Trabalho Prático de Organização de Computadores I

Professor: Antônio Otávio Fernandes

Elaborado por: Nestor D. O. Volpini, Thiago Rodrigues B. da S. Soares, Fernando Carvalho da Silva Coelho.

Monitor: Fernando Carvalho da Silva Coelho
fccoelho@dcc.ufmg.br

1. Introdução

Neste trabalho, o objetivo é a implementação de um processador multiciclo de 32 bits, similar ao MIPS com instruções de tamanho fixo e cinco estágios de processamento.

Dessa forma, para implementar esse processador, deverá ser utilizada a linguagem de descrição de Hardware (HDL) Verilog, cuja utilização é comum nos projetos envolvendo desenvolvimento de hardware.

Assim, o aluno terá a oportunidade de aprender a utilizar ferramentas comuns para esse propósito, além de entrar em contato com o processo de desenvolvimento de um processador simples, que reforçará os conteúdos vistos na disciplina de Organização de Computadores I.

2. Módulos do processador

O trabalho será segmentado em módulos funcionais para facilitar a organização do desenvolvimento do processador.

Nessas condições, cada módulo contem interfaces externas com sinais de entradas e saidas que se comunicam com outros módulos do processador.

2.1. Unidade Lógica Aritmética (ULA)

A Unidade Lógica Aritmética (ULA), em inglês ALU (Arithmetic Logic Unit) é um circuito digital que faz as operações aritiméticas simples e as operações lógicas binárias em números inteiros. Esse circuito é uma parte fundamental de qualquer unidade de processamento, tais como CPUs e GPUs.

Neste trabalho, a ULA desenvolvida terá duas entradas de dados de 32 bits e uma entrada de códigos de operação de 3 bits, além de uma saida de resultados de 32 bits e um sinal informando se a operação executada resultou no valor zero, conforme representado na figura 1 e explicados na tabela 1. Assim, esse módulo será capaz de executar cinco operações, em dados de 32 bits, conforme descrição na tabela 2.

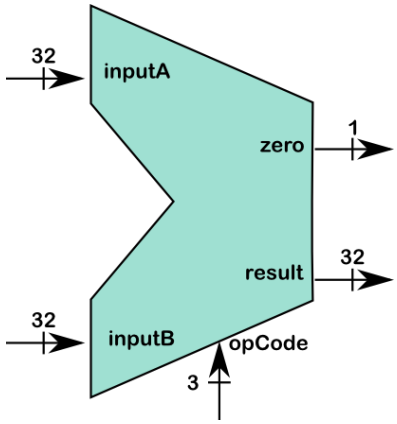


Figura 1: Representação da unidade lógica aritmética.

Nome do sinal	Tamanho	Tipo	Descrição
inputA	32 bits	Input	Operando 1 da operação
inputB	32 bits	Input	Operando 2 da operação
opCode	3 bits	Input	Seletor da operação a ser realizada
zero	1 bit	Output	Se a operação realizada resultar em zero, esse sinal será ligado em 1 caso contrário será 0.
result	32 bits	Output	Resultado da operação executada na ULA.

Tabela 1: Descrição da interface da ULA.

Operação (opCode)	Mnemônico	Descrição
000	ADD	Adição
001	SUB	Subtração
010	AND	Operação de AND Lógico
011	OR	Operação de OR Lógico
100	XOR	Operação de XOR Lógico
101	NOP	Nenhuma operação
110	NOP	Nenhuma operação
111	NOP	Nenhuma operação

Tabela 2: Descrição das operações da ULA.

2.2. Banco de Registradores

O Banco de registradores é um vetor de registradores dentro do processador, utilizados para acesso rápido de dados.

Nesse trabalho, haverá um vetor de 32 registradores 32 bits a serem usados nas operações da ULA. Para acessar esses registradores, o módulo contem dois sinais de 5 bits para definir o endereço do registrador a ser lido, um sinal de 5 bits que define o endereço do registrador a ser escrito, uma entrada de dados de 32 bits, duas saidas de dados de 32 bits e um sinal de 1 bit que define se o banco de registradores deve armazenar os dados que chegam ou não. Na figura 3, está a representação da interface do módulo.

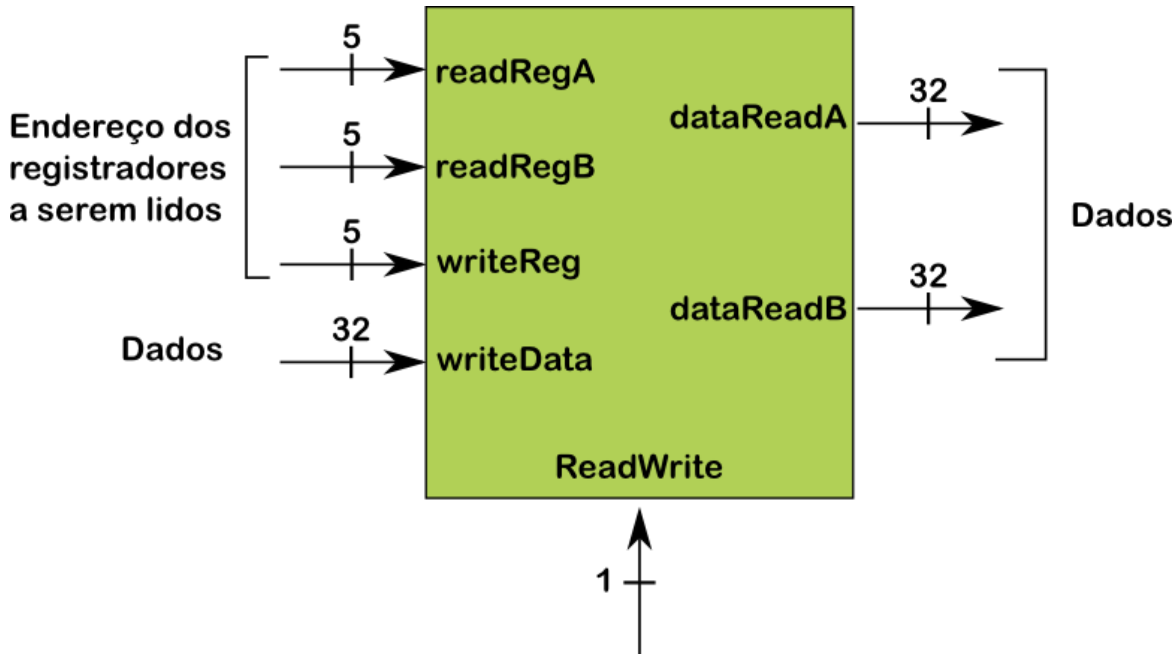


Figura 3: Interface do banco de registradores.

Nome do sinal	Tamanho	Tipo	Descrição
readRegA	5 bits	Input	Define o endereço do registrador a ser ligado na saida `dataReadA`
readRegB	5 bits	Input	Define o endereço do registrador a ser ligado na saida `dataReadB`
writeReg	5 bits	Input	Define o endereço do registrador que receberá os dados colocados em `writeData`
readWrite	1 bit	Input	Quando estiver ligado em 1, os dados de `writeData` deverão ser escritos no registrador endereçado por `writeReg`
writeData	32 bits	Input	Recebe os dados a serem escritos no registrador endereçado por `writeReg`
dataReadA	32 bits	Output	Saida dos dados armazenados no registrador endereçado em `readRegA`
dataReadB	32 bits	Output	Saida dos dados armazenados no registrador endereçado em `readRegB`

Apêndice

1. Exemplo de referência

Para facilitar a apresentação da linguagem Verilog aos alunos, nessa seção há alguns trechos de código de exemplo.

Dessa forma, crie um arquivo `counter.v` para o módulo contador, com o seguinte código:

```
module counter(clock, clear, out);
input            clear;
input            clock;
output reg [1:0] out;

always @(posedge clock)
begin
      if(clear == 1)
            out <= 2'b00;
      else
            out <= out + 1'b1;
end
endmodule
```

Depois, crie um arquivo `testbench.v` para o módulo que servirá de testbench.

```
`include "counter.v"
module testbench;

reg        clock = 0;
reg        clear = 0;
wire [1:0] out;

always #1 clock = !clock;

initial $dumpfile("testbench.vcd");
initial $dumpvars(0, testbench);

counter c(clock, clear, out);

initial begin
      //These events must be in chronological order.
      # 5  clear = 1;
      # 1  clear = 0;
      # 20 $finish;
end
endmodule
```

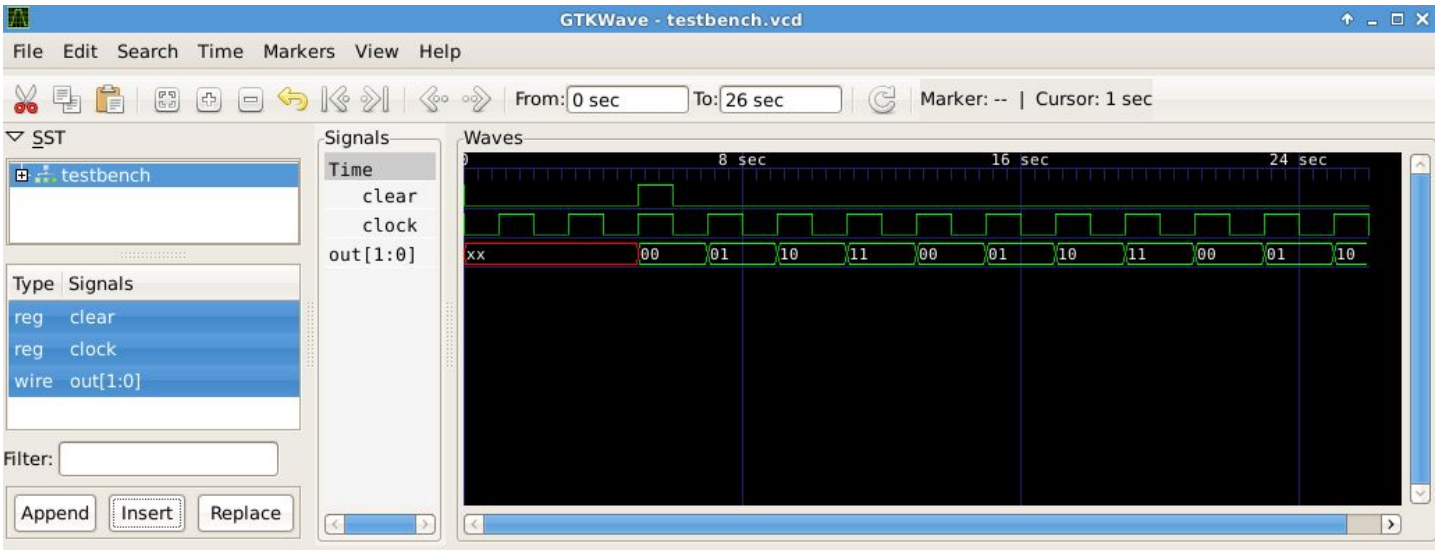
Compile os módulos utilizando o iverilog:

```
iverilog testbench.v
```

Execute o programa gerado utilizando o aplicativo vvp:
`vvp a.out`

Utilize o GTKwave para visualizar a forma de onda das saidas do seu programa:
`gtkwave testbench.vcd`

A saida será a seguinte:



2. Softwares de apoio

Para o desenvolvimento é necessário o uso do sofware de apoio Quartus II da empresa Altera (link para download http://dl.altera.com/?edition=web)