

Arquitetura e Organização de Computadores

Turma C - 2021/02

Projeto das Memórias de Dados e Instruções

Objetivo: projetar, simular e sintetizar as memórias de dados e instruções do RISC-V.

Resumo

O RISC-V pipeline utiliza duas memórias internas para armazenar programa e dados. Neste trabalho deverá ser desenvolvido um modelo VHDL para cada memória. A **memória de dados** deve ser uma memória tipo RAM, com processos de leitura e escrita. No caso da **memória de instruções**, os sinais de escrita não são necessários, pois é uma memória de apenas leitura (ROM), que deve ser iniciada com instruções RV lidas de um arquivo texto.

Descrição

O bloco de memória ROM tem a seguinte interface:

- um barramento de endereço (suportar 8 bits de endereço)
- um barramento de dados de saída (32 bits)

O bloco de memória RAM tem a seguinte interface:

- um barramento de endereço (suportar 8 bits de endereço)
- um barramento de dados de entrada (32 bits)
- um barramento de dados de saída (32 bits)
- sinal de habilitação de escrita (*wren write-enable*)

Com relação ao modelo VHDL, é sugerida a seguinte abordagem:

1. Utilizar uma descrição reutilizável. Isso pode ser obtido definindo-se a interface da seguinte maneira:

```
entity rom_rv is
  port (
    address : in  std_logic_vector;
    dataout  : out std_logic_vector
  );
end entity rom_rv;

entity ram_rv is
  port (
    clock    : in  std_logic;
    we       : in  std_logic;
    address  : in  std_logic_vector;
    datain   : in  std_logic_vector;
    dataout  : out std_logic_vector
  );
end entity ram_rv;
```

Neste caso, os parâmetros devem ser obtidos a partir das propriedades dos sinais conectados às portas:

```
architecture RTL of ram_rv is
  Type mem_type is array (0 to (2**address'length)-1) of std_logic_vector(datain'range);
  signal ram : mem_type;
  signal read_address : std_logic_vector(address'range);
```

Arquitetura e Organização de Computadores

Turma C - 2021/02

```
architecture RTL of rom_rv is
  Type mem_type is array (0 to (2**address'length)-1) of std_logic_vector(dataout'range);
  signal rom : rom_type;
  signal read_address : std_logic_vector(address'range);
```

2. Permitir leitura imediatamente após a escrita, ou seja, se um endereço for escrito e lido ao mesmo tempo, o dado escrito é retornado pela leitura.

Isso pode ser realizado atrasando-se um delta ciclo a leitura, utilizando-se o sinal interno `read_address`:

```
Write: process(clock) begin
  ...
  if we = '1' then
    ram(to_integer(unsigned(address))) <= datain;
  end if;
  read_address <= address;
  ...
end process;

dataout <= ram(to_integer(unsigned(read_address)));
```

3. No caso da memória de instruções, não existe processo de escrita, apenas a leitura das instruções. A carga das instruções a partir de um arquivo texto, contendo em cada linha uma palavra de 32 bits em hexadecimal, pode ser realizada conforme indicado no link abaixo:

- a. <https://vhdlwhiz.com/initialize-ram-from-file/>

obs: note que é necessário utilizar o padrão VHDL-2008. No ModelSim, para indicar a utilização desse padrão deve-se clicar com o botão direito do mouse nos arquivos e selecionar a opção PROPERTIES, aba VHDL. Selecionar 1076-2008.

No EdaPlayground a versão do VHDL é indicada na entrada “Compile Options” e, por padrão, é 2008.

4. *Testbench* para verificação das memórias:
 - a. Memória de dados: escrever e ler uma sequência de valores.
 - b. Memória de instruções: carregar um arquivo com instruções do RV para a ROM e exibí-los.
 - c. A visualização das saídas pode ser feita com a janela de formas de onda do ModelSim / EdaPlayground.
 - d. A geração de estímulos pode ser feita com o auxílio do comando *for loop*:

```
for i in 0 to 255 loop
  address <= std_logic_vector(to_unsigned(i,8));
  datain <= std_logic_vector(to_unsigned(i,30)) & "00";
  wait for 1 ns;
end loop;
```

Entrega:

- Arquivo “.zip” tendo o número de matrícula como nome, incluindo os arquivos VHDL, o arquivo com as instruções lidas para a ROM.