UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia

Departamento de Computação

Arquitetura e Organização de Computadores 1

Prática 8 Processador MIPS Monociclo

Professores: Luciano de Oliveira Neris e Mauricio Fernandes Figueiredo

Autores (Grupo R)

Guilherme Campos Marques, 727338 e Engenharia da Computação

Leticia Bossatto Marchezi, 791003 e Ciência da Computação

Marcos Cardoso Vendrame, 790725 e Ciência da Computação

Mateus Grota Nishimura Ferro, 771043 e Ciência da Computação

São Carlos, 19 de Junho de 2021

1. Seção A (ori)

Para a implementação da instrução ori (or immediate) é necessário adicionar um bit a mais em **controls(mips.sv)** pois alusrc passa a ter 2 bits Além disso, os sinais de controle de ori foram inseridos no case para op=001101 seguindo a ordem da **tabela 1**.

regwrite	regdst	alusrc	branch	memwrite	memtoreg	jump	aluop
1	0	10	0	0	0	0	11

Tabela 1: Sinais de controle para operação ORI.

```
    begin
      reset <= 1; # 22; reset <= 0;
    end

  // generate clock to sequence tests
  always
    begin
      clk <= 1; # 5; clk <= 0; # 5;
    end

  // check that 7 gets written to address 64
  always@(negedge clk)
    begin
      if(memwrite) begin
        if(dataadr === 64 & writedata === 7) begin
          $display("Simulation succeeded");
          $stop;
        end else if (dataadr !== 64) begin
          $display("Simulation failed");
          $stop;
        end
      end
    end
endmodule
```

```
Log  Share
# KERNEL: SLP loading done - time: 0.0 [s].
# KERNEL: Warning: You are using the Riviera-PRO EDU Edition. The performance
# KERNEL: Warning: Contact Aldec for available upgrade options - sales@aldec.c
# KERNEL: SLP simulation initialization done - time: 0.0 [s].
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 4695 kB (elbread=427 elab2=4133 kernel=134 s
# KERNEL: ASDB file was created in location /home/runner/dataset.asdb
# KERNEL: Simulation succeeded
```

Imagem 1: Resultado do testbench

Código em Assembly de uma implementação de ORI:

addi $2, $0, 0x7 # initialize $v0 = 8

ori $3, $2, 0x4 # 111 | 100 = 111

sw $3, 64($0)   # write mem[64] = 111 = 7

```
1  addi $2, $0, 0x7 # initialize $v0 = 8
2  ori $3, $2, 0x4  # 111 | 100 = 111
3  sw $3, 64($0)    # write mem[64] = 111 = 7
```

Imagem 2: código em Assembly

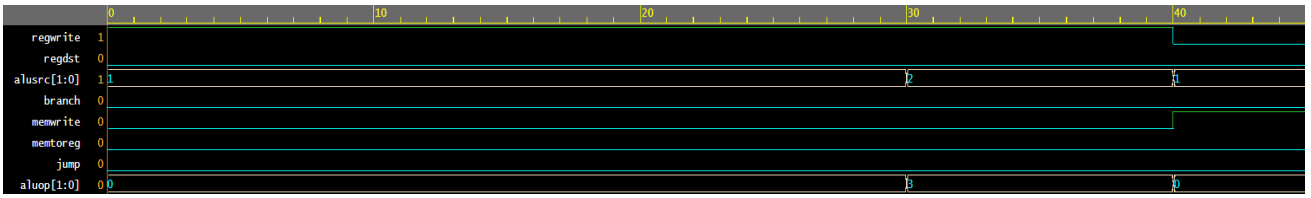

Imagem 3: Diagrama de ondas da simulação, ori realizado em t=30ns

Link para a implementação: https://edaplayground.com/x/hnDv

Link para diffchecker: https://www.diffchecker.com/10o6N7Rx

2. Seção B (bne)

Para a implementação da instrução bne(branch if not equal) é necessário adicionar um bit a mais em **controls(mips.sv)** para armazenar o novo sinal de controle que indica a execução da instrução bne. Além disso, os sinais de controle de bne foram inseridos no case para op=000101 seguindo a ordem da **tabela 2**. Para realizar o desvio, o módulo **controller** foi modificado acrescentando um teste lógico *or* entre a operação lógica anterior e a expressão (bne & !zero), que executa o desvio quando bne é verdadeiro e o conteúdo dos dois registradores forem diferentes, pois zero é falso.

regwrite	regdst	alusrc	branch	memwrite	memtoreg	jump	bne	aluop
0	0	0	0	0	0	0	1	01

Tabela 2: Sinais de Controle para a operação BNE.

Já na simulação referente a instrução bne foram usadas instruções para adicionar os valores 8 e 16, respectivamente, nos registradores $v0 e $v1. Logo após isso, a instrução bne deve realizar um desvio caso o $v0 seja diferente de $v1, o que é verdade e causa o salto do programa para realizar uma subtração e posteriormente armazenar seu valor no endereço de memória 40.

```
    begin
      reset <= 1; # 22; reset <= 0;
    end

  // generate clock to sequence tests
  always
    begin
      clk <= 1; # 5; clk <= 0; # 5;
    end

  // check that 8 gets written to address 40
  always@(negedge clk)
    begin
      if(memwrite) begin
        if(dataadr === 40 & writedata === 8) begin
          $display("Simulation succeeded");
          $stop;
        end else if (dataadr !== 40) begin
          $display("Simulation failed");
          $stop;
        end
      end
    end
endmodule
```

Log Share

```
# KERNEL: SLP simulation initialization done - time: 0.0 [s].
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 4694 kB (elbread=427 elab2=4132 kernel=134 sdf=0)
# KERNEL: ASDB file was created in location /home/runner/dataset.asdb
# KERNEL: Simulation succeeded
```

Imagem 4: Resultado do testbench

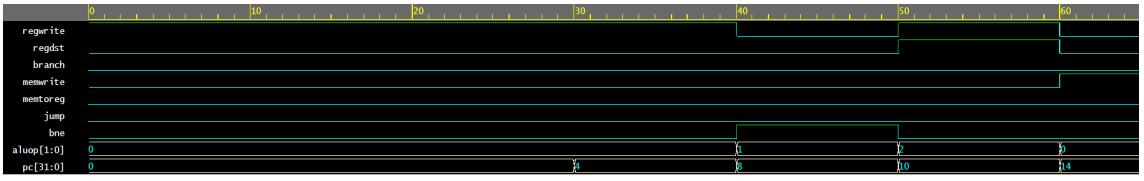

Imagem 5: Diagrama de ondas da simulação de bne, desvio realizado de pc=8 para pc=10

Código em Assembly de uma implementação de BNE:

addi $2, $0, 0x8 # initialize $v0 = 8

addi $3, $0, 0x10 # initialize $v1 = 16

bne $2, $3, L1 # branch if $v0 != $v1 to L1

add $4, $3, $2 # add $a1 = $v1 + $v1 = 24

L1:sub $4, $3, $2 # branch, sub $a1 = $v1 - $v0 = 8

sw $4, 32($4) # write mem[40] = 8

```
1  addi $2, $0, 0x8  # initialize $v0 = 8
2  addi $3, $0, 0x10 # initialize $v1 = 16
3  bne $2, $3, L1    # branch if $v0 !=  $v1 to L1
4  add $4, $3, $2    # add $a1 = $v1 + $v1 = 24
5  L1:sub $4, $3, $2 # branch, sub $a1 = $v1 - $v0 = 8
6  sw $4, 32($4)     # write mem[40] = 8
```

Imagem 6: código em Assembly

Link para a implementação: https://edaplayground.com/x/uand

Link para o diffchecker: https://www.diffchecker.com/8w8zawJV