Resumo DesComp

Autor: Enricco Gemha

Arquitetura baseada em acumulador

O resultado, de qualquer operação, sempre é armazenado no acumulador (AC ou A), está implícito o uso desse registrador. As instruções possuem apenas um endereço, ou sempre será uma posição de memória ou valor imediato.

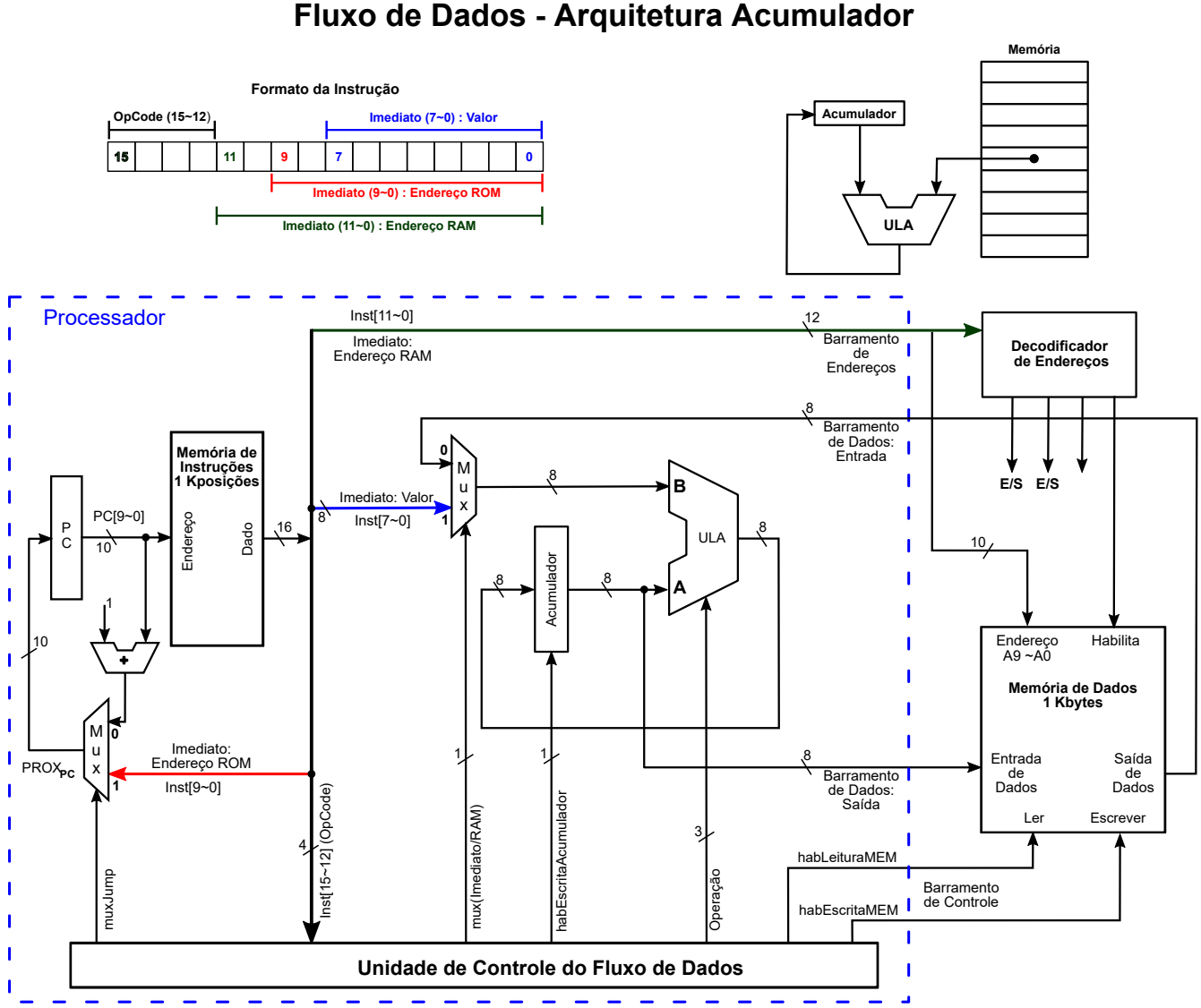
- `AC` é o único registrador, acumulador
- `Mem[y]` é o conteúdo da posição de memória número `y`

`ADD 0X1FFF` | `AC = AC + Mem[0x1FF]` | `Load 0x1FF` | `AC = Mem[0x1FF]` | `STORE 0X1FF` | `Mem[0x1FF] = AC`

Como a instrução possui apenas um endereço de operando (da memória), a implementação (organização) do processador necessita somente de um barramento.

```
Opcode (15 ~ 12) | Imediato (11 ~ 0): ram | Imediato (9 ~ 0): rom | imediato (7 ~ 0): valor
```

Exemplo: Processador com 16 instruções e endereçamento de 1024 de memória. Se tivermos um total de 16 instruções ou menos, precisaremos de 4 bits para codificar as instruções. Para endereçar todas as 1024 posições de memória, precisaremos de 10 bits ($1024 = 2^{10}$) O formato das instruções terá 14 bits, divididos e, 4 bits para o opcode da instrução e 10 bits para endereço de memória.



Arquitetura baseada em registrador e memória

O processador possui uma quantidade `N` de registradores, onde `N` > 1.

As operações ocorrem entre um registrador e uma posição de memória ou um valor imediato. O resultado é armazenado no mesmo registrador utilizado como um dos argumentos. Essa é a arquitetura do x86.

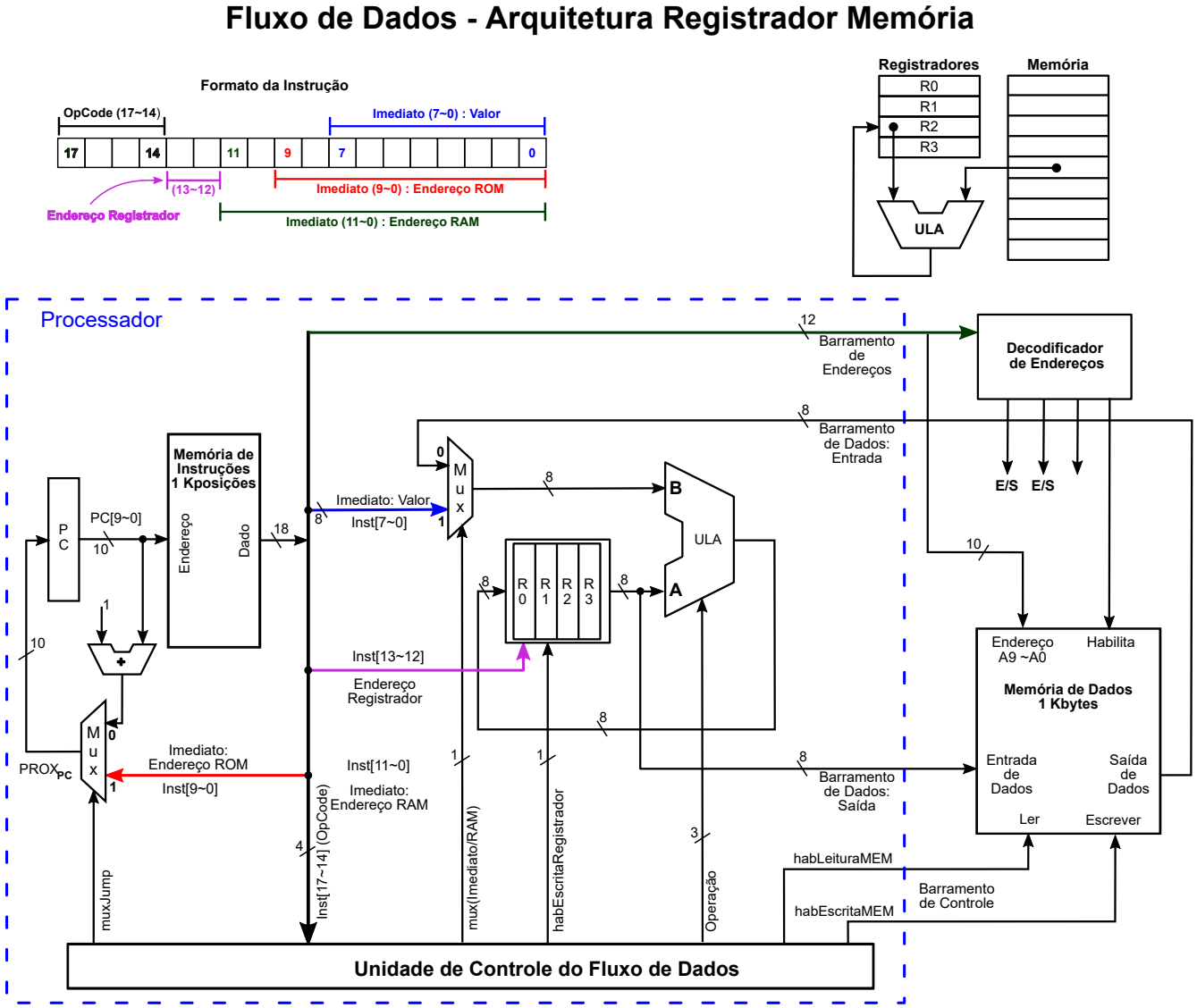
`Add R1, 0x1FF` | `R[1] = R[1] + Mem[0x1FF]` | `Load R1, 0x1FF` | `R[1] = Mem[0x1FF]` | `Store 0x1FF, R1` | `Mem[0x1FF] = R[1]`

Como a instrução possui um endereço para o registrador e outro para o operando (da memória), a implementação (organização) do processador necessita de dois barramentos:

- O barramento da RAM, constituído pelos bits 11 até 0 do imediato
- O barramento de endereços de registradores, constituído pelos bits 13 e 12 da instrução.

Exemplo: Temos um processador de 32 instruções, 8 registradores e endereçamento de 2048 posições de memória. Se tivermos um total de 32 instruções ou menos, precisaremos de 5 bits para codificar as 32 instruções. Para definir qual registrador usar, precisaremos de 3 bits (2^3). Para endereçar todas as 2048 posições, precisaremos de 11 bits. O formato da instrução de 19 bits, divididos em:

- 5 bits para opcode
- 3 bits para endereço do registrador
- 11 bits para o endereço de memória utilizado (ou 8 bits para o imediato)



Arquitetura baseada em registrador de uso geral

Chamada de registrador-registrador ou load-store. As operações só ocorrem entre registradores e os valores para esses registradores devem ser carregados da memória (ou imediato) através de uma instrução específica:

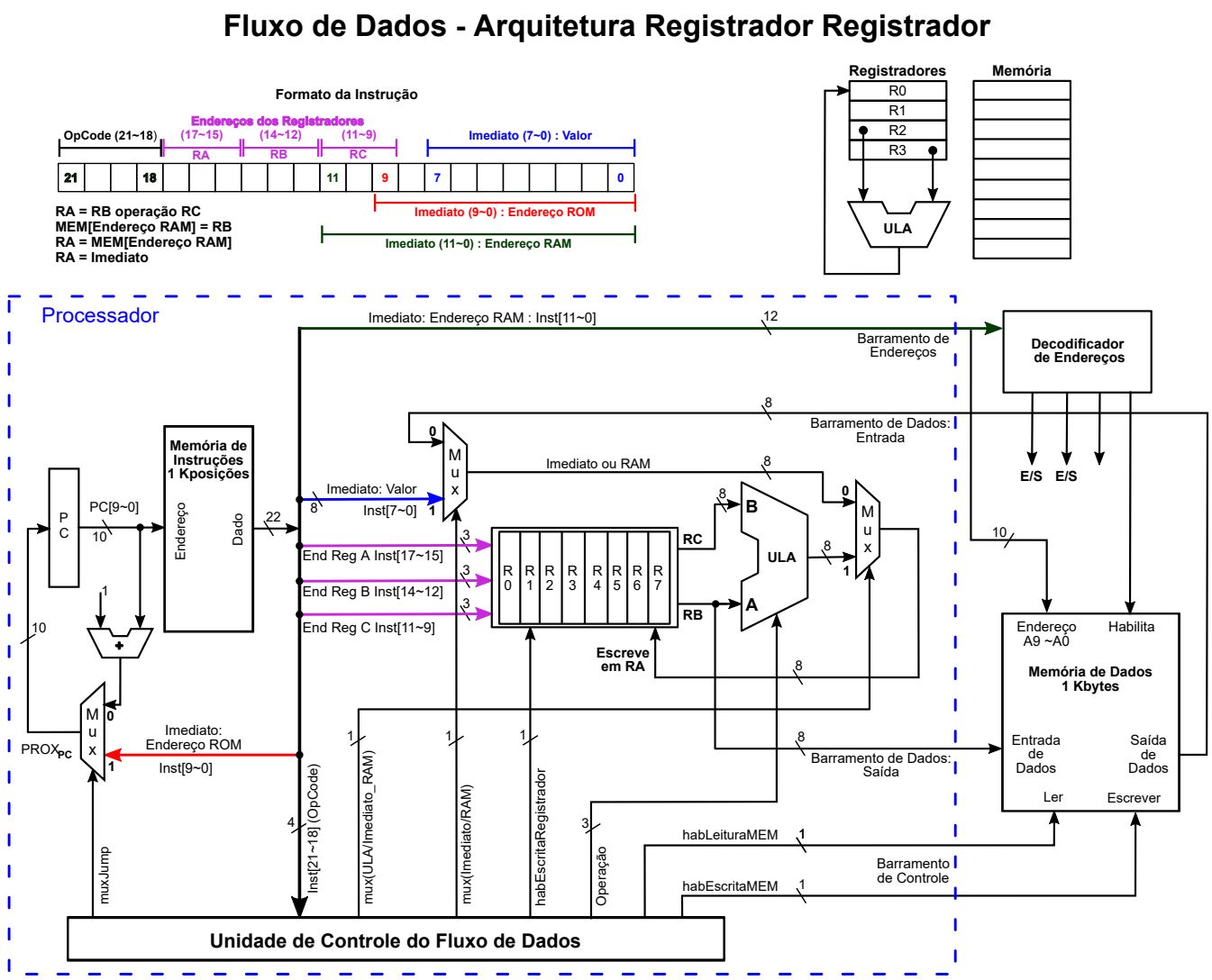
- Carregar os valores da memória usa uma instrução de carga (`load`); O resultado é armazenado em um registrador definido e pode ser guardado na memória usando uma instrução específica.
- O retorno do resultado para a memória é feito com uma instrução de armazenamento (`store`). `Add R1, R2, R3` | `R[1] = R[2] + R[3]` | `Load R1, 0x1FF`. Temos, dois tipos de instruções diferentes:
    - A instruções de acesso à memória (carga e armazenamento) que possuem dois argumentos: endereço de memória e registrador;
    - As instruções que executam uma operação lógica ou aritmética {possuem três argumentos: endereços dos 3 registradores}

Exemplo lógico aritmético: temos um processador de 64 instruções, 32 registradores e endereçamento de 4096 posições de memória.

Se tivermos um total de 64 instruções ou menos, precisaremos de 6 bits para decodificar. Para definir quais registradores usar, precisaremos de 5 bits (2^5) um para cada registrador.

O formato de instrução terá, no mínimo, 21 bits, divididos em: 6 bits para instrução e 5 para para endereço de cada registrador, totalizando 15 bits.

Exemplo acesso memória: o formato da instrução terá, no mínimo, 24 bits, divididos em: 6 bits para o opcode da instrução, 5 bits para o endereço do registrador, 13 bits para o endereço de memória.



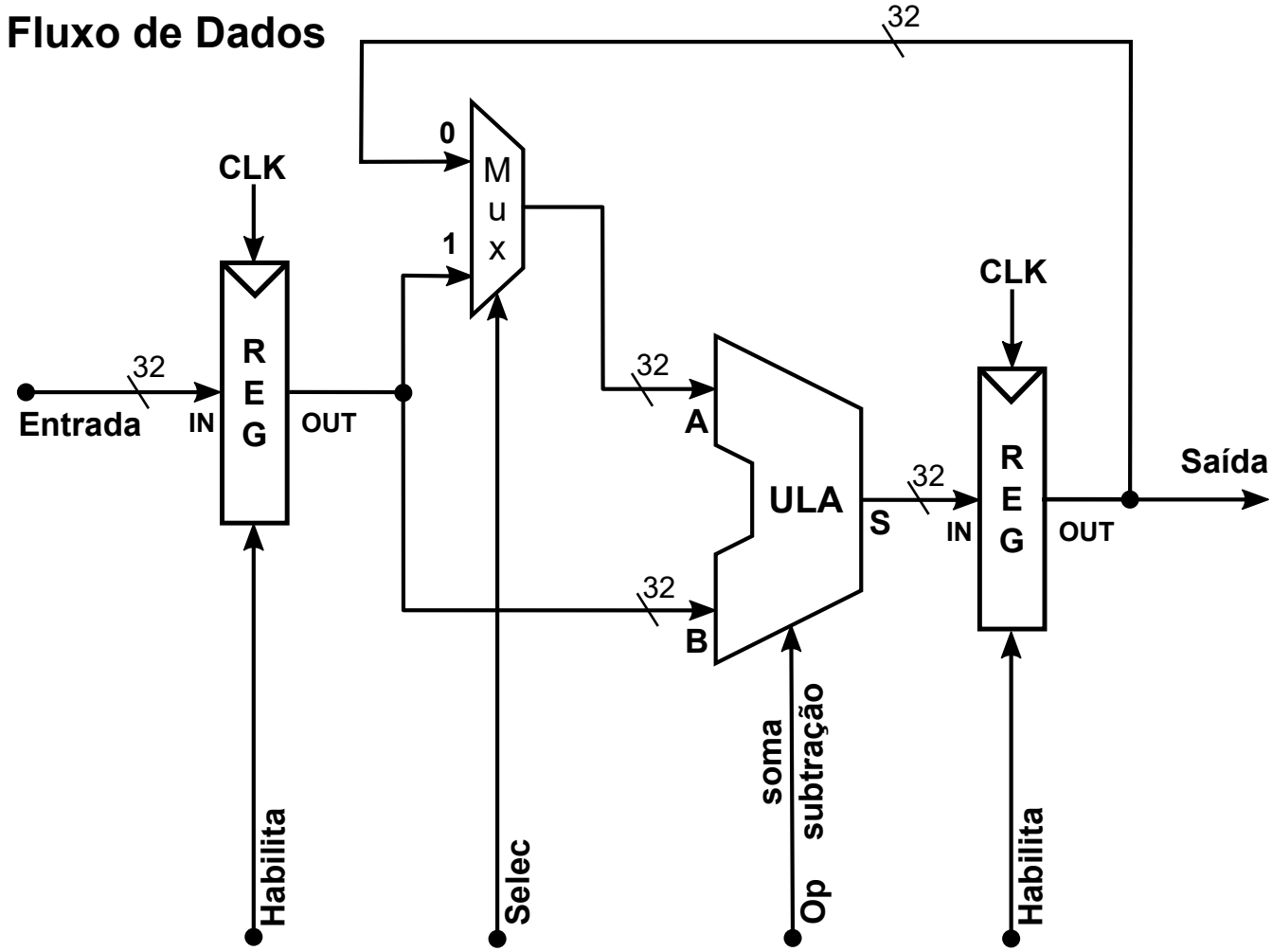
Exercício 1 sobre VHDL

Para o circuito abaixo, faça a codificação em VHDL considerando que:

- Todos os componentes já existem e possuem os nomes indicados;
- Só é necessário fazer o top_level, ou seja, instanciar e conectar os componentes;
- Eles foram criados utilizando `generics`:

Basta instanciar e definir a largura do barramento de dados;

Fluxo de Dados




```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity top_level is
  generic   (
    DATA_WIDTH  : natural :=  32
  );

  port   (
    clk : in  std_logic;
    dataIN   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    enableIn : in  std_logic;
    Selec    : in  std_logic;
    Op       : in  std_logic;
    enableOUT : in  std_logic;
    dataOUT  : out  std_logic_vector(DATA_WIDTH-1 downto 0)
);
end entity;


architecture arch_name of top_level is

  signal regin_ula : std_logic_vector(DATA_WIDTH-1 downto 0);
```

```
  signal mux_ula : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal ula_regout : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal dataOUT_MUX : std_logic_vector(DATA_WIDTH-1 downto 0);

begin

  -- Para instanciar, a atribuição de sinais (e generics) segue a ordem:
(nomeSinalArquivoDefinicaoComponente => nomeSinalNesteArquivo)
  regIN:  entity work.registradorGenerico generic map (larguraDados => DATA_WIDTH)
          port map (DIN => dataIN, DOUT =>  regin_ula, enable =>  enableIn, clk =>
clk, rst => '0');

  MUX :  entity work.muxGenerico2x1 generic map (larguraDados => DATA_WIDTH)
        port map( entradaA_MUX => dataOUT_MUX,
                  entradaB_MUX =>  regin_ula,
                  seletor_MUX => Selec,
                  saida_MUX => mux_ula);

  ULA : entity work.ULASomaSub  generic map(larguraDados => DATA_WIDTH)
          port map (entradaA => mux_ula, entradaB =>  regin_ula, saida =>
ula_regout, seletor => Op);

  regOUT: entity work.registradorGenerico generic map (larguraDados => DATA_WIDTH)
          port map (DIN => ula_regout, DOUT =>  dataOUT_MUX, enable =>  enableOUT,
clk =>  clk, rst => '0');

dataOUT <= dataOUT_MUX;

end architecture;
```

Exercício 1 sobre Endereçamento

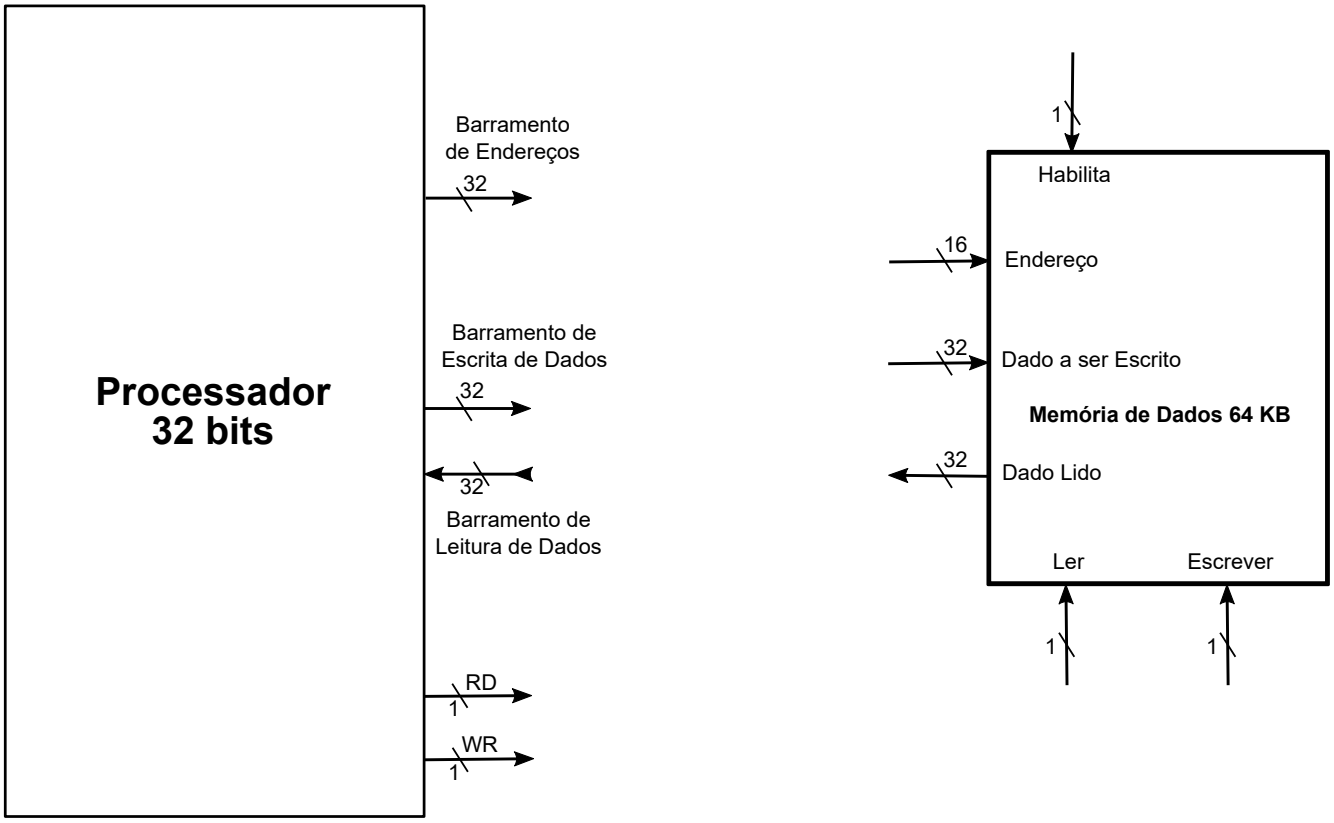
Dado um processador de 32 bits, mostrado a seguir, pede-se:

Fazer o decodificador de endereços que permita: Que a memória esteja somente na faixa de endereços de 0 a 64KB-1.

Que a escrita na posição de memória 128K ative 8 Leds dependendo do conteúdo escrito. Não se esqueça de colocar um registrador.

Que a leitura da posição de memória 128K+1 obtenha o estado de 8 chaves. Utilize uma porta tristate com 8 entradas. Caso não conheça essa porta, faça uma pesquisa.

Mostre as alterações necessárias no Hardware (desenho).



Solução:

A ligação da memória ao circuito do processador é uma conexão entre sinais de mesmo nome/significado.

Assim, teremos:

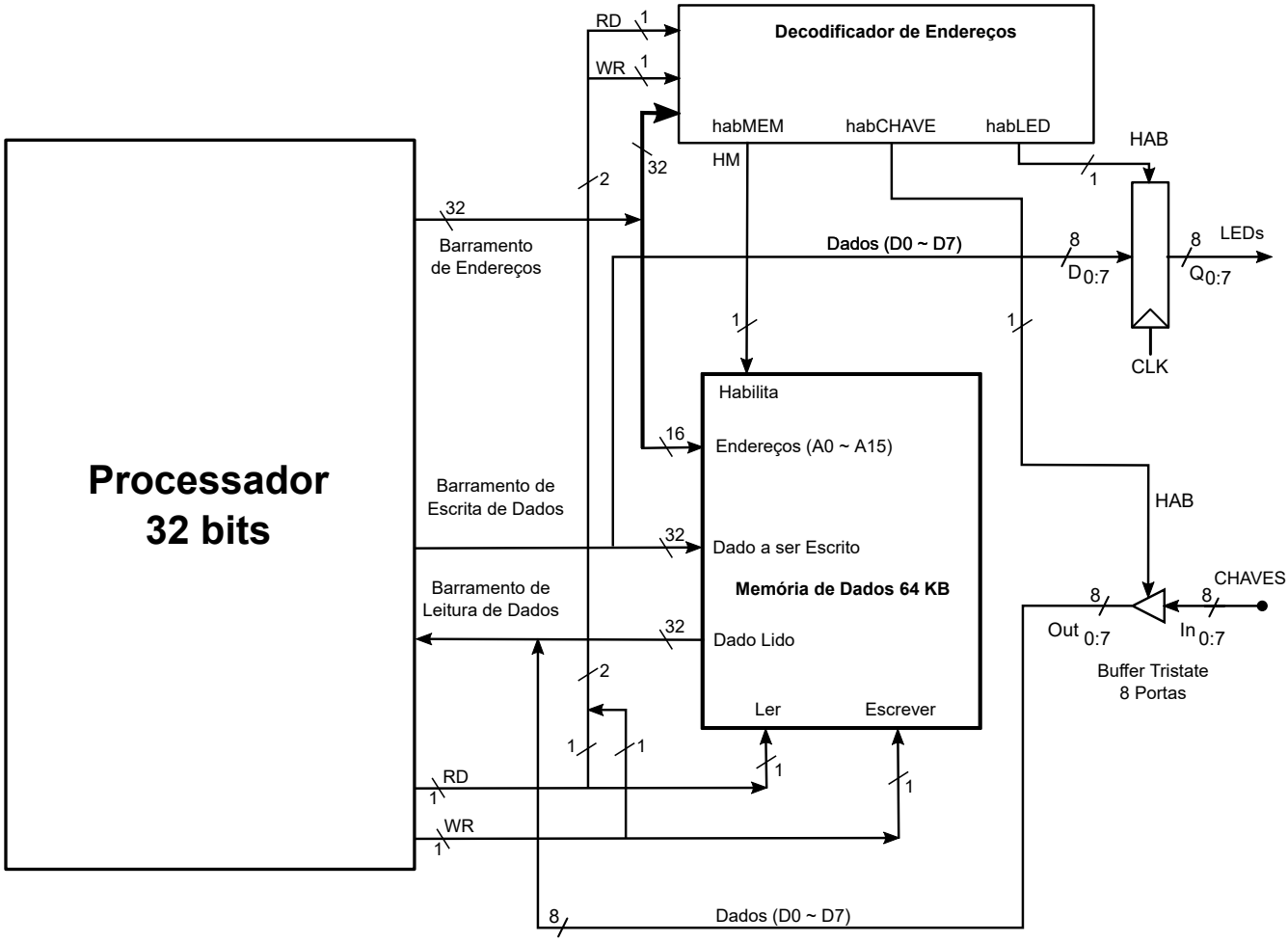
Barramentos de endereços do processador (A0 a A15) => Barramento de endereços da memória, A0 a A15.

Barramento de Escrita de Dados e Leitura de Dados (D0 a D31) => Memória nos respectivos barramentos (D0 a D31).

Sinal de leitura (RD), com 1 bit, conecta na entrada de habilitação de leitura da Memória.

Sinal de leitura (WR), com 1 bit, conecta na entrada de habilitação de escrita da Memória.

Barramentos de endereços do processador (A0 a A31) => Decodificador (A0 a A31).



Acessando a memória RAM.

Para que a memória esteja somente na faixa de endereços de 0 a 64KB-1, precisamos decodificar toda a faixa de endereços, ou seja:

Os 16 bits mais significativos (A16 ~ A31) são decodificados:

Pelo Decodificador de Endereços, gerando a saída HabMEM. Os 16 bits menos significativos (A0 ~ A15) são decodificados:

Dentro da memória, que está habilitada a funcionar se a sua entrada “habilita” estiver em nível alto. O decodificador responsável pela ativação da memória será uma porta NOR com 16 bits de entrada. Os bits mais significativos do endereçamento (A16 ~ A31) são a entrada dessa porta NOR e o sinal habMEM é a sua saída.

Assim, quando todos bits de A16 ~ A31 forem nível baixo, a saída da NOR habilita a memória. Caso qualquer bit dessa faixa (A16 ~ A31) passar para nível alto, a memória será desabilitada. Isso significa que a sua saída passará para tristate (alta impedância de saída) e ela não fará nenhuma operação de escrita ou leitura.

A escrita na posição de memória 128K.

Para ligar ou desligar os LEDs, precisamos escrever no registrador adequado.

Para definir o endereço desse registrador, precisaremos decodificar os 32 bits do endereço. Para tanto, precisamos converter o endereço de decimal para binário e fazer decodificador para esse endereço.

O endereço é 131072 em decimal, que equivale a uma palavra de 32 bits com somente o bit 17 em nível alto.

Como o registrador só deve ser habilitado na escrita, adicionaremos o sinal WR. Podemos implementar esse decodificador com uma porta NOR de 33 bits.

A leitura da posição de memória 128K+1.

Para obter o endereço em binário da posição 128K+1 ativamos o bit 0 do endereço obtido no item anterior.

Da mesma forma, usamos uma porta NOR de 33 bits.