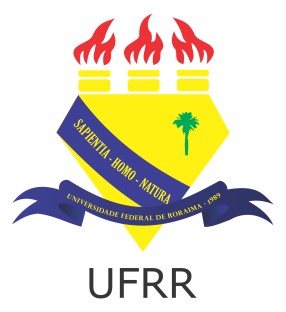
****

**PODER EXECUTIVO**

**MINISTÉRIO DA EDUCAÇÃO**

**UNIVERSIDADE FEDERAL DE RORAIMA**

**DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO**

**ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES**

**RELATÓRIO DO PROJETO: PROCESSADOR NEO**

**ALUNOS:**

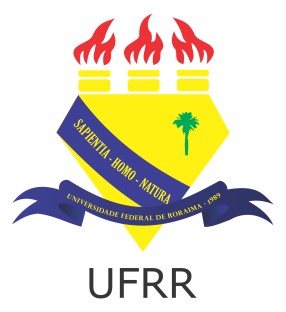
**Gabriel Carvalho de Araújo - 2201524449**

**Hermino Barbosa de Freitas Júnior - 2201524475**

**Jeovane Araujo da Silva – 2201524420**

**Janeiro de 2018**

**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 NEO**

**Janeiro de 2018**

**Boa Vista/Roraima**

**Resumo**

Neste trabalho será abordado os principais pontos da construção e implementação do processador multiciclo NEO de 8 bits baseado na arquitetura MIPS. Será descrito com detalhes cada etapa do processo de sua construção levando em conta todos os componentes necessários para o seu funcionamento e os testes realizados durante a implementação. O processador terá capacidade de executar 16 instruções, já incluso, a soma de ponto flutuante, dando a possibilidade bastante abrangente de executar algoritmos.

A implementação do processador foi feita integralmente com a linguagem de descrição de hardware VHDL e os teste foram analisados através simulador ModelSim que gera waveforms, demonstrando assim o comportamento do processador.

**Conteúdo**

Sumário

[1 Especificação 7](#_Toc503625940)

[2 Plataforma de desenvolvimento 7](#_Toc503625941)

[3 Conjunto de instruções 7](#_Toc503625942)

[3.1 Tipo de Instruções: 8](#_Toc503625943)

[3.2 Visão geral das instruções do Processador NEO: 9](#_Toc503625944)

[4 Descrição do Hardware 9](#_Toc503625945)

[4.1 Registrador Flip Flop 9](#_Toc503625946)

[4.2 Memória de Instrução 10](#_Toc503625947)

[4.3 Banco de Registradores 11](#_Toc503625948)

[4.4 Extensor de Sinal de 2 para 8 bits 12](#_Toc503625949)

[4.5 Extensor de Sinal de 4 para 8 bits. 12](#_Toc503625950)

[4.6 Multiplexador de 2 entradas 13](#_Toc503625951)

[4.7 Multiplexador de 3 entrada 13](#_Toc503625952)

[4.8 ULA 14](#_Toc503625953)

[4.9 Somador de Ponto Flutuante 15](#_Toc503625954)

[4.10 Memória de Dados 17](#_Toc503625955)

[4.11 Unidade de Controle 18](#_Toc503625956)

[4.12 Bloco Operativo 20](#_Toc503625957)

[5 Datapath 21](#_Toc503625958)

[5.1 Datapath idealizado 22](#_Toc503625959)

[5.2 Datapath 22](#_Toc503625960)

[6 Simulações e Testes 23](#_Toc503625961)

[6.1 N-ésimo termo de uma P.A: 23](#_Toc503625962)

[6.2 Fatorial: 24](#_Toc503625963)

[6.3 Verificação dos resultados no relatório da simulação 24](#_Toc503625964)

[7 Considerações finais 25](#_Toc503625965)

**Lista de Tabelas**

[Tabela 1 – Tabela que mostra o formato da instrução do tipo R. 8](#_Toc504569838)

[Tabela 2 - Tabela que mostra a separação dos bits para intruções do tipo R. 8](#_Toc504569839)

[Tabela 3 - Tabela que mostra o formato da instrução do tipo J. 8](#_Toc504569840)

[Tabela 4 - Tabela que mostra a separação dos bits do tipo J. 8](#_Toc504569841)

[Tabela 5 - Tabela que mostra a lista de Opcodes utilizadas pelo processador NEO. 9](#_Toc504569842)

[Tabela 6 - Organização de ponto flutuante para 32 bits. 16](#_Toc504569843)

[Tabela 7 - Adaptação e organização de ponto flutuante para 8 bits. 16](#_Toc504569844)

[Tabela 8 - Demonstração da adaptação do ponto flutuante parte 1. 16](#_Toc504569845)

[Tabela 9 - Demonstração da adaptação do ponto flutuante parte 2. 17](#_Toc504569846)

[Tabela 10 - Código N-ésimo termo de uma P.A. para o processador NEO. 23](#_Toc504569847)

[Tabela 11 - Código Fatorial para o processador NEO. 25](#_Toc504569848)

[Tabela 12 - Código para calcular a soma de ponto flutuante do processador NEO. 26](#_Toc504569849)

**Lista de Figura**

[Figura 1 - Especificações no Quartus. 7](#_Toc504569888)

[Figura 2 - Trecho de código da entidade do componente Flip Flop. 9](#_Toc504569889)

[Figura 3 - RTL view do componente Flip Flop gerado pelo Quartus. 10](#_Toc504569890)

[Figura 4 - Trecho de código da entidade do componente Memória de Instrução. 10](#_Toc504569891)

[Figura 5 - RTL view do componente Memória de Instrução gerado pelo Quartus. 10](#_Toc504569892)

[Figura 6 - Trecho de código da entidade do componente Banco de Registradores. 11](#_Toc504569893)

[Figura 7 - RTL view do componente Banco de Registradores gerado pelo Quartus. 11](#_Toc504569894)

[Figura 8 - Trecho de código da entidade do componente Extensor de Sinal de 2 para 8 bits. 12](#_Toc504569895)

[Figura 9 - RTL view do componente Extensor de Sinal de 2 para 8 bits gerado pelo Quartus. 12](#_Toc504569896)

[Figura 10 - Trecho de código da entidade do componente Extensor de Sinal de 4 para 8 bits. 12](#_Toc504569897)

[Figura 11 - RTL view do componente Extensor de Sinal de 4 para 8 bits gerado pelo Quartus. 12](#_Toc504569898)

[Figura 12 - Trecho de código da entidade do componente Multiplexador de 2 entradas. 13](#_Toc504569899)

[Figura 13 - RTL view do componente Multiplexador de 2 entradas gerado pelo Quartus. 13](#_Toc504569900)

[Figura 14 - Trecho de código da entidade do componente Multiplexador de 3 entradas. 13](#_Toc504569901)

[Figura 15 - RTL view do componente Multiplexador de 3 entradas gerado pelo Quartus. 14](#_Toc504569902)

[Figura 16 - Trecho de código da entidade do componente Multiplexador de 3 entradas. 14](#_Toc504569903)

[Figura 17 - RTL view do componente ULA gerado pelo Quartus. 15](#_Toc504569904)

[Figura 18 - Trecho de código do Somador de Ponto Flutuante. 15](#_Toc504569905)

[Figura 19 - RTL view do componente Somador de Ponto Flutuante gerado pelo Quartus. 17](#_Toc504569906)

[Figura 20 - Trecho de código da Memória de Dados. 17](#_Toc504569907)

[Figura 21 - RTL view do componente Memória de dados gerado pelo Quartus. 18](#_Toc504569908)

[Figura 22 - Trecho de código da Unidade de Controle. 18](#_Toc504569909)

[Figura 23 - RTL view do componente Unidade de Controle gerado pelo Quartus. 19](#_Toc504569910)

[Figura 24 - Máquina de Estados da Unidade de Controle gerada pelo Quartus. 19](#_Toc504569911)

[Figura 25 - Trecho de código do Bloco Operativo. 20](#_Toc504569912)

[Figura 26 - RTL view do componente Bloco Operativo gerado pelo Quartus. 21](#_Toc504569913)

[Figura 27 - Datapath idealizado após a construção dos componentes. 22](#_Toc504569914)

[Figura 28 - Trecho de código datapath do processador NEO. 22](#_Toc504569915)

[Figura 29 - RTL view do Datapath gerado pelo Quartus. 23](#_Toc504569916)

[Figura 30 - Waveform do teste do algoritmo N-ésimo termo de uma P.A. 24](#_Toc504569917)

[Figura 31 - Waveform do teste do algoritmo N-ésimo termo de uma P.A. 25](#_Toc504569918)

[Figura 32 - Waveform do teste do algoritmo para calcular fatorial. 26](#_Toc504569919)

[Figura 33 - Waveform do teste do algoritmo de Soma de Ponto Flutuante. 27](#_Toc504569920)

# Especificação

Nesta seção é apresentado o conjunto de itens para o desenvolvimento do processador multiciclo NEO de 8 bits, bem como a descrição detalhada de cada etapa da construção do processador.

# Plataforma de desenvolvimento

Para a implementação do processador NEO foi utilizado a IDE: Quartus Prime, versão 16.1, com o simulador ModelSim-Altera e toda a descrição do hardware foi feita com a linguagem VHDL.

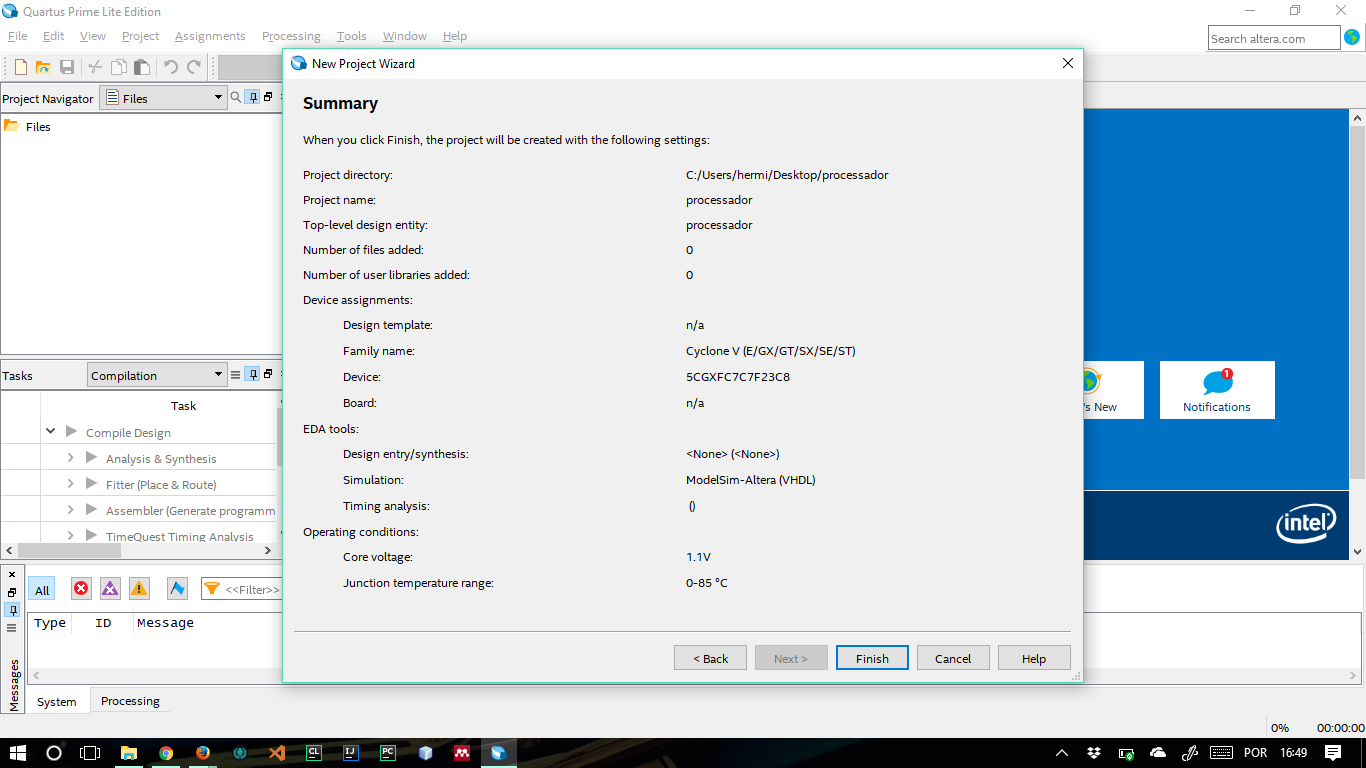


Figura 1 - Especificações no Quartus.

# Conjunto de instruções

O processador NEO possui 4 registradores no banco de registradores: S0, S1, S2, S3. Assim como 2 formatos de instruções de 8 bits cada, instruções do tipo **R** e **J**, seguem algumas considerações sobre as estruturas contidas nas instruções:

* **Opcode**: a operação básica a ser executada pelo processador, tradicionalmente chamado de código de operação;
* **RS**: o registrador contendo o primeiro operando fonte e adicionalmente para alguns tipos de instruções (ex. instruções do tipo R) é o registrador de destino;
* **RT**: o registrador contendo o segundo operando fonte;
* **IMM**: endereço de memória, para instruções do tipo jump condicional e incondicional.

## Tipo de Instruções:

**- Formato do tipo R:** Este formato aborda instruções de Load, Store e instruções baseadas em operações aritméticas e instruções aritméticas imediatas.

Formato para escrita de código na linguagem NEO:

Tabela 1 – Tabela que mostra o formato da instrução do tipo R.

|  |  |  |
| --- | --- | --- |
| Tipo da Instrução | RS | RT |

Formato para escrita em código binário:

Tabela 2 - Tabela que mostra a separação dos bits para intruções do tipo R.

|  |  |  |  |
| --- | --- | --- | --- |
| OPCODE | RS | RT | IMM |
| 7-4 | 3-2 | 1-0 | - |
| 4 | 2 | 2 | - |

**- Formato do tipo J:** Este formato aborda instruções do tipo jump condicional e incondicional.

Formato para escrita de código na linguagem NEO:

Tabela 3 - Tabela que mostra o formato da instrução do tipo J.

|  |  |
| --- | --- |
| Tipo da Instrução | LABEL |

Formato para escrita em código binário:

Tabela 4 - Tabela que mostra a separação dos bits do tipo J.

|  |  |  |  |
| --- | --- | --- | --- |
| OPCODE | RS | RT | IMM |
| 7-4 | - | - | 3-0 |
| 4 | - | - | 4 |

## Visão geral das instruções do Processador NEO:

O número de bits do campo **Opcode** das instruções é igual a quatro, sendo assim obtemos um total () de 16 **Opcodes (0-16)** que são distribuídos entre as instruções, assim como é apresentado na Tabela 5.

Tabela 5 - Tabela que mostra a lista de Opcodes utilizadas pelo processador NEO.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Tipo | OP | RS | RT | IMM | Exemplo | Instrução |
| R | 0000 | 00-11 | 00-11 | - | **add** $s1, $s2 | **add** |
| R | 0001 | 00-11 | 00-11 | - | **addi** $s1, WORD | **addi** |
| R | 0010 | 00-11 | 00-11 | - | **sub** $s1, $s2 | **sub** |
| R | 0011 | 00-11 | 00-11 | - | **subi** $s1, WORD | **subi** |
| R | 0100 | 00-11 | 00-11 | - | **mult** $s1, $s2 | **mult** |
| R | 0101 | 00-11 | 00-11 | - | **multi** $s1, WORD | **multi** |
| R | 0110 | 00-11 | 00-11 | - | **eq** $s1, $s2 | **eq** |
| R | 0111 | 00-11 | 00-11 | - | **eqi** $s1, WORD | **eqi** |
| R | 1000 | 00-11 | 00-11 | - | **move** $s1, $s2 | **move** |
| R | 1001 | 00-11 | 00-11 | - | **movi** $s1, WORD | **movi** |
| R | 1010 | 00-11 | 00-11 | - | **lw** $s1, WORD | **lw** |
| R | 1011 | 00-11 | 00-11 | - | **sw** $s1, WORD | **sw** |
| R | 1100 | 00-11 | 00-11 | - | **addf** $s1, $s2 | **addf** |
| J | 1101 | - | - | 0000-1111 | **bne** Label | **bne** |
| J | 1110 | - | - | 0000-1111 | **beq** Label | **beq** |
| J | 1111 | - | - | 0000-1111 | **j** Label | **j** |

# Descrição do Hardware

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

## Registrador Flip Flop

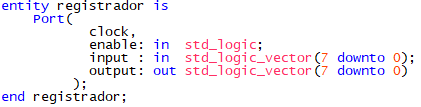


Figura 2 - Trecho de código da entidade do componente Flip Flop.

O registrador flip flop funciona da seguinte maneira: Caso o **clock** esteja em borda altae **enable** possuir valor 1, o valor da entrada **input** é registrado e o **output** recebe o **input.**

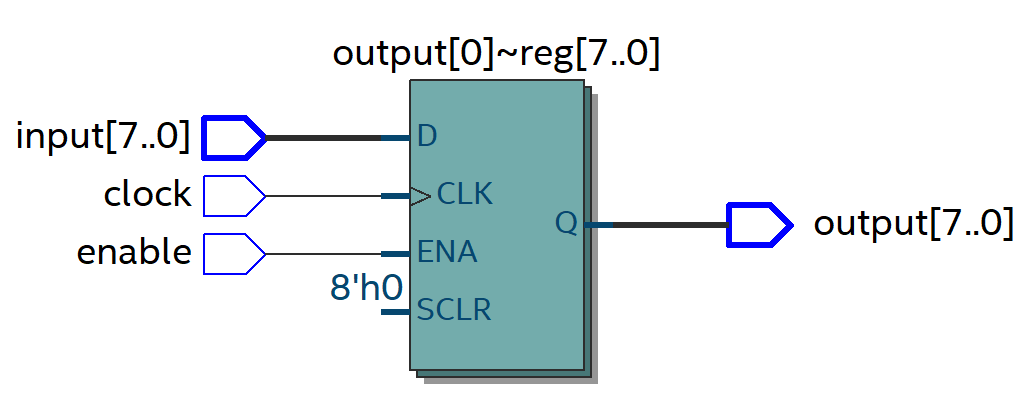
****

Figura 3 - RTL view do componente Flip Flop gerado pelo Quartus.

## Memória de Instrução

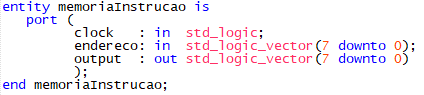


Figura 4 - Trecho de código da entidade do componente Memória de Instrução.

A memória de instrução funciona da seguinte maneira: Caso o **clock** esteja em borda alta, a saída **output** irá conter o valor da memória no endereço da entrada **endereco.**

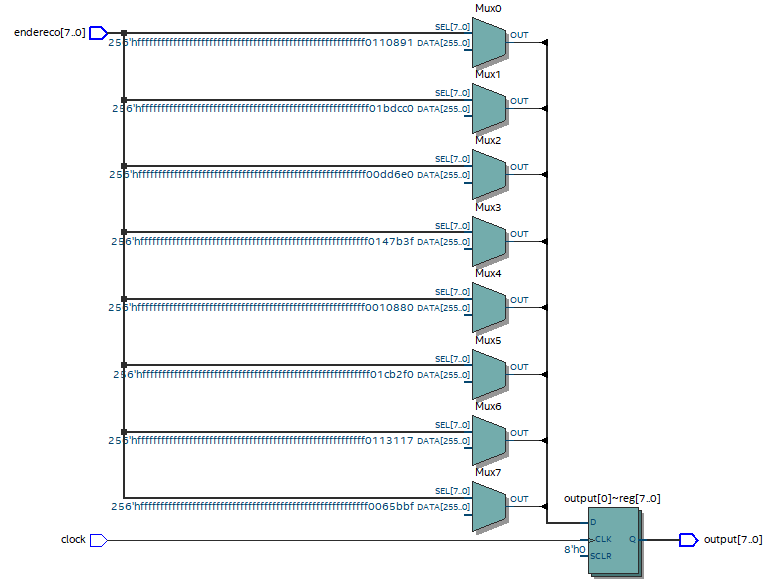
****

Figura 5 - RTL view do componente Memória de Instrução gerado pelo Quartus.

## Banco de Registradores

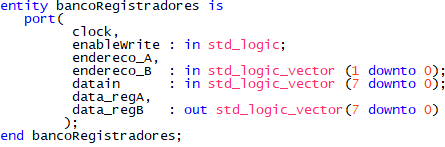


Figura 6 - Trecho de código da entidade do componente Banco de Registradores.

O banco de registradores funciona da seguinte maneira: Funciona como um seletor de dados, caso o **enableWrite** esteja em 0, é procurado os dados registrados no **endereco\_A** e **endereco\_B**. Caso o **enableWrite** esteja em 1, o banco de registrador irá guardar o dado contido no **datain** no endereç**o endereco\_A** através do demultiplexador.

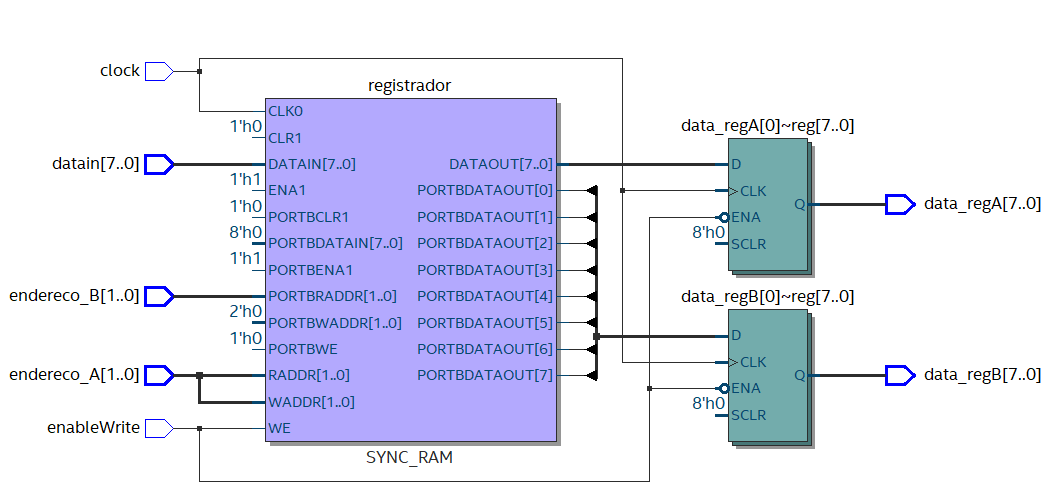


Figura 7 - RTL view do componente Banco de Registradores gerado pelo Quartus.

## Extensor de Sinal de 2 para 8 bits

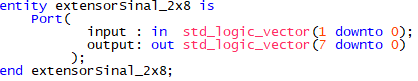


Figura 8 - Trecho de código da entidade do componente Extensor de Sinal de 2 para 8 bits.

O extensor de sinal de 2 para 8 bits funciona da seguinte maneira: A entrada **input** com 2 bitsserá convertida para uma saída **output** com 8 bits.

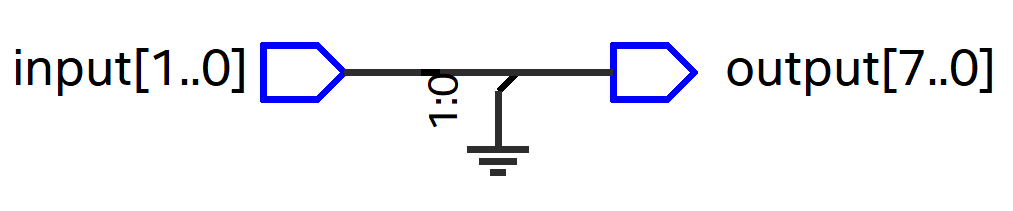


Figura 9 - RTL view do componente Extensor de Sinal de 2 para 8 bits gerado pelo Quartus.

## Extensor de Sinal de 4 para 8 bits.

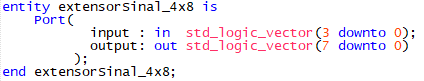


Figura 10 - Trecho de código da entidade do componente Extensor de Sinal de 4 para 8 bits.

O extensor de sinal de 4 para 8 bits funciona da seguinte maneira: A entrada **input** com 4 bitsserá convertida para uma saída **output** com 8 bits.

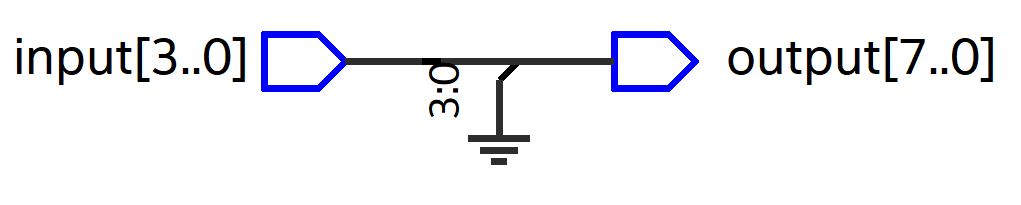
****

Figura 11 - RTL view do componente Extensor de Sinal de 4 para 8 bits gerado pelo Quartus.

## Multiplexador de 2 entradas

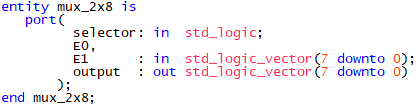


Figura 12 - Trecho de código da entidade do componente Multiplexador de 2 entradas.

O multiplexador de 2 entradas funciona da seguinte maneira: É um componente onde terá 2 entradas e apenas uma saída que será definida através do **selector** dependendo do seu valor: caso o selector seja "00", a saída do MUX terá o valor de E0. Caso o seletor seja "01", a saída do MUX terá o valor de E1. Obs.: todos os casos necessitam do evento de borda alta no ciclo de clock.

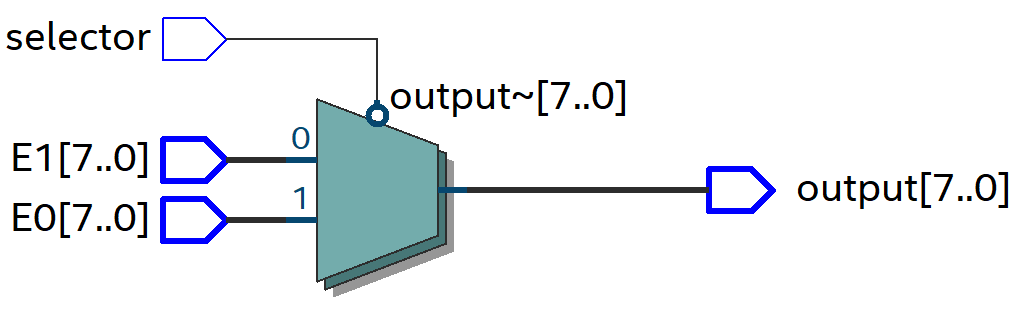
****

Figura 13 - RTL view do componente Multiplexador de 2 entradas gerado pelo Quartus.

## Multiplexador de 3 entrada

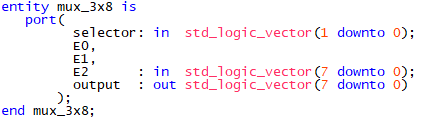


Figura 14 - Trecho de código da entidade do componente Multiplexador de 3 entradas.

O multiplexador de 2 entradas funciona da seguinte maneira: É um componente onde terá diversas entradas e apenas uma saída que será definida através do selector dependendo do seu valor: caso o selector seja "00", a saída do MUX terá o valor de E0. caso o seletor seja "01", a saída do MUX terá o valor de E1. caso o seletor seja "10", a saída do MUX terá o valor de E2. Obs.: todos os casos necessitam do evento de borda alta no ciclo de clock

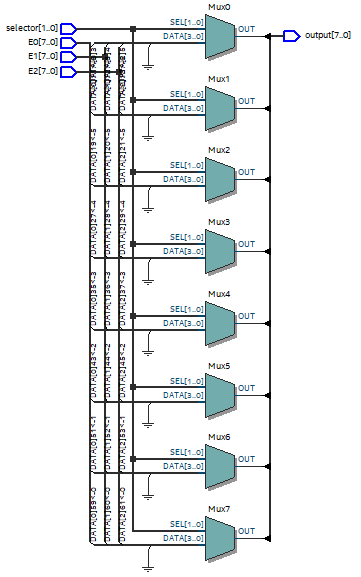


Figura 15 - RTL view do componente Multiplexador de 3 entradas gerado pelo Quartus.

## ULA

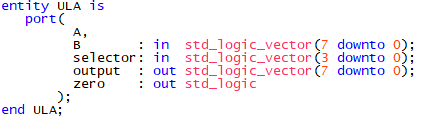


Figura 16 - Trecho de código da entidade do componente Multiplexador de 3 entradas.

O componente ULA (Unidade Lógica Aritmética) tem como principal objetivo efetuar as principais operações aritméticas, dentre elas: soma, subtração, multiplicação. Adicionalmente a ULA efetua operações de comparação de valor igual. O componente ULA recebe como entrada três valores: **A** – dado de 8bits para operação; **B** - dado de 8bits para operação e **OP** – identificador da operação que será realizada de 4bits. A ULA também possui duas saídas: **zero** – identificador de resultado (2bit) para comparações (1 se verdade e 0 caso contrário); e **result** – saída com o resultado das operações.

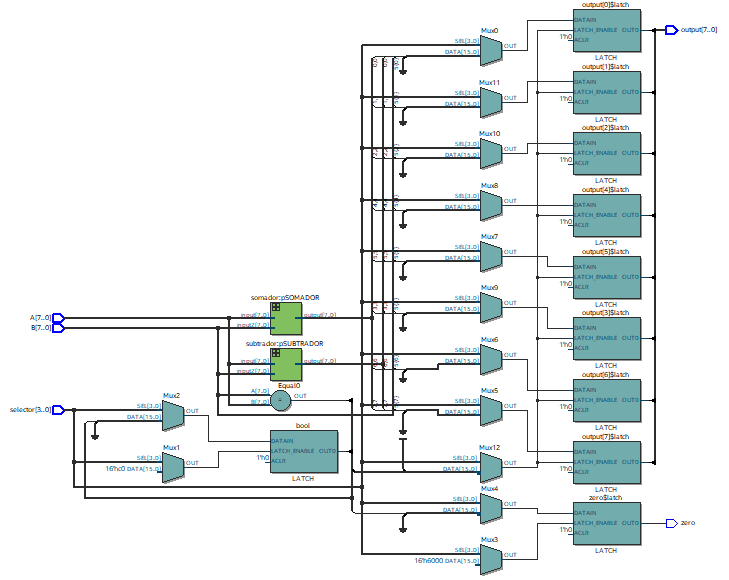


Figura 17 - RTL view do componente ULA gerado pelo Quartus.

## Somador de Ponto Flutuante

Somador com entradas e saída usando a adaptação da representação para ponto flutuante que serão apresentadas neste documento.

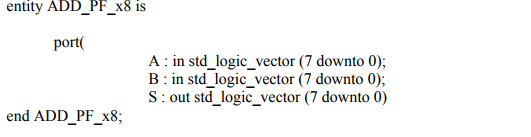


Figura 18 - Trecho de código do Somador de Ponto Flutuante.

O padrão IEEE 754 é uma forma de normalizar as operações com pontos flutuantes, tal padrão dá suporte apenas para arquiteturas de 16, 32 e 64 bits. Para um processador de 8 bits se fez necessário uma adaptação, que será mostrada a seguir.

Tabela 6 - Organização de ponto flutuante para 32 bits.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |
| S | Expoente | | | | | | | | | | | Significado | | | | | | | | | | | | | | | | | | | |

Para uma representação de 8 bits foi necessária uma distribuição onde há o máximo de precisão para essa arquitetura, foram distribuídos 3 bits para o expoente e 5 bits para o significando, nesse caso o bit de representação de sinal não foi levado em consideração, pois não há utilidade no processador NEO.

Tabela 7 - Adaptação e organização de ponto flutuante para 8 bits.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Expoente | | | Significando | | | | |

O número 5,3 em decimal equivale a 101,0100 x 20 em binário, normalizando este número ficaria:

**10,1010 × 21**

**-1,01010 × 22**

Agora o número está normalizado, pois há apenas um único 1 após a virgula. Para converter este número para a representação de 8 bits mostrada anteriormente são necessários passos simples:

Primeiro separamos o significando ignorando o 1 implícito em um binário normalizado.

Tabela 8 - Demonstração da adaptação do ponto flutuante parte 1.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  |  |  | **0** | **1** | **0** | **1** | **0** |
| Expoente | | | **Significando** | | | | |

Depois é separado o expoente, o expoente deve sempre ser somado com 4, ou seja, 4 + exp. Essa notação é chamada de Notação de excesso que é utilizada no padrão IEEE 754, logo, para o exemplo em questão ficaria 2 + 4 = 6, esse número em binário é igual a 110.

Tabela 9 - Demonstração da adaptação do ponto flutuante parte 2.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| *1* | *1* | *0* | **0** | **1** | **0** | **1** | **0** |
| *Expoente* | | | **Significando** | | | | |

Logo abaixo podemos ver a estrutura do somador de ponto flutuante na RLT view:

Uma imagem contendo captura de tela

Descrição gerada com muito alta confiança

Figura 19 - RTL view do componente Somador de Ponto Flutuante gerado pelo Quartus.

## Memória de Dados

Uma imagem contendo captura de tela

Descrição gerada com alta confiança

Figura 20 - Trecho de código da Memória de Dados.

Podemos ver acima um trecho do código da memória de dados que demostra todas as entradas e saídas. Basicamente ela funciona da seguinte maneira: quando houver um ciclo de borda alta no clock a escrita será habilitada, logo após o dado datain é armazenado no endereco estabelecido, no próximo ciclo de borda alta em que que o endereco lido for o estabelecido obteremos o valor contido naquele endereço no dataout. Abaixo podemos ver a RTL view do componente especificado.

Uma imagem contendo captura de tela

Descrição gerada com muito alta confiança

Figura 21 - RTL view do componente Memória de dados gerado pelo Quartus.

## Unidade de Controle

Uma imagem contendo texto

Descrição gerada com muito alta confiança

Figura 22 - Trecho de código da Unidade de Controle.

Acima podemos ver um trecho do código da Unidade de Controle do processador NEO. A unidade de controle é componente responsável por ativar as flags e manter o fluxo correto das instruções, podendo suportar instruções do tipo R e J, e capaz de executar os estados básicos do mips, são eles: busca, decodificação, execução, cálculo de memória e escrita no registrador. Abaixo podemos ver na Figura 23 a RTL view e na Figura 24 a máquina de estados da unidade de controle.

Uma imagem contendo texto, mapa

Descrição gerada com muito alta confiança

Figura 23 - RTL view do componente Unidade de Controle gerado pelo Quartus.

Uma imagem contendo mesa

Descrição gerada com alta confiança

Figura 24 - Máquina de Estados da Unidade de Controle gerada pelo Quartus.

## Bloco Operativo

Uma imagem contendo captura de tela, texto

Descrição gerada com muito alta confiança

Figura 25 - Trecho de código do Bloco Operativo.

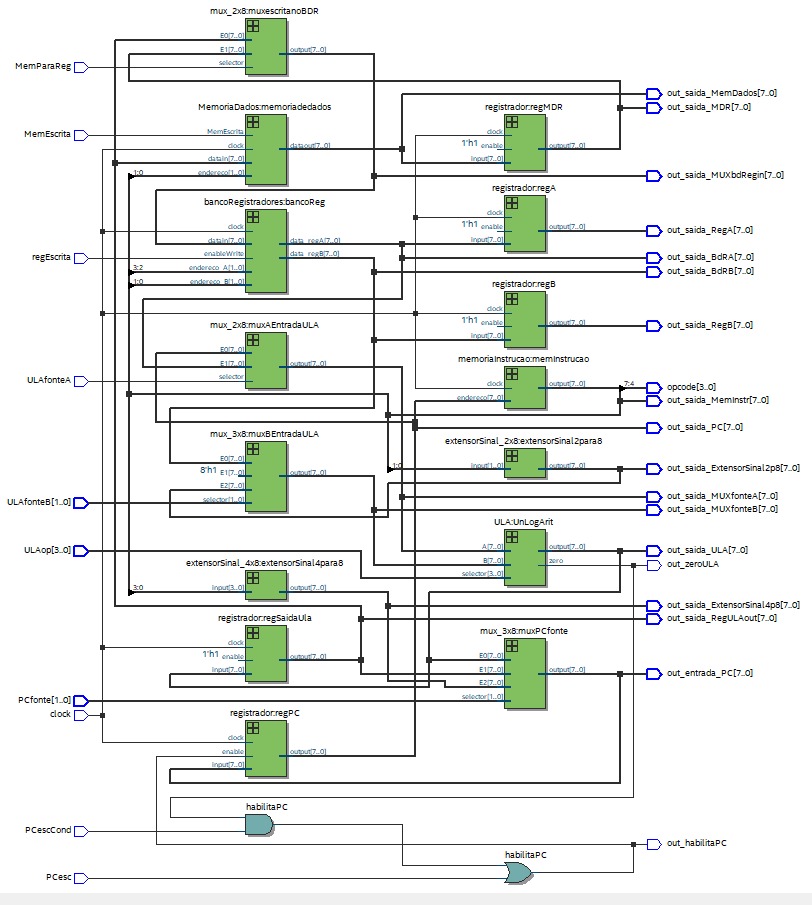


Figura 26 - RTL view do componente Bloco Operativo gerado pelo Quartus.

# Datapath

É a conexão entre 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 tipos de instruções. Para o processador NEO foi decido colocar a memória de dados e a memória de instruções, pois ambas possibilitam um gerenciamento melhor dos testes facilitando no momento da execução dos algoritmos que foram codificados.

## Datapath idealizado

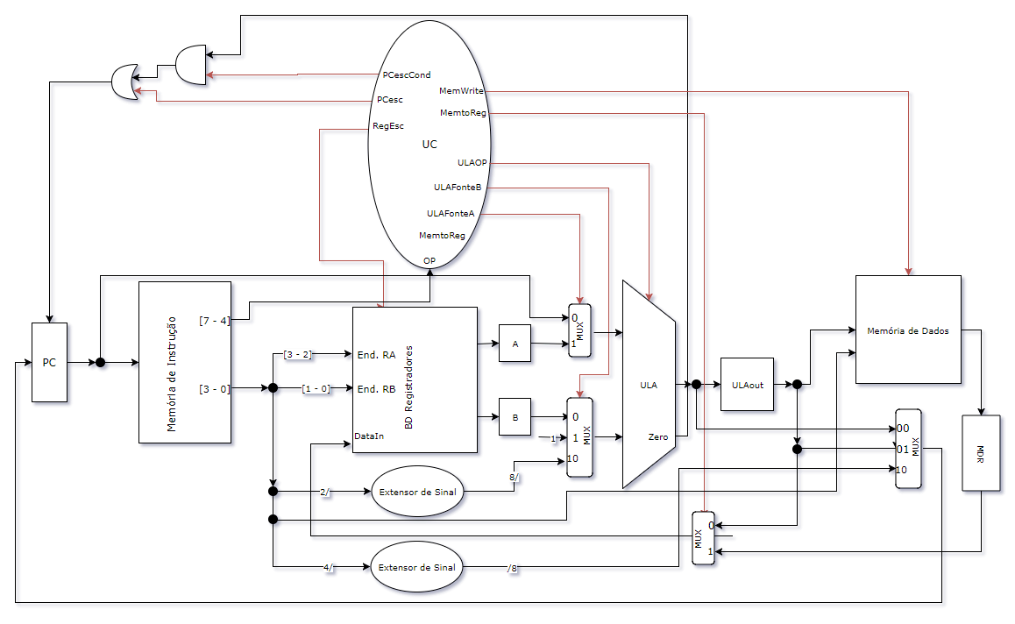


Figura 27 - Datapath idealizado após a construção dos componentes.

## Datapath

Uma imagem contendo texto, captura de tela

Descrição gerada com muito alta confiança

Figura 28 - Trecho de código datapath do processador NEO.

Uma imagem contendo captura de tela

Descrição gerada com alta confiança

Figura 29 - RTL view do Datapath gerado pelo Quartus.

# 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 NEO utilizaremos como exemplo o código para calcular o N-ésimo termo de uma P.A, o fatorial de 3 e a soma de números de ponto flutuante. Os testes podem ser verificados logo abaixo.

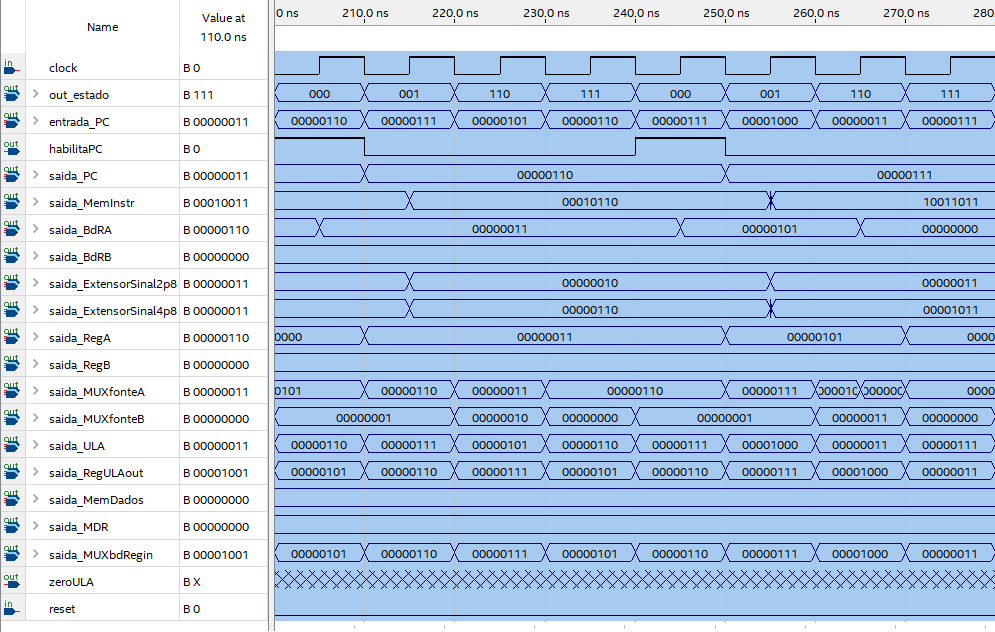
## N-ésimo termo de uma P.A:

Tabela 10 - Código N-ésimo termo de uma P.A. para o processador NEO.

|  |  |  |
| --- | --- | --- |
| Código | Endereço | Binário |
| main: movi $s1, 3 | 0000 | 1001 00 11​ |
| addi $s1, 3​ | 0001 | 0001 00 11​ |
| addi $s1, 3 | 0010 | 0001 00 11​ |
| addi $s1, 1​ | 0011 | 0001 00 01​ |
| movi $s2, 3 | 0100 | 1001 01 11 |
| addi $s2, 2 | 0101 | 0001 01 10 |
| movi $s3, 3 | 0110 | 1001 10 11 |
| loop: add $s1, $s3​ | 0111 | 0000 00 10 |
| subi $s2, 1 | 1000 | 0011 01 01 |
| eq $s2, 0​ | 1001 | 0111 01 00 |
| bne loop | 1010 | 1101 01 11​ |

2

1



3 

4 

­Figura 30 - Waveform do teste do algoritmo N-ésimo termo de uma P.A.

Na porta out\_estado, podemos verificar em qual estado a instrução está, neste exemplo temos um trecho de onde é executado duas instruções do tipo R, no quadro 1, podemos verificar que a instrução (addi $s1, 2) que está no estado S0 (busca da instrução e incremento do PC), passa pelo S1 (decodificação da instrução), passa pelo S6 (Execução da Instrução) e termina em S7 (Escrita no Registrador). O mesmo pode ser observado no quadro 2, onde começa a segunda instrução (movi $s2, 3). No estado S6 (Execução da Instrução), podemos verificar na porta saída\_ULA, no quadro 3 que o resultado da primeira instrução é cinco, e no quadro 4 é onde podemos verificar que o resultado da segunda instrução é três.

Uma imagem contendo captura de tela

Descrição gerada com muito alta confiança

5

Figura 31 - Waveform do teste do algoritmo N-ésimo termo de uma P.A.

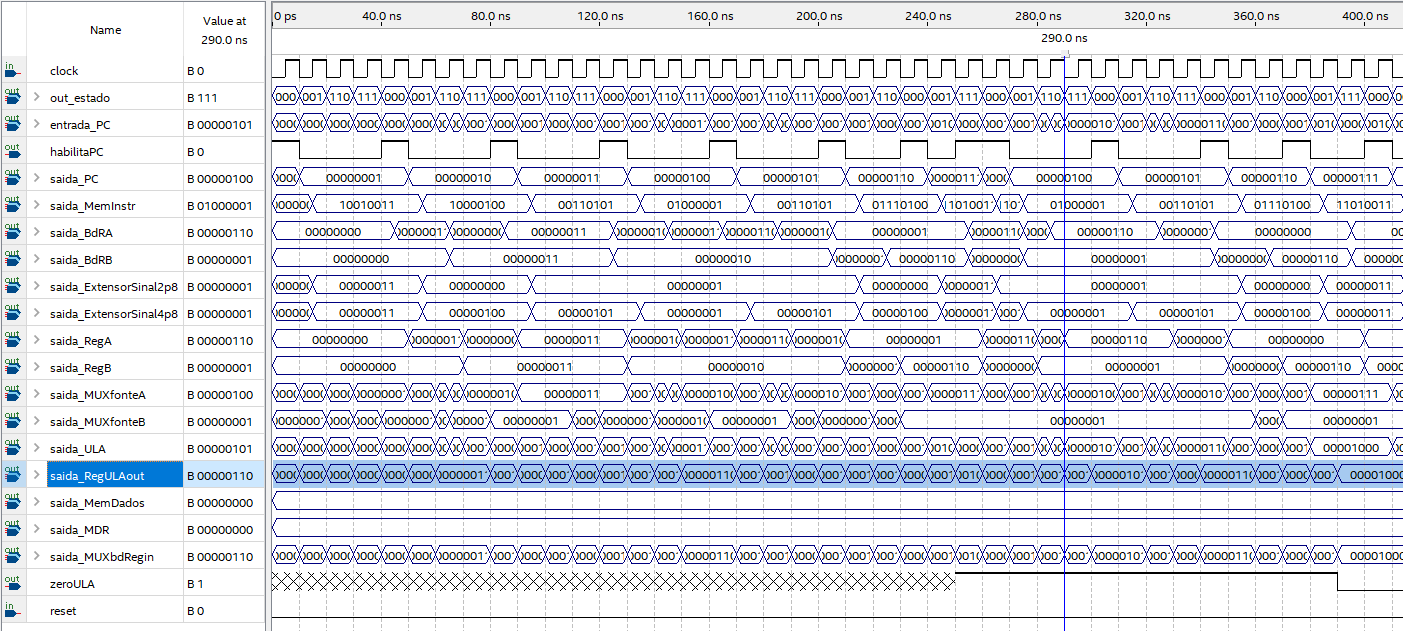
No quadro **5** podemos analisar na porta **saída\_RegULAout** o resultado referente ao algoritmo do N-ésimo termo de uma P.A. Este algoritmo tem duração de 980ns.

## Fatorial:

Tabela 11 - Código Fatorial para o processador NEO.

|  |  |  |
| --- | --- | --- |
| Código | Endereço | Binário |
| main: movi $s0, 3;​ | 0000 | 1001 00 11​ |
| move $s1, $s0;​ | 0001 | 1000 01 00 |
| subi $s1, 1;​ | 0010 | 0011 01 01​ |
| loop: mult $s0, $s1;​ | 0011 | 0100 00 01 |
| subi $s1, 1;​ | 0100 | 0011 01 01 |
| eqi $s1,0;​ | 0101 | 0111 01 00​ |
| bne loop; | 0110 | 1101 0110 |
| exit: ​ | - | - |

Logo abaixo na Figura 32, podemos verificar os testes realizado com este algoritmo. No quadro **1** na porta **saída\_RegULAout** podemos verificar o resultado obtido ao realizar o fatorial de 3. Neste algoritmo o tempo de duração de 400ns.

****

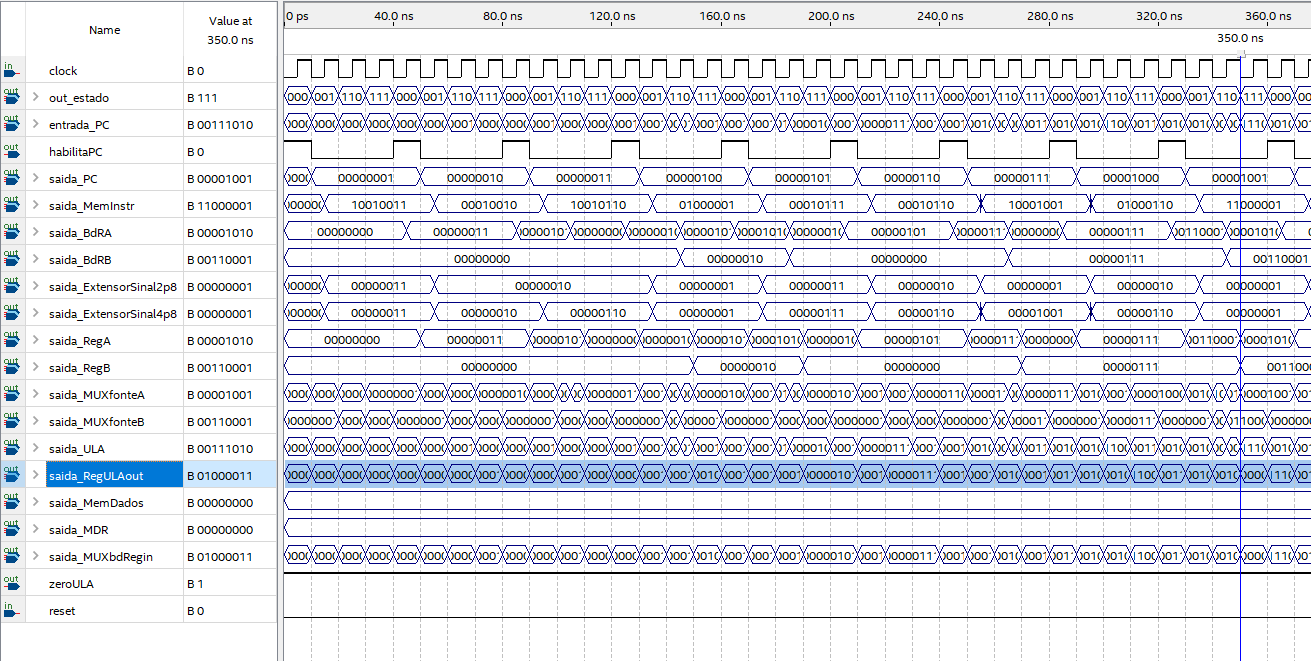
1

Figura 32 - Waveform do teste do algoritmo para calcular fatorial.

## Soma Ponto Flutuante:

Tabela 12 - Código para calcular a soma de ponto flutuante do processador NEO.

|  |  |  |
| --- | --- | --- |
| Código | Endereço | Binário |
| main: movi $s0, 3;​ | 0000 | 1001 00 11​ |
| addi $s0, 2;​ | 0001 | 0001 00 10 |
| movi $s1, 2;​ | 0010 | 1001 01 10​ |
| mult $s0, $s1;​ | 0011 | 0100 00 01 |
| addi $s1, 3;​ | 0100 | 0001 01 11 |
| addi $s1, 2;​ | 0101 | 0001 01 10​ |
| move $s2, $s1; | 0110 | 1000 10 01 |
| mult $s1, $s2; | 0111 | 0100 01 10 |
| addf $s0, $s1; | 1000 | 1100 00 01 |

****

1

Figura - Waveform do teste do algoritmo de Soma de Ponto Flutuante.

No quadro **1** podemos analisar o resultado na porta **saída\_RegULAout** é 0,28 em decimal.

# Considerações finais

Este trabalho apresentou o projeto e implementação do processador de 8 bits denominado de NEO que é a junção dos conhecimentos adquiridos durante o período de ensino da disciplina Arquitetura e Organização de Computadores. Os conhecimentos passados pelo professor Herbert, foram suficientes para concluir a construção do processador com êxito.

O nome escolhido para o processador faz referência a trilogia matrix que tem como protagonista o personagem Neo interpretado pelo ator Keanu Reeves que possui a seguinte simbologia e significado: A palavra ‘Neo’ é um [anagrama](https://pt.wikipedia.org/wiki/Anagrama) da palavra ‘One’. Neo também é uma palavra latim que significa "novo", sugerindo assim uma pista para a sua missão na Matrix. Ainda no filme Neo é a pessoa escolhida da profecia, onde essa pessoa é chamada de “O escolhido” (The One). Por guarda tantos significados e por se tratar de uma referência a uma excelente trilogia escolhemos o nome NEO para o processador de 8 bits.