Trabalho 2 de OAC
ULA e Gerador de Imediatos

Alunos:
Gabriel Matheus - 17/0103498
Guilherme Braga - 17/0162290
Victor Castor - 17/0115127

Professor: Marcelo Grandi Mandelli
Entrega: 10/10/2019

1- Marco Teórico: ULA e Gerador de Imediatos

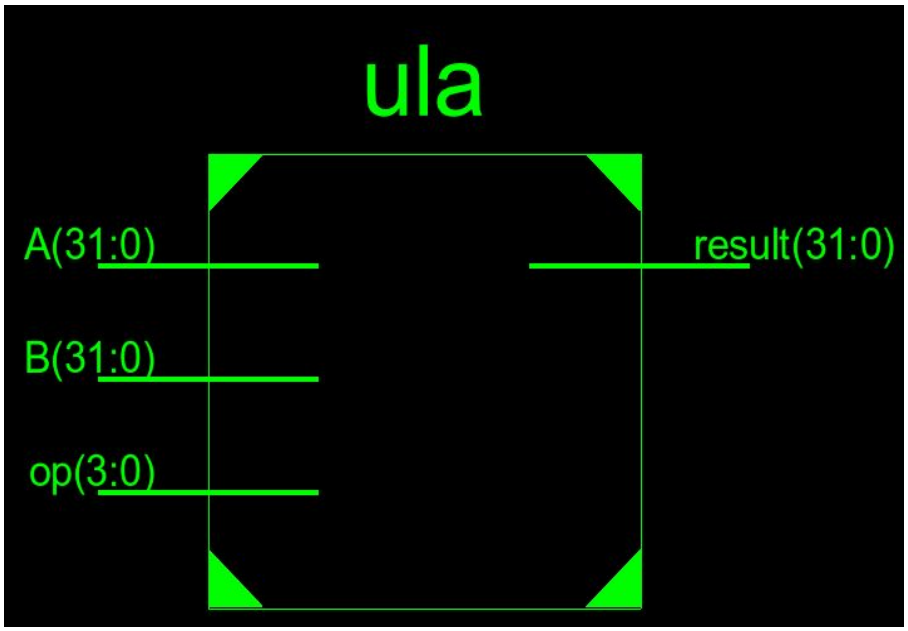
1.1- ULA

A unidade lógica e aritmética (proposta pelo matemático John von Neumann), em inglês Arithmetic Logic Unit (ALU), é um circuito digital que realiza um conjunto predefinido de operações lógicas, dado um input. É considerada a mais simples unidade de processamento central. A ULA aqui proposta segue a seguinte estrutura, utilizando funções prontas da linguagem padrão do VHDL:

Operação	**Significado**	**Op code**
ADD	A + B	0000
SUB	A - B	0001
SLL	A deslocado B bits para a esquerda	0010
SLT	A < B? (com sinal)	0011
SLTU	A < B? (sem sinal)	0100
XOR	A XOR B	0101
SRL	A deslocado B bits para a direita sem sinal	0110

SRA	A deslocado B bits para a direita com sinal	0111
OR	A OR B	1000
AND	A AND B	1001
SEQ	A == B?	1010
SNE	A != B?	1011
SGE	A >= B? (com sinal)	1100
SGEU	A >= B? (sem sinal)	1101

Após a execução do código, teremos um módulo com esta estrutura:



1.2 - Gerador de Imediatos

RISC-V é um conjunto de instruções (ISA) aberto e baseado em RISC ("Reduced Instruction Set Computing"). Para este trabalho iremos considerar a geração de instruções de 32 bits, onde, dependendo da instrução, será necessária a extração de uma constante, o imediato. Ou seja, este módulo em VHDL gerará o imediato para alguns tipos de instruções.

Após a execução em VHDL, teremos um módulo com esta estrutura:

2- Procedimento

2.1- ULA

O código da ULA se concentra em duas partes: a definição de sinais e a designação destes sinais e de operações “inline”. Para operações de comparação, o retorno será uma informação do tipo booleana. A partir desta informação, designamos para a resposta um vetor composto unicamente de zeros (para indicar o sinal negativo) ou um vetor com o 1 em sua posição inicial (para indicar o sinal positivo).

Por sua importância, destacamos o seguinte trecho de código. Este serve para efetivamente designar os sinais desejados caso o opcode seja correspondido:

```
result <= std_logic_vector(signed(A) + signed(B))                          when op="0000" else
          std_logic_vector(signed(A) - signed(B))                          when op="0001" else
          std_logic_vector(shift_left(unsigned(A), to_integer(unsigned(B)))) when op="0010" else
          sinal_A_menor_com_sinal                                          when op="0011" else
          sinal_A_menor_sem_sinal                                          when op="0100" else
          std_logic_vector(unsigned(A) xor unsigned(B))                    when op="0101" else
          std_logic_vector(shift_right(unsigned(A), to_integer(unsigned(B)))) when op="0110" else
          std_logic_vector(shift_right(signed(A), to_integer(unsigned(B)))) when op="0111" else
          std_logic_vector(A or B)                                         when op="1000" else
          std_logic_vector(A and B)                                        when op="1001" else
          sinal_A_igual_B                                                  when op="1010" else
          sinal_A_dif_B                                                    when op="1011" else
          sinal_A_maiorigual_B_com_sinal                                   when op="1100" else
          sinal_A_maiorigual_B_sem_sinal                                   when op="1101" else
          sinal_default;
end ula;
```

2.2- Gerador de Imediatos

Para o gerador de imediatos, separamos a resposta necessária em pré-determinada ordem para poder definir o sinal correto, nos valendo dos recursos de concatenação do VHDL. Sinais auxiliares foram criados para a fácil interpretação do código.

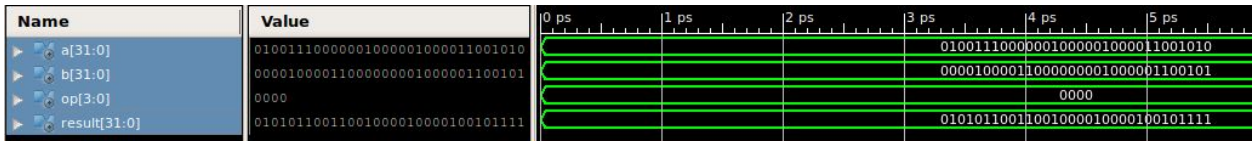
Por sua importância, destacamos o seguinte trecho de código. Este serve para efetivamente designar os sinais desejados caso o opcode seja correspondido:

```
imm32 <= (aux & inst(31 downto 20))                                    when opcode = "0000011" else
      (aux & inst(31 downto 20))                                       when opcode = "0010011" else
      (aux & inst(31 downto 20))                                       when opcode = "1100111" else
      (aux & inst(31 downto 25) & inst(11 downto 7))                   when opcode = "0100011" else
      (aux & inst(7) & inst(30 downto 25) & inst(11 downto 8) & '0')   when opcode = "1100011" else
      (inst(31 downto 12) & "000000000000")                            when opcode = "0010111" else
      (inst(31 downto 12) & "000000000000")                            when opcode = "0110111" else
      (aux_j & inst(19 downto 12) & inst(20) & inst (30 downto 21) & '0')  when opcode = "1101111" else
       default;
```

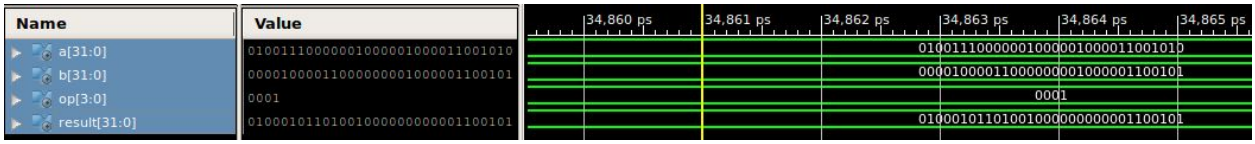
3- Resultados

3.1- Para a ULA:

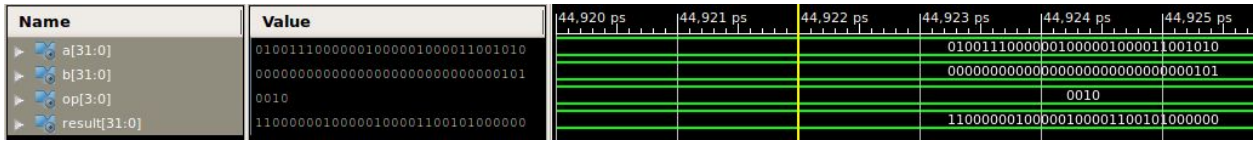
- **A+B**



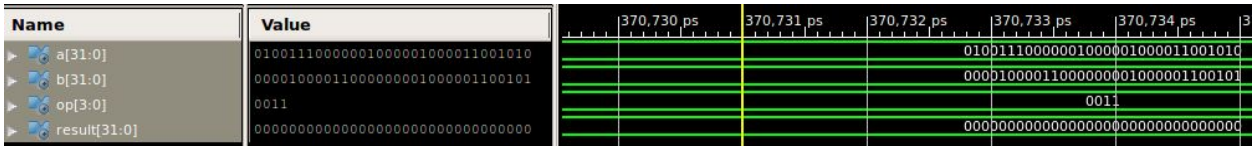
- **A-B**



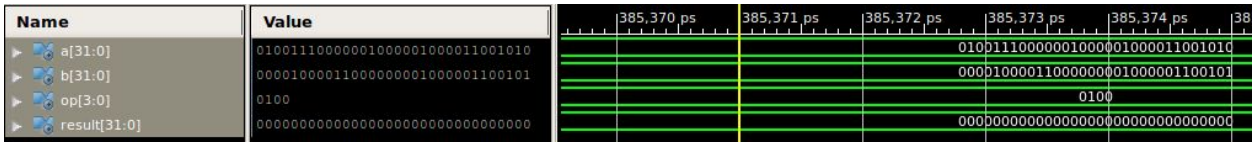
- **SLL (A deslocado à esquerda “B” vezes)**



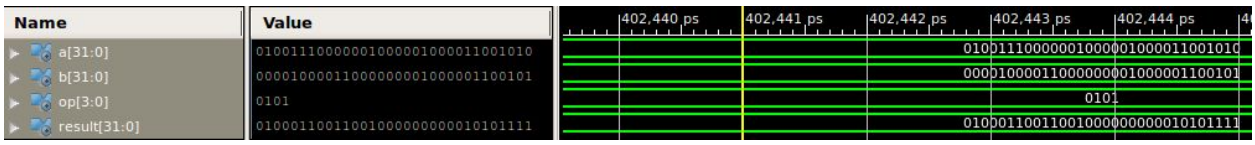
- **SLT (A<B com sinal)**



- **SLTU (A<B sem sinal)**

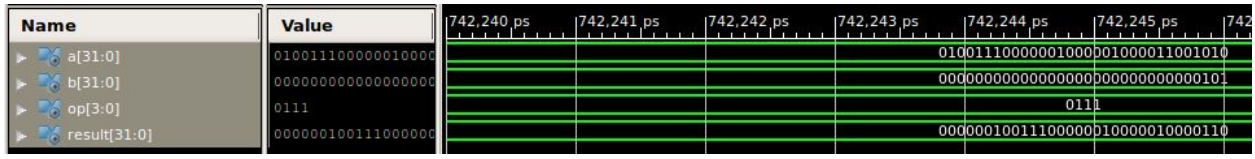


- **A XOR B**

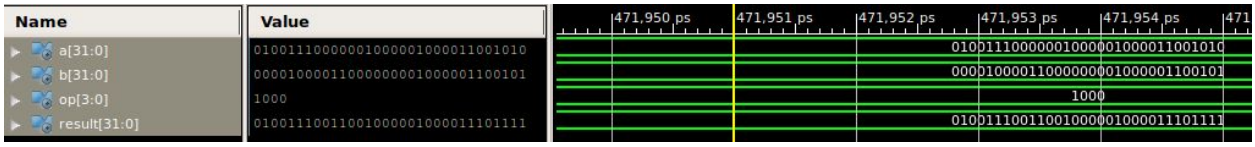


- **SRL (A deslocado à direita “B” vezes sem sinal)**

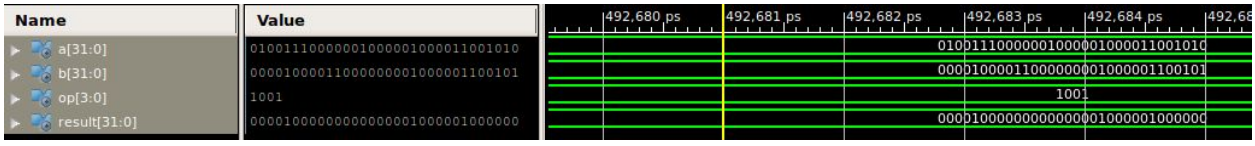
- **SRA (A deslocado à direita “B” vezes com sinal)**



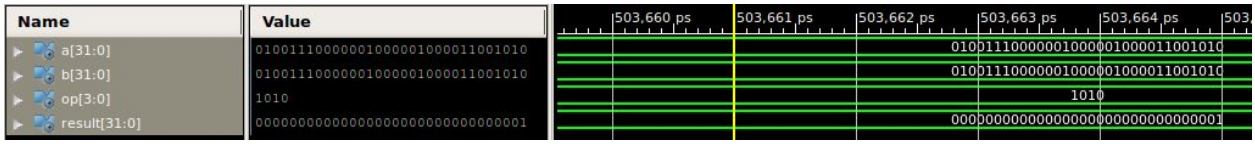
- **A OR B**



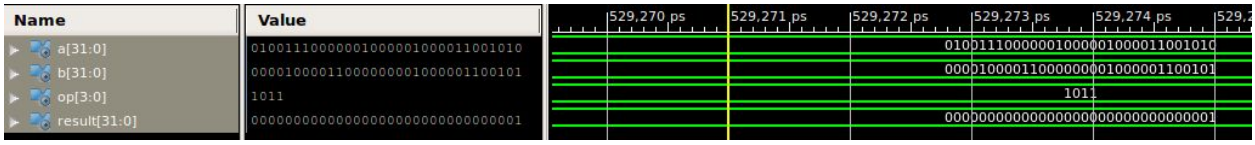
- **A AND B**



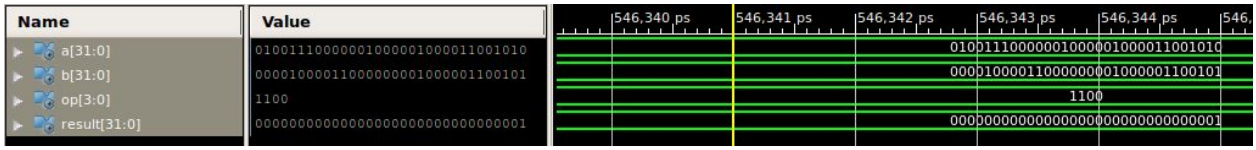
- **SEQ (comparação A == B)**



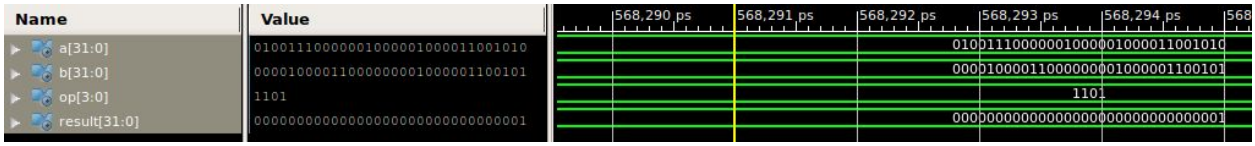
- **SNE (comparação A != B)**



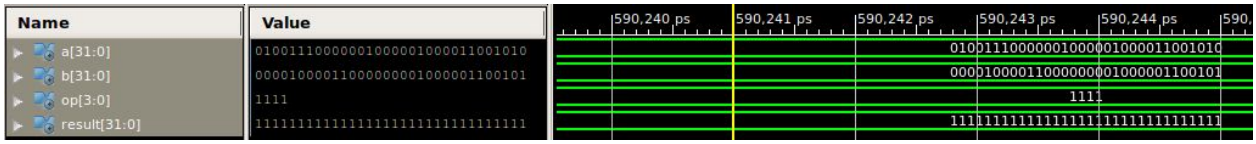
- **SGE (A >= B com sinal)**



- **SGEU (A >= B sem sinal)**



- **SINAL DEFAULT**



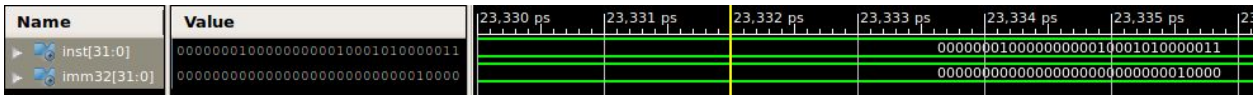
3.2 - Para o gerador de imediatos:

- **Instrução tipo R**

RISC-V: add t0, zero, zero
Instrução: 0x000002B3
Imediato: 0x00000000

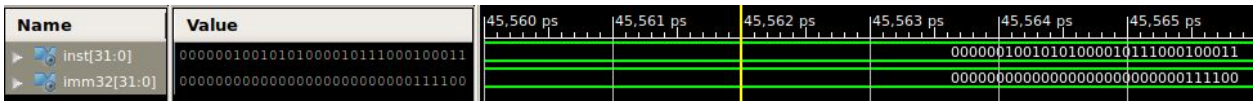
- **Instrução tipo I**

RISC-V: lw t0, 16(zero)
Instrução: 0x01002283
Imediato: 0x00000016



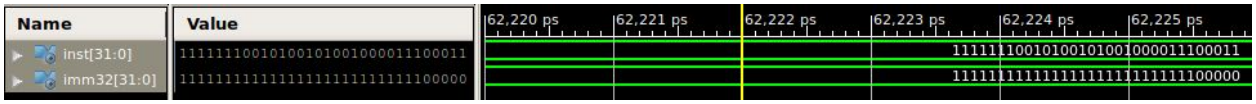
- **Instrução tipo S**

RISC-V: sw t0 60(s0)
Instrução: 0x02542E23
Imediato: 60



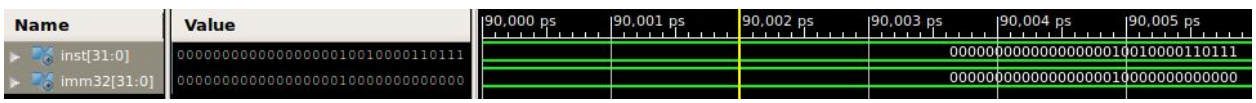
- **Instrução tipo B**

RISC-V: bne t0, t0, main
Instrução: 0xFE5290E3
Imediato: -32



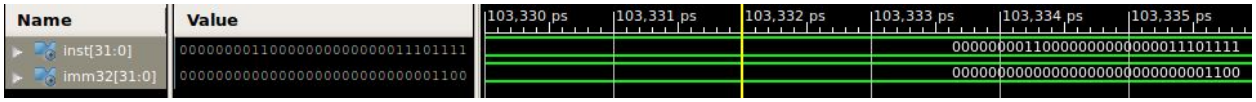
- **Instrução tipo U**

RISC-V: lui s0, 2
Instrução: 0x00002437
Imediato: 0x2000



- **Instrução tipo J**

RISC-V: jal rot
Instrução: 0x00C000EF
Imediato: 12



4- Link para o trabalho no Github:

https://github.com/therealguib545/OAC_Trabalho2