PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE RORAIMA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

RELATÓRIO DO PROJETO: PROCESSADOR GE_22

ALUNOS:
EDUARDO HENRIQUE DE ALMEIDA IZIDORIO – 2020000315
MARCIA GABRIELLE BONIFÁCIO DE OLIVEIRA – 2020011319

Março de 2022
Boa Vista/Roraima

PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE RORAIMA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

RELATÓRIO DO PROJETO: PROCESSADOR GE_22

Março de 2022
Boa Vista/Roraima

Resumo

O projeto aborda a elaboração e implementação do processador GE_22 uniciclo de 8 bits baseado na arquitetura do processador MIPS, será feita a descrição de todos os componentes necessários para o funcionamento dele. O processador em questão tem a capacidade de executar 13 instruções, dentre elas se encontra instruções de comparação, salto na memória e aritméticas, com as instruções disponíveis no processador é possível executar diversas operações distintas. Toda a parte de implementação foi feita utilizando a linguagem VHDL e o software Quartus Prime Lite, e a parte de testes dos componentes foi feita utilizando o software ModelSim Altera através das waveforms, demonstrando assim todas as funcionalidades de cada componente.

Palavras-chave: MIPS. Instruções. Quartus Prime Lite.

Conteúdo

Lista de Figuras

Lista de Tabelas

1. Especificação

Nesta seção é apresentado o conjunto de itens para o desenvolvimento do processador GE_22, bem como a descrição detalhada de cada etapa da construção do processador.

1.1 Plataforma de desenvolvimento

Flow Status	Successful - Sun Mar 13 16:00:52 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	processador_GE_22
Top-level Entity Name	processador_GE_22
Family	MAX 10
Device	10M08DAF484C8G
Timing Models	Final
Total logic elements	480
Total registers	80
Total pins	78
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	0
Total PLLs	0
UFM blocks	0
ADC blocks	0

Figura 1 – Especificações no Quartus

Para a implementação do processador GE_22 foi utilizado a IDE: Quartus Prime Lite, versão 18.1.0 e o simulador ModelSim Altera.

1.2 Conjunto de instruções

O processador GE_22 possui 4 registradores: S0 E S1. Assim como 3 formatos de instruções de 8 bits cada, instruções do tipo R, I e J, que seguem algumas considerações sobre as estruturas contidas nas instruções:

- **Opcode:** é destinado para identificação do código de operação que será realizada, tradicionalmente chamado de código de operação;
- **Reg1:** o registrador contendo o primeiro operando fonte e adicionalmente para alguns tipos de instruções (ex. instruções do tipo R) é o registrador de destino;
- **Reg2:** o registrador contendo o segundo operando fonte.

1.2.1 Tipo de Instruções

- **Tipo R:** este formato aborda as instruções baseadas em operações aritméticas e lógicas, soma, subtração, multiplicação.

Formato para escrita de código de alto nível:

Tipo da Instrução	Reg1	Reg2

Formato para escrita em código binário:

Opcode	**Reg1**	**Reg2**
4 bits	**2 bits**	**2 bits**
7-4	**3-2**	**1-0**

- **Tipo I:** este formato aborda as instruções baseadas em operações com valores imediatos, desvios condicionais e operações relacionadas à memória, soma e subtração imediata, BEQ, BNE, Load e Store.

Formato para escrita de código em alto nível:

Tipo da Instrução	Reg1	Reg2

Formato para escrita em código binário:

Opcode	**Reg1**	**Imediato**
4 bits	**2 bits**	**2 bits**
7-4	**3-2**	**1-0**

- **Tipo J:** este formato aborda as instruções de desvios incondicionais, exemplo o Jump.

Formato para escrita de código em alto nível:

Tipo de Instrução	Endereço

Formato para escrita em código binário:

Opcode	Endereço
4 bits	**4 bits**
7-4	**3-0**

Visão geral das instruções do processador GE_22:

O número de bits do campo **Opcode** das instruções é quatro, assim, obtemos um máximo ($Bit(0e1)^4 \therefore 2^4 = 16$) de 16 Opcodes (0 a 15) que podem ser implementados. A Tabela 1 Apresenta as instruções associadas aos Opcodes.

Tabela 1 – Lista de Opcodes utilizadas pelo processador GE_22

Opcode	Nome	Formato	Breve Descrição	Exemplo
0000	add	R	Soma	add $s0, $s1, ou seja: $s0 := $s0 + $s1
0001	addi	I	Soma Imediata	addi $s0 3, ou seja: $s0 := $s0 + 3
0010	sub	R	Subtração	sub Ss0, $s1, ou seja: $s0 := $s0 - $s1
0011	subi	I	Subtração Imediata	subi $s0 3, ou seja: $s0 := $s0 - 3
0100	lw	I	Load Word	lw $s0 memória(00), ou seja: $s0 := valor memória(00)
0101	sw	I	Store Word	sw $s0 memória(00), ou seja: memória(00) := $s0
0110	move	R	Mover	move $s0 $s1, ou seja: $s0 := $s1
0111	li	I	Load Imediato	li $s0 1, ou seja: $s0 := 1
1000	beq	I	Desvio Condicional	beq endereço, ou seja: if($s0 == $s1)
1001	bne	I	Desvio Condicional	bne endereço, ou seja: if($s0 != $s1)
1010	If beq e bne	R	Condição para salto	if $s0 $s1, ou seja: if($s0 == $s1)
1011	mul	R	Multiplicação	mul $s0 $s1, ou seja: $s0 := $s0 * $s1
1111	j	J	Desvio Incondicional	J endereço(0000)

1.3 Descrição do Hardware

Nesta seção são descritos os componentes do hardware que compõem o processador GE_22, incluindo uma descrição de suas funcionalidades, valores de entrada e saída.

1.3.1 ALU ou ULA

A ULA é responsável por realizar as operações aritméticas, dentre elas: soma, subtração, multiplicação e eventualmente a ULA efetua operações de comparação de valor para realizar desvios condicionais, a ULA possui 4 entradas, o clock, ula_op que recebe qual operação a mesma vai realizar, um in_port_A e in_port_B onde vão ser recebidos os dois dados de 8 bits cada para realizar as operações, uma saída out_ula_result onde sairá os 8 bits do resultado das operações, zero onde sairá se ocorrerá um desvio condicional e a saída overflow que sai 1 se ocorrer um overflow durante a execução das operações.

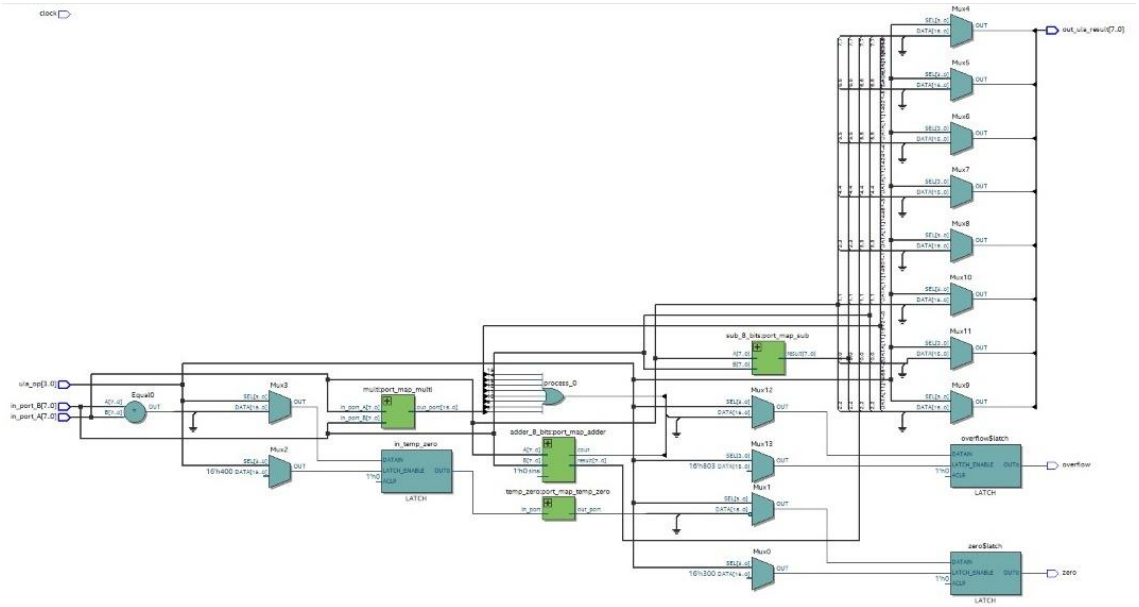

Figura 2 - RTL viewer do componente ULA

1.3.2 Banco de Registradores

O banco de registradores é responsável por armazenar os dados que são usados na execução das operações que utilizam o mesmo, possui 4 registradores que podem armazenar valores de 8 bits, possui como entrada, o clock, que vai ativar o componente, reg_write que ativa a opção de escrever dados no registrador, write_data que recebe o valor de 8 bits do dado a ser escrito no registrador de destino, o endereco_reg_A recebe 2 bits para o endereço do primeiro registrador, o endereco_reg_B recebe 2 bits para o

endereço do segundo registrador, reg_out_A resulta em uma saída de 8 bits do valor armazenado no endereco_reg_A e o re_out_A vai ter uma saída de 8 bits do valor armazenado no endereco_reg_B.

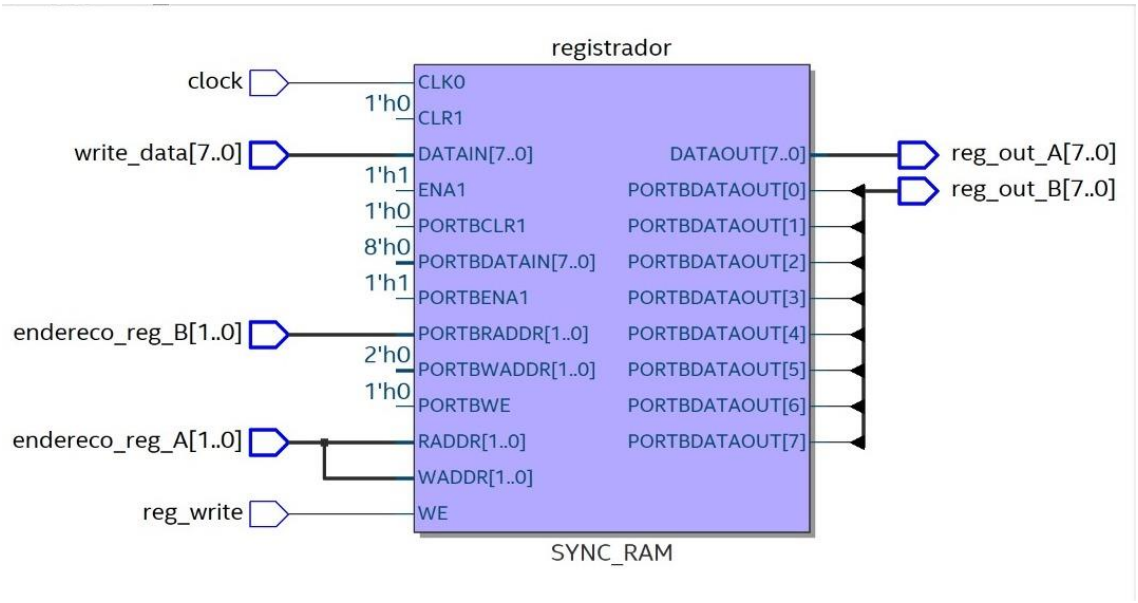

Figura 3 - RTL viewer do componente banco de registradores

1.3.3 Unidade de Controle

O componente unidade de controle tem como objetivo realizar o controle de todos os componentes do processador de acordo com o opcode de 4 bits que é recebido no opcode, esse controle é feito através das flags de saída abaixo:

• **Jump:** sinal que vai decidir se vai ocorrer um desvio incondicional.
• **Branch:** sinal para decidir se vai ocorrer um desvio condicional.
• **Mem_read:** sinal para decidir se será lido um dado da memória RAM.
• **Mem_to_reg:** sinal para decidir se o dado que será escrito no banco de registradores vai ser enviado pela ULA ou pela memória RAM.
• **Ula_op:** sinal de 4 bits que será enviado para a ULA decidir qual operação será realizada.
• **Mem_write:** sinal para decidir se será escrito um dado da memória RAM.
• **Ula_src:** sinal para decidir se o dado que entrará na ULA vai ser enviado pelo banco de registradores ou pelo extensor de sinal.
• **Reg_write:** sinal para decidir se o banco de registradores vai escrever um dado na posição do registrador de destino.

Abaixo segue a tabela, onde é feita a associação entre os opcodes e as flags de controle:

Comando	Jump	Branch	M_read	M_to_reg	Ula_op	M_write	Ula_src	Reg_write
add	0	0	0	0	0000	0	0	1
addi	0	0	0	0	0001	0	1	1
sub	0	0	0	0	0010	0	0	1
subi	0	0	0	0	0011	0	1	1
lw	0	0	1	1	0100	0	0	1
sw	0	0	0	0	0101	1	0	0
move	0	0	0	0	0110	0	0	1
li	0	0	0	0	0111	0	1	1
Beq	0	1	0	0	1000	0	0	0
bne	0	1	0	0	1001	0	0	0
If bne beq	0	0	0	0	1010	0	0	0
mul	0	0	0	0	1011	0	0	1
jump	1	0	0	0	1111	0	0	0

Tabela 2 - Detalhes das flags de controle do processador

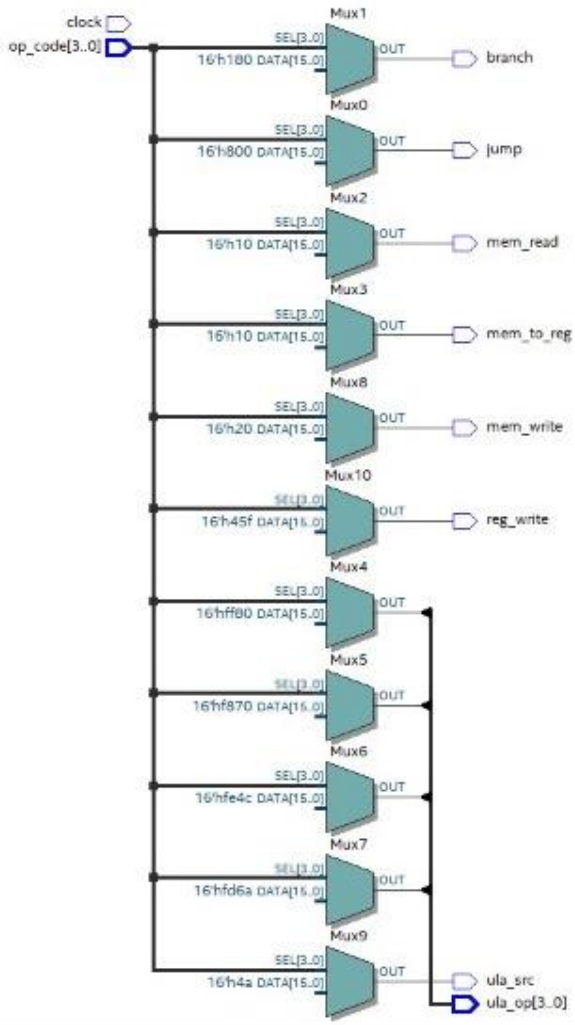

Figura 4 - RTL viewer do componente Unidade de Controle

1.3.4 Memória de dados

A memória RAM é responsável por armazenar temporariamente os dados que são usados durante a execução das instruções, possui 5 entradas, o clock que ativa o componente, in_port recebe o dado de 8 bits que será armazenado temporariamente na RAM, mem_write recebe 1 bit para saber se vai armazenar dados na RAM, mem_read recebe 1 bit para saber se será lido algum dado da memória RAM, endereco recebe 8 bits da posição da memória RAM onde o dado deve ser escrito ou lido e possui 1 saida out_port onde sai um dado de 8 bits da posição que foi recebida no endereço, se a mem_read estiver setada em 1.

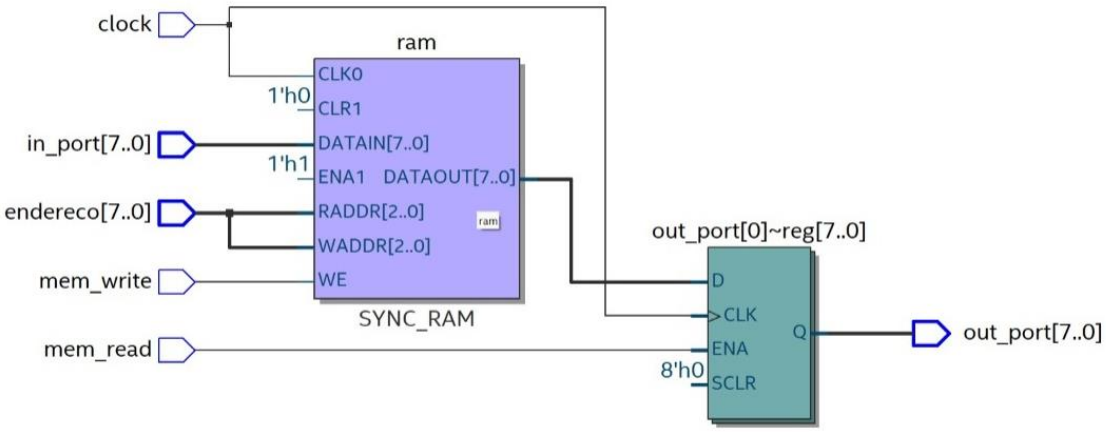

Figura 5 - RTL viewer do componente memória de dados

1.3.5 Memória de Instruções

A memória ROM tem a função de armazenar as instruções que vão ser executadas pelo processador, possui duas entradas, o clock e o in_port, que é o endereço de 8 bits da instrução que será enviada para a execução, possui uma saída out_port de 8 bits da instrução que será executada.

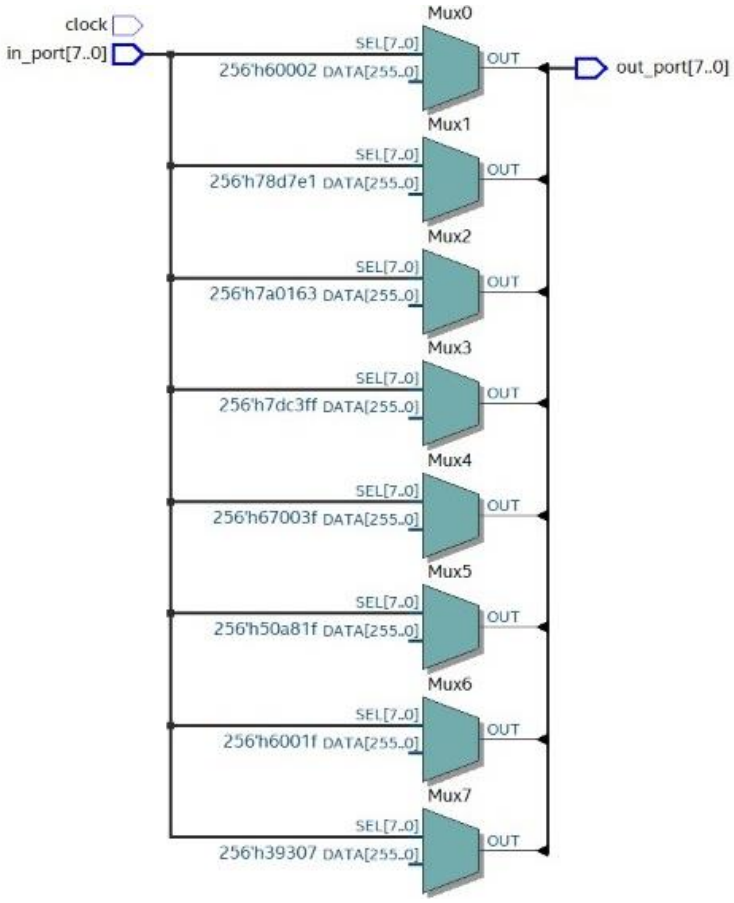

Figura 6 - RTL viewer do componente memória de instruções

1.3.6 Multiplexador 2x1

Um multiplexador tem duas entradas de 8 bits, in_A e in_B que recebera os dados e uma entrada in_port que decidirá qual dado vai sair, caso o in_port for 0 o dado que sairá na out_port será o dado in_A se for 1 sairá o dado in_B.

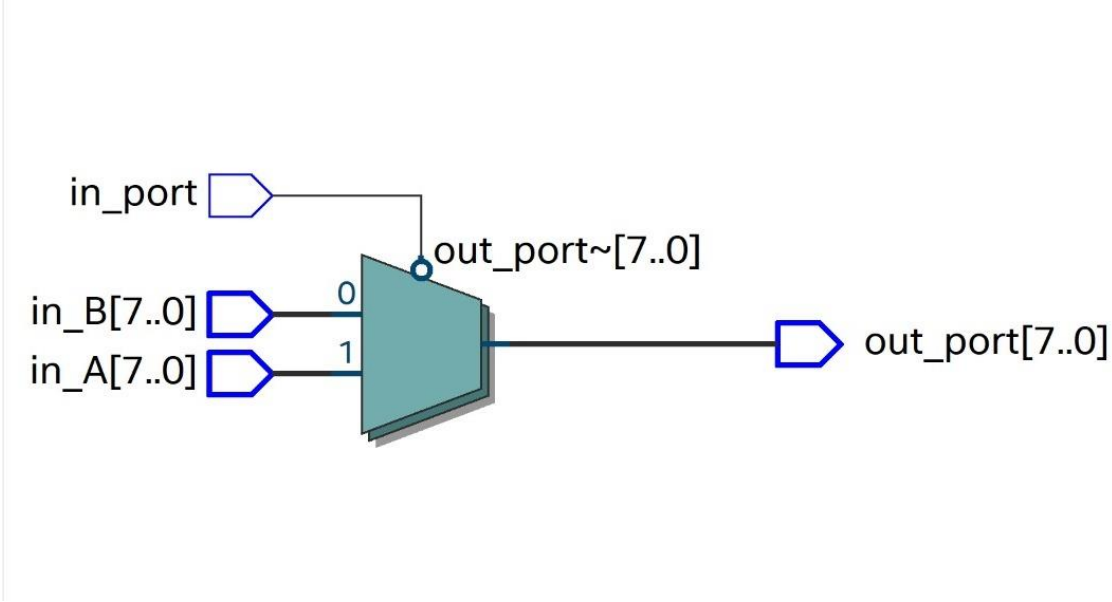

Figura 7 - RTL viewer do componente multiplexador 2x1

1.3.7 Extensor de Sinal 2_8

O extensor de sinal é responsável por estender o número de bits da entrada in_port de 2 bits para 8 bits para a saída out_port, seu resultado é usado para operações na ULA e como endereço da memória RAM.

Figura 8 - RTL viewer do componente extensor de sinal 2_8

1.3.8 Extensor de Sinal 4_8

O extensor de sinal é responsável por estender o número de bits da entrada in_port de 4 bits para 8 bits para a saída out_port, seu resultado é usado para desvios condicionais e incondicionais.

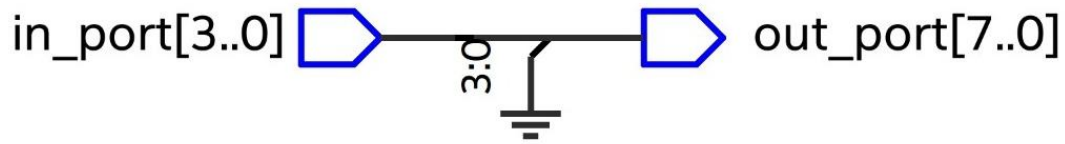

Figura 9 - RTL viewer do componente extensor de sinal 4_8

1.3.9 PC

A função do componente PC é armazenar o endereço de 8 bits da instrução que será executada, possui dois valores de entrada, o clock, que é o responsável por ativar o componente e o in_port, onde entra o endereço da instrução a ser executada e possui uma saída, out_port, onde sai o endereço da instrução a ser executada.

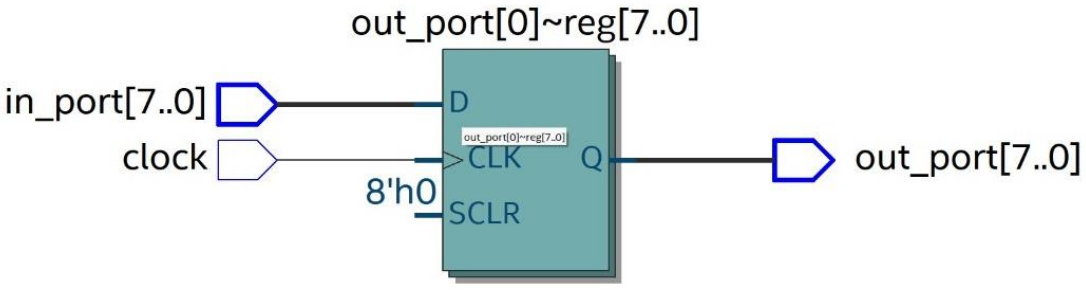

Figura 10 - RTL viewer do componente PC

1.3.10 PC Counter

O PC Counter é o componente que adiciona 1 passo ao PC. Em operações normais, ou seja, sem saltos, ele apenas adiciona 1 ao endereço recebido pelo PC, fazendo o programa armazenado avançar 1 passo. O componente recebe o endereço do PC e adiciona 1 a esse valor.

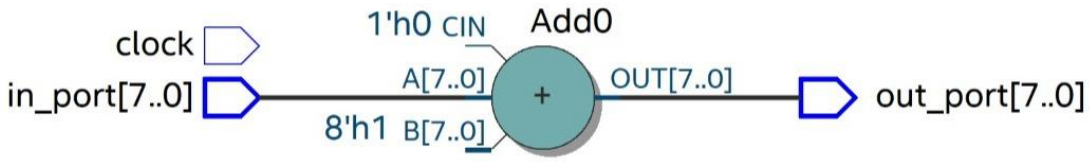

Figura 11 - RTL viewer do componente PC Counter

1.3.11 Zero

O zero fornece uma flag que indica se um valor é igual ou diferente do que foi comparado. O componente Zero fica dentro da ULA, e é utilizado apenas no caso de operações comparativas. Sua função é apenas inicializar a flag necessária para realizar a comparação.

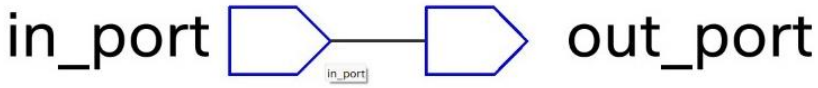

Figura 12 – RTL viewer do componente zero

1.3.12 And

O and tem a função de decidir se vai ocorrer um desvio condicional, ele tem duas entradas, in_port_A e in_port_B se as duas estiverem recebendo 1 o mesmo sairá 1 no out_port assim passando o valor 1 para o multiplexador fazendo com que aconteça um desvio condicional.

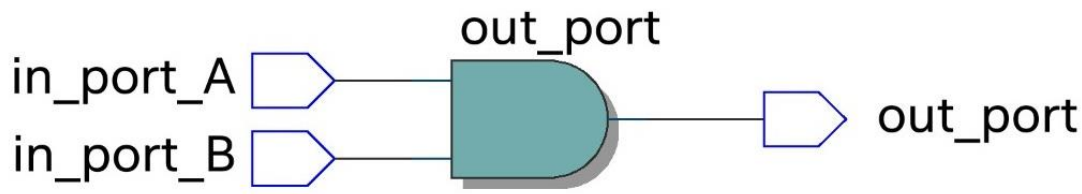

Figura 13 - RTL viewer do componente And

1.3.13 Divisor de Instruções

A divisor de instruções é responsável por dividir os 8 bits que recebe da memória ROM no in_port, out_op_code receberá os bits que entrarão na Unidade de Controle, out_jump recebe os últimos 4 bits da instrução que serão utilizados para desvio condicional e incondicional, out_rs recebe os 2 bits do registrador onde será armazenado o resultado das operações, out_rt pode ser tanto o dado do registrador ou o endereço da memória RAM para realizar load e store.

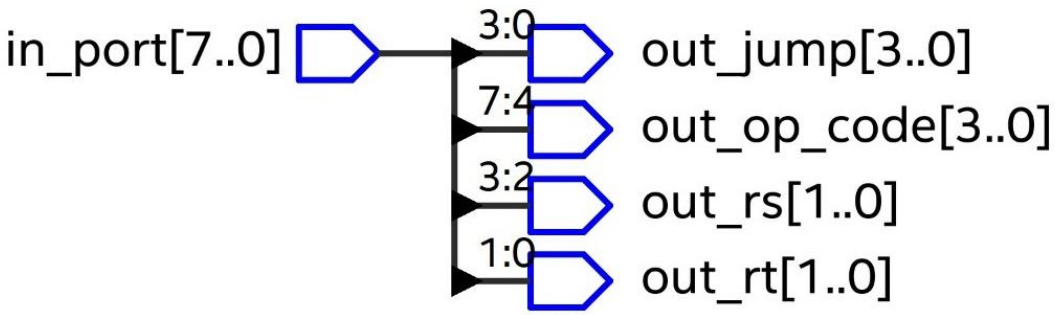

Figura 14 - RTL viewer do componente Divisor de Instruções

1.4 Datapath

É a conexão entre os as unidades funcionais formando um único caminho de dados e acrescentando uma unidade de controle responsável pelo gerenciamento das ações que serão realizadas para diferentes classes de instruções.

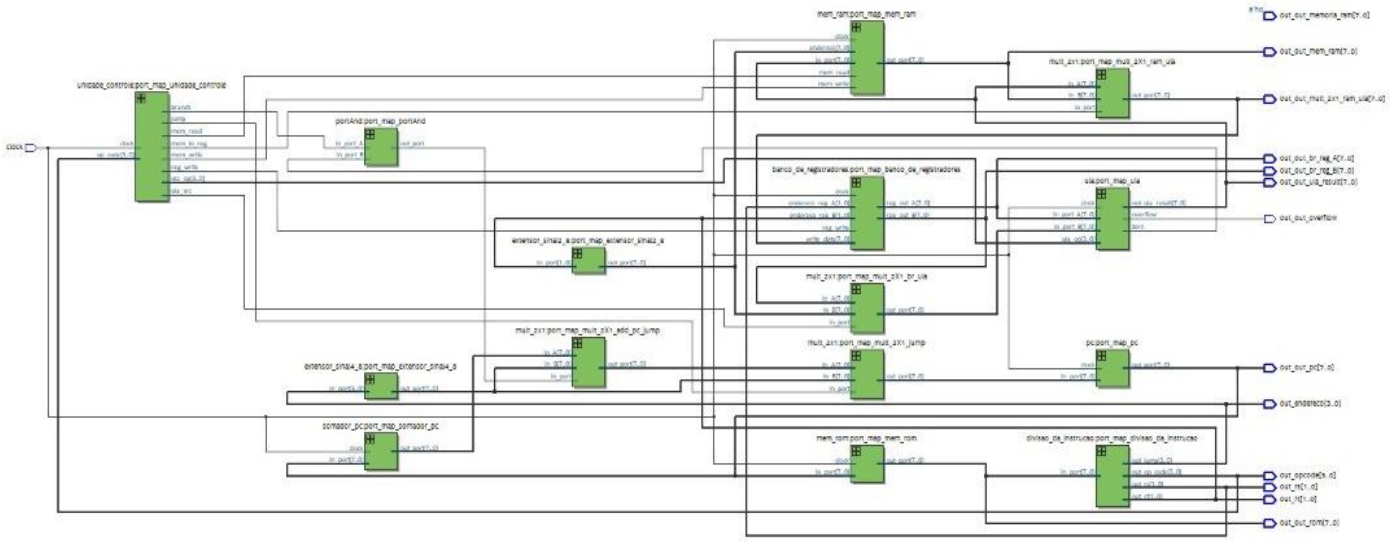

Figura 15 – RTL viewer do processador GE_22

2. Simulações e Testes

Objetivando analisar e verificar o funcionamento do processador, efetuamos alguns testes analisando cada componente do processador em especifico, em seguida efetuamos testes de cada instrução que o processador implementa. Para demonstrar o funcionamento do processador GE 22 utilizaremos como exemplo o código para calcular o número da sequência de Fibonacci.

Endereço	Linguagem de Alto Nível	Binário		
		Opcode	Reg1	Reg2
			Endereço	
0	**li** S3 3	0111	11	11
1	**mul** S3 S3	1011	11	11
2	**addi** S3 1	0001	11	01
3	**addi** S3 2	0001	11	10
4	**addi** S3 2	0001	11	10
5	**li** S2 0	0111	10	00
6	**li** S0 0	0111	00	00

7	**sw** S0 ram(00)	0101	00	00
8	**li** S0 1	0111	00	01
9	**sw** S0 ram(01)	0101	00	01
10	**lw** S0 ram(00)	0100	00	00
11	**add** S1 S0	0000	01	00
12	**lw** S0 ram(01)	0100	00	01
13	**add** S1 S0	0000	01	00
14	**sw** S0 ram(00)	0101	00	00
15	**sw** S1 ram(01)	0101	01	01
16	**addi** S2 1	0001	10	01
17	**if** S2 == S3	1010	10	11
18	**bne** S2 != S3	1001	1010	

Tabela 3 – Código do Fibonacci para o processador GE_22

Verificação dos resultados no relatório da simulação: Após a compilação e execução da simulação, o seguinte relatório é exibido.

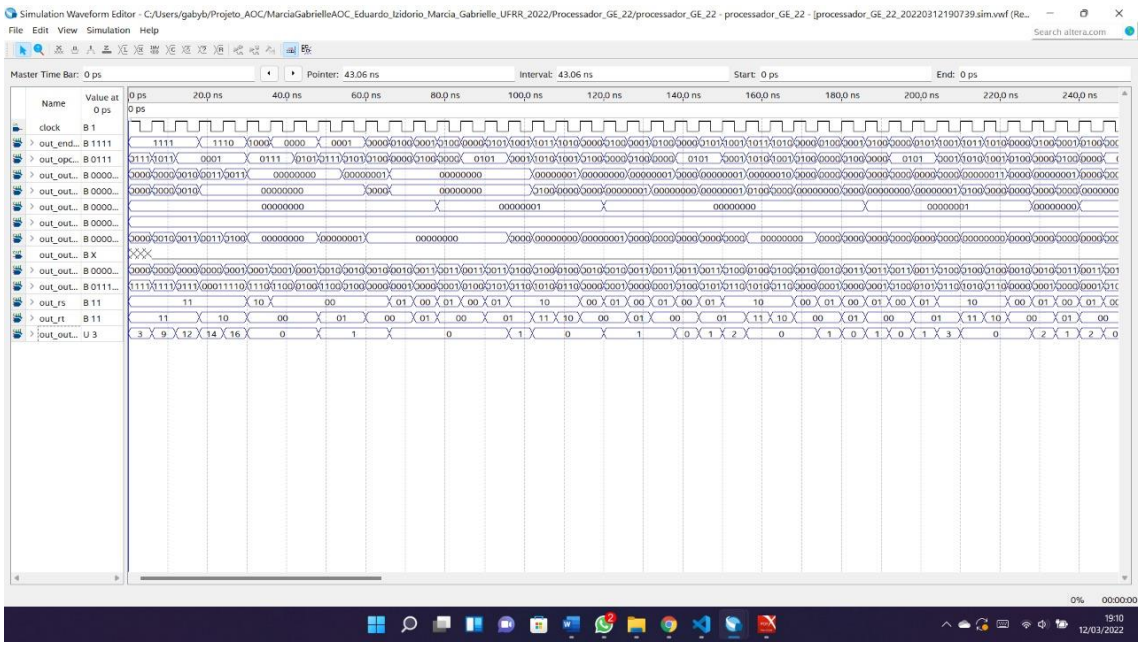

Figura 16 – Resultado da 1 parte do Fibonacci

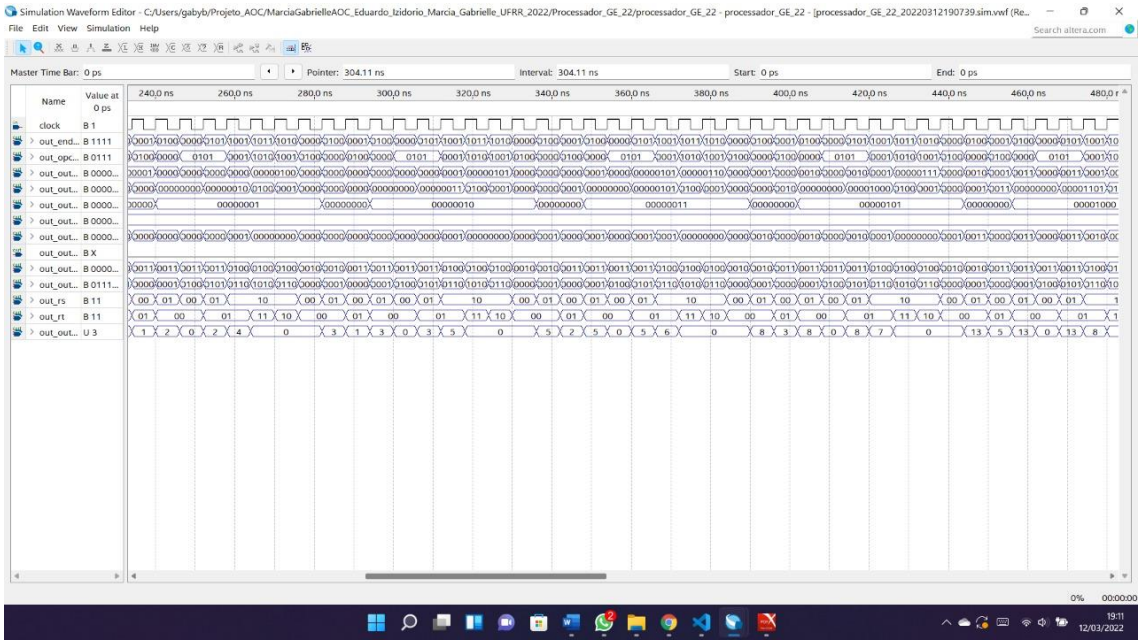

Figura 17 – Resultado da 2 parte do Fibonacci

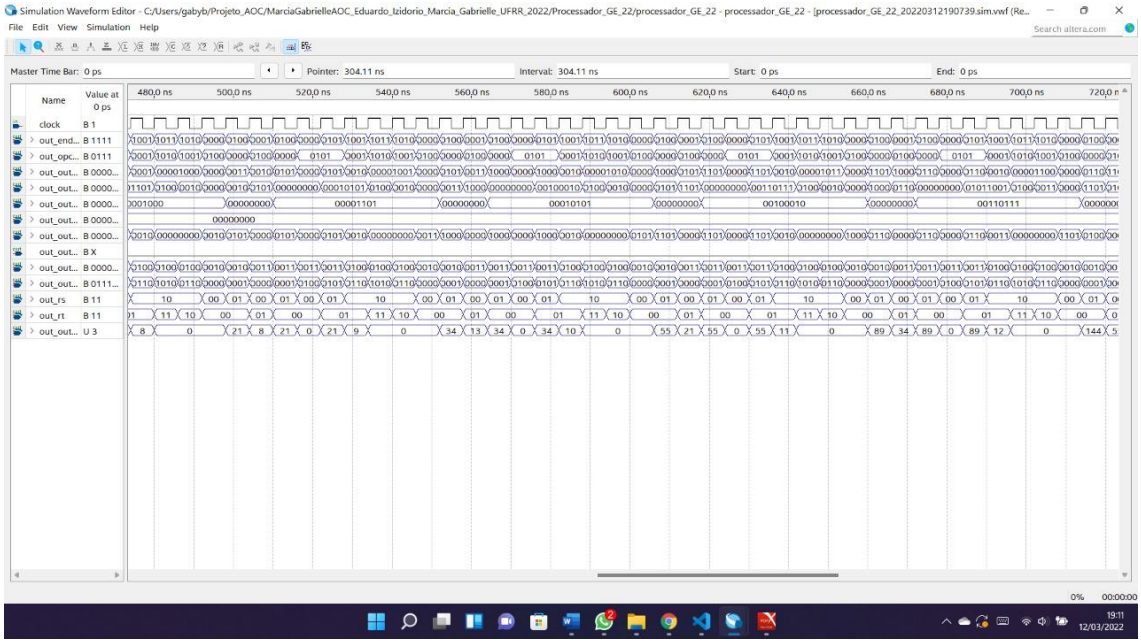

Figura 18 – Resultado da 3 parte do Fibonacci

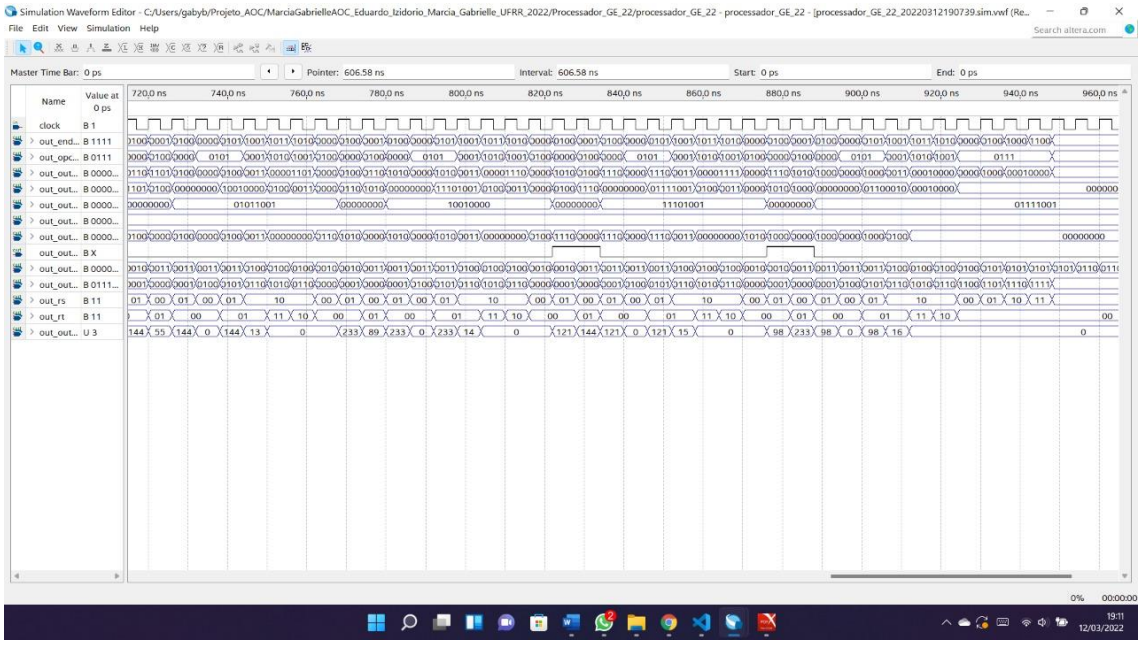

Figura 19 – Resultado da 4 parte do Fibonacci

3. Considerações finais

Este trabalho apresentou o projeto e implementação do processador de 8 bits denominado GE 22 referente ao projeto final de disciplina de Arquitetura e Organização de computadores.

O processador contém todos os requisitos para a realizações de operações aritméticas, memória de dados, condicionais, controle de dados e afins. E demonstração do funcionamento do processador GE 22 com o código para calcular o número da sequência de Fibonacci se mostrou bem funcional.

As limitações mais notáveis, como a baixa janela de valores que podem sair da ALU e a baixa capacidade de transmissão de informações em bits reduzem o escopo de programar que podem ser implementados no PU8B. Outra limitação é o load e store que funcionam em somente uma instrução e devido a limitação de 8 bits, somente dois bits são usados para acessar a memória RAM, o que nos limita a poder acessar somente as 4 primeiras posições da memória RAM.