Universidade de Brasília

Trabalho 3: Geração de Dados Imediatos no RISC-V

Universidade de Brasília
Departamento de Ciência da Computação
Organização e Arquitetura de Computadores
Evelyn Soares Pereira
17/0102785
soares.evelynp@gmail.com

1. DESCRIÇÃO DO TRABALHO

Este trabalho consiste na implementação de um módulo em VHDL para gerar dados imediatos usados nas instruções do processador RISC-V. A arquitetura do RISC-V utiliza diversos formatos de instrução, como R, I, S, SB, U e UJ, cada um com diferentes campos que influenciam na geração dos dados imediatos. O objetivo deste trabalho é criar um componente capaz de identificar o formato da instrução de entrada e gerar o dado imediato correspondente.

O código em VHDL consiste na declaração da entidade genImm32 com uma entrada instr de 32 bits e uma saída imm32 de 32 bits representando a instrução, de entrada, e o dado imediato, de saída, respectivamente. É dado inicio a arquitetura com o processo senível a mudança da entrada, então é definido uma indentificação do formato da instrução com "case" com base nos bits 6 a 0 da entrada. Para instruções do tipo I o dado imediato é gerado a partir dos bits 31 a 20, com extensão de sinal. Para o tipo S, o dado imediato é formado pelos bits 31 a 25 e 11 a 7 da instrução, com extensão de sinal. Para o formato SB-type, o dado imediato é composto pelos bits específicos da instrução, com extensão de sinal, como mostra a figura 1. Para o formato U-type, o dado imediato é formado pelos bits 31 a 12 da instrução, com extensão de sinal. Para outros formatos não mencionados, considera-se o formato I type*, gerando o dado imediato de acordo com as regras específicas. O código utiliza operadores VHDL como para agregação de bits e a função resize para realizar a extensão de sinal quando necessário. A lógica do código está estruturada para lidar com os diferentes formatos de instrução definidos pela arquitetura RISC-V.

2. TESTES REALIZADOS

Para verificar o funcionamento do módulo, foram realizados testes utilizando instruções representativas do conjunto de instruções RISC-V, as intruções usadas foram as disponíveis na especificação do trabalho, como está na figura 3. Cada teste envolve uma instrução específica com um formato correspondente. Os imediatos gerados foram comparados com os valores esperados, considerando as regras definidas para cada formato de instrução.

3. PERGUNTAS DO RELATÓRIO

Embaralhamento dos bits do imediato no RISC-V: O embaralhamento dos bits no imediato em instruções RISC-V ocorre para simplificar a decodificação das instruções e melhorar a eficiência do hardware. Os bits são agrupados de maneira específica para facilitar a extração de campos importantes, como opcode, registradores, e imediatos, durante a fase de decodificação.

Exclusão do bit 0 em alguns imediatos: A exclusão do bit 0 em alguns imediatos pode ocorrer devido à natureza das instruções e ao formato dos campos imediatos. Em algumas instruções, o bit 0 pode ser predefinido como zero ou não ser relevante para a operação em questão. A exclusão desse bit pode economizar espaço e simplificar a implementação do hardware.

Extensão de sinal em imediatos de operações lógicas: Em geral, os imediatos de operações lógicas no RISC-V seguem a extensão de sinal. Isso significa que o bit mais significativo (MSB) do imediato é replicado para os bits mais significativos adicionados durante a extensão. Isso garante que a operação lógica seja aplicada corretamente, mantendo a semântica da instrução.

Implementação da instrução NOT no RISC-V: A arquitetura RISC-V não possui uma instrução NOT dedicada. Em vez disso, a negação lógica (NOT) pode ser realizada usando uma instrução XOR com um operando imediato contendo todos os bits definidos como 1. A lógica é que XORing cada bit com 1 inverte o valor do bit. Por exemplo, para inverter o valor de um registrador x, você pode usar:

XORI x, x, -1

Dado um registrador x, nesta instrução, o imediato -1 terá todos os bits definidos como 1, efetivamente invertendo todos os bits do registrador x.

Format	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R-type	funct7							rs2					rs1					funct3			rd					opcode							
I-type	11	10	9	8	7	6	5	4	3	2	1	0	rs1					funct3			rd					opcode							
I-type*		1						4	3	2	1	0	rs1					funct3			rd					opcode							
S-type	11	10	9	8	7	6	5	rs2					rs1					funct3			4	3	2	1	0	opcode							
SB-type	12	10	9	8	7	6	5	rs2					rs1					funct3			4	3	2	1	11	opcode							
UJ-type	20	10	9	8	7	6	5	4	3	2	1	11	19	18	17	16	15	14	13	12	rd					opcode							
U-type	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	rd					opcode							

Figure 1. Tabela de formatos de instruções do RISC-V.

OAC_T3.vhd | tb_OAC_T3.vhd*

```
        imm32 : out signed(31 downto 0)
    );
end genImm32;

architecture Behavioral of genImm32 is
begin
    process(instr)
    begin
        -- Identificação do formato
        case instr(6 downto 0) is
            when "0110011" =>
                -- R-type
                imm32 <= (others => '0');
            when "0000011" =>
                -- I-type
                imm32 <= signed(resize(unsigned(instr(31 downto 20)), 32));
            when "0100011" =>
                -- S-type
                imm32 <= signed(resize(unsigned(instr(31 downto 25) & instr(11 downto 7)), 32));
            when "1100011" =>
                -- SB-type
                imm32 <= signed(resize(unsigned(instr(31) & instr(7) & instr(30 downto 25) & instr(11 downto 8)), 32));
            when "1101111" =>
                -- UJ-type
                imm32 <= signed(resize(unsigned(instr(31) & instr(19 downto 12) & instr(20) & instr(30 downto 21)), 32));
            when "0110111" =>
                -- U-type
                imm32 <= signed(resize(unsigned(instr(31 downto 12)), 32));
            when others =>
                -- I_type*
                imm32 <= signed(resize(unsigned(instr(11 downto 7) & "00000" & instr(31 downto 5)), 32));
        end case;
    end process;
end Behavioral;

```

Figure 2. Código em VHDL.

Instrução RISC-V	Código	Formato	Imediato	Imm hex
add t0, zero, zero	0x000002b3	R-type	inexiste: 0	0x00000000
lw t0, 16(zero)	0x01002283	I-type0	16	0x00000010
addi t1, zero, -100	0xf9c00313	I-type1	-100	0xFFFFFF9C
xori t0, t0, -1	0xfff2c293	I-type1	-1	0xFFFFFFFF
addi t1, zero, 354	0x16200313	I-type1	354	0x00000162
jalr zero, zero, 0x18	0x01800067	I-type2	24	0x00000018
srai t1, t2, 10	0x40a3d313	I-type*	10	0x0000000A
lui s0, 2	0x00002437	U-type	0x2000	0x00002000
sw t0, 60(s0)	0x02542e23	S-type	60	0x0000003C
bne t0, t0, main	0xfe5290e3	SB-type	-32_C	0xFFFFFFE0
jal rot	0x00c000ef	UJ-type	12	0x0000000C

Figure 3. Instruções testadas.

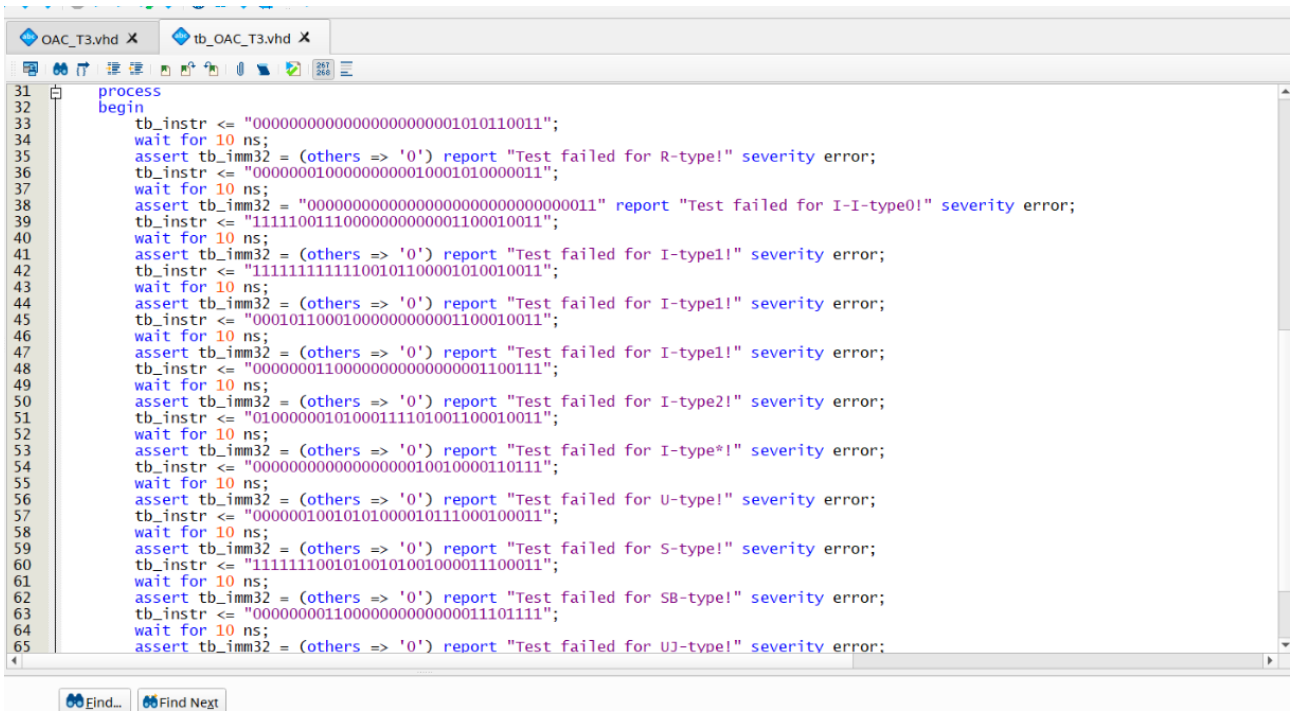

Figure 4. Código do testbench.