µProcessador 7 Memória de Dados + Validação

Vamos lá garotada, chegou a hora de vocês brilharem! É o final da paradinha!

Implemente no circuito uma memória RAM e instruções para usá-la. As instruções deverão ser exatamente aquelas de leitura e escrita de memória no processador escolhido.

As datas de entrega, primeiro a RAM e depois a validação, estão no cronograma.

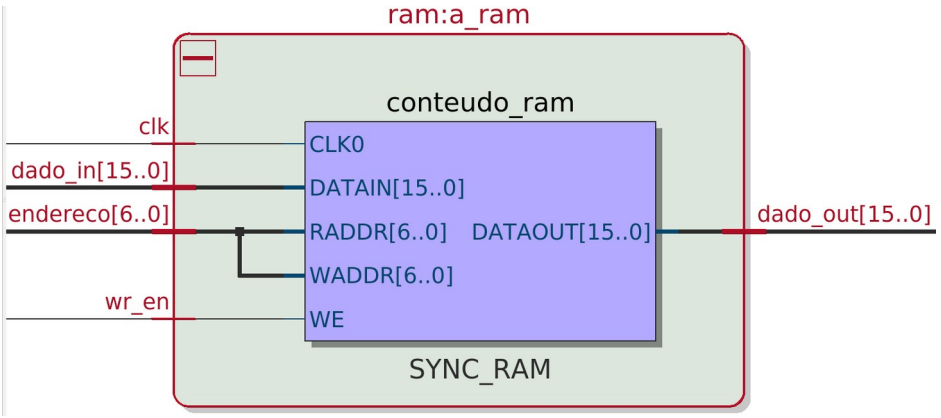
Requisitos

O endereço a ser lido ou escrito deve obrigatoriamente usar um registrador como ponteiro, para, por exemplo, poder fazer *loops* sobre vetores.[1]

Abaixo um exemplo com instruções que não incluem constantes:

```
addi   $r3,$r0,0x1E      # endereço 1E16 carregado no reg 3
lw     $r1,($r3)         # dado lido da RAM carregado no reg 1
```

Note que a RAM deverá ter barramentos de dados e endereços explícitos. O barramento de controle pode ser apenas o *clock* e o *write enable* que já está bom. Pra facilitar, o *data bus* pode ser dividido entre dados para serem escritos e dados lidos, como na figura abaixo:



Fora isso, o que tem pra fazer é decodificar as novas instruções e gerar os sinais de controle, seguindo o esquema do livro de preferência. Um mux dá as caras nos dados a serem escritos no Banco de Registradores. Por fim, não esqueça de verificar os momentos de leitura da RAM e escrita no Banco; altere a máquina de estados se for necessário.

Testes

Faça testes simples mas não seja otário:

- Faça várias escritas com dados variados em endereços espaçados, bem aleatório.
- Usar o valor do endereço para o dado ("número 3 no endereço 3") pode dar "falso ok."
- Evite testar com leitura subsequente à escrita: o valor pode estar sobrando ainda em algum barramento ou registrador temporário.
- Evite usar os mesmos registradores: use todos, um pra cada coisa, com valores embaralhados.
- Fazer *uma só* escrita e leitura subsequente é coisa de novato não promissor, sabe?

Teste direito. É fácil deixar um *bug* sem detecção.

1 Pode ser incluída uma constante de deslocamento, como no lw e sw do MIPS. Ou não, você é quem sabe,

Detalhes de Implementação

A RAM completa em VHDL[2] segue:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
--------------------------------------------------------------------------------
entity ram is
   port(
         clk      : in std_logic;
         endereco : in unsigned(6 downto 0);
         wr_en    : in std_logic;
         dado_in  : in unsigned(15 downto 0);
         dado_out : out unsigned(15 downto 0)
   );
end entity;
--------------------------------------------------------------------------------
architecture a_ram of ram is
   type mem is array (0 to 127) of unsigned(15 downto 0);
   signal conteudo_ram : mem;
begin
   process(clk,wr_en)
   begin
      if rising_edge(clk) then
         if wr_en='1' then
            conteudo_ram(to_integer(endereco)) <= dado_in;
         end if;
      end if;
   end process;
   dado_out <= conteudo_ram(to_integer(endereco));
end architecture;
```

Note que, desta forma, a escrita é síncrona e a leitura é assíncrona[3].

Há alternativas de *templates* diferentes, como os do Quartus II, ou componentes proprietários (IP – *Intellectual Property*) caso não se use o *ghdl*.

Arquivos a Entregar

Para a entrega do lab #7, envie um arquivo compactado com:

- ✔ Arquivos VHDL do projeto e seus *testbenches*
- ✔ Páginas do manual com as instruções de memória escolhidas em destaque
- ✔ Especificação atualizada da codificação das instruções
- ✔ Código *assembly* e codificação na ROM para o programa-teste pedido acima

Validação do µProcessador – Última Entrega

Implemente um programa que cospe os números primos em algum pino do *top-level*.

O quê? Como? Onde?

Use o Crivo de Eratóstenes[4], ele é *brother*. Sério, use o Crivo. Assim:

1. Coloque na memória RAM, com um *loop* em *assembly*, os números de interesse:
   - no mínimo até o 32;
   - coloque o número 1 no endereço 1, o 2 no 2, o 3 no 3... (facilita).
2. Elimine da lista, com um *loop*, todos os múltiplos de 2.
3. Idem para 3 e 5.

2 Não tem muito como, nesta disciplina, ensinar sem dar tudo pronto. Então que vá. É "só" colocar pra funcionar.

3 A escrita sempre ocorre numa rampa de subida do *clock*; já a leitura reflete instantaneamente na saída de dados qualquer alteração na entrada de endereços.

4 http://pt.wikipedia.org/wiki/Crivo_de_Erat%C3%B3stenes

4. Seria decente seguir o algoritmo completo (“tente” eliminar todos os não primos, como os múltiplos de 7, 11 e etc.), então faça isso se der.
5. Faça um *loop* para ler a RAM do endereço 2 ao 32.
6. Ponha um pino extra para visualizar as coisas ou simplesmente jogue o dado lido da RAM na saída da ULA, para que possamos acompanhar o resultado.

O algoritmo é simples, mas, meu colega, o *debug...* Aiaiai, é *realmente trabalhoso* se algo der errado. Não é à toa que isso vale cerca de dois pontos na média.

Entregue tudo seguindo o padrão estabelecido nestes últimos labs.

Dicas:

- Gere os *opcodes* com cuidado. **Confira** com muita atenção. É muito raro não errar uma ou duas instruções, ou seja:
  - *Assuma que você errou algum opcode.*
  - Faça a listagem de *assembly/opcodes* no formato *mais fácil possível* para conferência.
- Se algo der errado, regras gerais de **depuração**:
  - Identifique a instrução responsável pelo mau funcionamento.
  - Coloque pinos extras de *debug* pra todo lado.
  - Verifique, em especial, os momentos de gravação de dados: veja as ondas dos *enables* de escrita e o dado na entrada no momento da rampa de gravação.
- Deixe *vários dias* disponíveis para o *debug*, ou seja, **adiante** o projeto.
- Não se assuste se descobrir ***bugs* passados** até então ocultos[5]. Eles são frutos de testes insuficientes.

A recuperação desta validação, caso você falhe em entregá-la no prazo, é botá-la pra funcionar, valendo 50% da nota original.

5 Eu perdi três horas de vida pra descobrir que minha leitura da RAM na verdade não tinha funcionado, porque meu teste não tinha sido esperto.