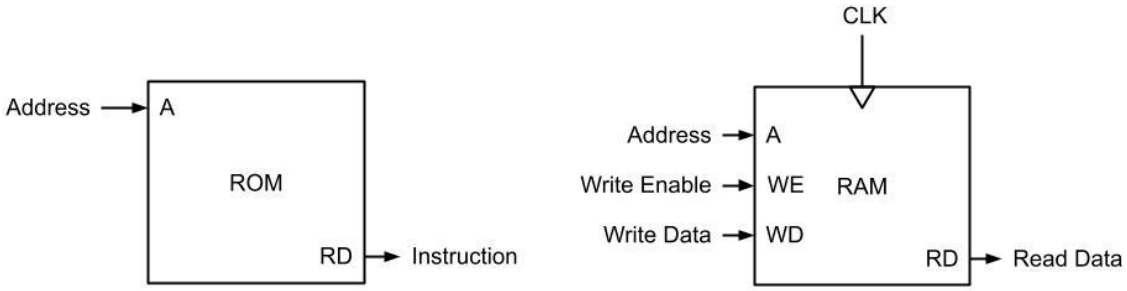
ORGANIZAÇÃO DAS MEMÓRIAS

O processador myRISCVv1 é baseado na arquitetura Harvard, ou seja, ele usa memórias separadas para instruções e dados. A Figura 3 mostra o diagrama de blocos correspondente às memórias de instrução (*Read Only Memory - ROM*) e dados (*Random Access Memory - RAM*).

Figura 3: Diagrama de Blocos da Memória ROM



MEMÓRIA DE INSTRUÇÃO

A memória de instruções atua para armazenar o programa, salvando todas as instruções correspondentes. Essa memória é somente leitura (ROM), o que significa que ela só pode ser acessada para leitura, não para escrita. É preciso uma entrada de endereço (address) para recuperar a instrução correspondente, que é representada com uma largura de bits predefinida de 32 bits, mas a memória só possui 64 KB. A saída é uma instrução de 32 bits.

O processador myRISCVv1, como o RISC-V, organiza a memória de instruções em bytes. No entanto, as próprias instruções são normalmente alinhadas aos limites de palavras (4 bytes para instruções de 32 bits). Uma leitura só acontece quando a entrada (address) é modificada. O código da Figura 4 mostra a descrição de uma memória ROM em VHDL.

MEMÓRIA DE DADOS

A memória de dados é a memória endereçável por byte em um sistema computacional que é usada para armazenar dados durante a execução do programa. Esta memória é comumente chamada de *Random Access Memory* (RAM), que permite operações de leitura e escrita. A memória RAM tem as seguintes entradas: clk (sinal do relógio), we (sinal de ativação de gravação), o endereço (a) para a operação de leitura ou gravação e os dados a serem gravados (wd). A saída consiste nos dados que estão sendo lidos (rd). Semelhante à memória de instruções, a largura do endereço e dos dados de entrada e saída são de 32 bits, mas a memória só possui 64 KB, de acordo com a Arquitetura de Conjunto de Instruções RISC-V (ISA) de 32 bits.

Figura 4: Descrição VHDL da Memória ROM

```
--------------------------------------------------------------------------------
-- A simple read only memory de 64KBx8bits                                    --
-- myRISCVv1                                                                  --
--                                                                            --
-- Prof. Max Santana  (2025)                                                  --
-- CEComp/Univasf                                                             --
--------------------------------------------------------------------------------

library ieee ;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
use std.textio.all;
use ieee.std_logic_textio.all;

entity rom is
  generic(
    DATA_WIDTH   : integer := 8;
    ADDRESS_WIDTH: integer := 16
    DEPTH        : integer := 65535
  );
  port(
    a : in std_logic_vector(31 downto 0);
    rd: out std_logic_vector(31 downto 0)
  );
end entity rom;

architecture behavior of rom is
  -- use array to define the bunch of internal temporary signals
  type rom_type is array (0 to DEPTH-1) of std_logic_vector(DATA_WIDTH-1 downto 0);
  signal imem : rom_type;
begin
  InitMem: process
    use std.textio.all;
    file f: text open read_mode is "myRISCV_example1.bin";
    variable l: line;
    variable value: std_logic_vector(DATA_WIDTH-1 downto 0);
    variable i: integer := 0;
  begin
    while not endfile(f) loop
      readline(f, l);
      read(l, value);
      imem(i) <= value;
      i:= i+1;
    end loop;
    wait;
  end process InitMem;

  process(a)
  begin
    rd(7 downto 0)   <= imem(to_integer(signed(a)));
    rd(15 downto 8)  <= imem(to_integer(signed(a))+1);
    rd(23 downto 16) <= imem(to_integer(signed(a))+2);
    rd(31 downto 24) <= imem(to_integer(signed(a))+3);
  end process;
end behavior;
```

Esta memória, como a memória de instrução, é organizada em bytes. O myRISCVv1 só suporta leitura de palavras de 32 bits, um pouco diferente do RISC-V que suporta endereçamento para bytes, meias palavras e outros tipos de dados. As palavras de dados são normalmente alinhadas aos limites das palavras, o que significa que ocupam 4 bytes para palavras de 32 bits. A memória RAM opera de forma síncrona, lendo dados na borda do relógio, enquanto a escrita ocorre quando o sinal de ativação de gravação (we) é ativado. O código da Figura 5 mostra a descrição de uma memória RAM em VHDL, como também a estrutura do tipo pilha que permite a chamada de rotinas de forma recursiva.

Figura 5: Descrição VHDL da Memória RAM

```
-----------------------------------------------------------------------------------------
-- A simple Random Access Memory de 64KBx8bits                                         --
-- myRISCVv1                                                                           --
--                                                                                     --
-- Prof. Max Santana  (2025)                                                           --
-- CEComp/Univasf                                                                      --
-----------------------------------------------------------------------------------------


------------- $sp (0xFF)
-- dynamic --
-- data    --
-------------     (0x7F)
-- Global  --
-- data    --
------------- $gp (0x00)

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
use std.textio.all;
use ieee.std_logic_textio.all;

entity ram is
  generic(
    DATA_WIDTH      : integer := 8;
    ADDRESS_WIDTH   : integer := 16;
    DEPTH           : integer := 65535;
  );
  port (
       clk: in std_logic;
       a  : in std_logic_vector(31 downto 0);
       wd : in std_logic_vector(31 downto 0);
       we : in std_logic; -- write when 1, read when 0
       rd : out std_logic_vector(31 downto 0)
       );
end entity;

architecture behavior of ram is
  type ram_type is array (0 to DEPTH-1) of std_logic_vector(DATA_WIDTH-1 downto 0);
  signal ram : ram_type;
begin
  process(clk, a, we)
  begin
    if (falling_edge(clk) and we = '1') then
        ram(to_integer(unsigned(a))) <= wd(7 downto 0);
        ram(to_integer(unsigned(a))+1) <= wd(15 downto 8);
        ram(to_integer(unsigned(a))+2) <= wd(23 downto 16);
        ram(to_integer(unsigned(a))+3) <= wd(31 downto 24);
    elsif (falling_edge(clk) and we = '0') then
        rd(7 downto 0)   <= ram(to_integer(unsigned(a)));
        rd(15 downto 8)  <= ram(to_integer(unsigned(a))+1);
        rd(23 downto 16) <= ram(to_integer(unsigned(a))+2);
        rd(31 downto 24) <= ram(to_integer(unsigned(a))+3);
    end if;
  end process;
end behavior;
```

Para atingir o objetivo de implementação de um processador funcional, a próxima etapa do projeto é definir a estrutura dos registradores de propósitos gerais para armazenamento temporário, para em seguida definir a estrutura e os blocos funcionais que compõem o processador.