Trabalho Prático de Organização de Computadores 1

Roberto G. Rosmaninho Neto[1], **Matheus T. Pimenta de Souza**[1], **Yasmin Araújo**[1], **Henry Tamekloe**[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte, MG – Brasil

{robertogomes,matheuspimenta,henrany,yasminaraujo}@dcc.ufmg.br

1. Introdução

O objetivo desse trabalho é implementar um processador em Verilog que emula o funcionamento de uma calculadora. Essa calculadora possui 5 unidades básicas, tais como: banco de registradores, unidade de memória, unidade lógica aritmética (ULA) e um controle de operações. Além disso, esse processador é monociclo, ou seja, cada instrução é processada em um único ciclo de *clock* e cada unidade funcional, bem como o processador inteiro, foram especificados utilizando a ordenação *little endin* para os bytes.
Esse processador foi escrito em Verilog, compilado com o **Icarus Verilog** e testando com o **GTK Wave**.

2. Implementação

Essa seção abordará as principais restrições, funcionalidades e peculiaridades de cada unidade lógica, além de explicar a necessidade de uma unidade de controle para o correto funcionamento do caminho de dados, este também será detalhado ao fim desta.

2.1. Banco de Registradores

O banco de registradores desse processador possui apenas 3 registrados de 32 bits cada. Estes foram nomeados, respectivamente, de RegFonteA, RegFonteB e RegAcumulador. Esta unidade funcional é descrita no arquivo **BancoReg.v**. Nesse arquivo, especificamos que cada escrita será realizada na borda de decida(negedge), enquanto cada leitura será realizada na bora de subida(posedge).
Além disso, foi definido que o banco possuiria 6 entradas e 2 saídas, sendo elas:

Nome	Tamanho	Tipo	Descrição
IdReg	2 bits	wire	Identificador do registrador a ser escrito.
Escrita	1 bit	wire	Realiza a escrita do dado no registrador caso o bit seja 1 ou a leitura do registrador caso o bit seja 0.
Dado	32 bits	wire	Dado a ser escrito no registrador, caso Escrita seja 1.
Fonte1	2 bits	wire	Identificador do primeiro registrador a ser lido.
Fonte2	2 bits	wire	Identificador do segundo registrador a ser lido.
Clock	1 bit	wire	Sinal do Clock utilizado para diferenciar o momento de Leitura/Escrita.

Tabela 1. Entrada de Dados - BancoReg.v

Nome	Tamanho	Tipo	Descrição
DadoLido1	32 bits	reg	Dado lido do primeiro registrador.
DadoLido2	32 bits	reg	Dado lido do segundo registrador.

Tabela 2. Saída de Dados - BancoReg.v

2.2. Unidade de Memória

A unidade de memoria é para a leitura e escrita de instruções, por ai temos os sinais que vamos utilizar na construção do da memoria são : Read Pc que é de tamanho 32 bits que é o endereço para a leitura de instrução , Read/write Addr que também é de 32 bits um endereço para leitura e escrita de instrucoes ,Op2En de 1 bit u sinal que indica se neste ciclo faremos um a segunda operação de memoria caso em nível alto, OpR2W de 1 bit que indica uma leitura em nível baixo e uma escrita em nível alto,Instrucition de 32 bits que é a saída para a instrução a ser executada,Data de 32 bits, um clock e um Datawrite para a escrita de instrução caso em nível alta.

Na implementação, os sinais sao declaradas como entrada e saída.Inicialmente, a leitura do conteúdo da memoria e depois faremos um loop para mostrar resultados apos a leitura. Depois disso, começamos com as operações para a escrita ou leitura da memoria, primeiro, verificamos se deve fazer uma segunda operação em nivel alto, e também verifica se faremos uma leitura ou escrita. se for de nível baixo ,escrevemos no sinal de saída Data e se for em nível baixo, faremos uma escrita no sinal Datawrite. Depois disso, a pegamos a instrução a ser executado pelo Instruction

2.3. Unidade Lógica Aritmética

A unidade lógica aritmética(ULA) é a principal unidade funcional desse processador, visto que este emula uma calculadora. A ULA normalmente é capaz de realizar operações lógicas como AND, OR, NOT e etc. No entanto, esse processador apenas realizará operações aritméticas, tais como, adição, subtração, divisão e multiplicação. Além disso, é importante observar que todas as operações serão realizadas utilizando os próprios operadores disponíveis no Verilog, os operandos são valores inteiros em 32 bits representados em complemento de dois, as operações ocorrem na subida de borda do clock (posedge) e, por fim, cada operação possui seu próprio código em 4 bits. Por fim, é possível sumarizar as principais propriedades dessa unidade nas seguintes tabelas que descrevem suas entradas e saída:

Nome	Tamanho	Tipo	Descrição
_op1	32 bits	wire	Primeiro dado da operação.
_op2	32 bits	wire	Segundo dado da operação .
_opcao	4 bits	wire	Controlador da operação a ser executada (one hot): 1000 - Soma 0100 - Subtração 0010 - Multiplicação 0001 - Divisão
_clock	1 bit	wire	Resultado da operação

Tabela 3. Entrada de Dados - ULA.v

Nome	Tamanho	Tipo	Descrição
_result	32 bits	reg	Resultado da operação.

Tabela 4. Saída de Dados - ULA.v

2.4. Unidade de Controle

A unidade de controle, apesar de não ter sido uma demandada do trabalho, foi implementada nesse processador visando simplificar e organizar o caminho de dados. Nela ocorrem os estágios de Fetch e Decode, ou seja, é nessa unidade que o código da instrução, 32 bits, é decodificado para ativar as outras unidades funcionais caso seja necessário. Assim, utilizando o padrão de instrução da Tabela 5, essa unidade é responsável por controlar qual operação será realizada, se e qual registrador será lido ou escrito, se uma posição da memória deve ser limpa(zerada) e se o processador deve parar de executar instruções. As principais funções realizadas pelo processador terão sua explicação detalhada posteriormente nesse documento.

31 29	28 27	26 25	24 0
Opcode – 3 bits	FonteA – 2 bits	FonteB – 2 bits	Imediato – 25 Bits

Tabela 5. Formato das Instruções

As entradas e saídas da unidade de controle são descritas nas tabelas abaixo:

Nome	Tamanho	Tipo	Descrição
_clock	1 bit	wire	Sinal de clock para realizar uma operação.
_instrucao	32 bits	wire	Instrução recebida pelo processador.

Tabela 6. Entrada de Dados - Controle.v

Nome	Tamanho	Tipo	Descrição
_ula_op	4 bits	reg	Saída com o código de operação para a ALU.
_mem_control	3 bits	reg	Saída com o código de operação para a memória.
_reg_dest	2 bits	reg	Saída com o identificador de registrador para destino do resultado da operação.
_imediato	25 bits	reg	Utilizado para zerar um registrador.

Tabela 7. Saída de Dados - Controle.v

As operações suportadas pela unidade de controle e, consequentemente, pelo processador são descritas na tabela abaixo:

.

Código	Nome	Descrição
000	Adição	Soma os valores em dois registradores.
001	Subtração	Subtrai o valor em um registrador pelo valor em outro.
010	Divisão	Divide o valor em um registrador pelo valor em outro.
011	Multiplicação	Multiplica o valor em um registrador pelo valor em outro.
100	Limpa Memória	Zera uma posição da memória.
101	Lê Memória	Lê uma posição da memória e salva em um registrador.
110	Escreve Memória	Escreve o dado em de um registrador em uma posição da memória.
111	Parada do Processador	Para o processador.

Tabela 8. Operações Suportadas pelo processador - Controle.v

3. Metodologia de Testes

Neste trabalho, há um arquivo ”TestBench”para cada unidade funcional. Dessa forma, foram realizados testes unitários para cada unidade de forma separada, estes foram feitos tanto utilizando a função *$monitor*, quanto a função *$display* durante a escrita do código e, quando essa foi finalizada, os resultados foram testados e avaliados utilizando o **GTK Wave**.
Além disso, após finalização de todos os testes das unidades funcionais e do caminho de dados, finalmente foi possível testar o processador inteiro, tanto utilizando as funções já mencionadas, quanto avaliando a forma da onda de entrada e saída de cada variável. Assim, alguns testes de usos gerais foram realizados por cada membro do grupo, visando a independência destes e a usabilidade da calculadora.
Por fim, os exemplos contidos no enunciado do trabalho foram testados no processador para completar os casos de testes necessários.

3.1. Avaliação

Neste trabalho, foram utilizados TestBenchs para simular o funcionamento do processador. Assim, as principais unidades independentes foram testadas de forma unitária, este é o caso do banco de registradores e da memória mostrados abaixo. Além disso, os resultados foram testados tanto utilizando as *flags* já mencionadas *$display $monitor*, também foi utilizado o *GTK Wave* para avaliar os formatos de onda das variáveis de acordo com as mudanças de *clock*.

Nas imagens abaixo, é possível observar que a da esquerda representa o resultado do teste da memória no Terminal com uso uso de uma das flags e na direita o uso do software para avaliar os resultados dos testes do banco de registradores.

.

```
1 `timescale 1ns/1ps
2 module memory_test();
3
4  wire Op2En,Op2RW;
5  reg [31:0] ReadPC,ReadWriteAddr,DataWrite;
6
7  wire [31:0] Data,Instruction;
8     reg [31:0] memory[0:1023];
9
10 reg _clk;
11     //instanciating
12 Memoria memoria(.clk(_clk),
13          .Op2En(Op2En),
14          .Op2RW(Op2RW),
15          .ReadPC(ReadPC),
16          .ReadWriteAddr(ReadWriteAddr),
17          .Data(Data),
18          .Instruction(Instruction));
19
20 //setting the clock
21 always
22 begin
23     _clk = 1; #10;
24     _clk = 0; #10;
25 end
26
27 initial begin
28   $dumpfile("Memory.vcd");
29   $dumpvars(0, memory_test);
30 end
31
32 //initial test
33 initial
34    begin
35        $readmemh("memory.txt", memory);
36        ReadWriteAddr[31:0] <= 5'b0;
37        #20;
38        ReadWriteAddr[31:0] <= 5'b0;
39        #20;
40
41 $finish;
42 //end of tests
43 end
44 endmodule
45
```

(a) Memória

```
1 module BancoReg_TestBench;
2
3   //Variaveis necessárias para iniciar o banco de registrador
4   reg  [1:0] _id, _f1, _f2;
5   reg  _escrita;
6   reg  [31:0] _dado;
7   wire [31:0] _dl1, _dl2;
8   reg clk;
9
10 //Instanciando o Banco de Registrador
11  BancoReg banco (.Clock(clk),
12                  .IdReg(_id), .Fonte1(_f1), .Fonte2(_f2),
13                  .Escrita(_escrita),
14                  .Dado(_dado),
15                  .DadoLido1(_dl1), .DadoLido2(_dl2));
16
17 //Gerando arquivo vara vizualição no GTKWave
18  initial begin
19    $dumpfile("BancoReg.vcd");
20    $dumpvars(0, BancoReg_TestBench);
21  end
22
23  //Testamos a lógica aqui
24  initial begin
25
26    //Insere as variáveis
27    _escrita = 1; #0;
28    _id = 2'b00; #0;
29    _dado = #0 32'b00000000000000000000011111100011;
30
31   //Escreve as variaveis na descida do clock
32   clk = #5 1'b0;
33
34   //Insirimos as variáveis
35   clk = #0 1'b0;
36   _escrita = 0; #5;
37   _f1 = 2'b00; #5;
38
39   //Lemos as variáveis na subida do clock
40     clk = #0 1'b1;
41  end
42 endmodule
```

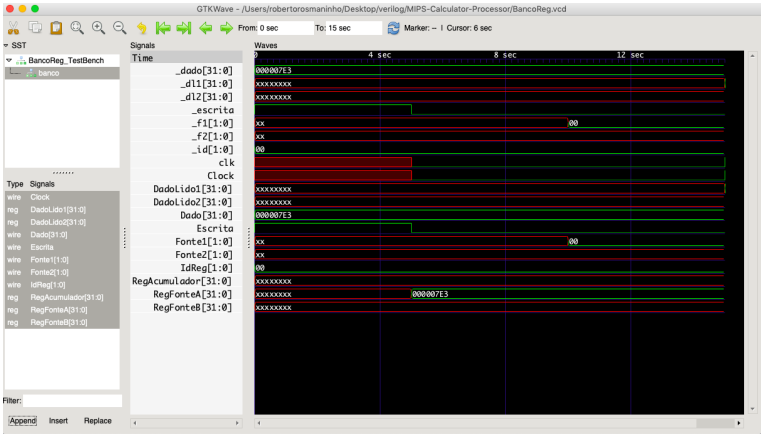
(b) Registradores

Figura 1. Exemplos de BenchTests

Os testes acima realizam as seguintes tarefas: No caso da memória é lido um arquivo com instruções em código binário e no caso do banco de registrados o número 2019 é escrito em binário em um registrador e depois lido deste, os resultados estão disponíveis abaixo:

```
results:
          0:10000002
          1:00000005
          2:00000005
          3:a0000000
```

(a) Resultado Memória



(b) Formato das Ondas das variáveis do Banco de Registradores

Finalmente, após a construção do *DataPath*, foi possível simular o funcionamento completo do processador e o rsultado utilizando o *GTK Wave* é mostrado abaixo:

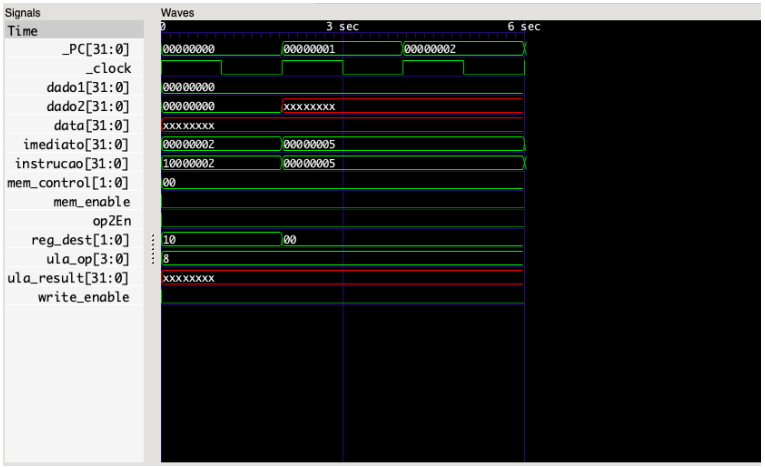


Figura 2. Caption

4. Conclusão

A implementação desse processador simplificado, ou seja, dessa calculadora se deu da seguinte maneira: Utilizamos as 3 unidades funcionais especificadas na descrição do trabalho e implementamos outras duas: Controle e *Datapath*. Dessa forma, foi possível compilar corretamente o código Verilog e executá-lo de forma que as conexões entre as unidades funcionais fosse simples e eficiente e assim os testes fossem realizados de forma correta obtendo os resultados esperados.