Gerador de Imediatos

Dérick Daniel S. de Andrade, 23/1003522

Dep. Ciência da Computação - Universidade de Brasília (UnB)

CIC0099 - Organização e Arquitetura de Computadores

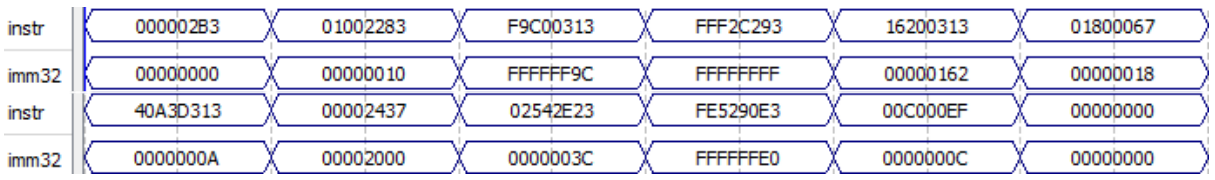
Implementação

Para implementar o gerador, utilizei as definições do roteiro e criei um sinal “a” do tipo std_logic_vector que vou usar como variável e manipular os bits e depois converter ele para integer e depois para signed. Usei um switch-case para verificar o opcode, e no caso do tipo I, usei um if-else para verificar se o opcode era 001 ou 101 para atribuir só os 5 primeiros bits, ou atribuir os 12 bits para o imediato; em todos os casos, fiz a extensão de sinal atribuindo o bit 31 a todos os bits que não foram atribuídos valores (others => instr(31)). Caso o opcode não for do tipo I, S, SB, U ou UJ, o default do switch-case atribui 0 a todos os bits do imediato.

Utilizei o Quartus II para fazer as simulações e depois fiz o testbench no EDA Playground.

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity genImm32 is
   port (
      instr : in std_logic_vector(31 downto 0);
      imm32 : out signed(31 downto 0)
   );
end genImm32;

architecture arquiteturaGerador of genImm32 is

   signal a: std_logic_vector(31 downto 0);

   begin
   process(instr)
      begin
      case instr(6 downto 0) is
         when "0000011" | "0010011" | "1100111" => -- Tipo I opcodes 0x03, 0x13 e 0x
            if ((instr(14 downto 12) = "101") or (instr(14 downto 12) = "001")) then
               a(4 downto 0) <= instr(24 downto 20);
               a(31 downto 5) <= (others => '0');

            else -- Todos os outros casos de tipo I
               a(11 downto 0) <= instr(31 downto 20);
               a(31 downto 12) <= (others => instr(31));
               a(31) <= instr(31);
            end if;

         when "0110111" => -- Tipo U
            a(31 downto 12) <= instr(31 downto 12);
            a(11 downto 0) <= (others => '0');

         when "0100011" => -- Tipo S
            a(11 downto 5) <= instr(31 downto 25);
            a(4 downto 0) <= instr(11 downto 7);
            a(31 downto 12) <= (others => instr(31));

         when "1101111" => -- Tipo UJ
            a(20) <= instr(31);
            a(19 downto 12) <= instr(19 downto 12);
            a(11) <= instr(20);
            a(10 downto 1) <= instr(30 downto 21);
            a(0) <= '0';
            a(31 downto 21) <= (others => instr(31));
         when "1100011" => -- Tipo SB
         a(12) <= instr(31);
         a(11) <= instr(7);
         a(10 downto 5) <= instr(30 downto 25);
         a(4 downto 1) <= instr(11 downto 8);
         a(31 downto 13) <= (others => instr(31));

         when others =>
            a <= (others => '0');
      end case;
   imm32 <= to_signed(to_integer(signed(a)), 32);
   end process;
end arquiteturaGerador;
```

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity tb_genImm32 is
end tb_genImm32;

architecture sim of tb_genImm32 is

  signal instr : std_logic_vector(31 downto 0);
  signal imm32 : signed(31 downto 0);

  component genImm32
    port (
      instr  : in std_logic_vector(31 downto 0);
      imm32  : out signed(31 downto 0)
    );
  end component;

begin
  UUT: genImm32
    port map (
      instr  => instr,
      imm32  => imm32
    );

  stimulus: process
  begin

    instr <= x"000002b3";
    wait for 10 ns;

    instr <= x"01002283";
    wait for 10 ns;

    instr <= x"f9c00313";
    wait for 10 ns;

    instr <= x"fff2c293";
    wait for 10 ns;

    instr <= x"16200313";
    wait for 10 ns;

    instr <= x"01800067";
    wait for 10 ns;
    instr <= x"40a3d313";
    wait for 10 ns;

    instr <= x"00002437";
    wait for 10 ns;

    instr <= x"02542e23";
    wait for 10 ns;

    instr <= x"fe5290e3";
    wait for 10 ns;

    instr <= x"00c000ef";
    wait for 10 ns;

    wait;
  end process;
```



Qual a razão do embaralhamento dos bits do imediato no Risc-V?

Os bits são embaralhados para a otimização do hardware. Por exemplo, o bit mais significativo do imediato sempre vai ser o bit mais significativo da instrução, então a extensão de sinal pode ser feita a partir do mesmo bit sempre. Os campos que definem os registradores sempre são os mesmos, então é possível acessar os registradores antes mesmo de saber qual a instrução a ser executada. Por isso, nas instruções tipo S e SB, o bit mais significativo do imediato é o primeiro, os seguintes possuem os mesmos índices de bit das instruções tipo I e UJ para manter o padrão, mas em vez de continuar a sequência, a posição dos registradores e funct3 possuem prioridade sobre isso; então o imediato continua após o funct3, no campo do registrador de destino, já que essas funções não escrevem em registrador. No caso das instruções UJ e SB, o bit 11, que foi deslocado para manter o padrão dos bits 10 até 5 do imediato nos bits 30 até 25 da instrução, é inserido onde ficaria o bit 0 do imediato, mas como se trata de um endereço de memória sabe-se que esse bit sempre vale 0 e pode ser inserido no valor do imediato depois. No tipo UJ, os bits de 10 até 1 do imediato seguem o padrão do tipo I, tem o bit 11 no lugar do bit 0 por sempre ser par e os bits restantes estão logo na sequência.

Por que alguns imediatos não incluem o bit 0?

Como dito anteriormente, alguns imediatos não possuem o bit 0 por se tratarem de endereços de memória, e portanto, são pares e o bit 0 sempre vale 0 e não há necessidade de ser representado na instrução. Com isso, o alcance dos endereços é dobrado.

Os imediatos de operações lógicas estendem o sinal?

Sim, imediatos de todas as operações lógicas estendem sinal.

Como é implementada a instrução NOT no Risc-V?

A instrução NOT é implementada a partir do XORI entre o registrador e o imediato -1 (0xFFFFFFFF). Ou seja, um XOR entre cada bit do registrador e o valor 1 vai inverter todos os bits.