Trabalho Final de Organização e Arquitetura de Computadores

Bruno Cordeiro Mendes (15/0007094), Bruno Justino Garcia Praciano (13/0006912) e Thales Gonçalves Grilo (14/0163603)

13 de dezembro de 2017

1 Objetivo

O projeto da disciplina consiste no desenvolvimento de uma versão do processador MIPS Pipeline em FPGA, utilizando a linguagem VHDL.

2 Implementação

2.1 Mux de 4 entradas

Um multiplexador de 4 entradas que será utilizado nas etapas Instruction Fetch (para selecionar o valor que será armazenado em PC), Execute (para selecionar o qual registrador será escrito), Write Back (seleciona o dado que escreverá no registrador destino). Esse mux também é utilizado para os seletores da FPGA, seletores 00 é o PC, 01 a instrução, 10 a saída de memória de dados e 11 é o valor da saida da ULA.

2.2 Mux de 2 entradas

Um multiplexador de 2 entradas que será utilizado em dois lugares na etapa de Execução, um mux servirá para escolher o dado que entrará na entrada A da ULA e o outro servirá para escolher o dado que entrará na entrada B da ULA, no caso se for um valor imediato o sinal AluSRC será ativado.

2.3 ULA

Esse módulo representará a Unidade Lógica Aritmética (ULA) de nosso processador. Nele serão feitas operações como AND, OR, ADD, SUB, SLT, NOR, XOR, LUI e ADDI. Além dos resultados de uma dessa operações, esse módulo fornecerá sinais zero e ovfl, que indicam que a operação gerou um resultado que tenha um bit a mais do que seus operandos, se o resultado é igual a 0 e se houve overflow na operação realizada, respectivamente.

2.4 Breg

Esse módulo representará o banco de registradores que contém 32 registradores. As operações com esse banco são limitadas a leitura de 2 registradores simultaneamente e a escrita em um registrador. A escrita possui duas limitações, ela só poderá ocorrer na borda de descida do clock e o registrador 0 possui um valor constante igual a 0, ou seja, não pode escrever nele.

2.5 PC

Registrador que armazena o valor de PC.

2.6 Somador

Esse componente tem duas entradas de 32 bits. A saída s dependerá do resultado soma dos dois números recebidos.

2.7 Memória de Instrução

A memória de instruções conterá todas as instruções do programa. O endereço vindo do módulo pc e servirá de referência à instrução nessa memória.

2.8 Memória de Dados

A memória de dados conterá todos os dados externos ao processador. As instruções que manipulam essa memória e acessam a mesma por meio de um endereço gerado com os campos da instrução e realizam a operação desejada. Da mesma forma que a memória de instruções, somente 8 bits são usados para o endereçamento.

3 Registradores de Pipeline

3.1 IF/ID

Esse registrador terá a tarefa de armazenar durante um ciclo de clock os valores de pc + 4 e da instrução.

3.2 ID/EX

Durante um período de clock, esse registrador deverá armazenar os sinais de controle que serão utilizados nas etapas EX, MEM e WB. Ele deverá armazenar também os valores de pc + 4, dos registradores rs e rt vindos do banco de registradores, do campo shamt da instrução, do valor estendido de kte e dos campos rt e rd da instrução.

3.3 EX/MEM

Durante um período de clock, esse registrador deverá armazenar os sinais de controle referentes às etapas MEM e WB. Ele também armazenará os valores de pc + 4, do resultado da ULA, de rt e do registrador destino escolhido por um mux4, no qual será escrito algum dado.

3.4 MEM/WB

Durante um período de clock, esse registrador deverá armazenar os sinais de controle referente à etapa WB. Ele também armazenará o resultado da ULA vindo do registrador EX/MEM, o dado lido da memória de dados, o valor de pc + 4 e o valor do registrador destino.

4 Controle

O controle depende da instrução recebida da memória de instruções. E então serão gerados sinais para controlar os componentes no processador. Os sinais de controle foram divididos em três subconjuntos, referentes às 3 últimas etapas do Pipeline (EX, MEM, WB). A lógica de controle é acionada na etapa de Decode da instrução. Nessa etapa existem alguns componentes que precisam de sinais de controle (sinais indicando se a instrução atual é um tipo de jump ou de branch). Então o controlador também mandará sinais para componentes no estágio ID.

5 Conclusão

Todas as instruções do Tipo-R, LUI, LW, SW, SLT, SLTi foram implementadas e testadas tanto na placa como no MODELSIM, as instruções de desvio (BEQ e BNE) funcionaram, mas com algumas limitações e de saltos (JAL, J, JR, JALR) foram implementadas, mas não tiveram o funcionamento correto.

6 Testes

6.1 Funções de Acesso a Memória

Para testar as funções de acesso e escrita a memória (SW e LW). Foi utilizado o código na figura 1. O código utilizado para verificação na placa não tem os NOPS, eles foram colocados apenas para espaçar os resultados durante o Testbench.

```
.text
    addi $t0, $zero, 4
    nop
    nop
    nop
    addi $t1, $zero, 777
    nop
    nop
    nop
    sw $t1, 0($t0)
    nop
    nop
    nop
    lw $t2, 0($t0)
```

Figura 1: Código em Assembly para teste de funções de memória.

Na primeira subida do clock temos:

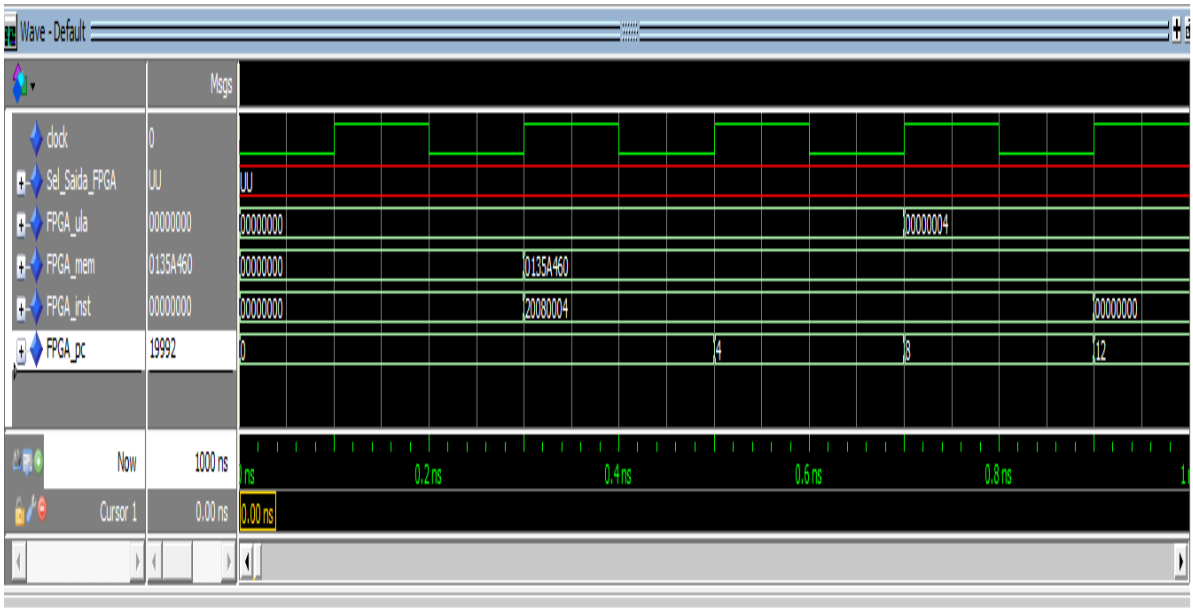

Figura 2: Primeiro ciclo de instruções.

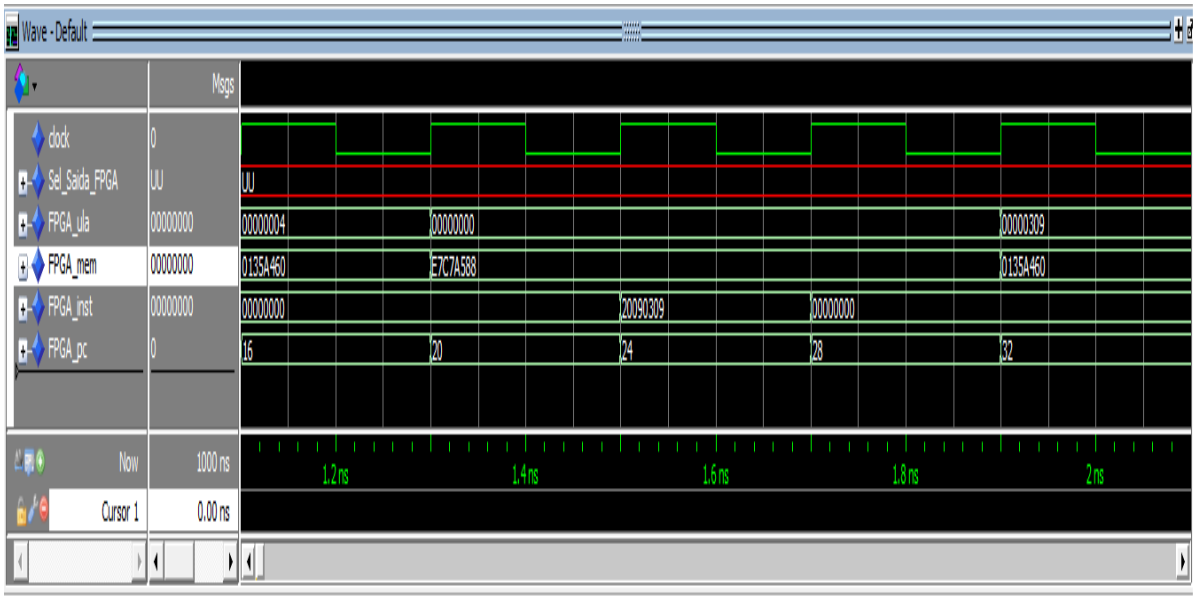

Figura 3: Segundo ciclo de instruções.

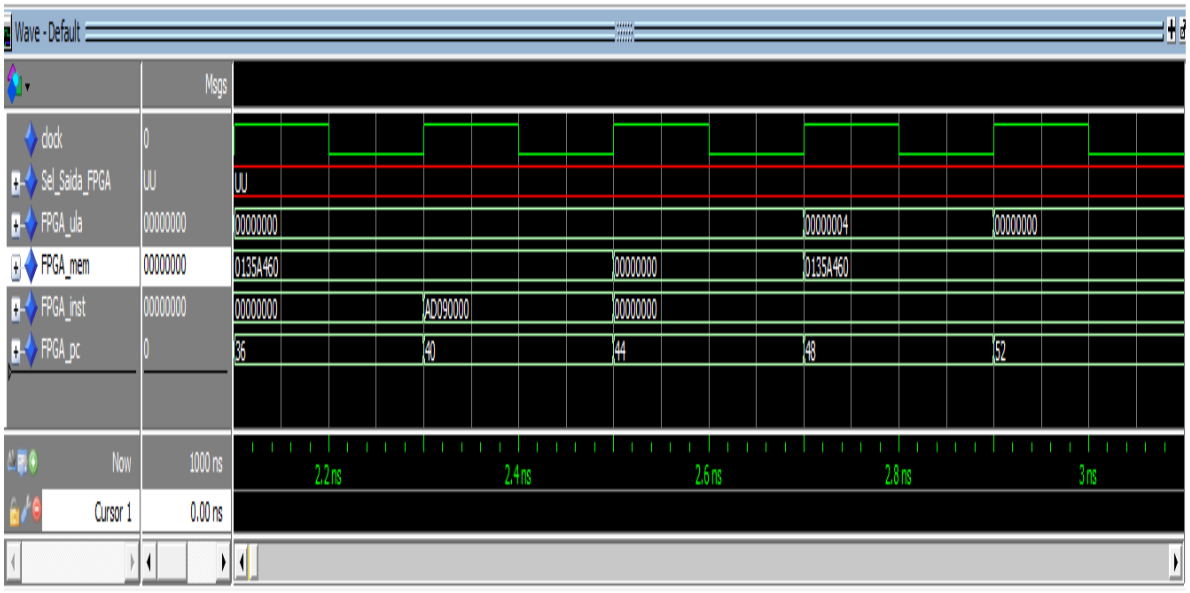

Figura 4: Terceiro ciclo de instruções.

6.2 Funções do Tipo-R, LUI, SLT e SLTi

Para as funções do Tipo-R foi utilizado o código na figura 6.

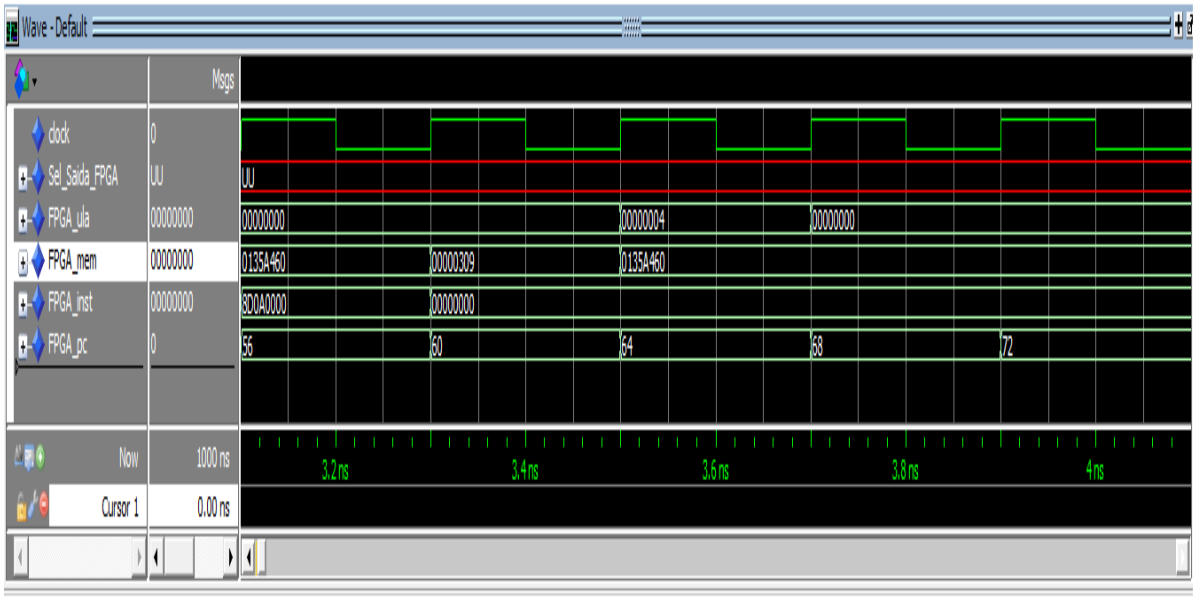

Figura 5: Ciclo final de instruções.

```
.text

        addi  $a0,  $zero,  2
        nop
        nop
        nop
        addi  $a1,  $zero,  10
        nop
        nop
        nop
        and  $t1,  $a0,  $a1
        nop
        nop
        nop
        or  $t2,  $a0,  $a1
        nop
        nop
        nop
        nor  $t3,  $a0,  $a1
        nop
        nop
        nop
        xor  $t4,  $a0,  $a1
        nop
        nop
        nop
        slt  $t5,  $a0,  $a1
        nop
        nop
        nop
        slti  $t6,  $a1,  12
```

Figura 6: Código utilizado para testar as funções do tipo-R.

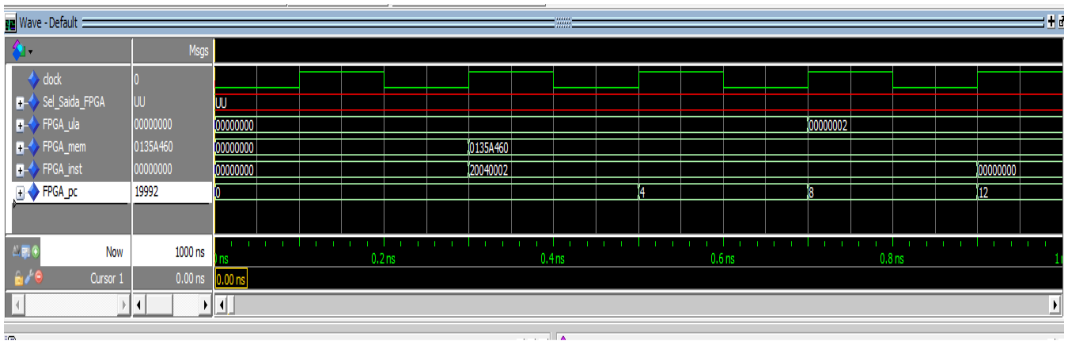

Figura 7: Testando a operação de AND.

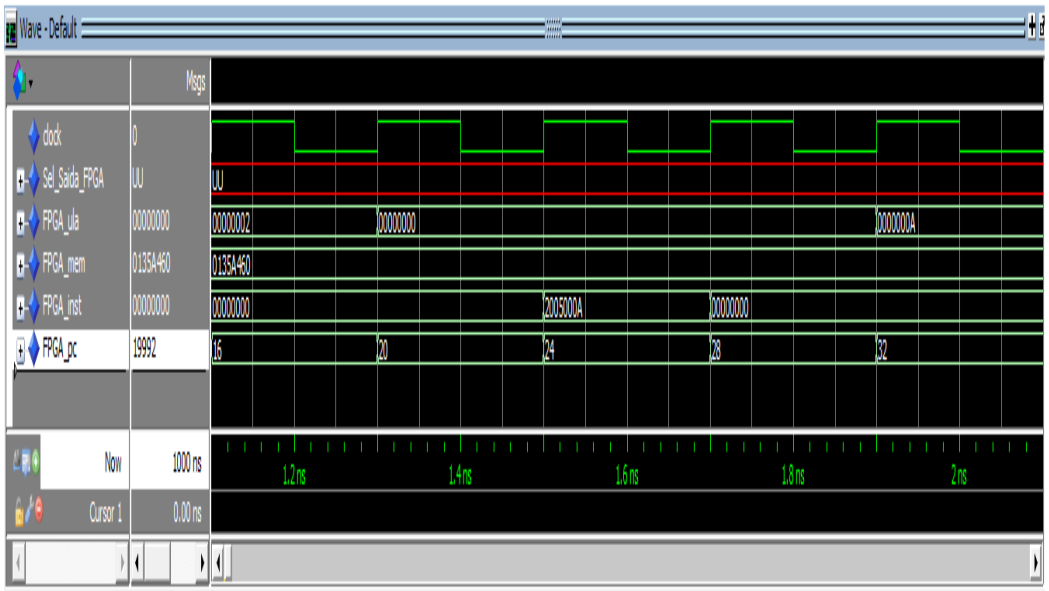

Figura 8: Testando a operação de OR.

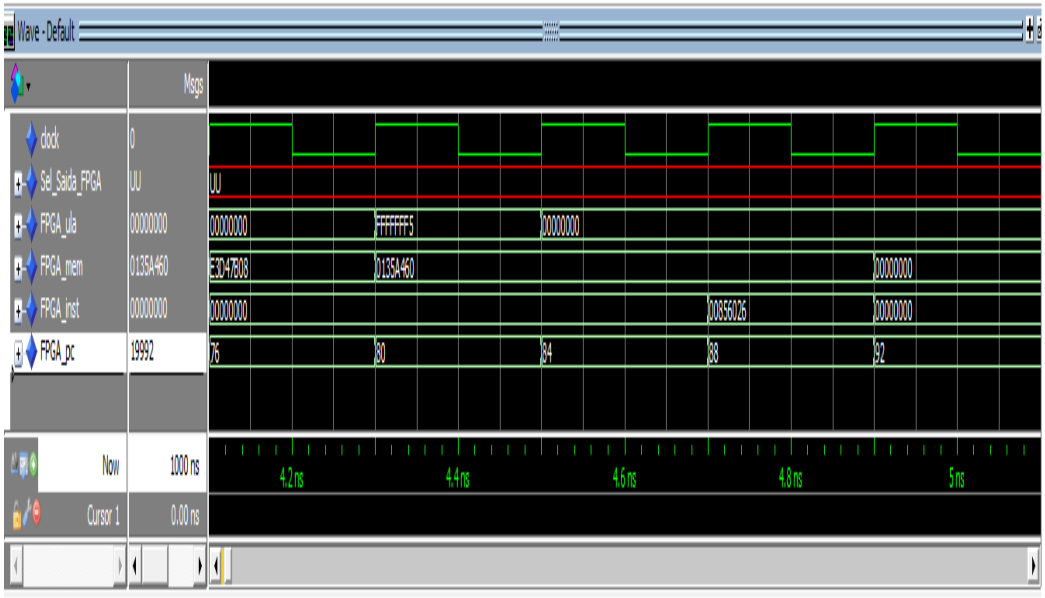

Figura 9: Testando a operação de NOR.

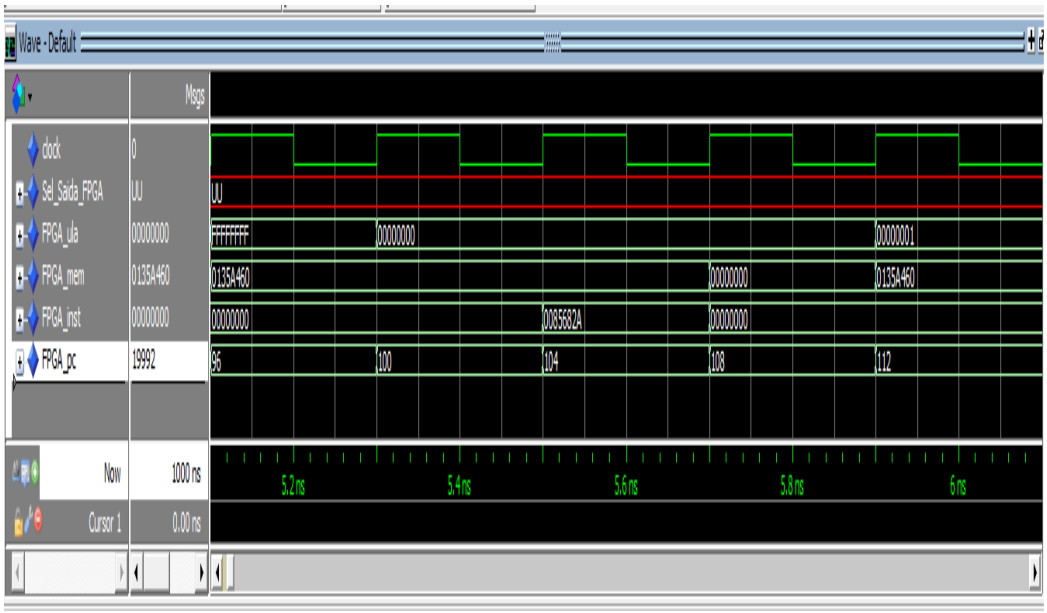

Figura 10: Testando a operação de XOR, SLT e SLTi.

```
.text

        add $a0, $zero, 2
        add $a1, $zero, 10#
        nop
        nop
        nop
        slt $t6, $a0, $a1
        slti $t7, $a1, 12
        lui $t5, 6
```

Figura 11: Código utilizado para testar a função LUI.

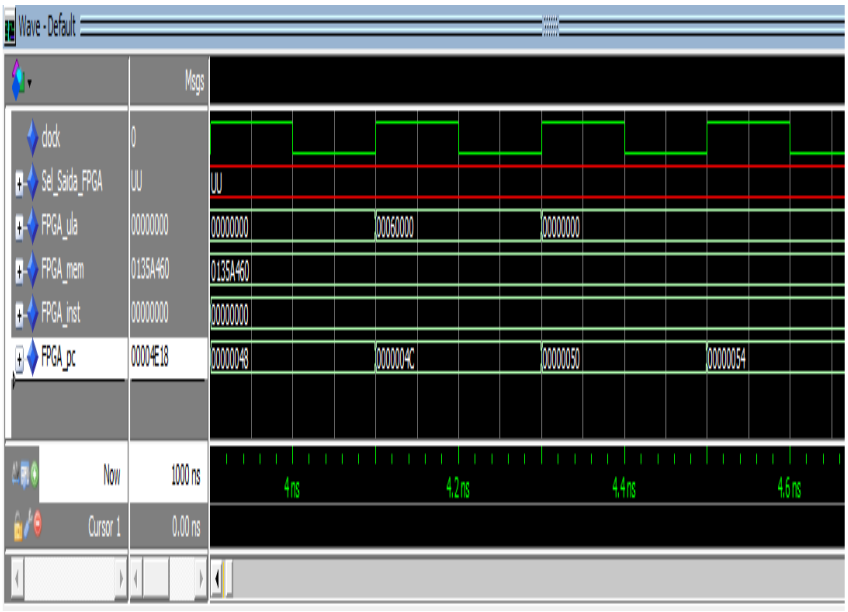

Figura 12: Testbench da função LUI.