Trabalho: Geração de Dados Imediatos no RISC-V

Aluno: Adriano Ulrich do Prado Wiedmann
Matrícula: 202014824

Objetivo:

Desenvolver um módulo em VHDL que gere os dados imediatos utilizados nas instruções do processador RISC-V.

Trabalho:

O trabalho foi realizado no ModelSim na versão 20.1.1.

Código:

Este trabalho consiste em dois arquivos *vhd*. O arquivo *genImm32.vhd* e *tbgenImm32.vhd* (*testbench*).

1. *Arquivo genImm32.vhd*

A *entity* utilizada neste arquivo é o mesmo que foi fornecido no PDF do trabalho.
Na *architecture* foi criado o *type FORMAT_RV*, como fornecido no PDF. Além disso, foi criado 2 variáveis para verificar o tipo *I_type**, a variável *inst30* para identificar o bit 30 e a variável *funct3* que recebe os bits 14 ao 12.
Além destas duas variáveis, foi criada o *opcode* para verificar o formato RV. E também, o *instr_format* do tipo *FORMAT_RV* para identificar o tipo da instrução e gerar o imediato. Figura 1.

```
architecture a of genImm32 is
        type FORMAT_RV is (R_type, I_type, S_type, SB_type, UJ_type, U_type);
        signal inst30 : std_logic_vector(0 downto 0);
        signal funct3 : std_logic_vector(2 downto 0);
        signal opcode : unsigned(6 downto 0);
        signal instr_format : FORMAT_RV;
begin
        opcode <= unsigned(instr(6 downto 0)); -- Extrai os 7 primeiros bits menos significativos (opcode)
        inst30 <= instr(30 downto 30); -- Bit 30, para vericação do caso I_type*
        funct3 <= instr(14 downto 12); -- funct3 recebe os bits 14 a 12, para verificação do formato I_type*
```

Figura 1. Variáveis criadas e extração de bits para algumas variáveis.

Neste código, foram utilizados dois *CASEs.* O primeiro *CASE* é para identificar o *opcode* e armazenar o tipo do formato RV na variável *instr_format.* Conforme a Figura 2.
O segundo case identifica o tipo da instrução através de *instr_format* e então gera o imediato. Figura 3.

```
case opcode is

        when "0110011" =>
                instr_format <= R_type;

        when "0000011" | "0010011" | "1100111" =>
                instr_format <= I_type;

        when "0100011" =>
                instr_format <= S_type;

        when "1100011" =>
                instr_format <= SB_type;

        when "1101111" =>
                instr_format <= UJ_type;

        when others =>
                instr_format <= U_type;

end case;
```

Figura 2. Primeiro case.

```
case instr_format is

        when R_type =>
                imm32 <= (others => '0'); -- Atribui zero a todos os bits

        when I_type =>
                if funct3 = "101" and inst30 = "1" then
                        imm32 <= resize(signed(instr(24 downto 20)), 32);
                else
                        imm32 <= resize(signed(instr(31 downto 20)), 32);
                end if;

        when S_type =>
                imm32 <= resize(signed(instr(31 downto 25) & instr(11 downto 7)), 32);

        when SB_type =>
                imm32 <= resize(signed(instr(31) & instr(7) & instr(30 downto 25) & instr(11 downto 8) & "0"), 32);

        when UJ_type =>
                imm32 <= resize(signed(instr(31) & instr(19 downto 12) & instr(20) & instr(30 downto 21) & "0"), 32);

        when others =>
                imm32 <= resize(signed(instr(31 downto 12) & "000000000000"), 32);

end case;
```

Figura 3. Segundo case.

2. *Arquivo tbgenImm32.vhd*

No *testbench* é realizando a associação (*port mapping*) dos sinais entre a unidade de teste (*tbgenImm32*) e a unidade de design (*genImm32*) da seguinte forma:

dut: genImm32 port map (instr => instr_tb, imm32 => imm32_tb);

E para verificar o funcionamento, é atribuído à *instr_tb,* em formato binário, as instruções fornecidas no PDF do trabalho disponibilizado no *Aprender3.* Por exemplo, a instrução ***add t0, zero, zero*** na Figura 4.

```
-- add t0, zero, zero
instr_tb <= "00000000000000000000001010110011";
wait for 5 ns;
assert imm32_tb = "00000000000000000000000000000000"
        report "Formato: R_type"
        severity error;
```

Figura 3. Instrução ***add t0, zero, zero*** em binário.

Testes:

Como dito anteriormente, os testes foram feitos através do que foi fornecido no PDF do trabalho. As instruções em binário.
Para verificar os testes no ModelSim, além de utilizar o *Compile All,* inicia-se a simulação através do *Start Simulation*. E selecionando em *work* o *tbgenImm32* (Figura 5) e clicando em OK.

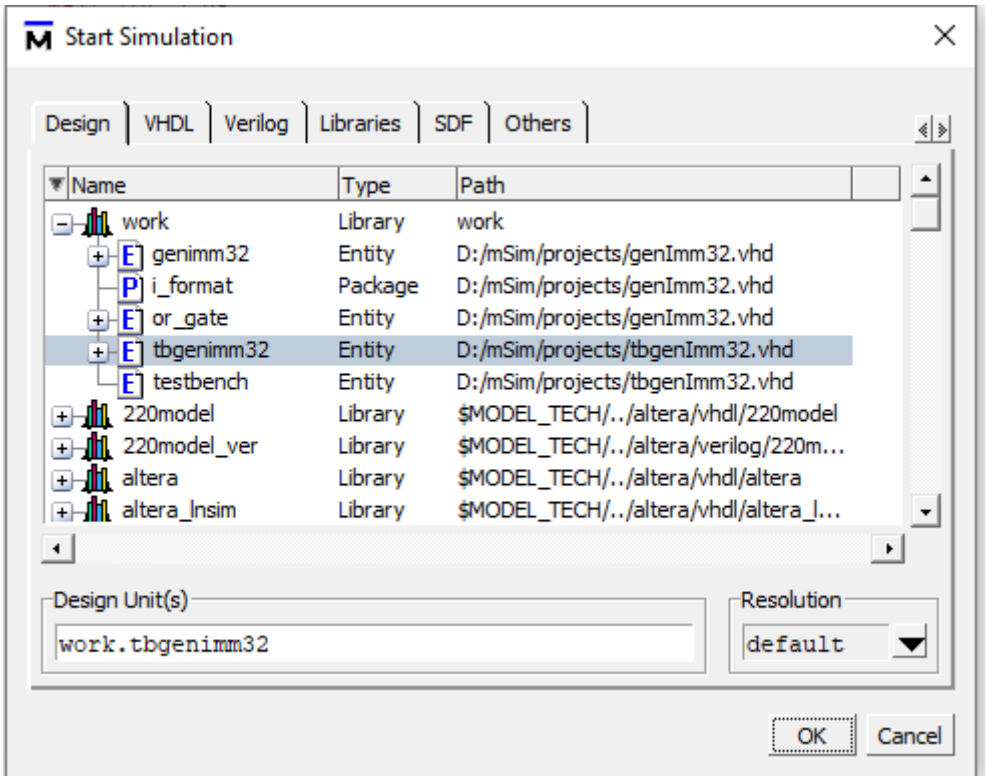


Figura 5. Start Simulation.

Seleciona-se *Add Wave* ao clicar com o botão direito do mouse em cima do *dut*. Figura 6

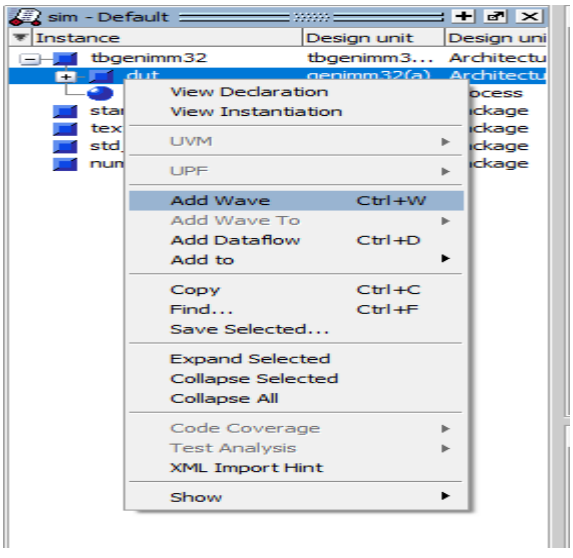
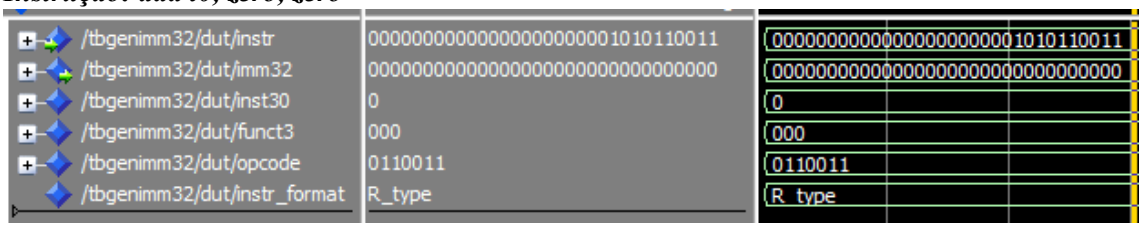


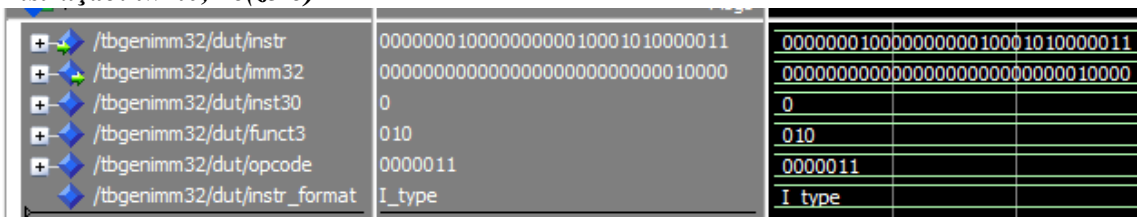
Figura 6. Add Wave.

Antes de rodar a simulação, define como *Run Lenght* igual a 55 ns. Ao rodar a simulação é possível verificar os testes.
Portanto, ao utilizar as instruções do RISC-V fornecidas, é possível verificar que o funcionamento do módulo está correto. A seguir, são fornecidos os resultados.

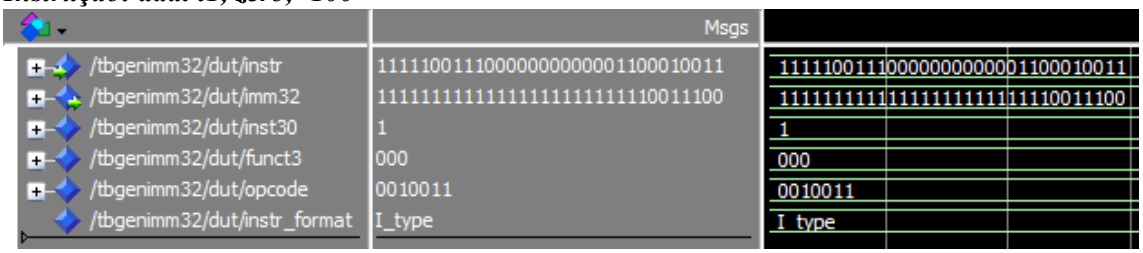
Instrução: add t0, zero, zero



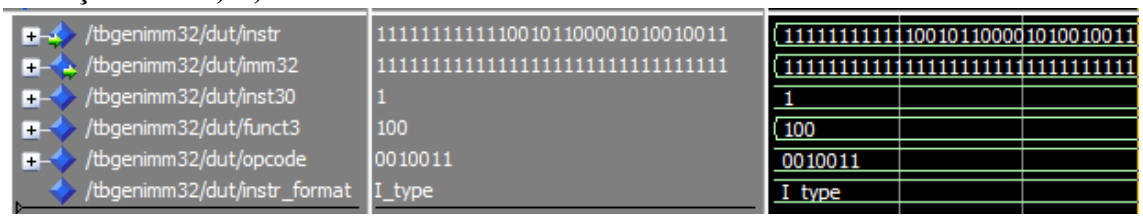
Instrução: lw t0, 16(zero)



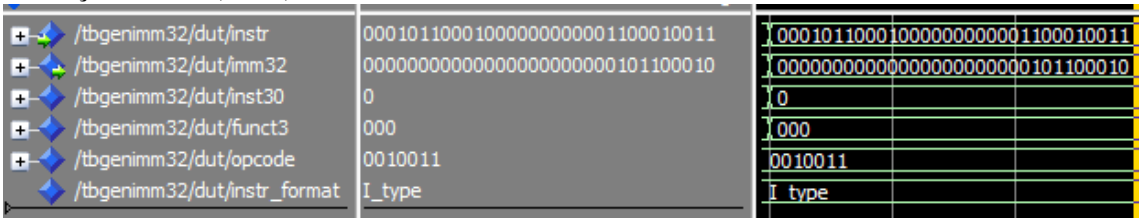
Instrução: addi t1, zero, -100



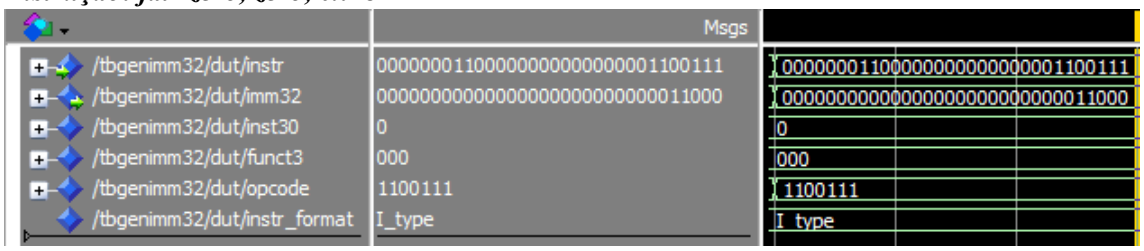
Instrução: xori t0, t0, -1



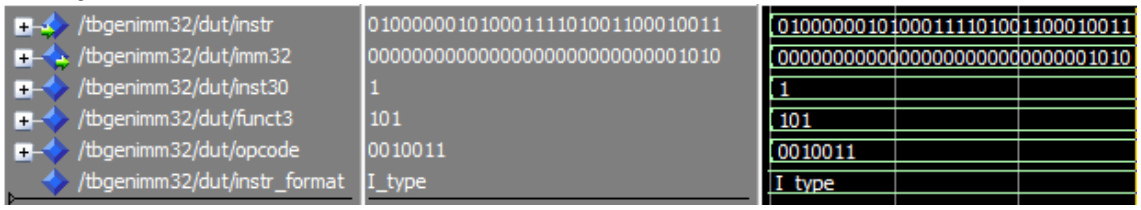
Instrução: addi t1, zero, 354



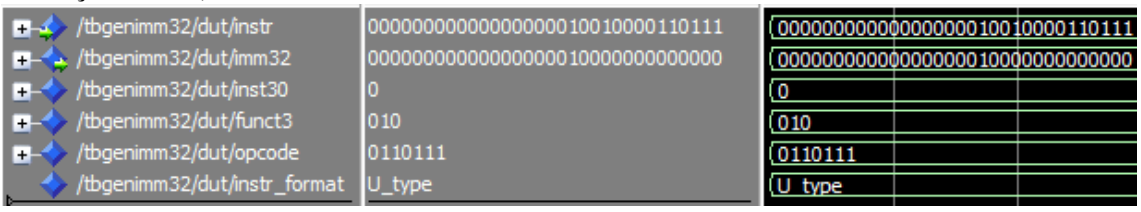
Instrução: jalr zero, zero, 0x18



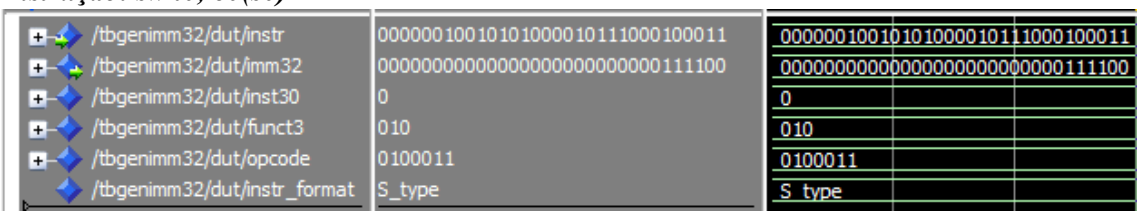
Instrução: srai t1, t2, 10



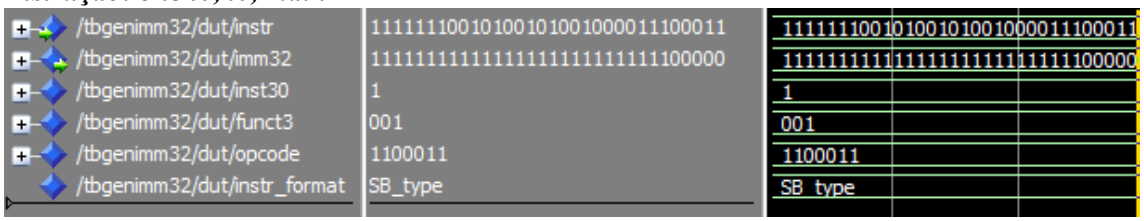
Instrução: lui s0, 2



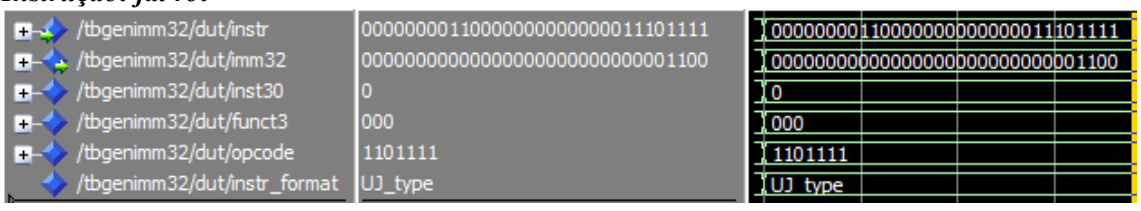
Instrução: sw t0, 60(s0)



Instrução: bne t0, t0, main



Instrução: jal rot



Conclusão:

Inicialmente, a intenção era desenvolver o módulo em VHDL com a utilização de apenas um *CASE,* mesmo sem ter noção de que funcionaria. Mas como foi fornecido a identificação dos formatos no arquivo PDF do trabalho, optou-se por implementar de dois CASEs. Isso proporcionou uma clara visualização do formato a ser utilizado com base no *Opcode* extraído da instrução.

Com a elaboração deste trabalho foi possível entender, com clareza, o funcionamento do gerador de imediatos de 32 bits do RISC-V.

Qual a razão do embaralhamento dos bits do imediato no RiscV?

Resposta: Os bits imediatos são embaralhados com o objetivo de diminuir o custo associado à decodificação do valor imediato. Isso resulta na redução do número de opções disponíveis para cada bit imediato de saída.

Por que alguns imediatos não incluem o bit 0?

Resposta: A decisão de não incluir o bit 0 em alguns imediatos pode estar relacionada à otimização do espaço de codificação e à simplificação do hardware. Em algumas instruções, o bit 0 pode ser considerado redundante ou ter um significado específico, o que permite economizar espaço na representação da instrução.

Os imediatos de operações lógicas estendem o sinal?

Resposta: Sim, o imediato é estendido para o tamanho do registrador antes da operação lógica.

Como é implementada a instrução NOT no RiscV?

Resposta: Para implementar a instrução NOT, pode-se utilizar a instrução XORI com o valor imediato de -1. Ou seja, XOR com -1 é equivalente a inverter todos os bits.
Por exemplo: ***xori rd, rs1, -1***