

**PODER EXECUTIVO**

**MINISTÉRIO DA EDUCAÇÃO**

**UNIVERSIDADE FEDERAL DE RORAIMA**

**DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO**

**ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES**

**RELATÓRIO DO PROJETO: PROCESSADOR NAV**

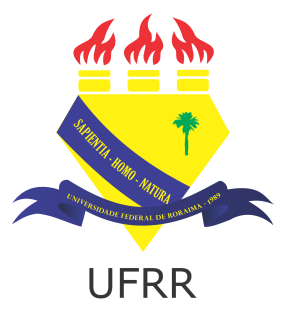
**ALUNOS:**

**Vinícius Nolêto de Araújo – 2020020185**

**Nataly Almeida dos Santos – 2019016330**

**Dezembro 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 NAV**

**Dezembro de 2022**

**Boa Vista/Roraima**

**Resumo**

Este trabalho aborda as soluções e métodos feitos para a construção do processador nav, desenvolvido por meio da ferramenta chamada *Logisim* (simulador digital de circuitos lógicos).

Os componentes que serão mostrados neste relatório servirão para base de estudos para alunos da Ciência da Computação juntamente com sua aplicação deste mesmo processador, contando também com avaliação da disciplina de Arquitetura e Organização de Computadores (AOC), ministrada pelo professor Herbert Oliveira Rocha.

**Conteúdo**

[1 Especificação 8](#_Toc17291)

[1.1 Plataforma de desenvolvimento 8](#_Toc3128)

[1.2 Conjunto de instruções 8](#_Toc10545)

[1.2.1 Formato de Instrução 8](#_Toc2553)

[1.2.2 Classificações de Instrução 9](#_Toc10874)

[1.3 Descrição do Hardware 10](#_Toc662)

[1.3.1](#_Toc1748) *[Clock](#_Toc1748)* [10](#_Toc1748)

[1.3.2](#_Toc26647) *[Flip-Flop D](#_Toc26647)* [10](#_Toc26647)

[1.3.3 Multiplexador 11](#_Toc32586)

[1.3.4 Demultiplexador 12](#_Toc14833)

[1.3.5 Valor de memória 8 bits 12](#_Toc13738)

[1.3.6 Linha de Comando 16 bits 13](#_Toc31267)

[1.3.7 PC 14](#_Toc5566)

[1.3.8 Memória ROM 15](#_Toc6154)

[1.3.9 Unidade de Controle 16](#_Toc8511)

[1.3.10 Banco de Registradores 18](#_Toc26374)

[1.3.11 Unidade de Controle de Operações Aritméticas Bit a Bit 19](#_Toc31853)

[1.3.11 Unidade de Operações Aritméticas 8 bits 20](#_Toc10030)

[1.3.12 Unidade de Operações Lógicas 20](#_Toc18555)

[1.3.13 Memória RAM 21](#_Toc6296)

[1.4 Datapath 23](#_Toc548)

[1.4.1 IMP 24](#_Toc5642)

[1.4.2 DR 26](#_Toc3623)

[1.4.3 SOMC 27](#_Toc23056)

[1.4.4 SOMR 29](#_Toc10886)

[1.4.5 SUBC 30](#_Toc27073)

[1.4.6 SUBR 32](#_Toc29357)

[1.4.7 GRA 33](#_Toc15040)

[1.4.8 CRA 35](#_Toc22005)

[1.4.9 SEIG 36](#_Toc5692)

[1.4.10 SEDIF 37](#_Toc6943)

[1.4.11 SEMA 38](#_Toc2303)

[1.4.11 SEMAIG 39](#_Toc8374)

[1.4.13 SEME 40](#_Toc28866)

[1.4.14 SEMEIG 41](#_Toc18811)

[1.4.15 SEPAR 42](#_Toc31078)

[1.4.16 J 43](#_Toc15695)

[2 Simulações e Testes 44](#_Toc12488)

[2.1 Programa](#_Toc3128) *[If, Elseif](#_Toc3128)* [e](#_Toc3128) *[Else](#_Toc3128)* [4](#_Toc3128)4

[2.2 Programa](#_Toc3128) *[While](#_Toc3128)* [4](#_Toc3128)5

[2.2 Programa Fibonacci 4](#_Toc3128)5

[3 Considerações finais 4](#_Toc10798)6

**Lista de Figuras**

Figura 1 - Especificações no Logisim

FIGURA 2-Onda de Clock

FIGURA 3 - Flip-Flop D

FIGURA 4 - MULTIPLEXADOR

FIGURA 5 - DEMULTIPLEXADOR

Figura 6 – Valor de memória 8 bits / Gerado no *Logisim* 9

Figura 7 – Linha de Comando 16 bits/ Gerado no *Logisim* 10

Figura 8– PC / Gerado no *Logisim* 11

Figura 9 – Memória ROM / Gerado no *Logisim* 12

Figura 10 – Unidade de Controle / Gerado no *Logisim* 13

Figura 11 – Banco de Registradores / Gerado no *Logisim*

Figura 12 – Unidade de Controle de Operações Aritméticas Bit a Bit

FIGURA 13 -Unidade de Operações Aritméticas (UOA) / Gerado no *Logisim* 14

Figura 14 – Unidade de Operações Lógicas (UOA) / Gerado no *Logisim* 14

Figura 15 – Memória RAM/ Gerado no *Logisim*

Figura 16 – Processador NAV/ Gerado no *Logisim*

FIGURA 17 -Instrução IMP Processador NAV / Gerado no Logisim

FIGURA 18 -Instrução DR Processador NAV / Gerado no Logisim

FIGURA 19- Instrução SOMR Processador NAV / Gerado no Logisim

FIGURA 20 - Instrução SOMC Processador NAV / Gerado no Logisim

FIGURA 21 - Instrução SUBC Processador NAV / Gerado no Logisim

**Lista de Tabelas**

Tabela 1 – Tabela de formato de instrução 9

[Tabela 2 - tabela de classe de instrução e1 9](#_haapch)

[Tabela 3 - tabela de classe de instrução e1v 9](#_2nusc19)

[tabela 4 - tabela de classe de instrução e2 9](#_2nusc19)

[tabela 5 - tabela de classe de instrução e2v 10](#_2nusc19)

[tabela 6 -tabela de Classe de Instrução V 10](#_2nusc19)

[tabela 7 - Tabela Verdade Flip-Flop D Simplificada 11](#_2nusc19)

[tabela 8 -Tabela Verdade Flip-Flop D Completa 11](#_2nusc19)

[tabela 9 -tabela verdade multiplexador simplificada 11](#_2nusc19)

[tabela 10 - tabela verdade demultiplexador 12](#_2nusc19)

[tabela 11 -Tabela Verdade Flags (Saídas de Controle) Componente UNIC / Gerado a partir de Testes 17](#_2nusc19)

[tabela 12 -Tabela Verdade Saídas de Endereçamento Componente UNIC / Gerado a partir de Testes 17](#_2nusc19)

[tabela 13 -Tabela Entradas e Saídas Componente COA / Gerado a partir de Testes 19](#_2nusc19)

[tabela 14 -Tabela Entradas e Saídas Componente UOA / Gerado a partir de Testes 20](#_2nusc19)

[tabela 15 -Tabela Entradas e Saídas Componente UOA / Gerado a partir de Testes 21](#_2nusc19)

[tabela 16 -Tabela de Instruções Processador NAV / Gerado a partir de Testes 23](#_2nusc19)

[tabela 17 -Tabela de Datapath de Instruções Processador NAV / Gerado a partir de Testes 24](#_2nusc19)

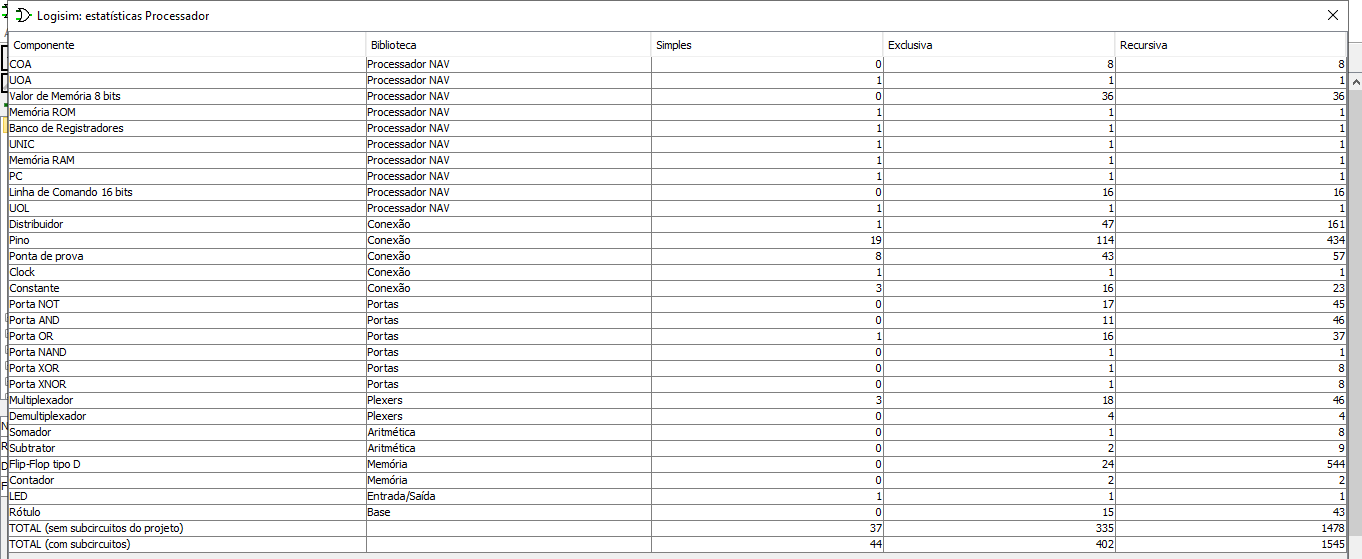
[tabela 18 -Tabela Programa If, Elseif e Else Processador NAV / Gerado a partir de Testes 44](#_2nusc19)

[tabela 19 -Tabela Programa If, Elseif e Else Processador NAV / Gerado a partir de Testes 45](#_2nusc19)

[tabela 20 -Tabela Programa Fibonacci Processador NAV / Gerado a partir de Testes 46](#_2nusc19)

# Especificação

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



**Figura 1 - Estatísticas do Processador NAV / Gerado no *Logisim***

## Plataforma de desenvolvimento

Para a implementação do processador NAV foi utilizado a IDE *Logisim.* O sistema é um simulador lógico que permite o desenho e a simulação de circuitos através de uma interface gráfica.

## Conjunto de instruções

Para o funcionamento do Processador NAV, de forma que ele possa realizar as operações matemáticas requisitadas, há regras e classificações quanto a formatação das instruções.

## 1.2.1 Formato de Instrução

O processador NAV possui 4 registradores, nomeados R1, R2, R3 e R4, 16 espaços de memória, nomeados M1[0], M1[1], M1[2], M1[3], M2[0]... M3[0]... M4[3], e 16 instruções de 8 bits, onde os bits são distribuídos da seguinte forma:

* **Código de Operação**: Também conhecido como “**OP CODE**” ou apenas “**OP**” é o valor relacionado a qual operação será realizada pelo Processador NAV, logo, pode ser considerado um *magic number* ou seja, um valor que essencialmente não possui relação com o resultado, mas é estabelecida que uma relação entre os mesmo para o funcionamento ideal do sistema;
* **Endereço 1**: Corresponde ao primeiro endereço utilizado na instrução, que normalmente está relacionado com um registrador, sendo este o registrador-destino dos resultados das operações aritméticas;
* **Endereço 2**: Corresponde ao segundo endereço utilizado na instrução, servindo para comparações, operações aritméticas e alocação de memória;

**Tabela 1 – Tabela de Formato de Instrução**

|  |  |  |
| --- | --- | --- |
| **4 Bits** | **2 Bits** | **2 Bits** |
| **Bits: 7-4** | **Bits: 3-2** | **Bits: 1-0** |
| **Código de Operação** | **Endereço 1** | **Endereço 2** |

## 1.2.2 Classificações de Instrução

As instruções do Processador NAV podem ser divididas em cinco classes, E1, E1V, E2, E2V e V, onde os bits são distribuídos da seguinte forma:

* **Tipo E1**: São instruções que usam apenas um endereço para a operação, logo apenas seis bits são úteis e os dois últimos bits não são utilizados, além disto não há valor de entrada. Um exemplo de instrução do tipo **E1** é a instrução **IMP.**

**Tabela 2 – Tabela de Classe de Instrução E1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Código de Operação** | **Endereço 1** | **Endereço 2** | **Valor de Entrada** |
| **XXXX** | **YY** | **--** | **--** |

* **Tipo E1V**: São instruções que usam apenas um endereço para a operação, logo apenas seis bits são úteis e os dois últimos bits não são utilizados, além disto não há valor de entrada. Um exemplo de instrução do tipo **E1V** é a instrução **DR.**

**Tabela 3 – Tabela de Classe de Instrução E1V**

|  |  |  |  |
| --- | --- | --- | --- |
| **Código de Operação** | **Endereço 1** | **Endereço 2** | **Valor de Entrada** |
| **XXXX** | **YY** | **--** | **ZZZZ ZZZZ** |

* **Tipo E2**: São instruções que usam dois endereços para a operação e não há valor de entrada. Um exemplo de instrução do tipo **E2** é a instrução **SOMR.**

**Tabela 4 – Tabela de Classe de Instrução E2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Código de Operação** | **Endereço 1** | **Endereço 2** | **Valor de Entrada** |
| **XXXX** | **YY** | **WW** | **--** |

* **Tipo E2V**: São instruções que usam dois endereços para a operação e há valor de entrada, logo é formato de instrução mais complexo do Processador NAV, pois usa a maior quantidade de bits. Um exemplo de instrução do tipo **E2V** é a instrução **SEIG.**

**Tabela 5 – Tabela de Classe de Instrução E2V**

|  |  |  |  |
| --- | --- | --- | --- |
| **Código de Operação** | **Endereço 1** | **Endereço 2** | **Valor de Entrada** |
| **XXXX** | **YY** | **WW** | **ZZZZ ZZZZ** |

* **Tipo V**: São instruções que usam apenas o valor de entrada para operação. Um exemplo de instrução do tipo **V** é a instrução **J.**

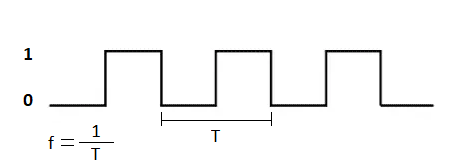
**Tabela 6 – Tabela de Classe de Instrução V**

|  |  |  |  |
| --- | --- | --- | --- |
| **Código de Operação** | **Endereço 1** | **Endereço 2** | **Valor de Entrada** |
| **XXXX** | -- | **--** | **ZZZZ ZZZZ** |

## Descrição do Hardware

Nesta secção são descritos os componentes do hardware que compõem o Processador NAV, incluindo uma descrição de suas funcionalidades, valores de entrada e saída.

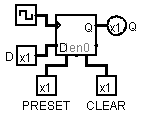
## *Clock*



**Figura 2 - Onda de Clock / Imagem gerada no *Paint* para ilustração**

O ***Clock*** determina o momento de atualização de estado do processador, alternando entre os valores 0 e 1 em uma determinada frequência de tempo, como pode ser visto na Figura X, que demonstra a onda quadrática que representa os valores de ***Clock*** em relação ao tempo.

## *Flip-Flop D*



**Figura 3 - *Flip-Flop* *D* / Gerado no *Logisim***

A estrutura do ***Flip-Flop D*** pode ser descrita como um ***Flip-Flop RS*** onde S possui o valor de ~R, de forma que o valor do *Flip-Flop*, mantém-se igual independentemente do valor de *Clock* e do valor anterior de saída Q (Qant) gerando a tabela Tabela X que, após simplificação pode ser reescrita como a Tabela X.

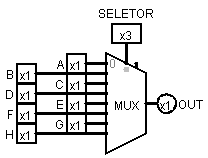
**Tabela 7 – Tabela Verdade *Flip-Flop D* Completa**

|  |  |  |
| --- | --- | --- |
| **D** | **Qant** | **Qs** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**Tabela 8 – Tabela Verdade *Flip-Flop D* Simplificada**

|  |  |
| --- | --- |
| **D** | **Qs** |
| 0 | 0 |
| 1 | 1 |

## Multiplexador



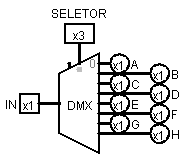
**Figura 4 - Multiplexador / Gerado no *Logisim***

O **Multiplexador** é um componente de controle de fluxo, onde a quantidade de entradas é igual a *2^(quantidade bits do seletor)* e o valor da saída corresponde ao endereço do seletor, como pode ser observado na Tabela X.

**Tabela 9 – Tabela Verdade Multiplexador Simplificada**

|  |  |
| --- | --- |
| **SELETOR** | **OUT** |
| **000** | A |
| **001** | B |
| **010** | C |
| **011** | D |
| **100** | E |
| **101** | F |
| **110** | G |
| **111** | H |

## Demultiplexador



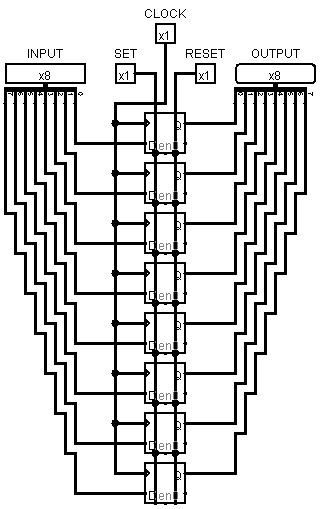
**Figura 5 -Demultiplexador / Gerado no *Logisim***

O **Demultiplexador** é um componente de controle de fluxo, onde a quantidade de saídas é igual a *2^(quantidade bits do seletor)* e possui apenas um único valor de entrada, de forma que os valores de saída serão iguais à entrada apenas quando o valor do seletor corresponder ao endereço do mesmo, como pode ser observado na Tabela X.

**Tabela 10 – Tabela Verdade Demultiplexador**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SELETOR** | **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** |
| **000** | IN | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **001** | 0 | IN | 0 | 0 | 0 | 0 | 0 | 0 |
| **010** | 0 | 0 | IN | 0 | 0 | 0 | 0 | 0 |
| **011** | 0 | 0 | 0 | IN | 0 | 0 | 0 | 0 |
| **100** | 0 | 0 | 0 | 0 | IN | 0 | 0 | 0 |
| **101** | 0 | 0 | 0 | 0 | 0 | IN | 0 | 0 |
| **110** | 0 | 0 | 0 | 0 | 0 | 0 | IN | 0 |
| **111** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | IN |

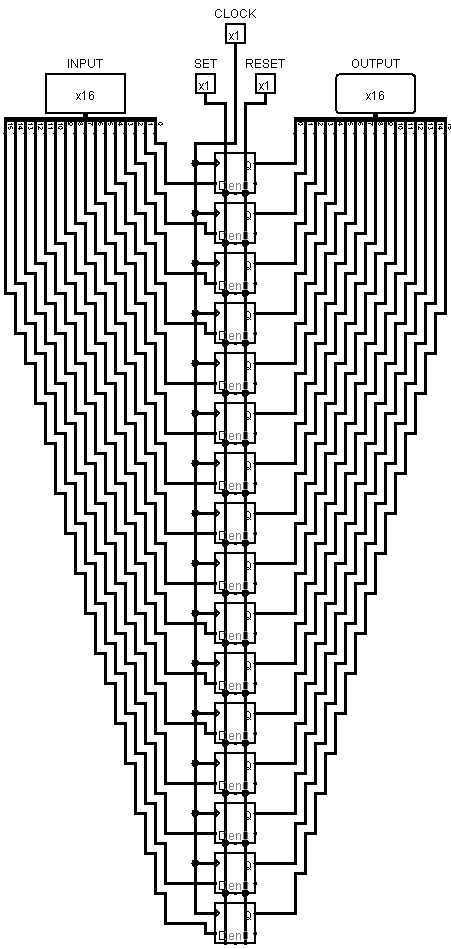
## 1.3.5 Valor de memória 8 bits

****

**Figura 6 – Valor de memória 8 bits / Gerado no *Logisim***

O componente **Valor de Memória** **8bits** está presente em diversas partes do Processador NAV, pois é capaz de armazenas 8 bits ordenados que representam um valor específico. O **Valor de Memória** é composto por quatro entradas, sendo elas: *INPUT*, *SET*, *RESET* e, por último, *Clock*, a entrada *INPUT* possui 8 bits de tamanho, referente ao valor a ser armazenado, as entradas *SET* e *RESET* são responsáveis por igualar todos os bits guardados a 1, no caso do *SET*, e a 0, no caso do *RESET*, e a entrada *Clock* corresponde ao *Clock* do processador, por outro lado há apenas uma saída no componente, o *OUTPUT*, que corresponde ao valor de oito bits armazenado. A estrutura do componente é composta por oito *Flip-Flops* *D* encadeados, onde suas entradas D estão conectadas ao distribuidor da entrada *INPUT*, que divide a mesma em oito conexões de um bit, e as saídas Q dos *Flip-Flops* *D* são conectadas com o distribuidor conectado a saída *OUTPUT*, formando, desta forma uma saída de oito bits. As entradas *SET* e *RESET* estão conectadas às entradas *PRESET* e *CLEAR* dos *Flip-Flops* *D* respectivamente. A entrada *Clock* é conectada à entrada de mesmo nome dos *Flip-Flops* *D*. O funcionamento do componente se dá pelo valor da entrada *INPUT*, que é armazenado nos *Flip-Flops* *D* apenas quando o *Clock* é acionado e seu valor é imediatamente enviado para o *OUTPUT*.

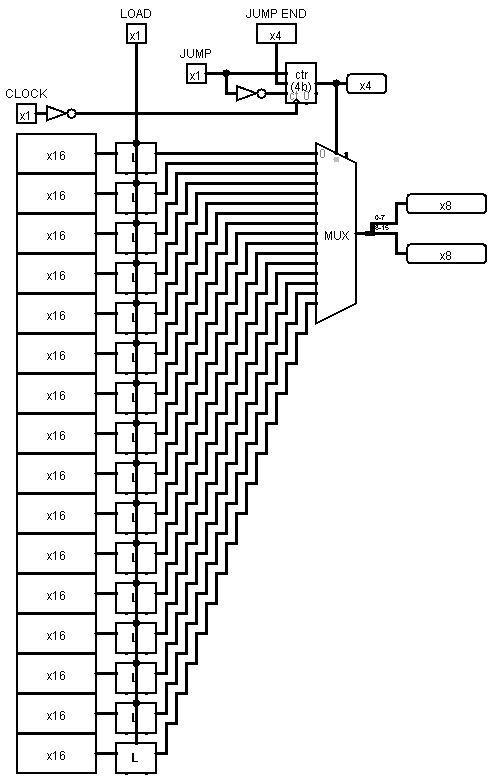
## 1.3.6 Linha de Comando 16 bits



**Fig*ura* 7 *– Linha de Comando 16 bits/ Gerado no Logisim***

O componente **Linha de Comando** **16 bits** é utilizada no componente **PC**, definido no ponto **1.3.3**, e a mesma possui uma estrutura parecida com o componente **Valor de Memória 8bits**, porém possui o dobro de bits, pois a mesma está relacionada com a instrução a ser lida pelo Processador NAV e o valor inteiro enviado em conjunto, de forma que os oito primeiros bits representam o valor (valor este em binário que representa no máximo o valor 255 em decimal) e os oito últimos representam a instrução que é dividida em quatro bits para código de operação e quatro bits para endereçamento e, além do valor de tamanho de bits, a entrada conectada ao *Clock* dos *Flip-Flops* *D* não é a entrada de *Clock*, mas sim a entrada chamada *LOAD*. A sua estrutura é composta por dezesseis *Flip-Flops* *D* encadeados, de forma que, assim como no componente **Valor de Memória** **8bits** a entrada *INPUT* é conectada a um distribuidor que conecta-se com todas as entradas D dos *Flip-Flops* *D* e suas saídas conectadas ao distribuidor da saída *OUTPUT*, possibilitando o armazenamento de um valor de 16bits.

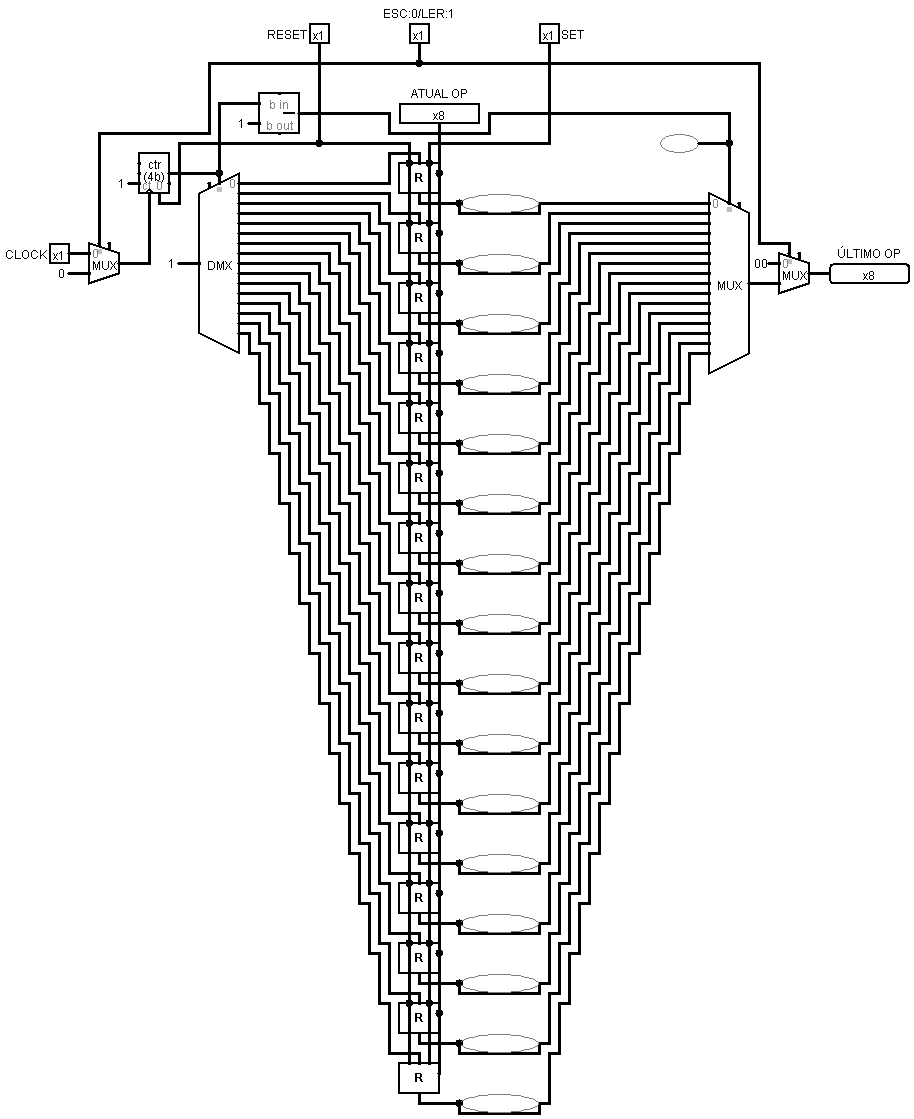
## 1.3.7 PC

****

**Figura 8 – PC / Gerado no *Logisim***

O componente **PC** é análogo ao conjunto de instruções de um programa que, como determinado pela arquitetura de *Von Neumann*, deve possuir o programa carregado em memória para que possa ser executado, prevenindo alterações durante a execução, logo possui um conjuntos de dezesseis **Linhas de Comando 16bits**, de forma a estruturar em linhas endereçáveis o programa executado. O componente possui quatro entradas, *CLOCK*, *JUMP*, *JUMP END* e *LOAD*, e duas saídas, VALOR e INSTRU, onde as entradas *JUMP* e *JUMP END* correspondem à próxima **Linha de Comando** que deve ser enviada para as saídas, a entrada *Clock* corresponde à negação do *Clock* do processador e a entrada *LOAD* é conectada a entrada de mesmo nome nas **Linhas de Comando** encadeadas, as saídas VALOR e INSTRU são referentes aos oito primeiros bits e aos oito últimos bits , respectivamente, da **Linha de Comando** lida. O funcionamento do **PC** se dá pela negação do *Clock*, de forma que a contagem de linha de comando possa ser realizada apenas após a realização do determinado comando, este valor é conectado a um Contador de 4 bits, incrementando-o, porém, caso a entrada *JUMP* seja igual a 1, o valor da entrada *JUMP END* é carregado no Contador e enviado para o seletor do Multiplexador, desta forma o valor de saída do **PC** para cada ciclo de *Clock* é dado de forma ordenada, porém com saltos determinados por instruções específicas, gerando um fluxo programável de leitura.

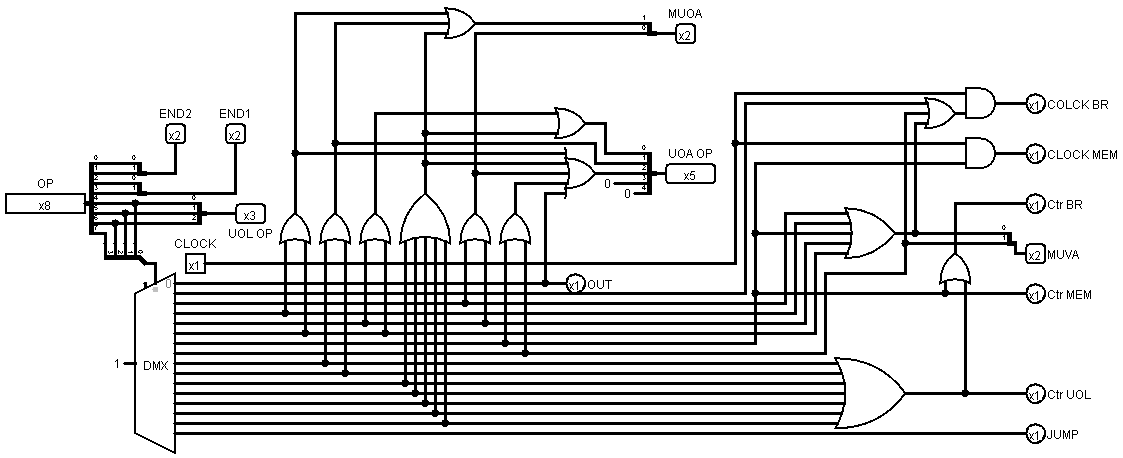
## 1.3.8 Memória ROM



**Figura 9 – Memória ROM / Gerado no *Logisim***

A Memória ROM é um componente que não possui utilidade para o Processador NAV, no quesito de processamento de instruções, porém foi acrescentado para o entendimento do funcionamento e posicionamento usual do mesmo. O componente possui quatro entradas, ATUAL OP, *SET*, *RESET* e *CLOCK*, e apenas uma saída, ULT OP*.* Sua estrutura possui dezesseis **Valores de Memória** encadeados, onde suas entradas *CLOCK* estão conectados ao Multiplexador conectado ao valor constante igual a “1”, suas entradas de *INPUT* estão conectadas à entrada ATUAL OP, suas entradas *SET* e *RESET* estão conectadas às entradas de mesmo da **Memória ROM** e suas saídas *OUTPUT* estão conectadas ao demultiplexador conectado a saída ULT OP. O Seletor do Multiplexador é conectado no valor de saída do Contador 4bits, que é incrementado pela entrada *CLOCK*, de forma que o endereço salvo na Memória ROM possa ser salvo de forma ordenada, por outro lado o valor do seletor do demultiplexador é dado pelo valor do contador subtraído de 1bit, logo, o valor do endereço anteriormente armazenado. Outros processadores usam da **Memória ROM** para basear o processamentos de instruções na instruções realizadas anteriormente e para guardar programas anteriormente compilados para evitar re-compilação, porém, como no Processador NAV este valor não é utilizado, sua saída é usada apenas para demonstração do armazenamento do último valor de instrução.

## 1.3.9 Unidade de Controle

****

**Figura 10 – Unidade de Controle / Gerado no *Logisim***

A Unidade de Controle, ou **UNIC**, é responsável por controlar o fluxo de dados do processador, utilizando diversas saídas que são conectadas a entradas de controle e de endereçamento de outros componentes. A **UNIC** possui duas entradas, o OP, que é um valor de 8bits que seleciona a instrução a ser realizada e o *CLOCK*, que corresponde ao *Clock* do processador, e treze saídas, dentre elas, sete *flags*, ou seja, saídas de controle, nomeadas *CLOCK* BR, *CLOCK* MEM, CTR BR, CTR MEM, *OUT*, CTR UOL e *JUMP*, e seis saídas de endereçamento, nomeadas END1, END2, UOL OP, MULA, UOA OP, MUVA.

As *flags* são saídas que controlam o funcionamento de outros componentes, logo, seus valores estão relacionados com o fluxo de funcionamento das instruções e seus **Códigos de Instrução**, ou ***OP CODE***, por isto, os seus valores correspondem a tabela verdade Tabela X. As saídas *CLOCK* BR e CTR BR estão conectadas com o **Banco de Registradores**, **BR**, de forma que a saída *CLOCK* BR está conectada com a entrada *CLOCK* e a saída CTR BR está conectada com a entrada ESC:0/LER:1, de forma que apenas quando o valor de *CLOCK* BR é igual a 1 que o *Clock* do **BR** pode ser realizado e, caso o valor de CTR BR seja igual a 0 o **BR** irá realizar a escrita de um valor e, caso seja igual a 1, irá realizar apenas a leitura de dois registradores. As saídas *CLOCK* MEM e CTR MEM funcionam de forma parecida com as saídas *CLOCK* BR e CTR BR, porém as mesmas são conectadas à **Memória RAM**, que possui uma única diferença em suas entradas, na sua relação de escrita e leitura, pois o valor 0 lê valores e o valor 1 escreve valores. O valor *OUT* de saída é utilizado no Multiplexador de impressão, de forma que, quando verdadeiro, a saída *PRINT* é igual ao valor do registrador escolhido. A saída CTR UOL é conectada à entrada CLOCK do componente **UOL** , de forma que apenas quando esta saída CTR UOL seja igual a 1 o valor calculador pela **UOL** será enviado para a saída *BOOLEAN* da mesma. Por último a saída *JUMP* é enviada diretamente para a entrada de mesmo nome do **PC**.

**Tabela 11 – Tabela Verdade *Flags* (Saídas de Controle) Componente UNIC / Gerado a partir de Testes**

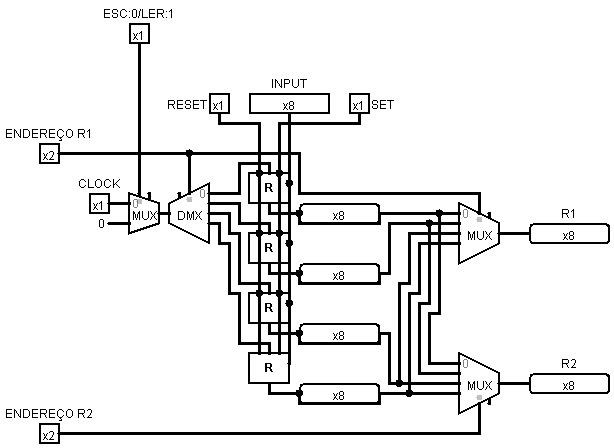
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **OP** | **CLOCK BR** | **CLOCK MEM** | **CTR BR** | **CTR MEM** | **OUT** | **CTR UOL** | **JUMP** |
| **0000XXYY** | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **0001XXYY** | *CLOCK* | 0 | 0 | 0 | 0 | 0 | 0 |
| **0010XXYY** | *CLOCK* | 0 | 0 | 0 | 0 | 0 | 0 |
| **0011XXYY** | *CLOCK* | 0 | 0 | 0 | 0 | 0 | 0 |
| **0100XXYY** | *CLOCK* | 0 | 0 | 0 | 0 | 0 | 0 |
| **0101XXYY** | *CLOCK* | 0 | 0 | 0 | 0 | 0 | 0 |
| **0110XXYY** | *CLOCK* | *CLOCK* | 1 | 1 | 0 | 0 | 0 |
| **0111XXYY** | *CLOCK* | 0 | 0 | 0 | 0 | 0 | 0 |
| **1000XXYY** | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| **1001XXYY** | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| **1010XXYY** | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| **1011XXYY** | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| **1100XXYY** | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| **1101XXYY** | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| **1110XXYY** | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| **1111XXYY** | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

As saídas de endereçamento correspondem ao endereço utilizado para seletores internos de outros componentes e para seletores de multiplexadores do processador que definem o fluxo de valores para realização de instruções, de forma que estas saídas seguem a tabela verdade da Tabela X.

**Tabela 12 – Tabela Verdade Saídas de Endereçamento Componente UNIC / Gerado a partir de Testes**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **OP** | **END.1** | **END.2** | **UOL OP** | **UOA OP** | **MUOA** | **MUVA** |
| **0000XXYY** | YY | XX | 000 | 00100 | 00 | 00 |
| **0001XXYY** | YY | XX | 001 | 00000 | 00 | 00 |
| **0010XXYY** | YY | XX | 010 | 00100 | 01 | 01 |
| **0011XXYY** | YY | XX | 011 | 00100 | 10 | 01 |
| **0100XXYY** | YY | XX | 100 | 00101 | 01 | 01 |
| **0101XXYY** | YY | XX | 101 | 00101 | 10 | 01 |
| **0110XXYY** | YY | XX | 110 | 00100 | 00 | 01 |
| **0111XXYY** | YY | XX | 111 | 00100 | 00 | 10 |
| **1000XXYY** | YY | XX | 000 | 00011 | 10 | 00 |
| **1001XXYY** | YY | XX | 001 | 00011 | 10 | 00 |
| **1010XXYY** | YY | XX | 010 | 00101 | 10 | 00 |
| **1011XXYY** | YY | XX | 011 | 00101 | 10 | 00 |
| **1100XXYY** | YY | XX | 100 | 00101 | 10 | 00 |
| **1101XXYY** | YY | XX | 101 | 00101 | 10 | 00 |
| **1110XXYY** | YY | XX | 110 | 00100 | 00 | 00 |
| **1111XXYY** | YY | XX | 111 | 00000 | 00 | 00 |

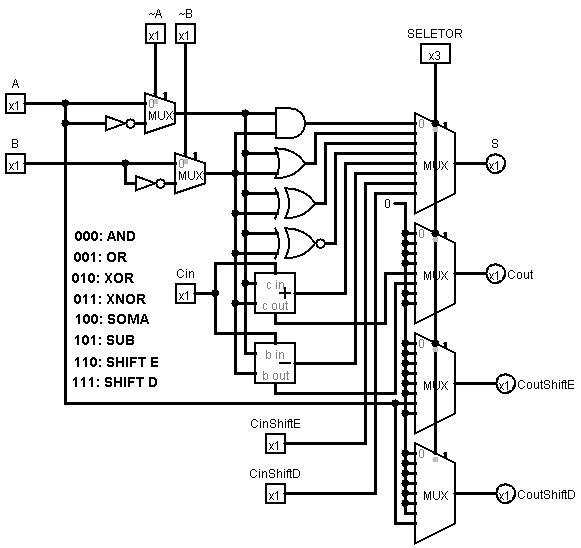
## 1.3.10 Banco de Registradores

****

**Figura 11 – Banco de Registradores / Gerado no *Logisim***

A Unidade **Banco de Registradores** armazena os registradores utilizados em todas a operações aritméticas e lógicas do Processador NAV. Possui cinco entradas principais, END 1, END 2, ESC: 0/LER: 1, *INPUT* e *CLOCK*, duas entradas menos utilizadas, *SET* e *RESET*, e duas saídas R1 e R2, onde as entradas END 1 e END 2 são entradas de endereço, possuindo 2bits cada, a entrada ESC: 0/LER: 1 é uma entrada de controle, onde quando igual a “0” um valor será escrito no momento em que o valor de *CLOCK* seja igual a “1” e quando igual a “1” independentemente do valor de *CLOCK* nenhum valor será escrito, a entrada INPUT é uma entrada de valor numérico de 8 bits utilizado para a armazenamento de registradores, a entrada *CLOCK* está relacionada com o *Clock* do processador, as entradas *SET* e *RESET* igualam todos os valores de todos os registradores a 1 e 0 respectivamente e as saídas R1 e R2 possuem 8bits, sendo referentes aos valores de dois registradores selecionados por endereços. A estrutura do **Banco de Registradores** possui quatro **Valores de Memória 8 bits** encadeados, onde as entradas *SET*, *RESET* e *INPUT* são conectadas às entradas de mesmo nome dos mesmos. A entrada ESC: 0/LER: 1 é conectada ao seletor do multiplexador conectado à entrada de *CLOCK*, de forma que apenas quando igual a “0” o valor de *CLOCK* será o valor de saída do multiplexador, que, por sua vez, é conectado ao demultiplexador que possui como seletor a entrada END 1 e suas saídas que conectam-se às entradas *CLOCK* dos **Valores de Memória 8 bits** e ,por último, as saídas dos mesmos são conectadas a dois multiplexadores que possuem como seletores as entradas END 1 e END 2 enviando desta forma os valores de dois registradores selecionados por endereços para as saídas R1 e R2.

## 1.3.11 Unidade de Controle de Operações Aritméticas Bit a Bit



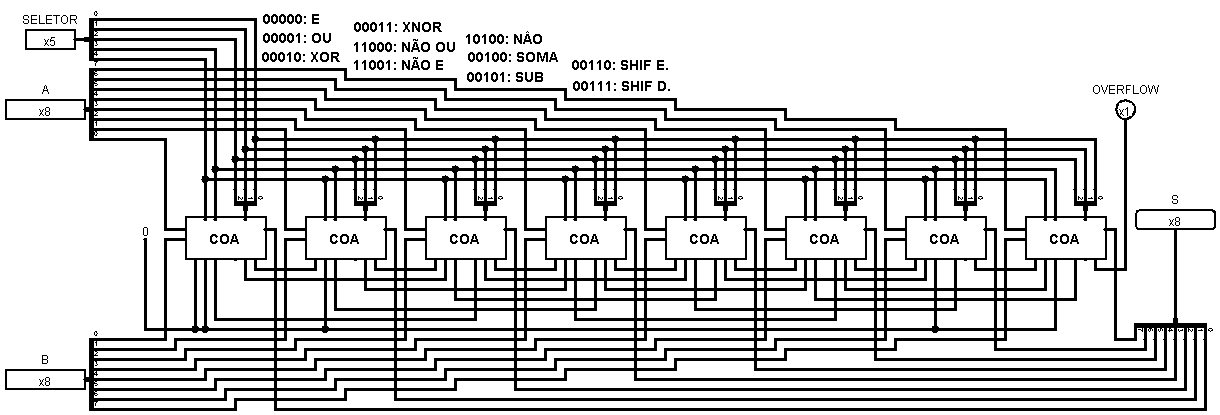
**Figura 12 – Unidade de Controle de Operações Aritméticas Bit a Bit (COA) / Gerado no *Logisim***

A **Unidade de Controle de Operações Aritméticas Bit a Bit**, ou **COA**, possui oito entradas, A, B, ~A, ~B, *Cin*, *CinShift E*, *CinShift D* e SELETOR, e quatro saídas S, *Cout*, *CoutShift E* e *CoutShift D*, de forma que as entradas A e B representam os dois valores que serão calculados pela unidade e ~A e ~B negam os valores das mesmas, sendo estas quatro entradas com o tamanho de 8 bits, as entradas *Cin*, *CinShift E*, *CinShift D* e as saídas *Cout*, *CoutShift E* e *CoutShift D* são entradas e saídas de cálculo em conjunto, utilizadas quando duas ou mais **Unidades de Controle de Operações Aritméticas Bit a Bit** são conectadas, de forma que seja possível realizar operações com dois valores com mais de um bit, a entrada SELETOR, que possui 3 bits, seleciona a operação que será realizada pelo componente e a saída S é correspondente ao valor final da operação realizada. A estrutura do componente relaciona as portas-lógicas com as entradas dos multiplexadores de forma que as operações relacionadas à um valor de entrada do SELETOR seja enviada para a saída S, desta forma os resultados da saída S correspondem a Tabela X. A saída *Cout* é conectada a um multiplexador com todas as entradas iguais a “0”, com exceção das entradas correspondentes às operações de soma e subtração, de forma que seu valor corresponde ao excedente da soma ou subtração realizada apenas quando o a operação em questão é selecionada. As saídas *CoutShift E* e *CoutShift D* são conectadas multiplexadores com todas as entradas iguais a “0”, com exceção das entradas correspondentes as suas respectivas operações, assim como no caso da saída *Cout*, entretanto os seus valores são determinados não por componentes de operação ou portas lógicas, mas sim por suas entradas dedicadas de nome *CinShift E*, *CinShift D*.

**Tabela 13 – Tabela Entradas e Saídas Componente COA / Gerado a partir de Testes**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **SELETOR** | **~A** | **~B** | **Entradas Utilizadas** | **Operação** | **Saída(s)** |
| **000** | **0** | **0** | A e B | A **E** B | S |
| **001** | **0** | **0** | A e B | A **OU** B | S |
| **000** | **1** | **1** | A e B | A **NÃO OU** B | S |
| **001** | **1** | **1** | A e B | A **NÃO E** B | S |
| **010** | **0** | **0** | A e B | A ***XOR*** B | S |
| **011** | **0** | **0** | A e B | A ***XNOR*** B | S |
| **100** | **0** | **0** | A, B e Cin | A **MAIS** B **MAIS** Cin | S e *Cout* |
| **101** | **0** | **0** | A, B e Cin | A **MENOS** B **MENOS** Cin | S e *Cout* |
| **110** | **0** | **0** | Cin Shift E | **SHIFT ESQUERDA** A **DOIS BITS** | S e *Cout Shift E* |
| **111** | **0** | **0** | Cin Shift D | **SHIFT DIREITA** A **DOIS BITS** | S e Cout Shift D |

## 1.3.11 Unidade de Operações Aritméticas 8 bits

****

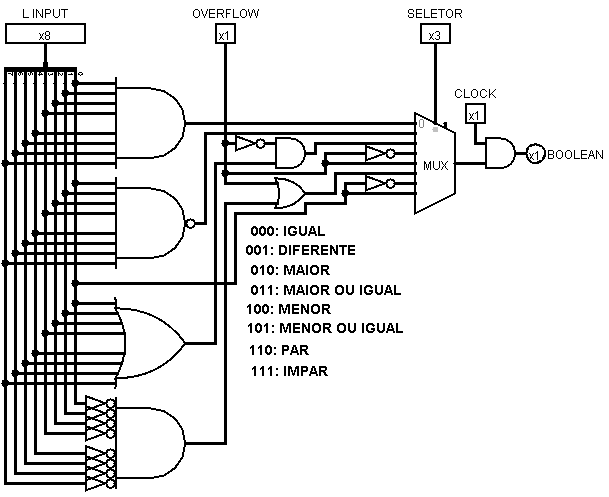
**Figura 13 – Unidade de Operações Aritméticas (UOA) / Gerado no *Logisim***

A **Unidade de Operações Aritméticas 8 bits**, ou **UOA**, é a unidade responsável pelas operações aritméticas do Processador NAV, tratando os valores por meio de oito **Unidades de Controle de Operações Aritméticas Bit a Bit** conectadas, de forma que as entradas A e B, conectam-se às entradas de mesmo nome nos componentes e a entrada SELETOR, que possui 5 bits, divide-se por meio de um distribuidor de forma que os dois primeiros bits conectam-se às entradas ~A e ~B e os três últimos conectam-se à entrada SELETOR. Os valores das saídas S das **Unidades de Controle de Operações Aritméticas Bit a Bit** conectam-se à saída de mesmo nome da **UOA**. O valor de *OVERFLOW* é igual ao valor de *Cout* do último **COA**, possibilitando a interpretação de valores como negativos. Considerando a padronização de operações contida na estrutura da **COA**, as operações realizadas pela unidade consequentemente correspondentes à Tabela X.

**Tabela 14 – Tabela Entradas e Saídas Componente UOA / Gerado a partir de Testes**

|  |  |  |  |
| --- | --- | --- | --- |
| **SELETOR** | **Entradas Utilizadas** | **Operação** | **Saída(s)** |
| **00000** | A e B | A **E** B | S |
| **00001** | A e B | A **OU** B | S |
| **11000** | A e B | A **NÃO OU** B | S |
| **11001** | A e B | A **NÃO E** B | S |
| **00010** | A e B | A ***XOR*** B | S |
| **00011** | A e B | A ***XNOR*** B | S |
| **00100** | A, B e Cin | A **MAIS** B **MAIS** Cin | S e *Cout* |
| **00101** | A, B e Cin | A **MENOS** B **MENOS** Cin | S e *Cout* |
| **00110** | Cin Shift E | **SHIFT ESQUERDA** A **DOIS BITS** | S e *Cout Shift E* |
| **00111** | Cin Shift D | **SHIFT DIREITA** A **DOIS BITS** | S e Cout Shift D |

## 1.3.12 Unidade de Operações Lógicas



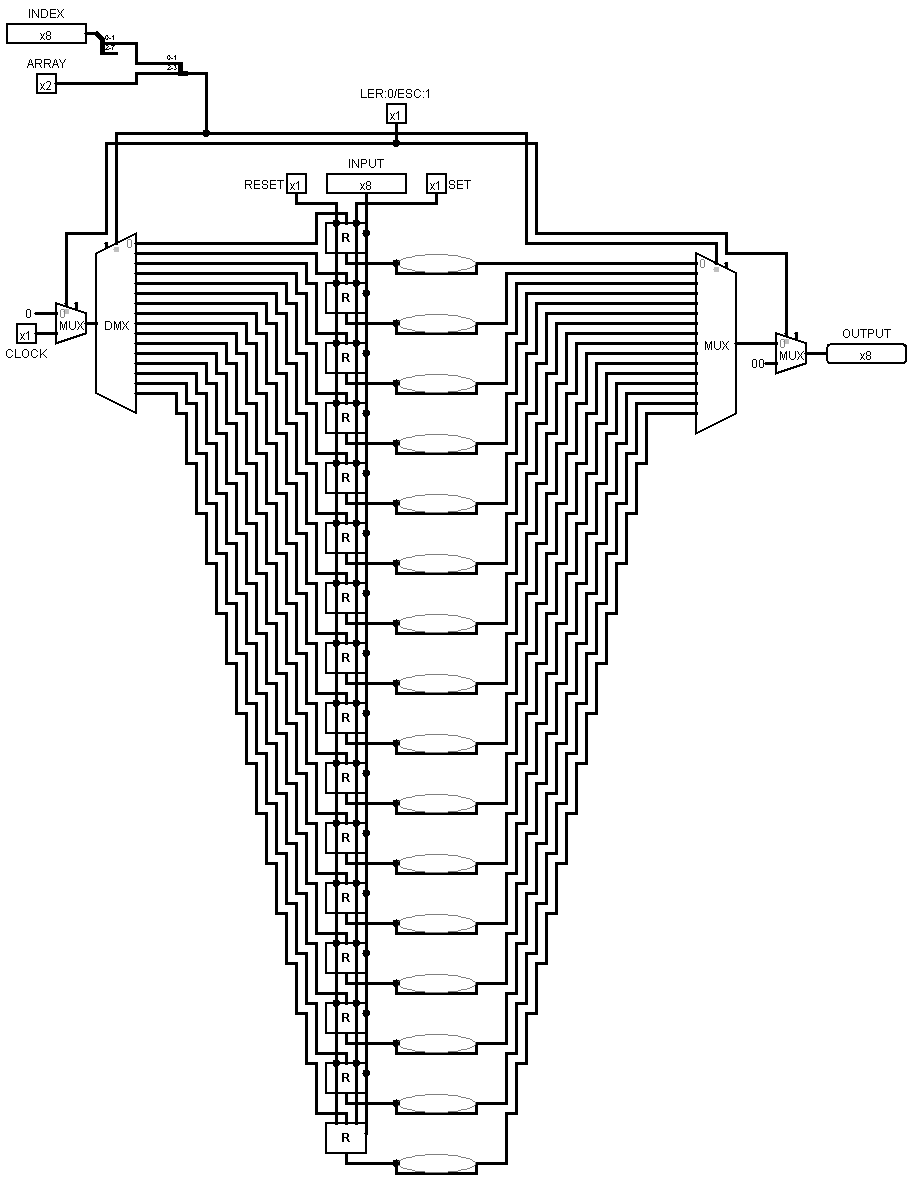
**Figura 14 – Unidade de Operações Lógicas (UOL) / Gerado no *Logisim***

A **Unidade de Operações Lógicas**, ou **UOL**, é responsável por realizar os testes booleanos de comparações, ou seja, determina de acordo com a comparação selecionada se a mesma é verdadeira ou não. A **UOL** possui quatro entradas, L INPUT, OVERFLOW, SELETOR e *CLOCK* e apenas uma única saída, BOOLEAN, onde as entradas L INPUT e OVERFLOW representam valores previamente processados pela **Unidade de Operações Aritméticas**, sendo a primeira a uma sequencia de 8 bits que advindos de uma operação XOR ou uma operação de Subtração e a segunda representa o valor excedente desta operação, que pode ser utilizado para a verificação se o valor do resultado é negativo, a entrada SELETOR é responsável por escolher qual operação será realizada de acordo a Tabela X, a entrada *CLOCK*, que está ligada com o *Clock* do computador e a saída BOOLEAN representa um valor de verdadeiro ou falso, validando a comparação para o desvio de fluxo condicional do **PC**.

**Tabela 15 – Tabela Entradas e Saídas Componente UOA / Gerado a partir de Testes**

|  |  |
| --- | --- |
| **SELETOR** | **Operação** |
| **000** | IGUAL |
| **001** | DIFERENTE |
| **010** | MAIOR |
| **011** | MAIOR IGUAL |
| **100** | MENOR |
| **101** | MENOR IGUAL |
| **110** | PAR |
| **111** | IMPAR |

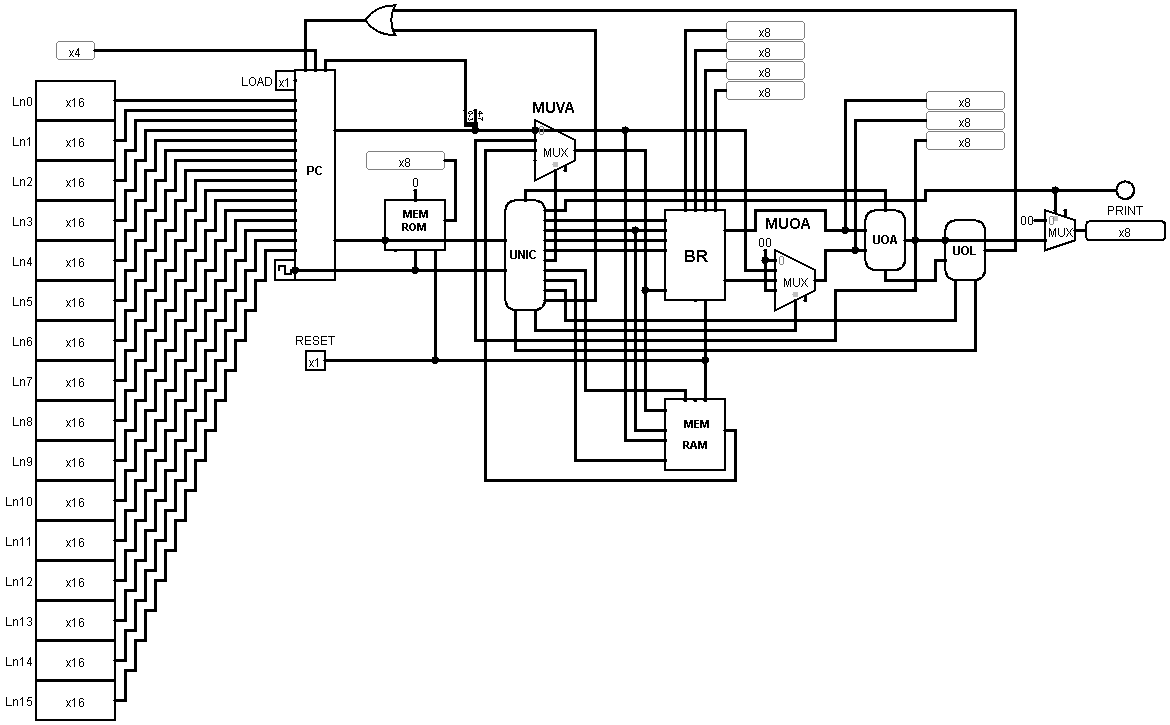
## 1.3.13 Memória RAM

****

**Figura 15 – Memória RAM/ Gerado no *Logisim***

A **Memória RAM** é uma memória de alocação arbitrária, logo, o local onde um determinado valor será salvo é determinado pelo valor de endereço utilizado na instrução e, por isto, a mesma possui dezesseis endereços de armazenamento associados a dezesseis **Valores de Memória 8 bits** encadeados, desta forma este componente é utilizado para armazenar valores de 8 bits que podem ser utilizados posteriormente pelo processador. A **Memória RAM** possui sete entradas, LER:0 / ESC:1, *INDEX*, *ARRAY*, *INPUT*, *CLOCK*, *SET* e *RESET* e apenas uma saída *OUTPUT*, onde as entradas *SET*, *RESET* e *INPUT* conectam-se as entradas de mesmo nome de todos os componentes **Valores de Memória 8 bits**, as entradas *INDEX* e *ARRAY* são entradas de endereço que se conectam com um distribuidor, formando uma única trilha de dados, de forma que os dois bits da entrada *ARRAY* representam os dois bits de maior relevância da trilha final e os dois bits de menor relevância da entrada *INPUT* igualmente representam os dois bits de menor relevância da trilha final, de forma que esse valor possa ser conectado com as entradas SELETOR do multiplexador e do demultiplexador de 4 bits de seleção, a entrada LER:0 / ESC:1 é uma entrada de controle que determina se um valor será lido ou escrito na **Memória RAM**, a mesma é conectada ao multiplexador conectado a entrada *CLOCK* e ao multiplexador conectado na saída *OUTPUT*, de forma que apenas quando o valor de LER:0 / ESC:1 é igual a “0” o valor de *OUTPUT* será igual ao valor do **Valor de Memória 8 bits** específico do endereço e, quando igual a “1” o valor de *CLOCK* será utilizado na entrada do multiplexador.

## Datapath



**Figura 16 – Processador NAV/ Gerado no *Logisim***

O Processador NAV, após sua estruturação e após realização pode ser programado seguindo os seguintes valores binários por instrução, onde “Z” simboliza o valor de entrada, “X” representa o primeiro endereço, “Y” representa o segundo endereço e “-” trata-se de um valor que pode ser desconsiderado para a instrução. Por último, após realização de teste sobre o fluxo de dados foi possível gerar o *Datapath* da Tabela X.

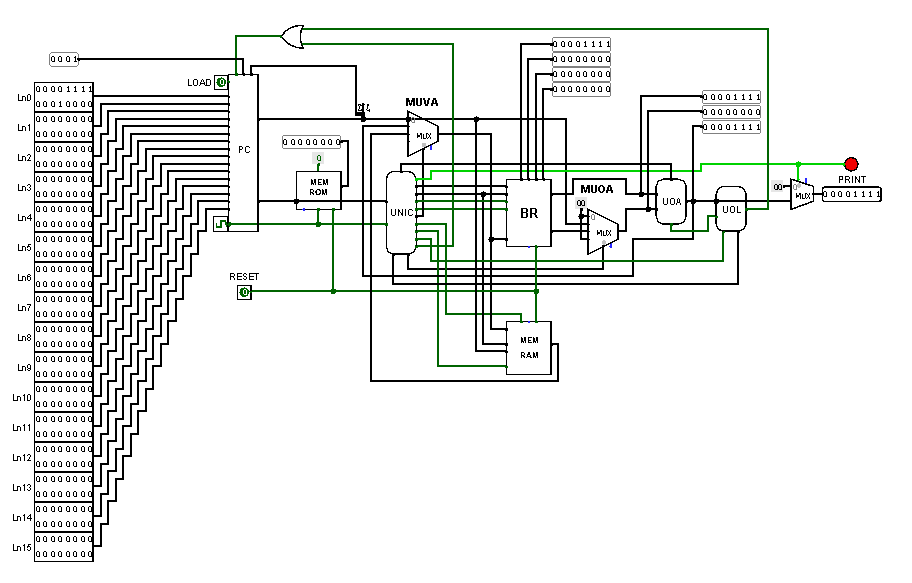
**Tabela 16 – Tabela de Instruções Processador NAV / Gerado a partir de Testes**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Instrução** | **Formato de Instrução** | **Classe de Instrução** | **Binário** | | | **Valor de Entrada** |
| **Código de Operação** | **Endereço 1** | **Endereço 2** |
| **IMP** | **IMP** R1 | E1 | 0000 | XX | -- | -------- |
| **DR** | **DR** R1: ZZZZ ZZZZ | E1V | 0001 | XX | -- | ZZZZ ZZZZ |
| **SOMC** | **SOMC** R1: ZZZZ ZZZZ | E1V | 0010 | XX | -- | ZZZZ ZZZZ |
| **SOMR** | **SOMR** R1, R2 | E2 | 0011 | XX | YY | -------- |
| **SUBC** | **SUBC** R1: ZZZZ ZZZZ | E1V | 0100 | XX | -- | ZZZZ ZZZZ |
| **SUBR** | **SUBR** R1, R2 | E2 | 0101 | XX | YY | -------- |
| **GRA** | **GRA** R1, M1[ZZ] | E2V | 0110 | XX | YY | ------ZZ |
| **CRA** | **CRA** R1, M1[ZZ] | E2V | 0111 | XX | YY | ------ZZ |
| **SEIG** | **SEIG** R1, R2: ZZZZ | E2V | 1000 | XX | YY | ----ZZZZ |
| **SEDIF** | **SEDIF** R1, R2: ZZZZ | E2V | 1001 | XX | YY | ----ZZZZ |
| **SEMA** | **SEMA** R1, R2: ZZZZ | E2V | 1010 | XX | YY | ----ZZZZ |
| **SEMAIG** | **SEMAIG** R1, R2: ZZZZ | E2V | 1011 | XX | YY | ----ZZZZ |
| **SEME** | **SEME** R1, R2: ZZZZ | E2V | 1100 | XX | YY | ----ZZZZ |
| **SEMEIG** | **SEMEIG** R1, R2: ZZZZ | E2V | 1101 | XX | YY | ----ZZZZ |
| **SEPAR** | **SEPAR** R1, R2: ZZZZ | E2V | 1110 | XX | YY | ----ZZZZ |
| **J** | **J**: ZZZZ | V | 1111 | -- | -- | ----ZZZZ |

**Tabela 17 – Tabela de *Datapath* de Instruções Processador NAV / Gerado a partir de Testes**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instrução** | **CLOCK BR** | **CLOCK MEM** | **CTR BR** | **CTR MEM** | **OUT** | **CTR UOL** | **JUMP** | **END.1** | **END.2** | **UOL OP** | **UOA OP** | **MUOA** | **MUVA** |
| **IMP** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | YY | XX | 000 | 00100 | 00 | 00 |
| **DR** | *CLOCK* | 0 | 0 | 0 | 0 | 0 | 0 | YY | XX | 001 | 00000 | 00 | 00 |
| **SOMC** | *CLOCK* | 0 | 0 | 0 | 0 | 0 | 0 | YY | XX | 010 | 00100 | 01 | 01 |
| **SOMR** | *CLOCK* | 0 | 0 | 0 | 0 | 0 | 0 | YY | XX | 011 | 00100 | 10 | 01 |
| **SUBC** | *CLOCK* | 0 | 0 | 0 | 0 | 0 | 0 | YY | XX | 100 | 00101 | 01 | 01 |
| **SUBR** | *CLOCK* | 0 | 0 | 0 | 0 | 0 | 0 | YY | XX | 101 | 00101 | 10 | 01 |
| **GRA** | *CLOCK* | *CLOCK* | 1 | 1 | 0 | 0 | 0 | YY | XX | 110 | 00100 | 00 | 01 |
| **CRA** | *CLOCK* | 0 | 0 | 0 | 0 | 0 | 0 | YY | XX | 111 | 00100 | 00 | 10 |
| **SEIG** | 0 | 0 | 1 | 0 | 0 | 1 | 0 | YY | XX | 000 | 00011 | 10 | 00 |
| **SEDIF** | 0 | 0 | 1 | 0 | 0 | 1 | 0 | YY | XX | 001 | 00011 | 10 | 00 |
| **SEMA** | 0 | 0 | 1 | 0 | 0 | 1 | 0 | YY | XX | 010 | 00101 | 10 | 00 |
| **SEMAIG** | 0 | 0 | 1 | 0 | 0 | 1 | 0 | YY | XX | 011 | 00101 | 10 | 00 |
| **SEME** | 0 | 0 | 1 | 0 | 0 | 1 | 0 | YY | XX | 100 | 00101 | 10 | 00 |
| **SEMEIG** | 0 | 0 | 1 | 0 | 0 | 1 | 0 | YY | XX | 101 | 00101 | 10 | 00 |
| **SEPAR** | 0 | 0 | 1 | 0 | 0 | 1 | 0 | YY | XX | 110 | 00100 | 00 | 00 |
| **J** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | YY | XX | 111 | 00000 | 00 | 00 |

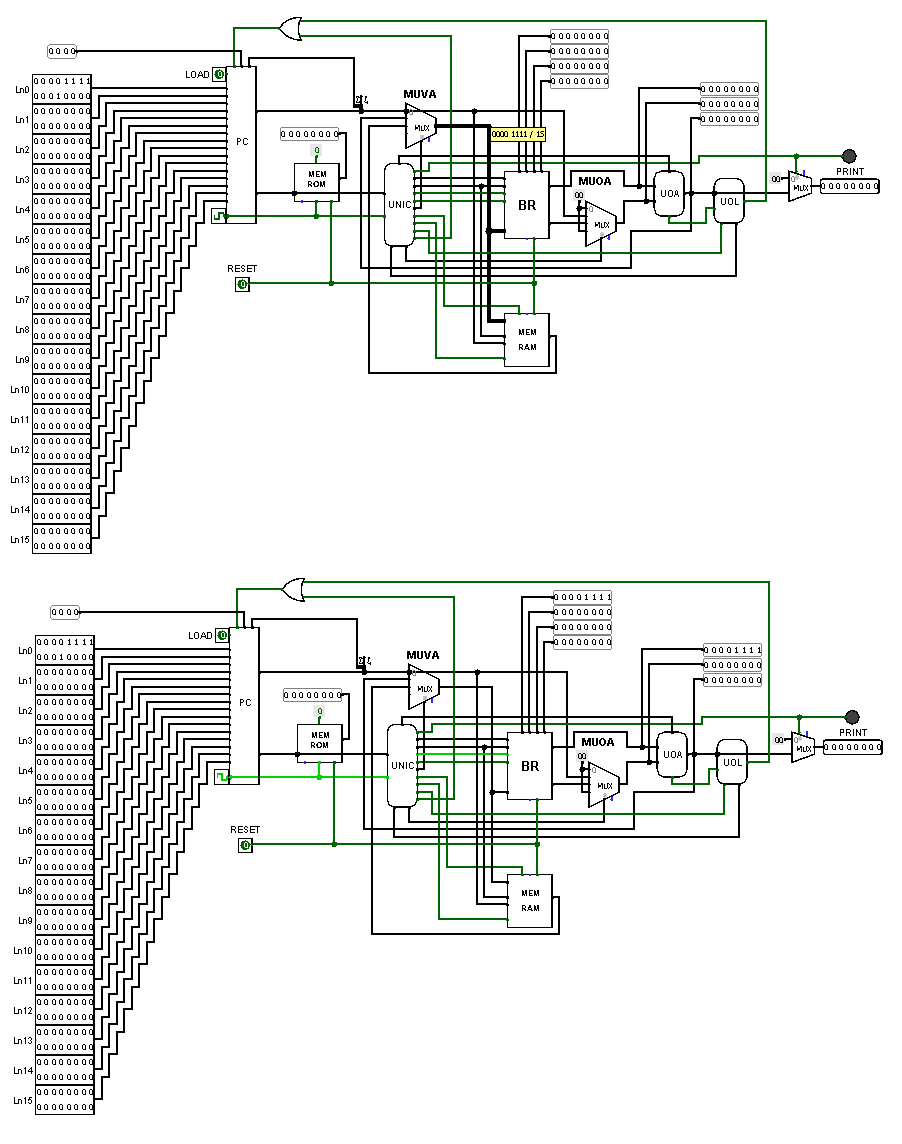
## 1.4.1 IMP



**Figura 17 – Instrução IMP Processador NAV / Gerado no *Logisim***

A instrução **IMP** é uma instrução para impressão de valor de registrador que não altera seu funcionamento de acordo com o valor do *Clock* do processador e para compreender seu funcionamento é preciso entender o fluxo de dados das unidades **UNIC**, **BR** e **UOA** em relação ao código de instrução “0000”. O fluxo inicia-se na unidade **UNIC**, onde apenas a *flag* PRINT é ativada e as saídas de endereço UOA OP, MUOA recebem os valores “00100” e “00” respectivamente, de forma que o valor o valor de END1 é conectado ao **Banco de Registradores** e o valor do registrador referente ao endereço de END1 é enviado para a saída R1 do **BR** e conecta-se à entrada A do **UOA**, simultaneamente o valor “00000000” é selecionado pelo multiplexador **MUOA** para a entrada B da unidade **UOA**, efetuando uma operação de soma com um valor nulo, logo, mantendo o valor original de A, enviando-o para o multiplexador conectado à saída *PRINT*, que seleciona o valor de A por conta da *flag* *PRINT* do **UNIC**.

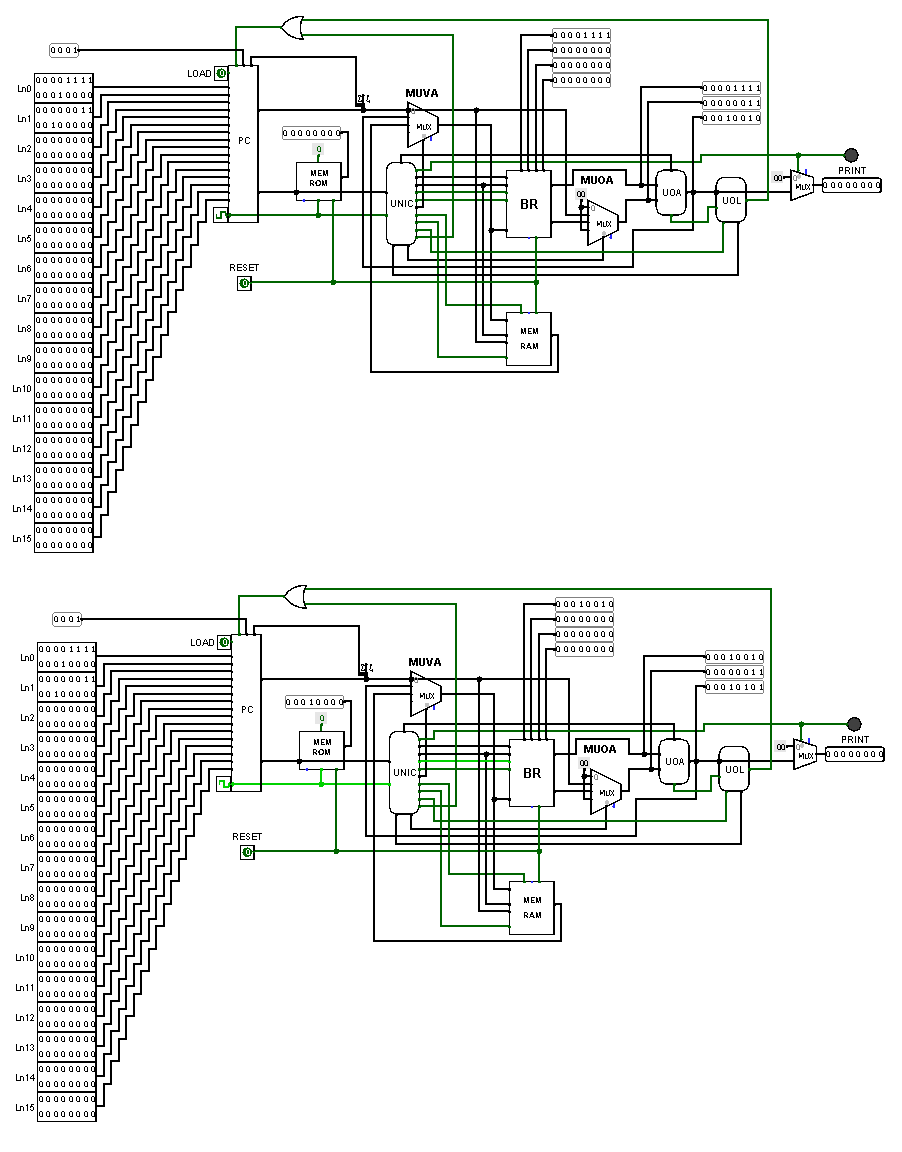
## 1.4.2 DR



**Figura 18 – Instrução DR Processador NAV / Gerado no *Logisim***

A instrução **DR** é uma instrução para declaração de valor de registrador que altera seu funcionamento de acordo com o valor do *Clock* do processador. Para compreender seu funcionamento é preciso entender o fluxo de dados das unidades **UNIC** e **BR** em relação ao código de instrução “0001”. O fluxo inicia-se com o valor de *Clock* igual a “0”, de forma que nenhuma *flag* é ativada na unidade **UNIC** e o endereço MUVA recebe o valor “00”, de forma que o valor de entrada da instrução é conectado à entrada *INPUT* do **Banco de Registradores**. Quando o valor de *Clock* do processador é igual a “1” a *flag* CLOCK BR é ativada, ativando desta forma a entrada CLOCK do **BR** e o valor já presente na entrada *INPUT* do mesmo é armazenado.

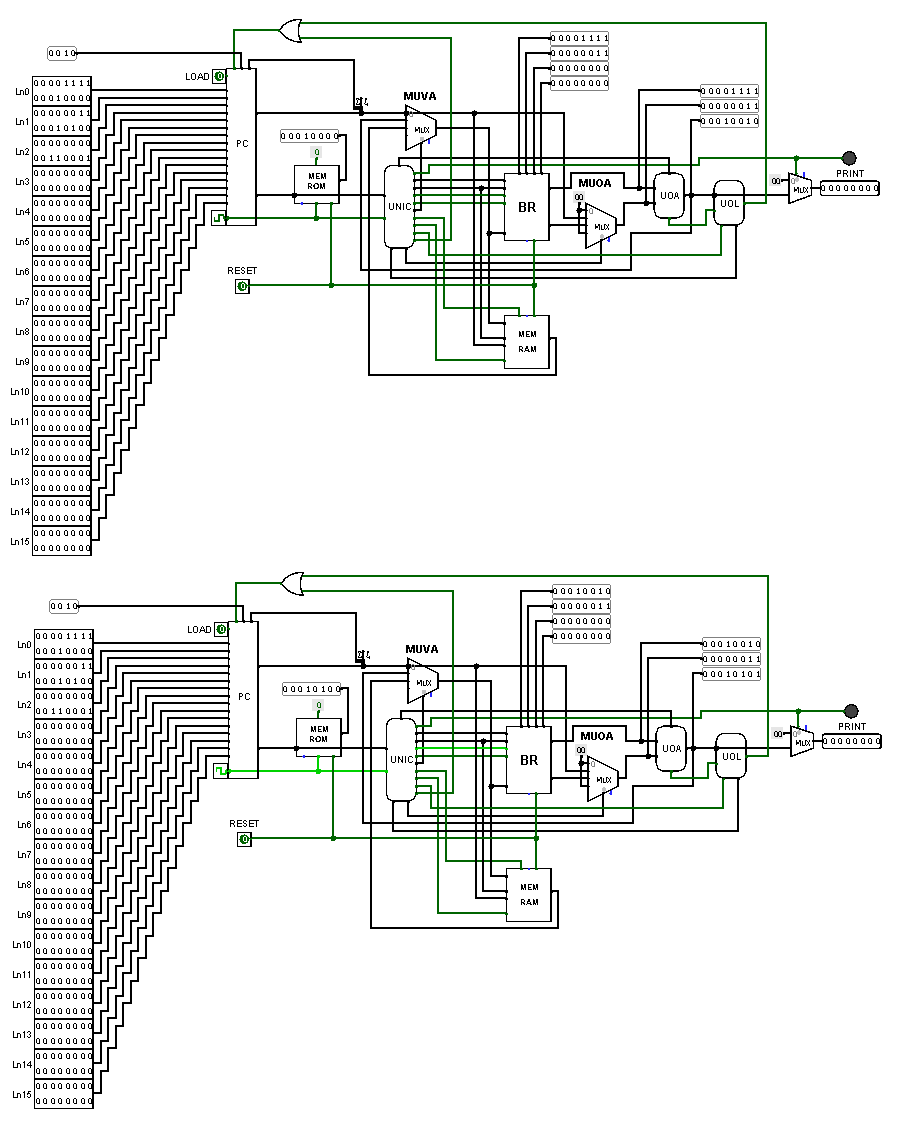
## 1.4.3 SOMC



**Figura 19 – Instrução SOMC Processador NAV / Gerado no *Logisim***

A instrução **SOMC** é uma instrução de soma de valor de registrador com constante que altera seu funcionamento de acordo com o valor do *Clock* do processador. Para compreender seu funcionamento é preciso entender o fluxo de dados das unidades **UNIC**, **BR** e **UOA** em relação ao código de instrução “0010”. O fluxo inicia-se com o valor de *Clock* igual a “0”, onde as saídas de endereço UOA OP, MUOA e MUVA da **UNIC** recebem os valores “00100”, “01” e “01” respectivamente, de forma que o valor de END1 é conectado ao **Banco de Registradores** e o valor do registrador referente ao endereço de END1 é enviado para a saída R1 do **BR** e conecta-se à entrada A do **UOA**, simultaneamente o valor de entrada da instrução é selecionado pelo multiplexador **MUOA** para a entrada B da unidade **UOA**, efetuando uma operação de soma de forma que o resultado é enviado para o multiplexador **MUVA** e o valor é conectado à entrada *INPUT* do **BR** . Quando o valor de *Clock* do processador é igual a “1” a *flag* CLOCK BR é ativada, ativando desta forma a entrada CLOCK do **BR** e o valor já presente na entrada *INPUT*, logo, o valor do resultado da soma é armazenado.

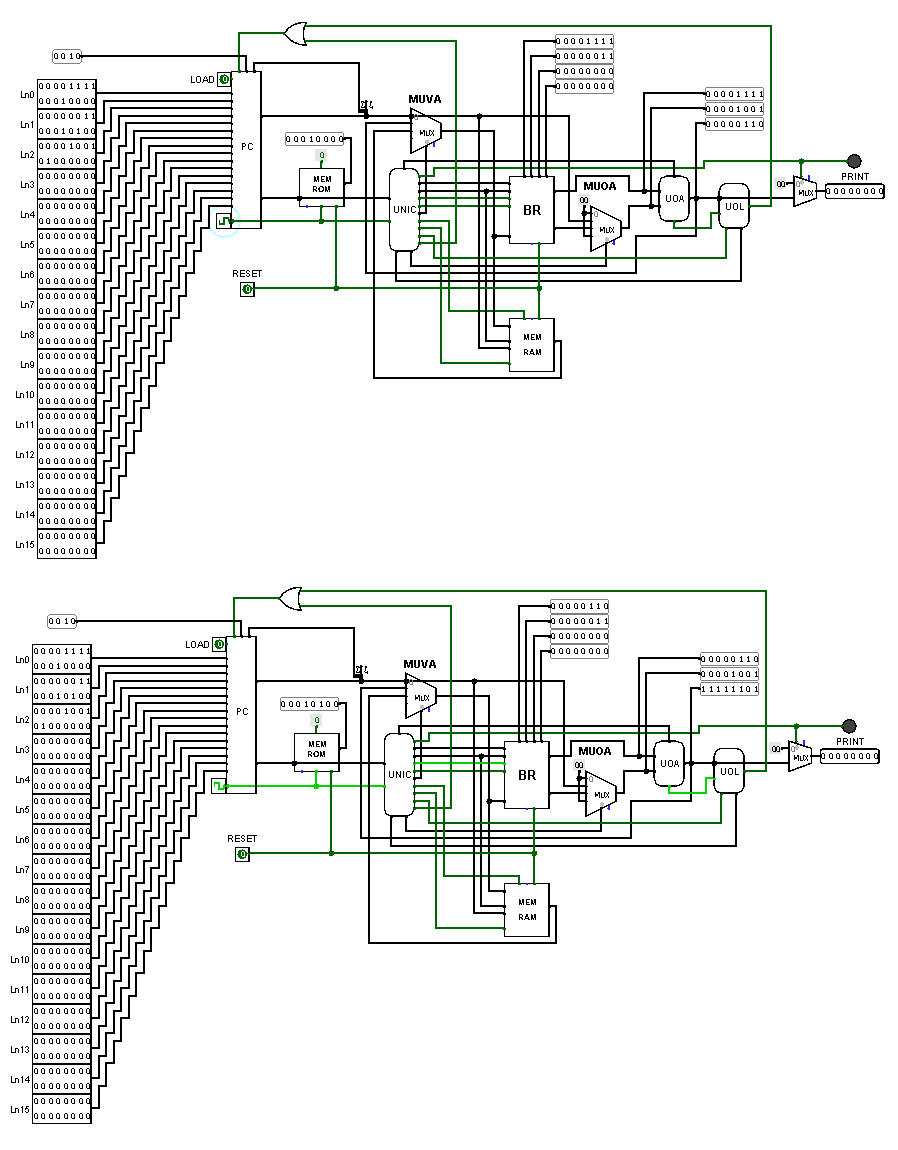
## 1.4.4 SOMR



**Figura 20 – Instrução SOMR Processador NAV / Gerado no *Logisim***

A instrução **SOMR** é uma instrução de soma de valor de dois registradores que altera seu funcionamento de acordo com o valor do *Clock* do processador. Para compreender seu funcionamento é preciso entender o fluxo de dados das unidades **UNIC**, **BR** e **UOA** em relação ao código de instrução “0011”. O fluxo inicia-se com o valor de *Clock* igual a “0”, onde as saídas de endereço UOA OP, MUOA e MUVA da **UNIC** recebem os valores “00100”, “10” e “01” respectivamente, de forma que os valores de END1 e END2 são conectados ao **Banco de Registradores** e o valor do registrador referente aos endereços de END1 e END2 são enviados para as saídas R1 e R2 do **BR** e conectam-se às entradas A e B do **UOA** por conta do multiplexador **MUOA** que seleciona o valor de R2 para B, simultaneamente o valor de entrada da instrução é selecionado pelo multiplexador **MUOA** para a entrada B da unidade **UOA**, efetuando uma operação de soma de forma que o resultado é enviado para o multiplexador **MUVA** e o valor é conectado à entrada *INPUT* do **BR**. Quando o valor de *Clock* do processador é igual a “1” a *flag* CLOCK BR é ativada, ativando desta forma a entrada CLOCK do **BR** e o valor já presente na entrada *INPUT*, logo, o valor do resultado da soma é armazenado no endereço END1.

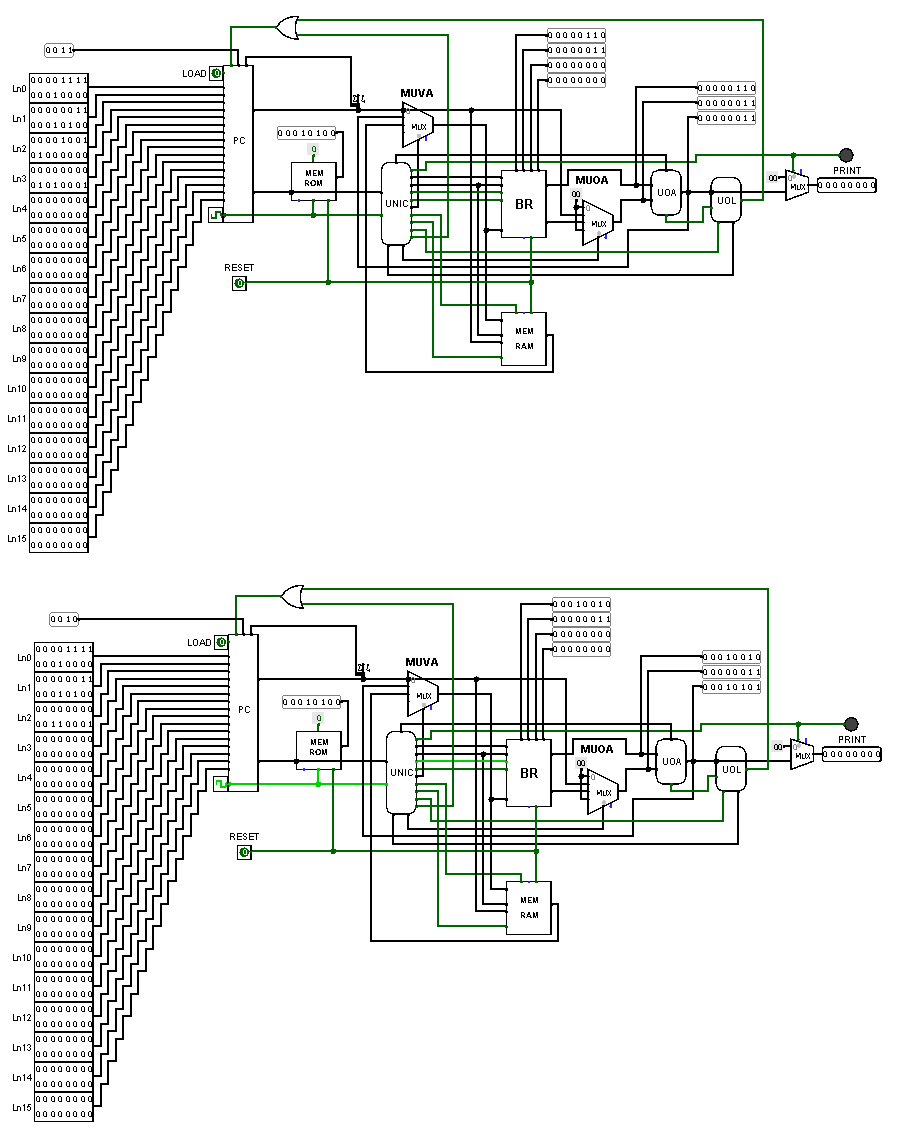
## 1.4.5 SUBC



**Figura 21 – Instrução SUBC Processador NAV / Gerado no *Logisim***

A instrução **SOMC** é uma instrução de subtração de valor de registrador com constante que altera seu funcionamento de acordo com o valor do *Clock* do processador. Para compreender seu funcionamento é preciso entender o fluxo de dados das unidades **UNIC**, **BR** e **UOA** em relação ao código de instrução “0100”. O fluxo inicia-se com o valor de *Clock* igual a “0”, onde as saídas de endereço UOA OP, MUOA e MUVA da **UNIC** recebem os valores “00101”, “01” e “01” respectivamente, de forma que o valor de END1 é conectado ao **Banco de Registradores** e o valor do registrador referente ao endereço de END1 é enviado para a saída R1 do **BR** e conecta-se à entrada A do **UOA**, simultaneamente o valor de entrada da instrução é selecionado pelo multiplexador **MUOA** para a entrada B da unidade **UOA**, efetuando uma operação de subtração de forma que o resultado é enviado para o multiplexador **MUVA** e o valor é conectado à entrada *INPUT* do **BR** . Quando o valor de *Clock* do processador é igual a “1” a *flag* CLOCK BR é ativada, ativando desta forma a entrada CLOCK do **BR** e o valor já presente na entrada *INPUT* é armazenado.

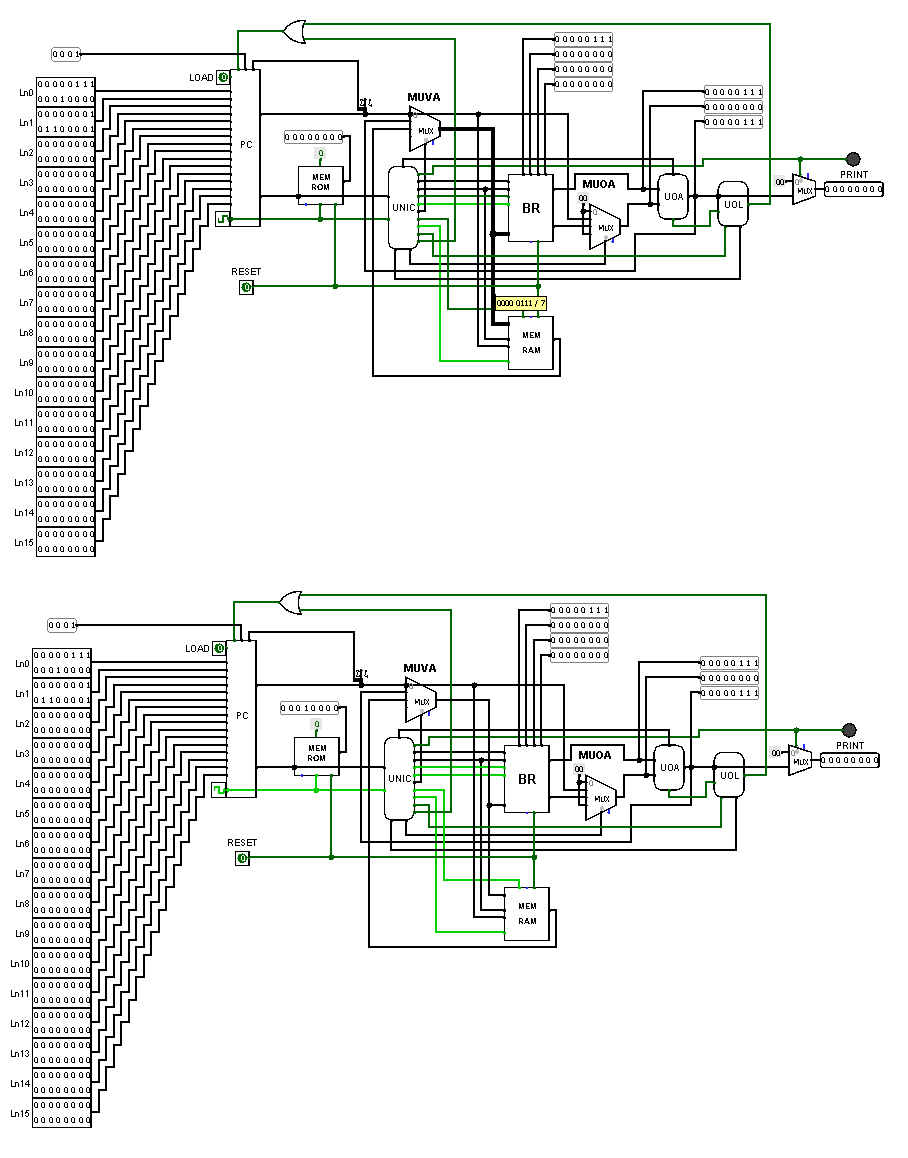
## 1.4.6 SUBR



**Figura 22 – Instrução SUBR Processador NAV / Gerado no *Logisim***

A instrução **SUBR** é uma instrução de subtração de valor de dois registradores que altera seu funcionamento de acordo com o valor do *Clock* do processador. Para compreender seu funcionamento é preciso entender o fluxo de dados das unidades **UNIC**, **BR** e **UOA** em relação ao código de instrução “0101”. O fluxo inicia-se com o valor de *Clock* igual a “0”, onde as saídas de endereço UOA OP, MUOA e MUVA da **UNIC** recebem os valores “00101”, “10” e “01” respectivamente, de forma que os valores de END1 e END2 são conectados ao **Banco de Registradores** e o valor do registrador referente aos endereços de END1 e END2 são enviados para as saídas R1 e R2 do **BR** e conectam-se às entradas A e B do **UOA** por conta do multiplexador **MUOA** que seleciona o valor de R2 para B, simultaneamente o valor de entrada da instrução é selecionado pelo multiplexador **MUOA** para a entrada B da unidade **UOA**, efetuando uma operação de subtração de forma que o resultado é enviado para o multiplexador **MUVA** e o valor é conectado à entrada *INPUT* do **BR**. Quando o valor de *Clock* do processador é igual a “1” a *flag* CLOCK BR é ativada, ativando desta forma a entrada CLOCK do **BR** e o valor já presente na entrada *INPUT*, logo, o valor do resultado da subtração é armazenado no endereço END1.

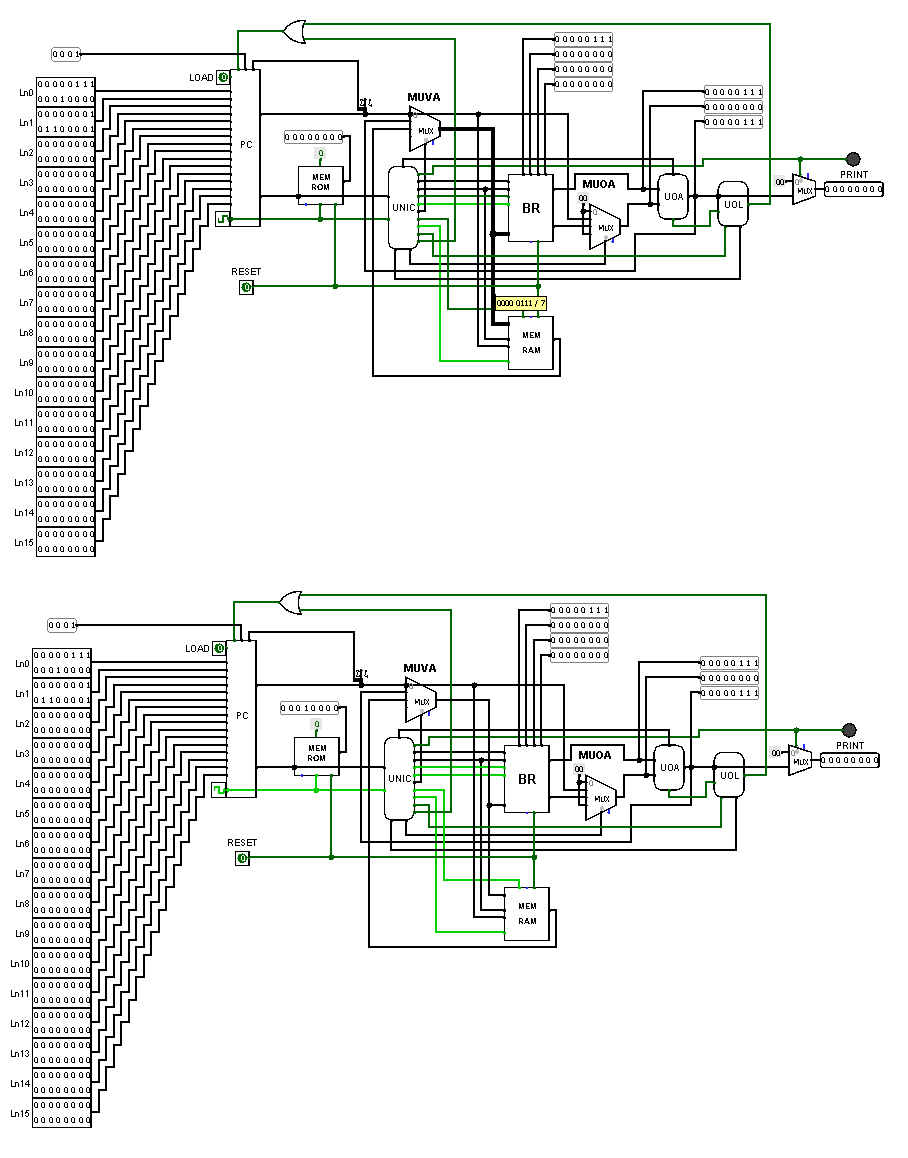
## 1.4.7 GRA



**Figura 23 – Instrução GRA Processador NAV / Gerado no *Logisim***

A instrução **GRA** é uma instrução de armazenamento de valor de registrador na **Memória RAM** que altera seu funcionamento de acordo com o valor do *Clock* do processador. Para compreender seu funcionamento é preciso entender o fluxo de dados das unidades **UNIC**, **BR**, **UOA** e **Memória RAM** em relação ao código de instrução “0110”. O fluxo inicia-se com o valor de *Clock* igual a “0”, onde as saídas de endereço UOA OP, MUOA e MUVA da **UNIC** recebem os valores “00100”, “00” e “01” respectivamente, e as *flags* CTR BR e CTR MEM, impedindo a escrita no banco de registradores e possibilitando a escrita na **MEM RAM**, de forma que o valor o valor de END1 é conectado ao **Banco de Registradores** e o valor do registrador referente ao endereço de END1 é enviado para a saída R1 do **BR** e conecta-se à entrada A do **UOA**, simultaneamente o valor “00000000” é selecionado pelo multiplexador **MUOA** para a entrada B da unidade **UOA**, efetuando uma operação de soma com um valor nulo, logo, mantendo o valor original de A, que por sua vez, é enviado para o multiplexador **MUVA** e o valor é conectado à entrada *INPUT* da **MEM RAM**. Quando o valor de *Clock* do processador é igual a “1” a *flag* CLOCK MEM é ativada, ativando desta forma a entrada CLOCK da **Memória RAM** que irá armazenar o valor no endereço formado pela junção dos dois números menos significativos do binário da instrução e do valor de entrada.

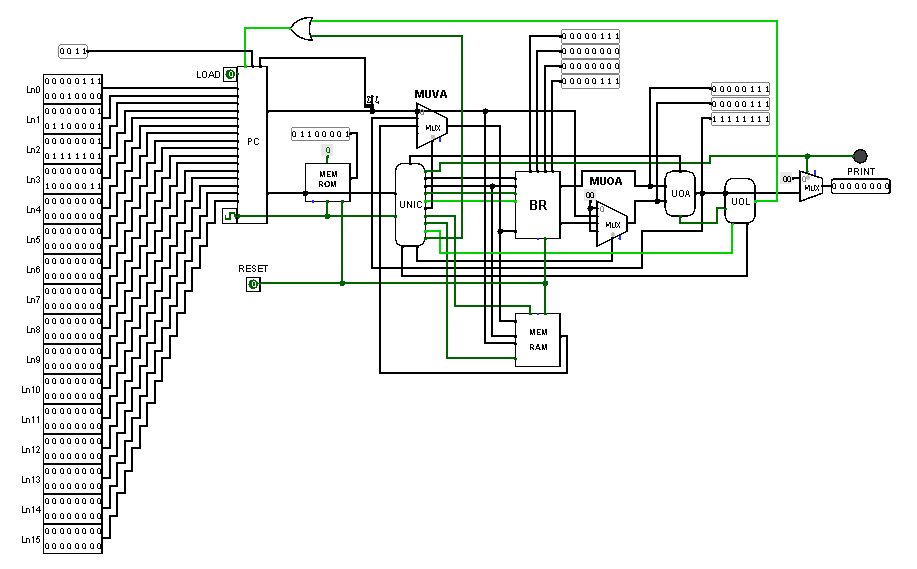
## 1.4.8 CRA



**Figura 24 – Instrução CRA Processador NAV / Gerado no *Logisim***

A instrução **CRA** é uma instrução de armazenamento de valor da **Memória RAM** em um determinado registrador que altera seu funcionamento de acordo com o valor do *Clock* do processador. Para compreender seu funcionamento é preciso entender o fluxo de dados das unidades **UNIC**, **BR**, **UOA** e **Memória RAM** em relação ao código de instrução “0111”. O fluxo inicia-se com o valor de *Clock* igual a “0”, onde as saídas de endereço UOA OP, MUOA e MUVA da **UNIC** recebem os valores “00100”, “00” e “10” respectivamente, e nenhuma *flag* é ativada, de forma que a junção de END2 e os dois bits menos significativos da entrada de instrução são utilizado para o endereço de leitura da **Memória RAM** de forma que o valor lido é enviado para o multiplexador **MUVA**, que se conecta com a entrada *INPUT* do **Banco de Rgistradores**. Quando o valor de *Clock* do processador é igual a “1” a *flag* CLOCK BR é ativada, ativando desta forma a entrada CLOCK do **BR** que irá armazenar o valor no endereço END1.

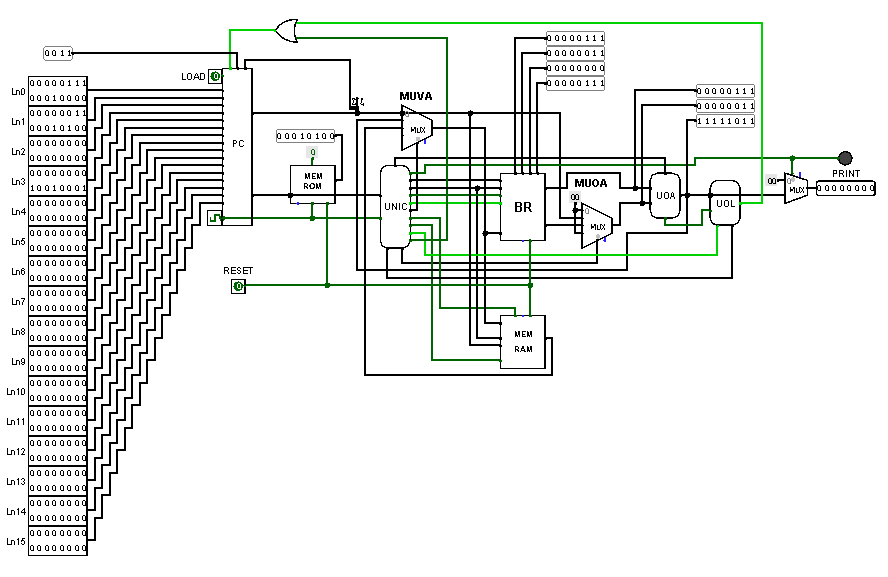
## 1.4.9 SEIG



**Figura 25 – Instrução SEIG Processador NAV / Gerado no *Logisim***

A instrução **SEIG** é uma instrução de comparação de valores de dois registradores que não altera seu funcionamento de acordo com o valor do *Clock* do processador. Para compreender seu funcionamento é preciso entender o fluxo de dados das unidades **UNIC**, **BR**, **UOA** e **UOL** em relação ao código de instrução “1000”. As saídas de endereço UOA OP e UOL OP da **UNIC** recebem os valores “00011” e “000” respectivamente, ativando as *flags* CTR BR e CTR UOL, que determinam que uma valor será comparádo e que nenhum novo registrador será escrito, de forma que os valores de END1 e END2 são conectados ao **Banco de Registradores** e o valor dos registradores referentes aos endereços de END1 e END2 são enviados para as saídas R1 e R2 do **BR** e conectam-se às entradas A e B do **UOA** por conta do multiplexador **MUOA** que seleciona o valor de R2 para B, efetuando uma operação de *xnor* de forma que os reusultados da comparação define “1” como igual e “0” como diferente, desta forma estes valores podem ser enviados para a **UOL** e, caso todos os valores sejam verdadeiros o valor da saída JUMP será verdadeiro também.

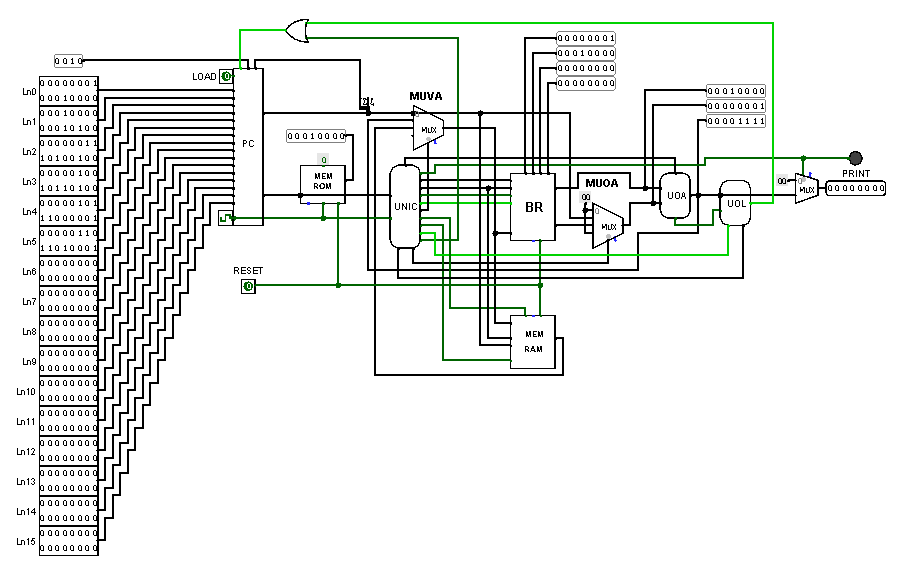
## 1.4.10 SEDIF



**Figura 26 – Instrução SEDIF Processador NAV / Gerado no *Logisim***

A instrução **SEDIF** é uma instrução de comparação de valores de dois registradores que não altera seu funcionamento de acordo com o valor do *Clock* do processador. Para compreender seu funcionamento é preciso entender o fluxo de dados das unidades **UNIC**, **BR**, **UOA** e **UOL** em relação ao código de instrução “1001”. As saídas de endereço UOA OP e UOL OP da **UNIC** recebem os valores “00011” e “001” respectivamente, ativando as *flags* CTR BR e CTR UOL, que determinam que uma valor será comparádo e que nenhum novo registrador será escrito, de forma que os valores de END1 e END2 são conectados ao **Banco de Registradores** e o valor dos registradores referentes aos endereços de END1 e END2 são enviados para as saídas R1 e R2 do **BR** e conectam-se às entradas A e B do **UOA** por conta do multiplexador **MUOA** que seleciona o valor de R2 para B, efetuando uma operação de *xnor* de forma que os reusultados da comparação define “1” como igual e “0” como diferente, desta forma estes valores podem ser enviados para a **UOL** e, caso qualquer valor seja falso o valor da saída JUMP será verdadeiro.

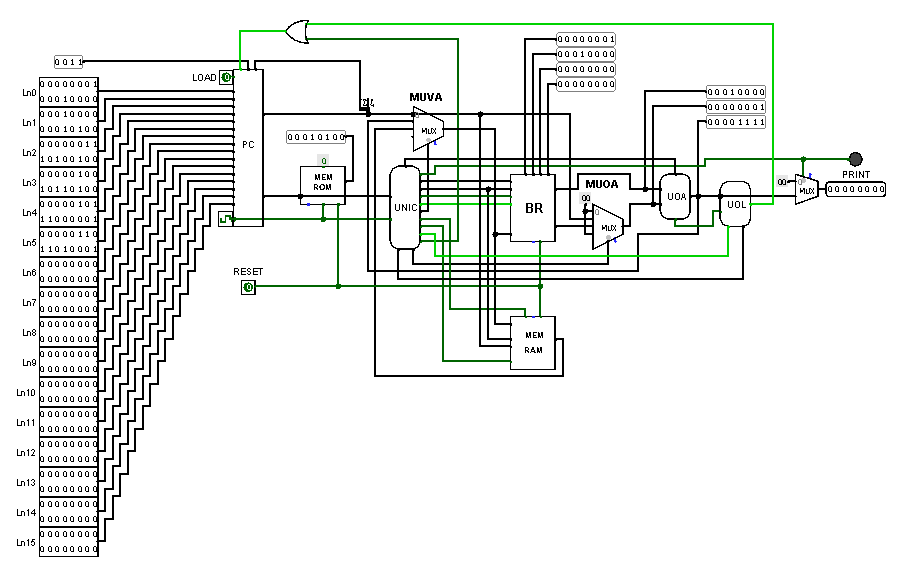
## 1.4.11 SEMA



**Figura 27 – Instrução SEMA Processador NAV / Gerado no *Logisim***

A instrução **SEMA** é uma instrução de comparação de valores de dois registradores que não altera seu funcionamento de acordo com o valor do *Clock* do processador. Para compreender seu funcionamento é preciso entender o fluxo de dados das unidades **UNIC**, **BR**, **UOA** e **UOL** em relação ao código de instrução “1010”. As saídas de endereço UOA OP e UOL OP da **UNIC** recebem os valores “00101” e “010” respectivamente, ativando as *flags* CTR BR e CTR UOL, que determinam que uma valor será comparádo e que nenhum novo registrador será escrito, de forma que os valores de END1 e END2 são conectados ao **Banco de Registradores** e o valor dos registradores referentes aos endereços de END1 e END2 são enviados para as saídas R1 e R2 do **BR** e conectam-se às entradas A e B do **UOA** por conta do multiplexador **MUOA** que seleciona o valor de R2 para B, efetuando uma operação de subtração de forma que os reusultados da operação e o valor da saída *OVERFLOW*, são utilizados no componente **UOL**, onde caso o valor final seja não-nulo (sendo nulo um valor onde todos os bits são iguais a “0”) e o valor de *OVERFLOW* também seja igual a “0” o valor da saída JUMP será verdadeiro.

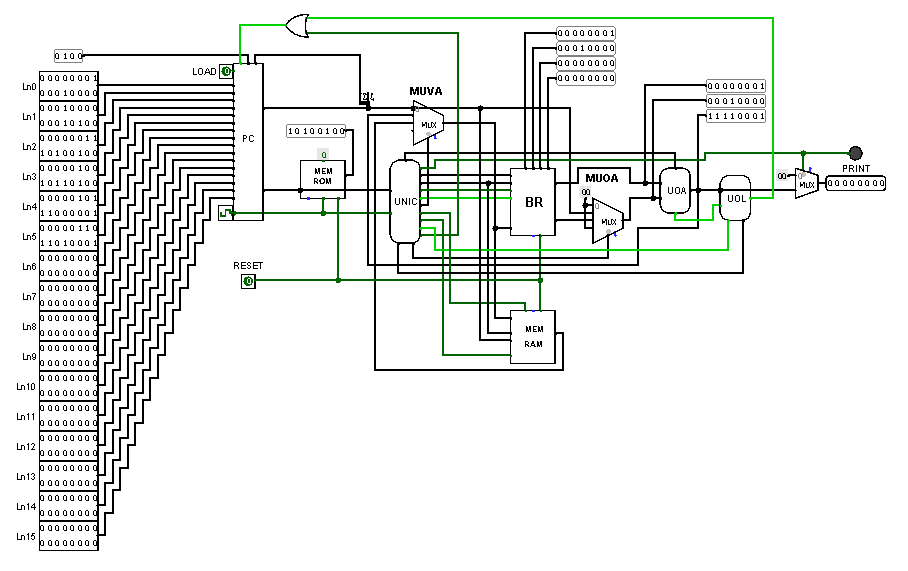
## 1.4.11 SEMAIG



**Figura 28 – Instrução SEMAIG Processador NAV / Gerado no *Logisim***

A instrução **SEMAIG** é uma instrução de comparação de valores de dois registradores que não altera seu funcionamento de acordo com o valor do *Clock* do processador. Para compreender seu funcionamento é preciso entender o fluxo de dados das unidades **UNIC**, **BR**, **UOA** e **UOL** em relação ao código de instrução “1011”. As saídas de endereço UOA OP e UOL OP da **UNIC** recebem os valores “00101” e “011” respectivamente, ativando as *flags* CTR BR e CTR UOL, que determinam que uma valor será comparádo e que nenhum novo registrador será escrito, de forma que os valores de END1 e END2 são conectados ao **Banco de Registradores** e o valor dos registradores referentes aos endereços de END1 e END2 são enviados para as saídas R1 e R2 do **BR** e conectam-se às entradas A e B do **UOA** por conta do multiplexador **MUOA** que seleciona o valor de R2 para B, efetuando uma operação de subtração de forma que os resultados da operação e o valor da saída *OVERFLOW*, são utilizados no componente **UOL**, onde caso o valor de *OVERFLOW* também seja igual a “0” o valor da saída JUMP será verdadeiro.

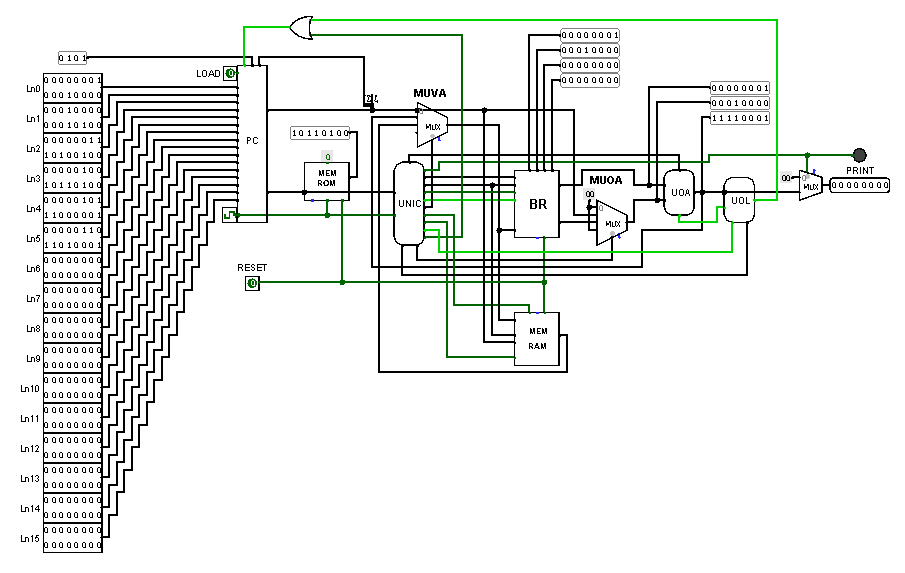
## 1.4.13 SEME



**Figura 29 – Instrução SEME Processador NAV / Gerado no *Logisim***

A instrução **SEME** é uma instrução de comparação de valores de dois registradores que não altera seu funcionamento de acordo com o valor do *Clock* do processador. Para compreender seu funcionamento é preciso entender o fluxo de dados das unidades **UNIC**, **BR**, **UOA** e **UOL** em relação ao código de instrução “1100”. As saídas de endereço UOA OP e UOL OP da **UNIC** recebem os valores “00101” e “100” respectivamente, ativando as *flags* CTR BR e CTR UOL, que determinam que uma valor será comparado e que nenhum novo registrador será escrito, de forma que os valores de END1 e END2 são conectados ao **Banco de Registradores** e o valor dos registradores referentes aos endereços de END1 e END2 são enviados para as saídas R1 e R2 do **BR** e conectam-se às entradas A e B do **UOA** por conta do multiplexador **MUOA** que seleciona o valor de R2 para B, efetuando uma operação de subtração de forma que os resultados da operação e o valor da saída *OVERFLOW*, são utilizados no componente **UOL**, onde caso o valor de *OVERFLOW* também seja igual a “1” o valor da saída JUMP será verdadeiro.

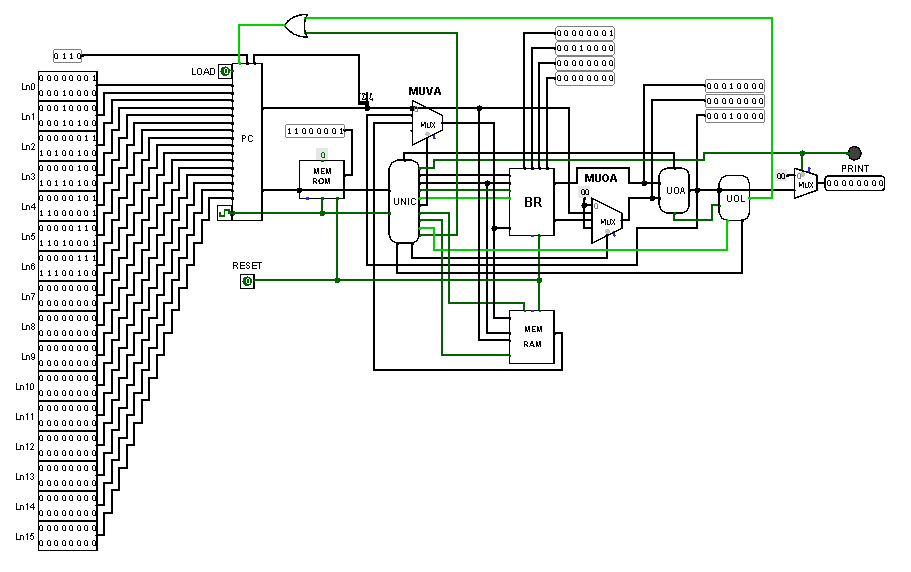
## 1.4.14 SEMEIG



**Figura 30 – Instrução SEMEIG Processador NAV / Gerado no *Logisim***

A instrução **SEMEIG** é uma instrução de comparação de valores de dois registradores que não altera seu funcionamento de acordo com o valor do *Clock* do processador. Para compreender seu funcionamento é preciso entender o fluxo de dados das unidades **UNIC**, **BR**, **UOA** e **UOL** em relação ao código de instrução “1101”. As saídas de endereço UOA OP e UOL OP da **UNIC** recebem os valores “00101” e “101” respectivamente, ativando as *flags* CTR BR e CTR UOL, que determinam que uma valor será comparádo e que nenhum novo registrador será escrito, de forma que os valores de END1 e END2 são conectados ao **Banco de Registradores** e o valor dos registradores referentes aos endereços de END1 e END2 são enviados para as saídas R1 e R2 do **BR** e conectam-se às entradas A e B do **UOA** por conta do multiplexador **MUOA** que seleciona o valor de R2 para B, efetuando uma operação de subtração de forma que os reusultados da operação e o valor da saída *OVERFLOW*, são utilizados no componente **UOL**, onde caso o valor de *OVERFLOW* também seja igual a “1” o valor da saída JUMP será verdadeiro ou o valor seja nulo(sendo nulo um valor onde todos os bits são iguais a “0”).

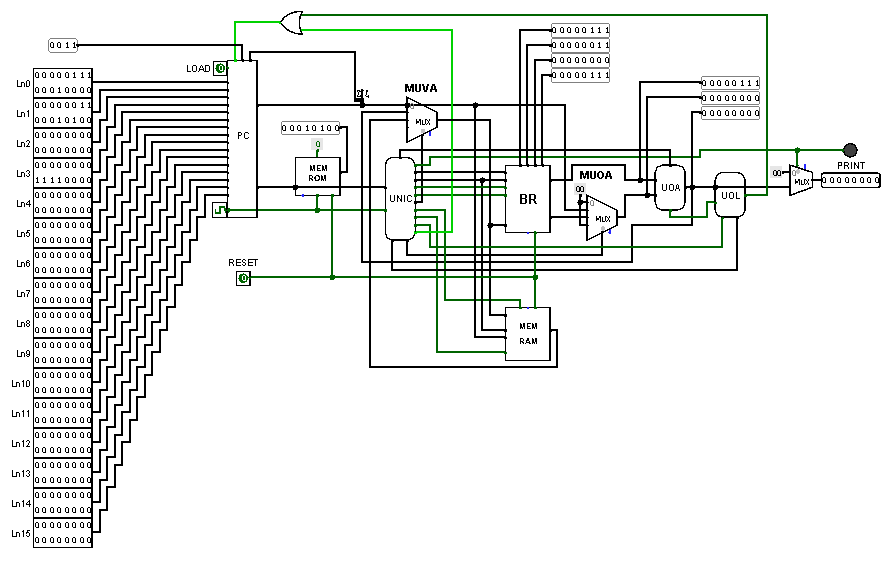
## 1.4.15 SEPAR



**Figura 31 – Instrução SEPAR Processador NAV / Gerado no *Logisim***

A instrução **SEPAR** é uma instrução de validação que verifica se um valor é par de valores de um registrador que não altera seu funcionamento de acordo com o valor do *Clock* do processador. Para compreender seu funcionamento é preciso entender o fluxo de dados das unidades **UNIC**, **BR**, **UOA** e **UOL** em relação ao código de instrução “1110”. O valor o valor de END1 é conectado ao **Banco de Registradores** e o valor do registrador referente ao endereço de END1 é enviado para a saída R1 do **BR** e conecta-se à entrada A do **UOA**, simultaneamente o valor “00000000” é selecionado pelo multiplexador **MUOA** para a entrada B da unidade **UOA**, efetuando uma operação de soma com um valor nulo, logo, mantendo o valor original de A, com o valor de A o mesmo pode ser lido pela **UOL**, onde caso o valor do bit menos significativo de A seja igual a “0 , o valor da saída JUMP será verdadeiro;

## 1.4.16 J



**Figura 32 – Instrução J Processador NAV / Gerado no *Logisim***

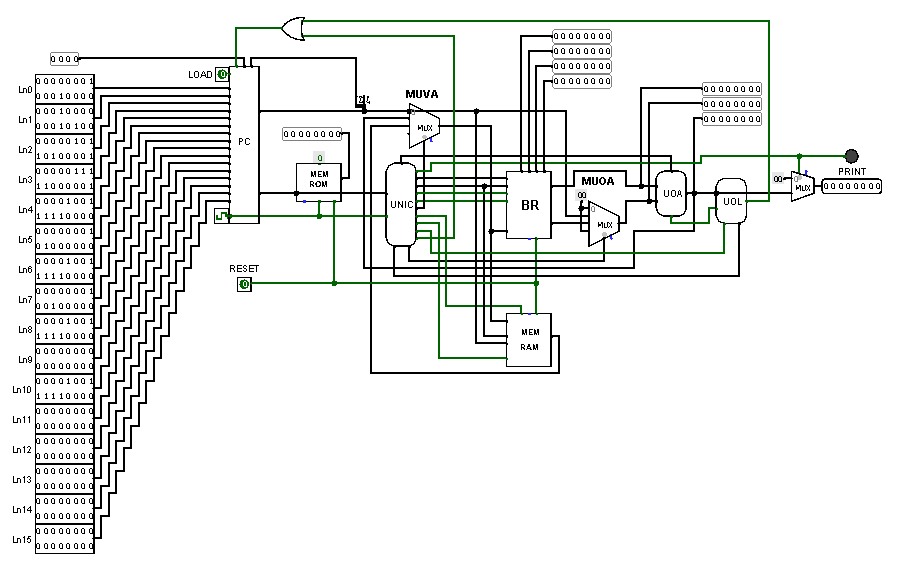
A instrução **J** é uma instrução de salto incondicional que utiliza apenas uma unidade, **UNIC**, que por sua vez utiliza apenas uma única *flag*, a *flag* JUMP, que é diretamente conectada à entrada de mesmo nome no componente **PC**, que utiliza os quatro bits de menor relevância da entrada de valor da instrução para realizar um salto para uma linha de instrução específica.

## 

## Simulações e Testes

Esta secção descreve os testes realizados para o funcionamento de dois programas simples mas amplamente vistos no ambiente de programação de alto nível. Sendo um programa que simula o funcionamento dos desvios condicionais *If* e *Elseif* e *Else*, demonstrando o funcionamento de desvios condicionais e incondicionais no Processador NAV, e um programa que simula o funcionamento do laço condicional *while*, demonstrando o funcionamento de laços de repetição no processador NAV.

* 1. **Simulação *If*, *Elseif* e *Else***



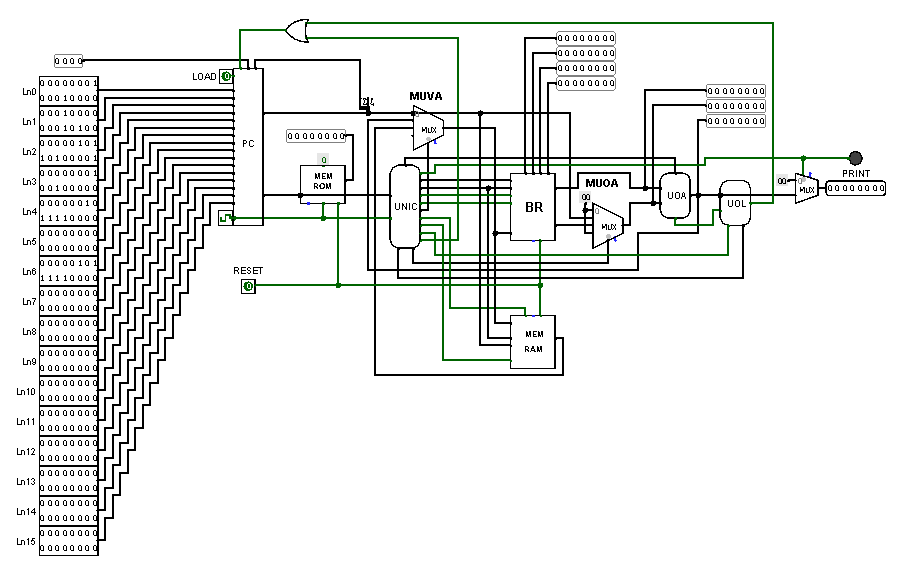
**Figura 33 – Programa *If*, *Elseif* e *Else* no Processador NAV / Gerado no *Logisim***

O programa *If*, *Elseif* e *Else* funciona a partir de desvios condicionais e incondicionais relacionados, de forma que a finalização de todas as condições apontem para o mesmo endereço de código, mas o funcionamento interno de cada condicional seja inacessível para todas as outras condições. Os passos do programa *If*, *Elseif* e *Else* são descritos na Tabela X.

**Tabela 18 – Tabela Programa *If*, *Elseif* e *Else* Processador NAV / Gerado a partir de Testes**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Linha de Código** | **Instrução** | **Classe de Instrução** | **Binário** | | | **Valor de Entrada** | **Descrição** |
| **Código de Operação** | **Endereço 1** | **Endereço 2** |
| **Ln0** | **DR** | E1V | 0001 | 00 | -- | 00000001 | Declara R1 |
| **Ln1** | **DR** | E1V | 0001 | 01 | -- | 00010000 | Declara R2 |
| **Ln2** | **SEMA** | E2V | 1010 | 00 | 01 | 00000101 | Salta para a linha 5 se R1 > R2 |
| **Ln3** | **SEME** | E2V | 1100 | 00 | 01 | 00000111 | Salta para a linha 7 se R1 < R2 |
| **Ln4** | **J** | V | 1111 | -- | -- | 00001001 | Salta para a linha 9 |
| **Ln5** | **SOMC** | E1V | 0010 | 00 | -- | 00000001 | Soma 1 em R1 |
| **Ln6** | **J** | V | 1111 | -- | -- | 00001001 | Salta para a linha 9 |
| **Ln7** | **SUBC** | E1V | 0100 | 00 | -- | 00000001 | Subtrai 1 de R1 |
| **Ln8** | **J** | V | 1111 | -- | -- | 00001001 | Salta para a linha 9 |
| **Ln9** | **IMP** | E1 | 0000 | **00** | **--** | -------- | Imprime R1 |

* 1. **Simulação *While***



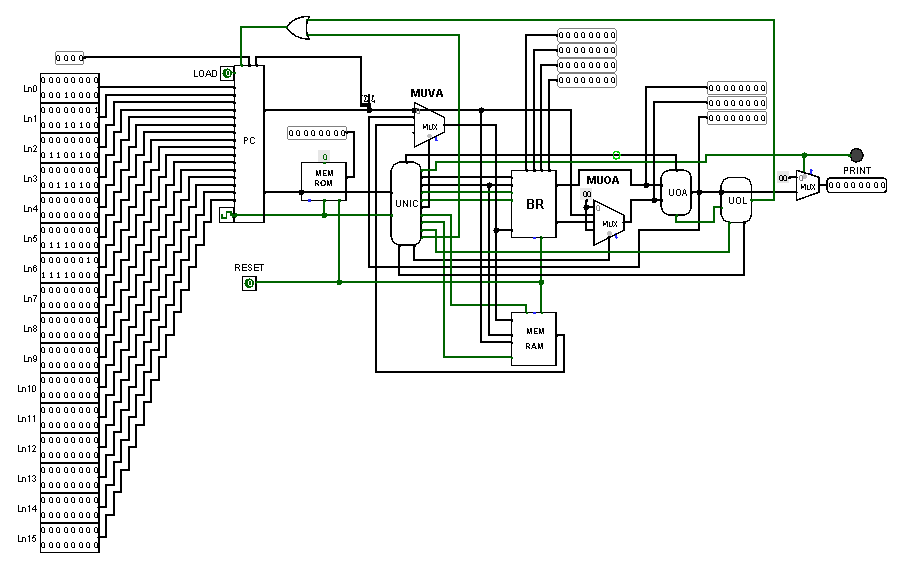
**Figura 34 – Programa *While* no Processador NAV / Gerado no *Logisim***

O programa *While* funciona a partir de desvios condicionais e incondicionais relacionados, de forma que apenas quando uma condição principal de laço seja satisfeita o programa será finalizado. Os passos do programa *While* são descritos na Tabela X.

**Tabela 19 – Tabela Programa *If*, *Elseif* e *Else* Processador NAV / Gerado a partir de Testes**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Linha de Código** | **Instrução** | **Classe de Instrução** | **Binário** | | | **Valor de Entrada** | **Descrição** |
| **Código de Operação** | **Endereço 1** | **Endereço 2** |
| **Ln0** | **DR** | E1V | 0001 | 00 | -- | 00000001 | Declara R1 |
| **Ln1** | **DR** | E1V | 0001 | 01 | -- | 00010000 | Declara R2 |
| **Ln2** | **SEMA** | E2V | 1010 | 00 | 01 | 00000101 | Salta para a linha 5 se R1 > R2 |
| **Ln3** | **SOMC** | E1V | 0010 | 00 | -- | 00000001 | Soma 1 em R1 |
| **Ln4** | **J** | V | 1111 | -- | -- | 00000010 | Salta para a linha 2 |
| **Ln5** | **IMP** | E1 | 0000 | 00 | **--** | -------- | Imprime R1 |
| **Ln6** | **J** | V | 1111 | -- | -- | 00000101 | Salta para a linha 5 |

* 1. **Simulação Fibonacci**



**Figura 35 – Programa Fibonacci no Processador NAV / Gerado no *Logisim***

O programa Fibonacci funciona a partir de desvios incondicionais, soma de registradores e alocação de valor de registradores na memória, de forma que os valores anteriores são salvos na memória e após isto são carregados nos registradores novamente. Os passos do programa Fibonacci são descritos na Tabela X.

**Tabela 20 – Tabela Programa Fibonacci Processador NAV / Gerado a partir de Testes**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Linha de Código** | **Instrução** | **Classe de Instrução** | **Binário** | | | **Valor de Entrada** | **Descrição** |
| **Código de Operação** | **Endereço 1** | **Endereço 2** |
| **Ln0** | **DR** | E1V | 0001 | 00 | -- | 00000000 | Declara R1 |
| **Ln1** | **DR** | E1V | 0001 | 01 | -- | 00000001 | Declara R2 |
| **Ln2** | **GRA** | E2V | 0110 | 01 | 00 | 00000000 | Armazena o valor de R2 em M1[0] |
| **Ln3** | **SOMR** | E2 | 0011 | 01 | 00 | -------- | Soma R2 com R1 e atribui para R2 |
| **Ln4** | **IMP** | E1 | 0000 | 00 | -- | -------- | Imprime o valor de R1 |
| **Ln5** | **CRA** | E2V | 0111 | 00 | 00 | 00000000 | Carrega o valor de M1[0] em R1 |
| **Ln6** | **J** | V | 1111 | -- | -- | 00000010 | Salta para a linha 2 |

## Considerações finais

Este trabalho apresentou o projeto e implementação do processador de 8 bits denominado de NAV. No decorrer da disciplina e de todo conteúdo apresentado, foi ensinado o quão a arquitetura de computadores influencia em cada passo dentro e fora da faculdade; neste projeto o aprofundamento foi ainda maior, proporcionando uma grande experiência para os alunos juntamente com o ilustríssimo Prof. Herbert Oliveira.