamar uma sub-rotina calc_abs que recebe como parâmetro um número e calcula o valor absoluto deste, retornando-o segundo as convenções do MIPS. O valor retornado pela sub-rotina deve ser armazenado, pelo programa principal, na mesma posição do array que continha o elemento original. Utilize a área de dados abaixo para escrever o seu programa. Dica: Como a subrotina calc_abs é claramente folha (não chama nenhuma outra), e como não se especifica como deve ser feita a passagem de parâmetros, não é necessário usar a pilha nesta questão.

.data

VET: .word 23 -43 55 -9 -7 21 -76 12 -45 -10

TAM: .word 10

ideia correta

Resposta:

Programa Principal main:

```
main:
                la      $s0,  VET
                la      $s1,  TAM
                lw      $s1,  0($s1)
loop:
                beq     $s1,  $zero,  fim
                Lw      $a0,  0($s0)
                subi    $sp, $sp, 8          Não é necessário
                Sw      $ra, 4($sp)          Não é necessário
                Sw      $fp, 0($sp)          Não é necessário
                move    $fp, $sp             Não é necessário
                Jal     calc_abs
                lw      $ra, 4($sp)          Não é necessário
                lw      $fp, 0($sp)          Não é necessário
                addi    $sp, $sp, 8          Não é necessário
                Sw      $v0,  0($s0)
                addiu   $s1,   $s1,   -1
                addiu   $s0,   $s0,   4
                j       Loop
fim:
                Li      $v0,10
                Syscall

## Subrotina de cálculo do valor absoluto
calc_abs:
                move    $v0,$a0
                beq     $v0,$zero, fim_ca
                subu    $v0,$zero,$v0
fim_ca:
                Jr      $ra

.data
VET: .word 23 -43 55 -9 -7 21 -76 12 -45 -10
TAM: .word 10
```

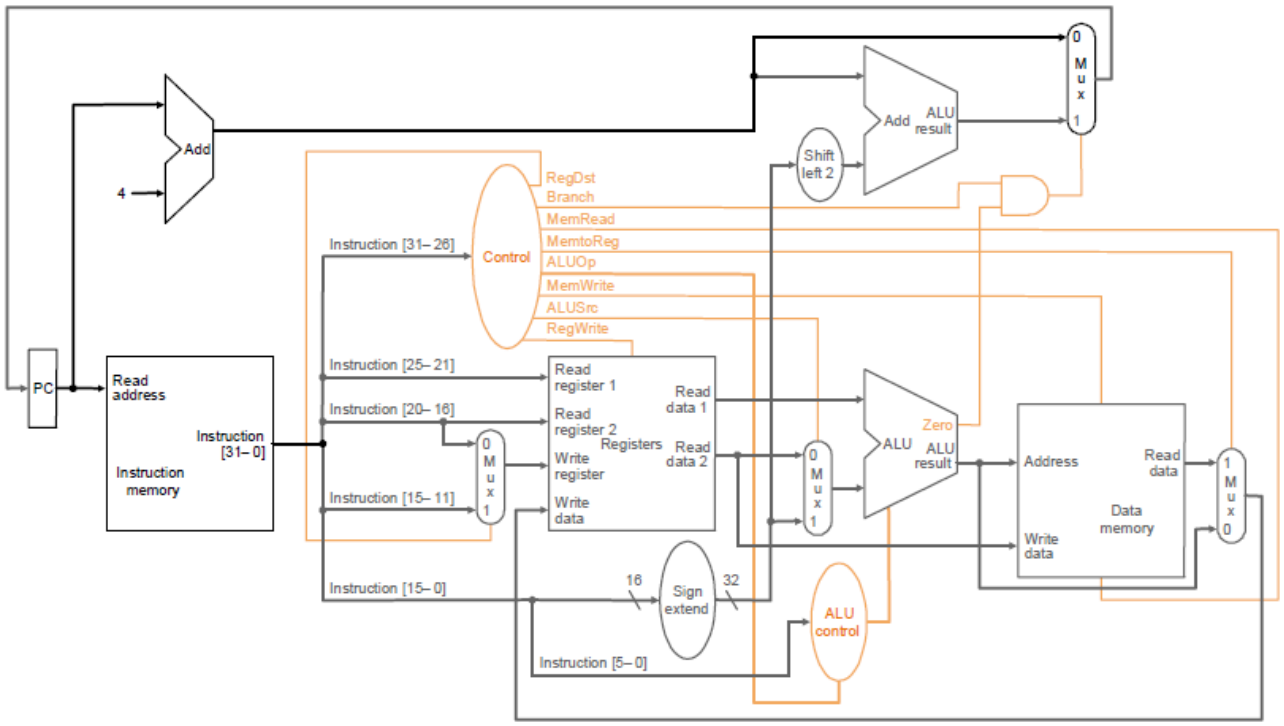
4a Questão) Considere a seguinte modificação na arquitetura do conjunto de instruções do processador MIPS: suponha que as instruções de acesso à memória de dados a palavra (lw/sw) não mais usam endereçamento base-deslocamento para acesso ao dado que deverá ser escrito no registrador destino, mas apenas endereçamento direto a registrador. As antigas instruções lw e sw passam a ser pseudo-instruções e deverão ser implementadas através de duas instruções. Por exemplo, uma pseudo-instrução lw $t0,100($t1) seria implementada pela seqüência de instruções:

addi $at,$t1,100 # Adiciona o deslocamento ao registrador base, gerando o endereço de acesso

lw $t0,$at # Usa a nova forma da instrução lw, ao invés do tradicional. Lê conteúdo de memória do

endereço contido em $at e o copia para $t0 lw $t0, 100($t1)

Quais modificações deveriam ser realizadas no Bloco de Dados e no Bloco de Controle para dar suporte a esta nova instrução? Considere a implementação monociclo do MIPS abaixo.

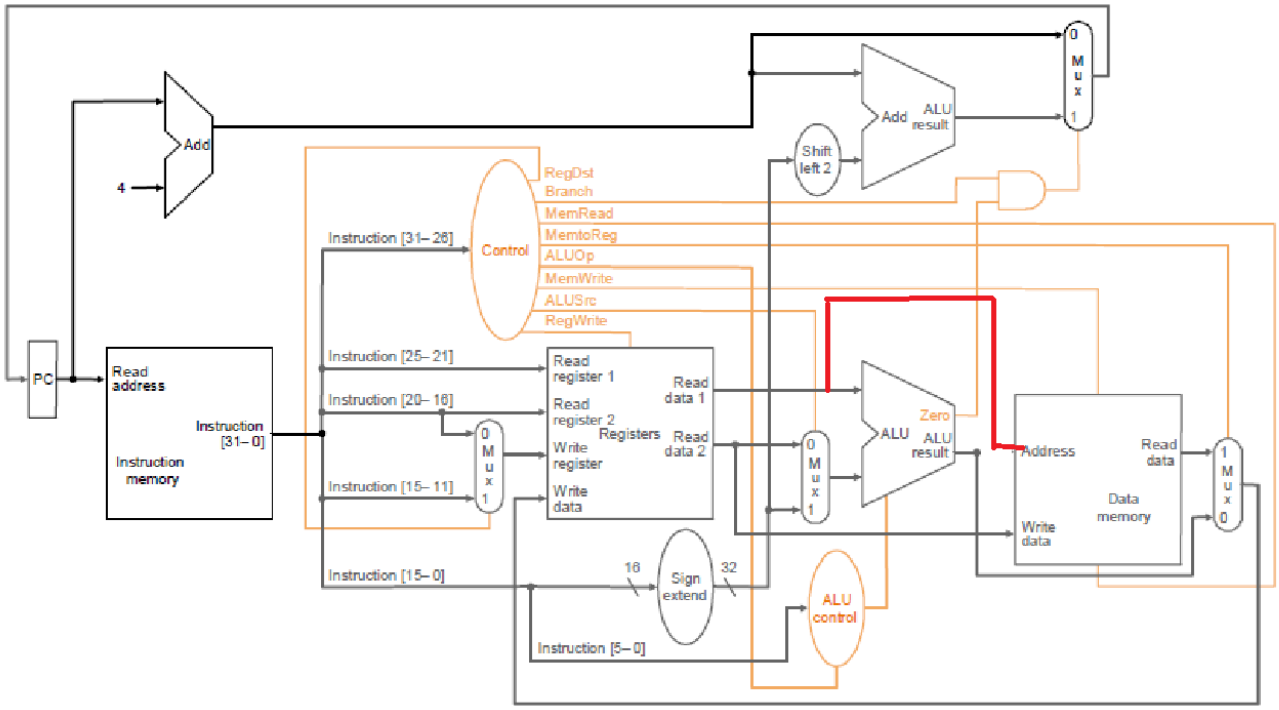


A principal modificação consiste em perceber que não será mais necessário passar através da ULA (cálculo do endereço) para só então acessar a memória de dados (1,0 ponto, se mencionar que precisa multiplexador descontar 0.2). Seria necessário acrescentar uma ligação direta entre a saída Read data 1 do banco de registradores e o barramento de endereços da memória (assumindo que o formato da instrução não será modificado) (0,5 ponto). A saída da ULA não precisa mais ser conectada ao barramento de endereços da memória de dados. O bloco de controle não precisa ser modificado (0,3 ponto), porém alguns sinais de controle podem ser don't cares para as instruções de acesso à memória, tais como ALUSrc, ALUOp0 e ALUOp1 (0.20 ponto).

Se mencionar que irar utilizar a ULA com sinal zero, considera apenas 1.0 (se não mencionar como controlar descontar 0,2)

Verificar sempre se menciona ALUSrc, ALUOp0 e ALUOp1, caso contrario -0,2

trocar Read Data 2 por Read Data 1 - 0,0 (Read Data 2 corresponde a entrada de dados para memoria e não pode ser alterada)


Add
4
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [31–26]
Control
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign extend
32
Shift left 2
Add
ALU result
ALU
Zero
ALU result
ALU control
Address
Read data
Data memory
Write data
M u x
0
1

Prova de Arquitetura de Computador (Prova B)

Aluno:

1a Questão) Analise o código do programa abaixo e explique o que este programa faz. Inicie mapeando que registradores são utilizados, dizendo como cada um é inicializado e conclua descrevendo a função destes no código do programa. Comente semanticamente o código. Com os dados fornecidos, que valor é escrito na memória de dados ao final da execução do programa?

```
[1] # Programa que calcula a soma dos termos de uma Progressão Aritmética (PA)
[2] #
[3] #
[4]
[5] .text
[6] .globl main
[7] main:
[8]                 xor     $s0,$zero,$zero     # $s0=reg que guarda a soma dos termos da PA
[9]                 la      $s1,SOMA            # Carrega em $s1 o endereço de SOMA
[10]                la      $t0,N               # Carrega em $t0 o endereço de N
[11]                lw      $t0,0($t0)          # t0=N, inicialmente o núm de termos a somar da PA
[12]                la      $t1,TI              # Carrega em $t1 o endereço de TI
[13]                lw      $t1,0($t1)          # $t1=TI, o Termo Inicial da PA
[14]                la      $t2,R               # Carrega em $t2 o endereço de R
[15]                lw      $t2,0($t2)          # $t2=R, a razão da PA
[16] prox_T:        blez    $t0,fim             # Se contador zerado, fim
[17]                addu    $s0,$s0,$t1         # Senão, adiciona termo à soma
[18]                addu    $t1,$t1,$t2         # Gera o próximo termo, somando R ao termo anterior
[19]                addiu   $t0,$t0,-1          # Decrementa o contador
[20]                j       prox_T              # Continue a executar o laço
[21] fim:           sw      $s0,0($s1)          # Estoca resultado da soma dos termos da PA em SOMA
[22] end:           li      $v0,10
[23]                syscall                     # E cai fora do programa
[24]
[25] .data                                      # segmento de dados
[26] TI:            .word   12                  # Termo Inicial da PA
[27] R:             .word   3                   # Razão da PA

[28] N:             .word   6                   # Número de termos a somar
```

Com os dados fornecidos o resultado escrito na variável soma é 12+15+18+21+24+27=117.

3a Questão) Escreva um programa em linguagem de montagem (assembly language) do processador MIPS que processa um vetor de números inteiros (VET), contando quantos elementos pares e ímpares contém o vetor. Os resultados devem ser armazenados nas

variáveis PAR e IMPAR em memória. Utilize a área de dados abaixo para escrever o seu programa. Implemente sua funcionalidade com instruções reais da arquitetura.

Solução:

```
        .text
            .globl      main
main:       la          $t0,VET         # Gera ponteiro para vetor em $t0
            la          $t1,TAM
            lw          $t1,0($t1)      # Inicializa $t1 com o tamanho do vetor
            add         $t2,$zero,$zero  # $t2=contador de pares, no início 0
            add         $t3,$zero,$zero # $t3=contador de ímpares, no início 0
            loop:       beq $t1,$zer0,end # Qdo cont. de elementos chega a 0, fim
            lw          $t4,0($t0)      # Lê elemento do vetor
            andi        $t4,$t4,1       # Este AND escreve 0 em $t4 se número
                                        # é par, senão e escreve 1 em $t4
            beq         $t4,$zero, ehpar # Salta se for par
            addi        $t3,$t3,1       # Senão, incrementa contador de ímpares
            j           loopend         # E vai finalizar laço
ehpar:      addi        $t2,$t2,1       # Incrementa contador de pares
loopend:    addi        $t0,$t0,4       # Avança ponteiro para próximo elemento
            addi        $t1,$t1,-1      # Decr contador de elementos a tratar
            j           loop            # Executa novo teste-iteração
end:        la          $t0,PAR         # Obtém ponteiro para PAR
            sw          $t2,0($t0)      # Escr. no. de pares na variável PAR
            la          $t0,IMPAR       # Obtém ponteiro para IMPAR
            sw $t3,0($t0)               # Escr. no. de ímpares na variável IMPAR
            li $v0,10
            syscall                     # E cai fora do programa
.data
VET: .word 23 -43 55 -9 -7 21 -76 12 -45 -10
TAM: .word 10
PAR: .word 0
IMPAR: .word
```

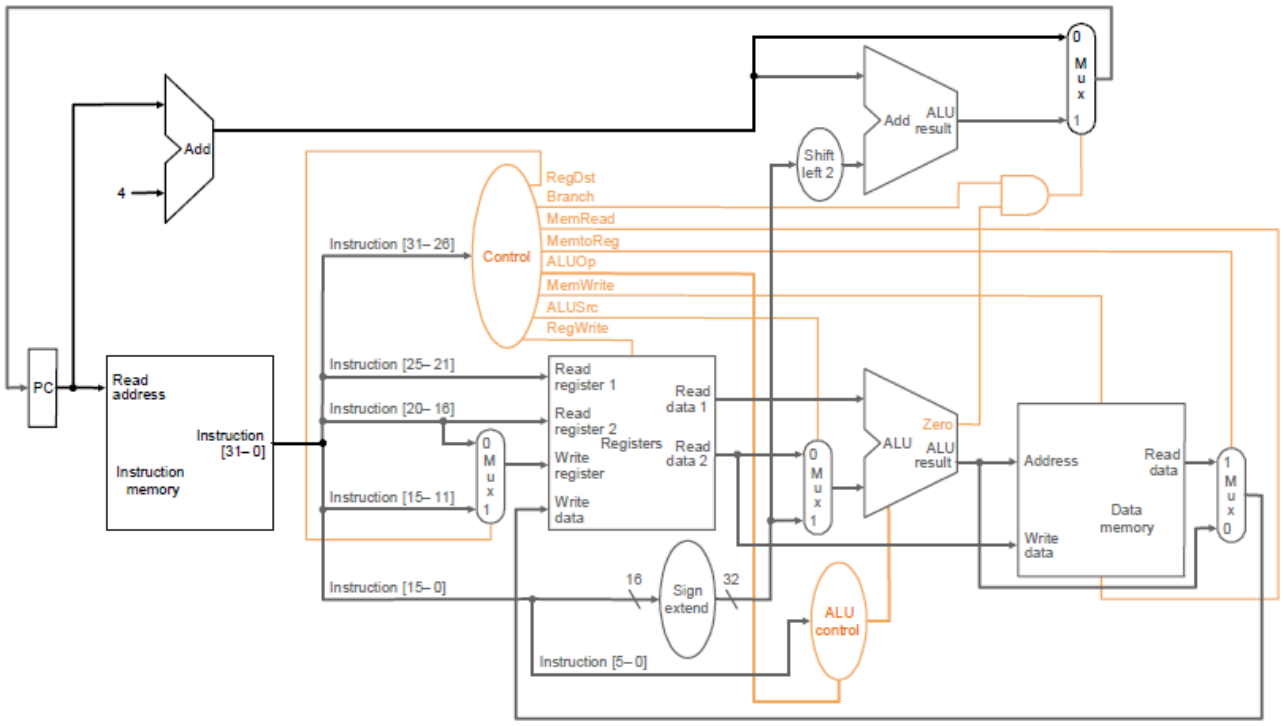
4a Questão) Considere a seguinte modificação na arquitetura do conjunto de instruções do processador MIPS: suponha que as instruções de acesso à memória de dados a palavra (lw/sw) não mais usam endereçamento base-deslocamento para acesso ao dado que deverá ser escrito no registrador destino, mas apenas endereçamento direto a registrador. As antigas instruções lw e sw passam a ser pseudo-instruções e deverão ser implementadas através de duas instruções. Por exemplo, uma pseudo-instrução lw $t0,100($t1) seria implementada pela seqüência de instruções:

```
addi $at,$t1,100        # Adiciona o deslocamento ao registrador base, gerando o endereço de acesso

lw $t0,$at              # Usa a nova forma da instrução lw, ao invés do tradicional. Lê conteúdo de memória do

                        # endereço contido em $at e o copia para $t0 lw $t0, 100($t1)
```

Quais modificações deveriam ser realizadas no Bloco de Dados e no Bloco de Controle para dar suporte a esta nova instrução? Considere a implementação monociclo do MIPS abaixo.

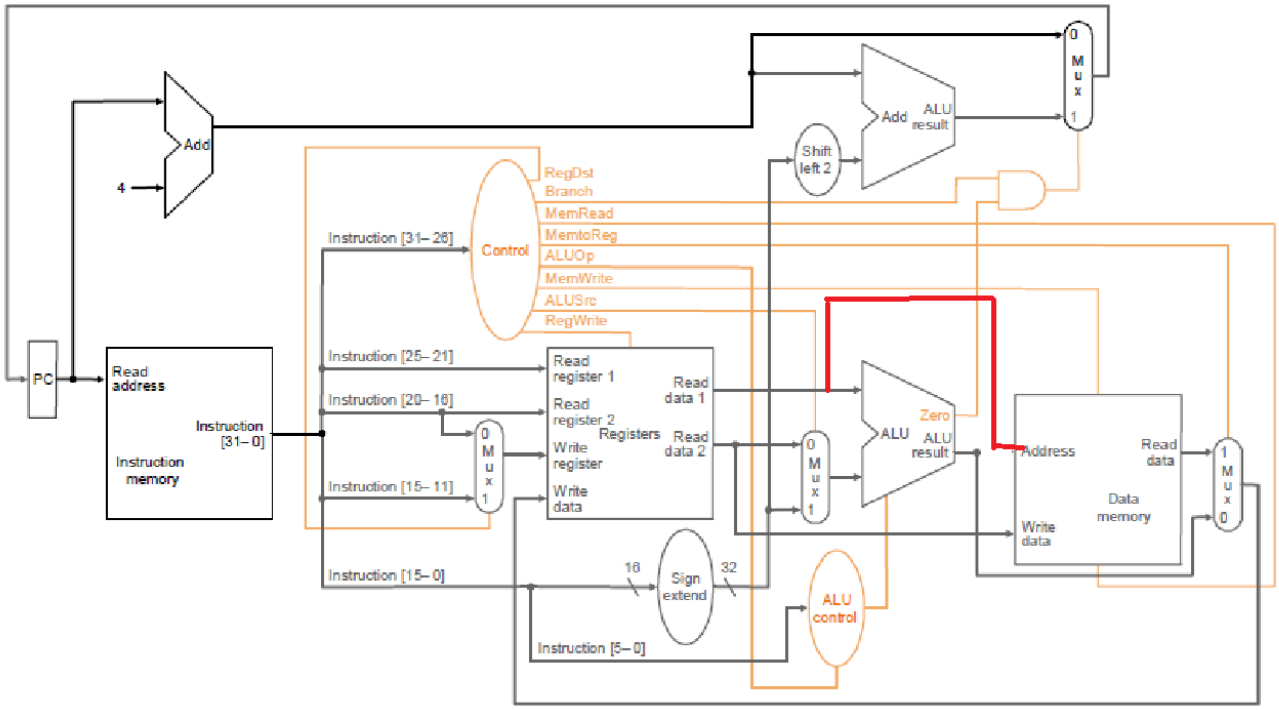


A principal modificação consiste em perceber que não será mais necessário passar através da ULA (cálculo do endereço) para só então acessar a memória de dados (1,0 ponto, se mencionar que precisa multiplexador descontar 0.2). Seria necessário acrescentar uma ligação direta entre a saída Read data 1 do banco de registradores e o barramento de endereços da memória (assumindo que o formato da instrução não será modificado) (0,5 ponto). A saída da ULA não precisa mais ser conectada ao barramento de endereços da memória de dados. O bloco de controle não precisa ser modificado (0,3 ponto), porém alguns sinais de controle podem ser don't cares para as instruções de acesso à memória, tais como ALUSrc, ALUOp0 e ALUOp1 (0.20 ponto).

Se mencionar que irar utilizar a ULA com sinal zero, considera apenas 1.0 (se não mencionar como controlar descontar 0,2)

Verificar sempre se menciona ALUSrc, ALUOp0 e ALUOp1, caso contrario -0,2

trocar Read Data 2 por Read Data 1 - 0,0 (Read Data 2 corresponde a entrada de dados para memoria e não pode ser alterada)


Add
4
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [31–26]
Control
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign extend
32
Shift left 2
Add
ALU result
ALU
Zero
ALU result
ALU control
Address
Read data
Data memory
Write data
M u x
0
1