

**PODER EXECUTIVO**

**MINISTÉRIO DA EDUCAÇÃO**

**UNIVERSIDADE FEDERAL DE RORAIMA**

**DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO**

**ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES**

**RELATÓRIO DO PROJETO: PROCESSADOR XLR-8**

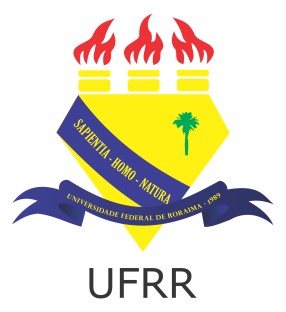
**ALUNOS:**

**Lucas Bessa Façanha Pereira – 2019005103**

**Rafael Nóbrega de Lima – 2019037555**

**Maio de 2021**

**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 XLR-8**

**Maio de 2021**

**Boa Vista/Roraima**

Resumo

O projeto aborda a elaboração e implementação do processador RISC (Reduced Instruction Set Computer) XLR-8 monociclo de 8 bits baseado na arquitetura do processador MIPS (Microprocessor without Interlocked Pipeline Stages). Este relatório abordará as descrições de todos os componentes básicos para o bom funcionamento do processador, tabelas exemplificando as instruções suportadas e mostrará todos os testes realizados durante a implementação. Toda a execução do projeto, incluindo imagens e testes gerados, ocorreu através da utilização da ferramenta Quatus Prime Lite da Intel Corporation e da linguagem de descrição de hardware VHDL.

**Lista de Figuras**

[**Figura 1 - Especificações do Quartus** 7](#_Toc72273492)

[**Figura 2 - RTL VIEWER da ULA** 10](#_Toc72273493)

[**Figura 3 - RTL VIEWER do Banco de Registradores** 11](#_Toc72273494)

[**Figura 4 - tempo de mudança de estado no clock**. 12](#_Toc72273495)

[**Figura 5 - RTL VIEWER da memória de dados** 13](#_Toc72273496)

[**Figura 6 - RTL VIEWER da Memória Rom.** 14](#_Toc72273497)

[**Figura 7 - RTL VIEWER do Program Counter (PC).** 14](#_Toc72273498)

[**Figura 8 - RTL VIEWER do somador do PC** 15](#_Toc72273499)

[**Figura 9 - RTL VIEWER do Multiplexador 2x1.** 15](#_Toc72273500)

[**Figura 10 - RTL VIEWER da porta and** 15](#_Toc72273501)

[**Figura 11 - Representação do datapath feito com draw.io** 17](#_Toc72273502)

[**Figura 12 - RTL VIEWER do processador** 18](#_Toc72273503)

[**Figura 13 - Teste de Fatorial 1 utilizando waveforms** 19](#_Toc72273504)

[**Figura 14 - Teste de Fatorial 2 utilizando waveforms** 20](#_Toc72273505)

[**Figura 15 - Teste de Fibonacci 1 utilizando waveforms** 20](#_Toc72273506)

[**Figura 16 - Teste de Fibonacci 2 utilizando waveforms** 21](#_Toc72273507)

[**Figura 17 - Teste de Fibonacci 3 utilizando waveforms** 21](#_Toc72273508)

**Lista de Tabelas**

[**Tabela 1 - Divisão de bits para instruções do tipo R em binário** 8](#_Toc72273535)

[**Tabela 2 - Instrução Omni do tipo R** 8](#_Toc72273536)

[**Tabela 3 - Divisão de bits para instruções do tipo I em binário** 8](#_Toc72273537)

[**Tabela 4 - Instrução Omni do tipo I** 8](#_Toc72273538)

[**Tabela 5 - Divisão de bits para instruções do tipo J em binário** 8](#_Toc72273539)

[**Tabela 6 - Instrução Omni do tipo J** 8](#_Toc72273540)

[**Tabela 7 - Descrição das instruções do processador XRL-8** 9](#_Toc72273541)

[**Tabela 8 - Flags de controle da Unidade de Controle** 13](#_Toc72273542)

[**Tabela 9 - Teste de cálculo do Fatorial** 19](#_Toc72273543)

[**Tabela 10 - Teste de cálculo dos valores da Sequência Fibonacci** 20](#_Toc72273544)

**Sumário**

Sumário

[Especificação 7](#_Toc72273979)

[Plataforma de desenvolvimento 7](#_Toc72273980)

[Conjunto de instruções 7](#_Toc72273981)

[Descrição do Hardware 9](#_Toc72273982)

[ULA ou ALU 9](#_Toc72273983)

[Banco de Registradores 10](#_Toc72273984)

[Clock 11](#_Toc72273985)

[Unidade de Controle 12](#_Toc72273986)

[Memória de dados 13](#_Toc72273987)

[Memória de instruções 14](#_Toc72273988)

[PC (Program Counter) 14](#_Toc72273989)

[Somador PC 14](#_Toc72273990)

[Multiplexador 2x1 15](#_Toc72273991)

[Porta And 15](#_Toc72273992)

[Datapath 17](#_Toc72273993)

[Simulações e Testes 19](#_Toc72273994)

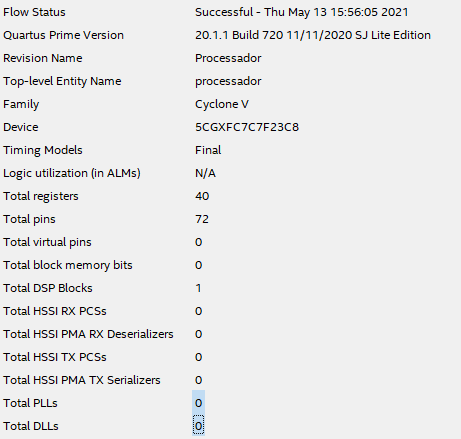
[Considerações Finais 22](#_Toc72273995)

## Especificação

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

## Plataforma de desenvolvimento

Para a implementação do processador XLR-8 foi utilizado a IDE:



**Figura 1 - Especificações do Quartus**

## Conjunto de instruções

O processador XLR-8 possui 4 registradores: $s0, $s1, $s2, $s3. Assim como 15 formatos de instruções de 8 bits cada. Primariamente, as instruções deste processador seguem um padrão de divisão de bits por blocos de funcionalidade, desta maneira cada bloco realiza uma função específica no barramento do processador:

* **Opcode**: bloco destinado para representar as operações básicas que serão executadas no processador, comumente chamado de código de operação;
* **Reg1**: representa o registrador que contém o primeiro operando fonte, e, em alguns tipos de instruções, como as do tipo R, é o registrador de destino;
* **Reg2**: representa o registrador contendo o segundo operando fonte;

Tipo de Instruções:

**- Formato do tipo R:** Formato padrão de instruções que realizam operações aritméticas e/ou lógicas entre os registradores.

Formato para escrita em código binário:

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

**Tabela 1 - Divisão de bits para instruções do tipo R em binário**

Formato para escrita em baixo nível:

|  |  |  |
| --- | --- | --- |
| **Operação**, | operando1 | operando2. |

**Tabela 2 - Instrução Omni do tipo R**

**- Formato do tipo I**: Formato padrão de instruções que realizam operações em memória.

Formato para escrita em código binário:

|  |  |  |
| --- | --- | --- |
| 4 bits | 2 bits | 2 bits |
| 7-4 | 3-2 | 1-0 |
| Opcode | Reg1 | Endereço/constante |

**Tabela 3 - Divisão de bits para instruções do tipo I em binário**

Formato para escrita em baixo nível:

|  |  |  |
| --- | --- | --- |
| **Operação**, | operando1 | Endereço/constante |

**Tabela 4 - Instrução Omni do tipo I**

**- Formato do tipo J:** Formato padrão de instruções que realizam operações de salto.

Formato para escrita em código binário:

|  |  |
| --- | --- |
| 4bits | 2 bits |
| 7-4 | 3-0 |
| Opcode | Endereço |

**Tabela 5 - Divisão de bits para instruções do tipo J em binário**

Formato para escrita em baixo nível:

|  |  |
| --- | --- |
| **Operação**. | endereço de salto |

**Tabela 6 - Instrução Omni do tipo J**

**Visão geral das instruções do Processador XLR-8:**

Como o processador XLR-8 é do tipo RISC 8 bits, a seleção de bits foi realizada estrategicamente e de forma abranger uma quantidade reduzida, porém primordial de instruções para o devido funcionamento do processador. A linguagem suportada pelo processador é chamada de Omni e esta suporta até 2 ^ (número de bits do opcode) - 1 instruções.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instrução | Tipo | Opcode | Sintaxe | Registradores |
| Add | R | (0000) | Add $s0,$s1 | 2 |
| Sub | R | (0001) | Sub $s0,$s1 | 2 |
| Lw | I | (0010) | Lw $s0, address | 2 |
| Sw | I | (0011) | Sw $s0,address | 2 |
| J | J | (0100) | J address | 0 |
| Beq | J | (0101) | Beq address | 0 |
| Bne | J | (0110) | Bne address | 0 |
| Addi | R | (0111) | Addi $s0,valor | 1 |
| mul | R | (1000) | Mul $s0,$s1 | 2 |
| And | R | (1001) | And $s0,$s1 | 2 |
| Or | R | (1010) | Or $s0,$s1 | 2 |
| Not | R | (1011) | Not $s0 | 1 |
| Li | R | (1100) | Li $s0,value | 2 |
| Move | R | (1101) | Move $s0,$s1 | 2 |
| JumpC | R | (1110) | JumpC $s0,$s1 | 2 |

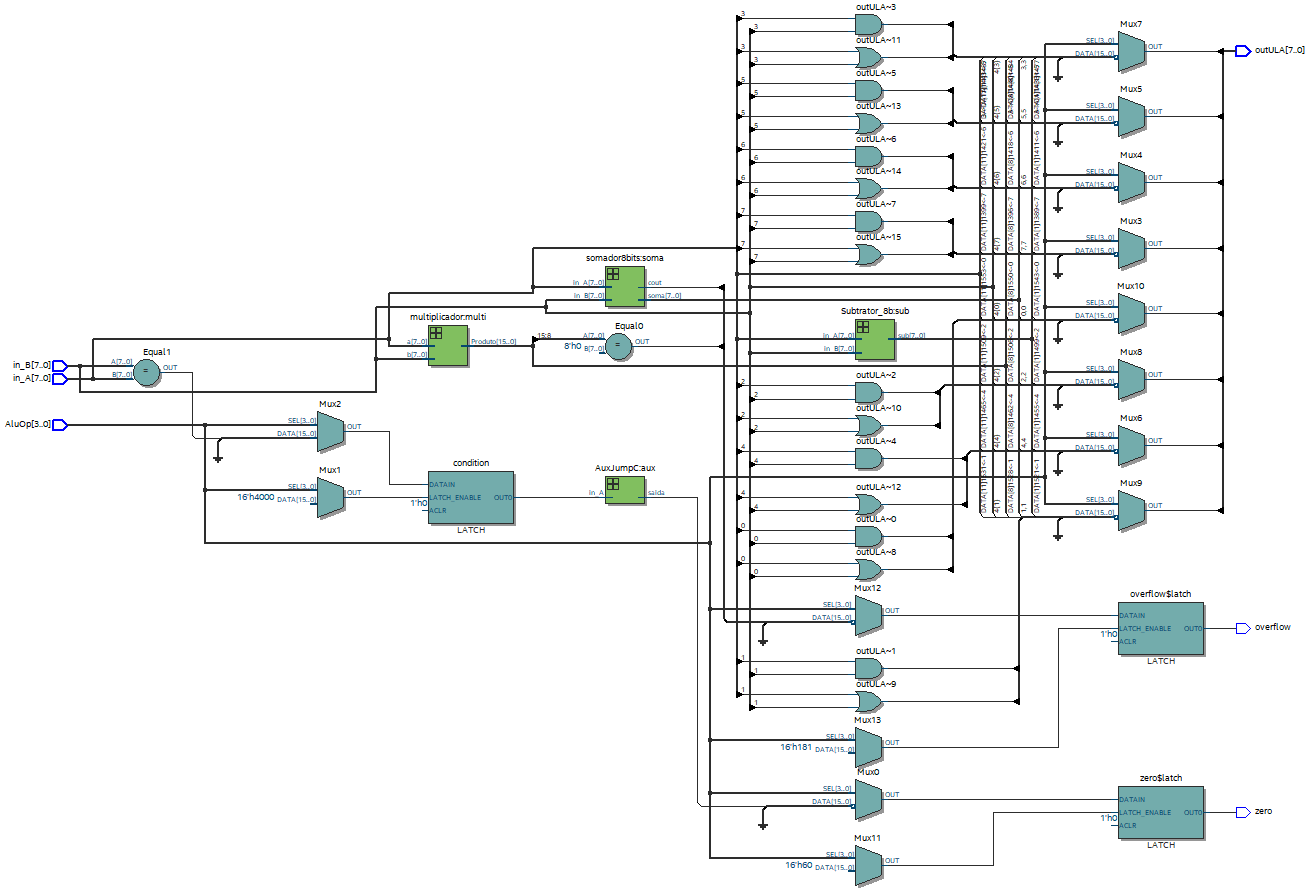
**Tabela 7 - Descrição das instruções do processador XRL-8**

## Descrição do Hardware

## ULA ou ALU

A ULA também conhecida como Unidade Lógica e aritmética (ou em Inglês Arithmetic Logic Unit), é um dispositivo que realiza operações lógicas e aritméticas com os valores passados na entrada. No processador XRL-8, este componente possui 3 entradas e 3 saídas entre elas:

* **Entrada 1:** Que recebe uma trilha de 8 bits vinda do banco de registradores.
* **Entrada 2:** Recebe uma trilha também de 8 bits vinda de um multiplexador.
* **AluOp:** Flag de controle da Unidade de Controle, que indica ao componente qual operação deve ser realizada.
* **Zero:** Flag da ULA que retorna para o barramento o sinal alto “1” toda vez que ocorrer um salto condicional. Desta maneira o sinal da flag “ZERO” em conjunto com o sinal da Unidade de controle “Branch” irão selecionar a trilha correta para que ocorra ou não o salto na memória de instruções.
* **Resultado:** trilha de 8 bits que passará para o barramento o resultado das operações realizadas na ULA.
* **Overflow:** Flag que indica ao barramento se houve ou não um estouro de memória na execução da operação na ULA.

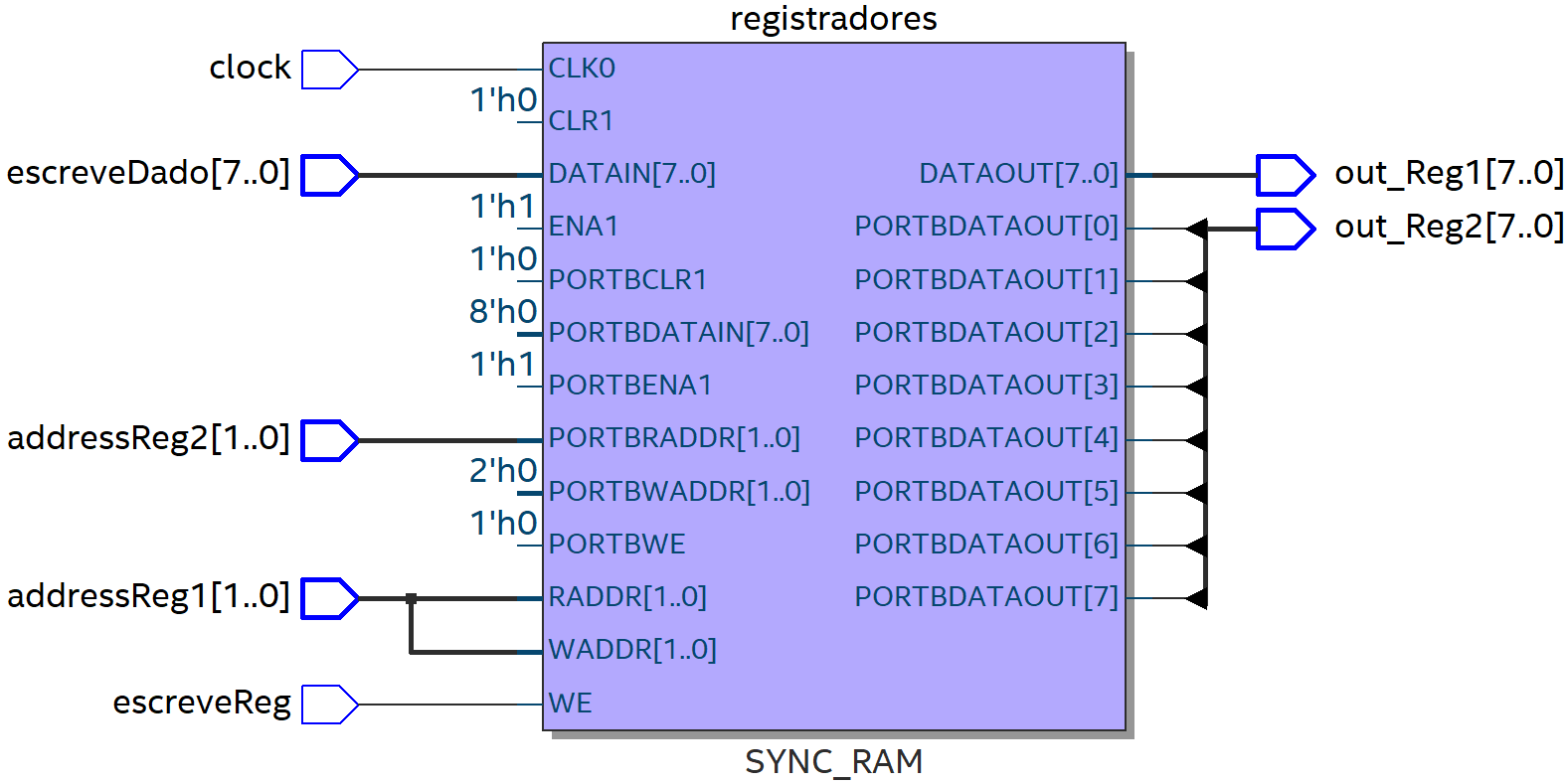


**Figura 2 - RTL VIEWER da ULA**

## Banco de Registradores

É um componente composto por um conjunto de registradores que podem ser acessados de maneira organizada. Podem ser executadas operações de leitura dos dados anteriormente gravados e de escrita de dados para modificar as informações internas. Por ter essas capacidades é um dos componentes mais importantes para o fluxo de dados no processador. O banco de Registradores, possui 3 entradas de dados, 2 entradas de saída de dados e 1 entrada de sinal vindo da unidade de controle:

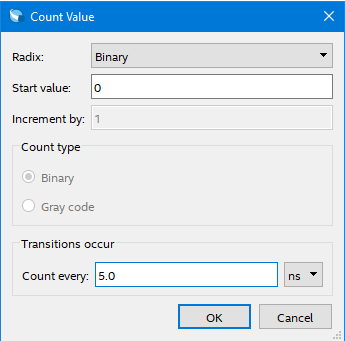
* Reg1: Entrada de 2 bits correspondente ao endereço do registrador que será acessado para a saída de dado 1 do componente.
* Reg2: Entrada de 2 bits correspondente ao endereço do registrador que será acessado para a saída de dado 2 do componente.
* Escrevedado: Entrada de 8 bits correspondente ao valor que será ou não armazenado em determinado registrador do banco. Em instruções do tipo R, o endereço do registrador que será guardado o valor é o que entra pela entrada Reg1. Além disso, o valor somente será guardado no processador se o clock estiver com valor alto.
* EscreveReg: Entrada de 1 bit vinda da Unidade de controle. Quando esta tiver valor alto “1”, o dado que entra em Escrevedado será guardado em determinado registrador.



**Figura 3 - RTL VIEWER do Banco de Registradores**

## Clock

É um temporizador que define no barramento quais componentes devem estrar ativos ou inativos. O período de ciclo de clock do processador XRL-8 é de 10 ns e como o processador é monociclo, todas as instruções serão executadas em um único ciclo de clock



**Figura 4 - tempo de mudança de estado no clock**.

## Unidade de Controle

Componente que terá função de controlar todo o fluxo de dados do barramento. A unidade possui uma trilha de 4 bits como valor de entrada, correspondente ao Opcode, e 8 flags de saída que desempenharão o controle do comportamento dos componentes para que determinados tipos de instruções ocorram de maneira esperada:

* **OrigAlu**: Flag que indica ao multiplexador se o valor da entrada da ULA virá do banco de registradores ou do extensor de sinal.
* **EscreveReg**: Utilizada para a escrita de dados no Banco de registradores. Quando ativa, um determinado dado é guardado em algum dos registradores.
* **EscreveMem**: Indica à memória de dados se um valor de entrada deve ou não ser guardado.
* **AluOp**: Flag vinda da unidade de controle onde será definida qual instrução será executada.
* **MemToReg**: Dependendo do sinal, determina qual valor será passado para guardar no Banco de Registradores.
* **LerMem**: Determina se o valor da memória de dados é passado para o barramento ou não.
* **Branch:** Dependendo da saída da Flag Zero, determina se um salto deverá ou não ser realizado.
* **Jump**: Uso exclusivo para a operação Jump, determina se um salto incondicional deve ou não ser realizado.

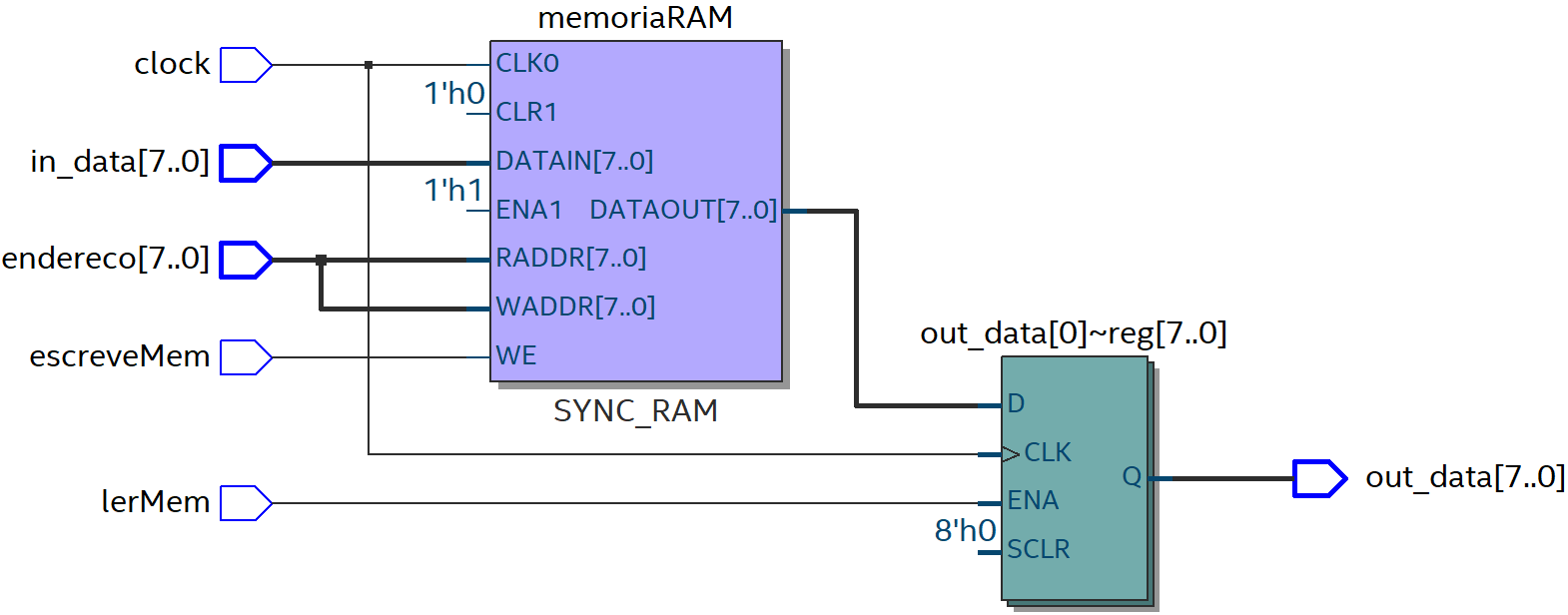
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Funcionamento da Unidade de Controle | | | | | | | | | |
| Instruções | Orig  Alu | Escreve  Reg | Escreve  Mem | Alu  Op | Mem  To  Reg | Ler  Mem | Branch | | Jump |
| Add | 0 | 1 | 0 | 0000 | 1 | 0 | 0 | 0 | |
| Sub | 0 | 1 | 0 | 0001 | 1 | 0 | 0 | 0 | |
| Lw | X | 1 | 0 | 0010 | 0 | 1 | 0 | 0 | |
| Sw | X | 0 | 1 | 0011 | 1 | 0 | 0 | 0 | |
| J | X | 0 | 0 | 0100 | 1 | 0 | 0 | 1 | |
| Beq | 0 | 0 | 0 | 0101 | 1 | 0 | 1 | 0 | |
| Bne | 0 | 0 | 0 | 0110 | 1 | 0 | 1 | 0 | |
| Addi | 1 | 1 | 0 | 0111 | 1 | 0 | 0 | 0 | |
| Mul | 0 | 1 | 0 | 1000 | 1 | 0 | 0 | 0 | |
| And | 0 | 1 | 0 | 1001 | 1 | 0 | 0 | 0 | |
| Or | 0 | 1 | 0 | 1010 | 1 | 0 | 0 | 0 | |
| Not | X | 1 | 0 | 1011 | 1 | 0 | 0 | 0 | |
| Li | 1 | 1 | 0 | 1100 | 1 | 0 | 0 | 0 | |
| Move | 0 | 1 | 0 | 1101 | 1 | 0 | 0 | 0 | |
| JumpC | 0 | 0 | 0 | 1110 | 1 | 0 | 1 | 0 | |
|  |  |  |  |  |  |  |  |  | |

**Tabela 8 - Flags de controle da Unidade de Controle**

## Memória de dados

Componente Funcional utilizado para armazenar dados gerais através do endereçamento. Este componente pode armazenar até 2^ (número de bits do processador) - 1 dados, valor equivalente a 255 espaços de 8 bits de armazenamento. Este componente, possui duas entradas de 8 bits, duas entradas de 1 bit e uma saída de 8 bits.

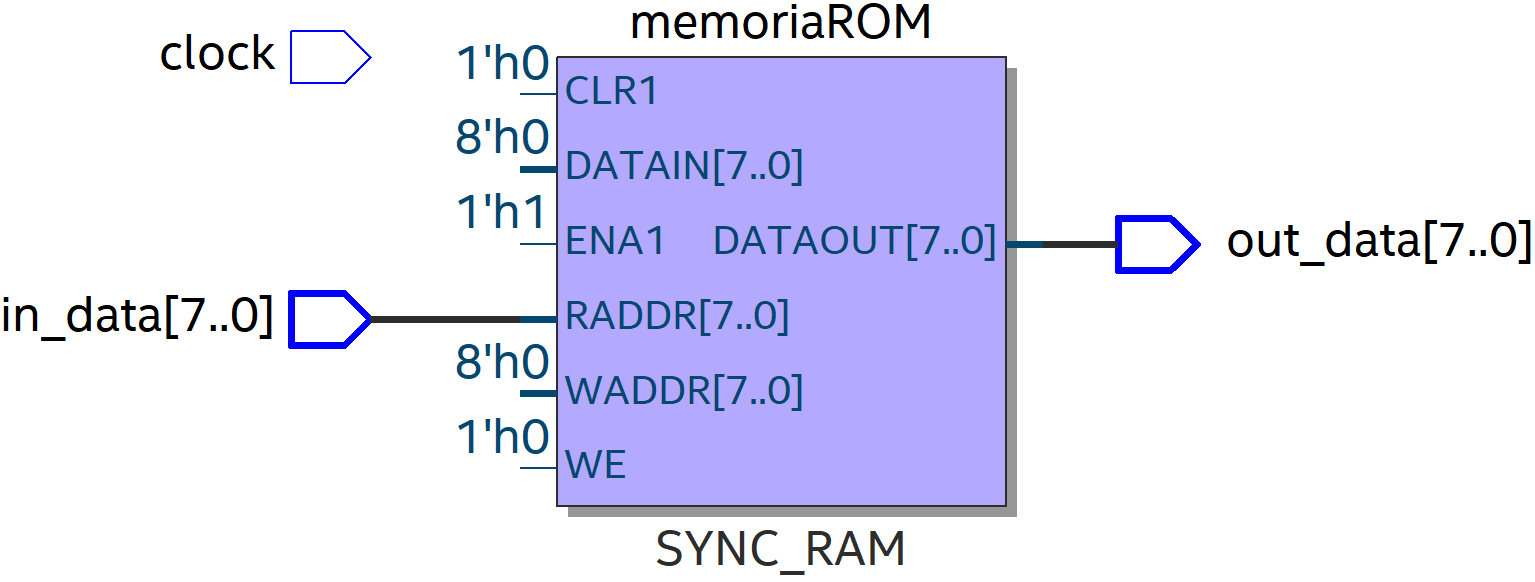
* In\_data: Entrada de dado de 8 bits que será ou não guardado na memória de acordo com o valor da flag EscreveMem.
* Endereço: Entrada de 8 bits correspondente ao endereço em que será guardado o dado da entrada in\_data.
* EscreveMem: Flag da Unidade de Controle que quando possuir valor alto fará com que o valor de in\_data seja guardado no endereço selecionado. Este valor somente será guardado somente de o clock possuir valor alto.
* LerMem: Flag da Unidade de Controle que quando possuir valor alto fará com que o valor do Endereço passado vá para a saída da memória.
* Out\_data: Saída de 8 bits da memória de dados.



**Figura 5 - RTL VIEWER da memória de dados**

## Memória de instruções

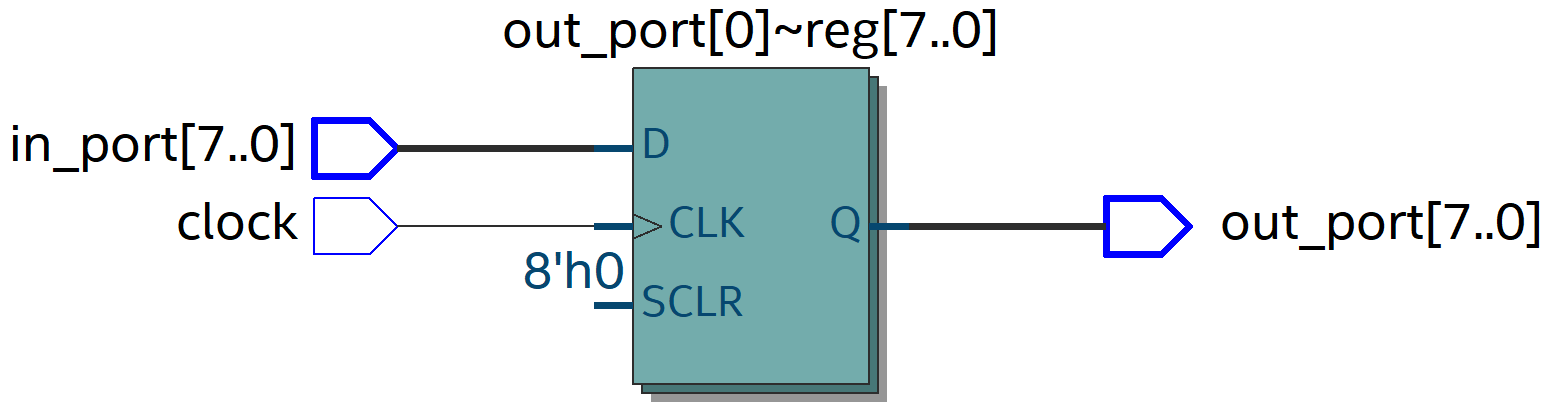
Componente do processador cuja principal função é alimentar o barramento com uma trilha de bits corresponde à instrução que será executada pelo processador.



**Figura 6 - RTL VIEWER da Memória Rom.**

## PC (Program Counter)

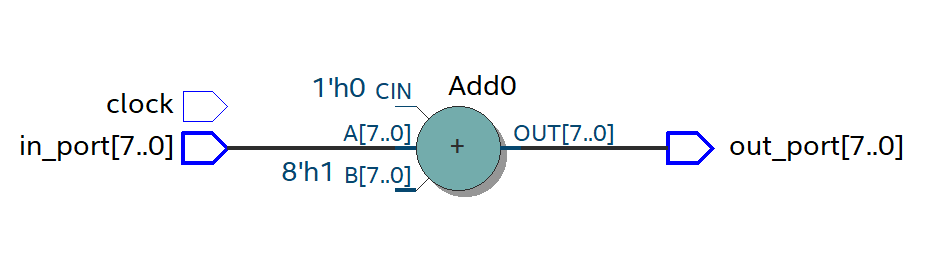
Componente do barramento cuja única função é passar uma trilha de bits para a memória de instruções correspondente aos endereços das instruções que serão executadas pelo processador.



**Figura 7 - RTL VIEWER do Program Counter (PC).**

## Somador PC

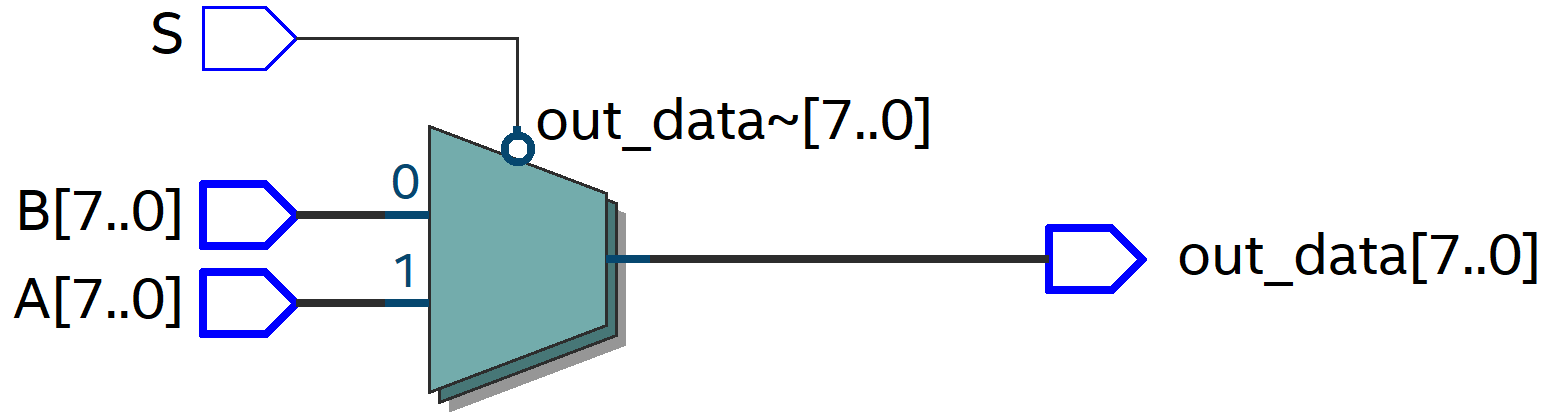
Componente que tem a função de incrementar o valor do PC e assim manter o fluxo de instruções no barramento. Possui uma entrada, correspondente à trilha de bits que vem da ULA e uma saída que corresponde a soma da trilha de entrada + 1, desta maneira o valor do PC vai aumentando constantemente até o fim da execução de todas as instruções da memória de instruções.



**Figura 8 - RTL VIEWER do somador do PC**

## Multiplexador 2x1

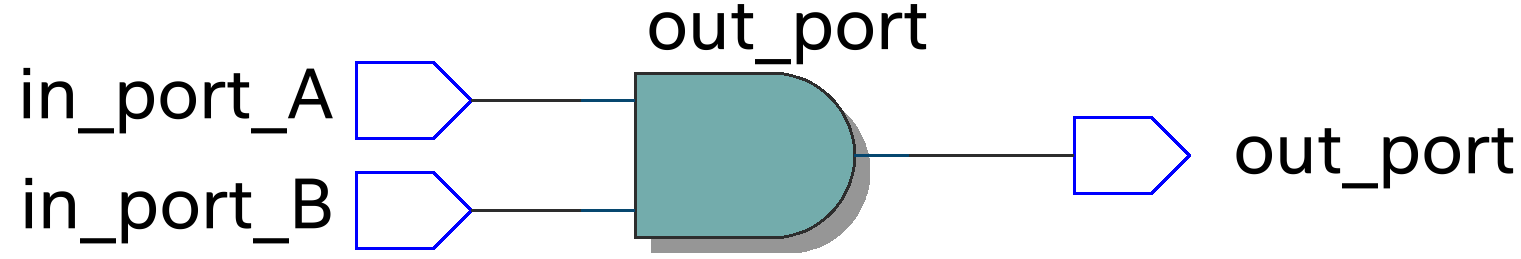
Componente cuja função é realizar uma seleção de trilhas. O componente possui 3 entradas e 1 saída. As entradas A e B são trilhas de 8 bits e a entrada S é uma entrada de 1 bits que realizará a seleção de trilhas. Assim, caso o valor de S seja 0, A trilha A vai para a saída, caso contrário a trilha B vai para a saída.



**Figura 9 - RTL VIEWER do Multiplexador 2x1.**

## Porta And

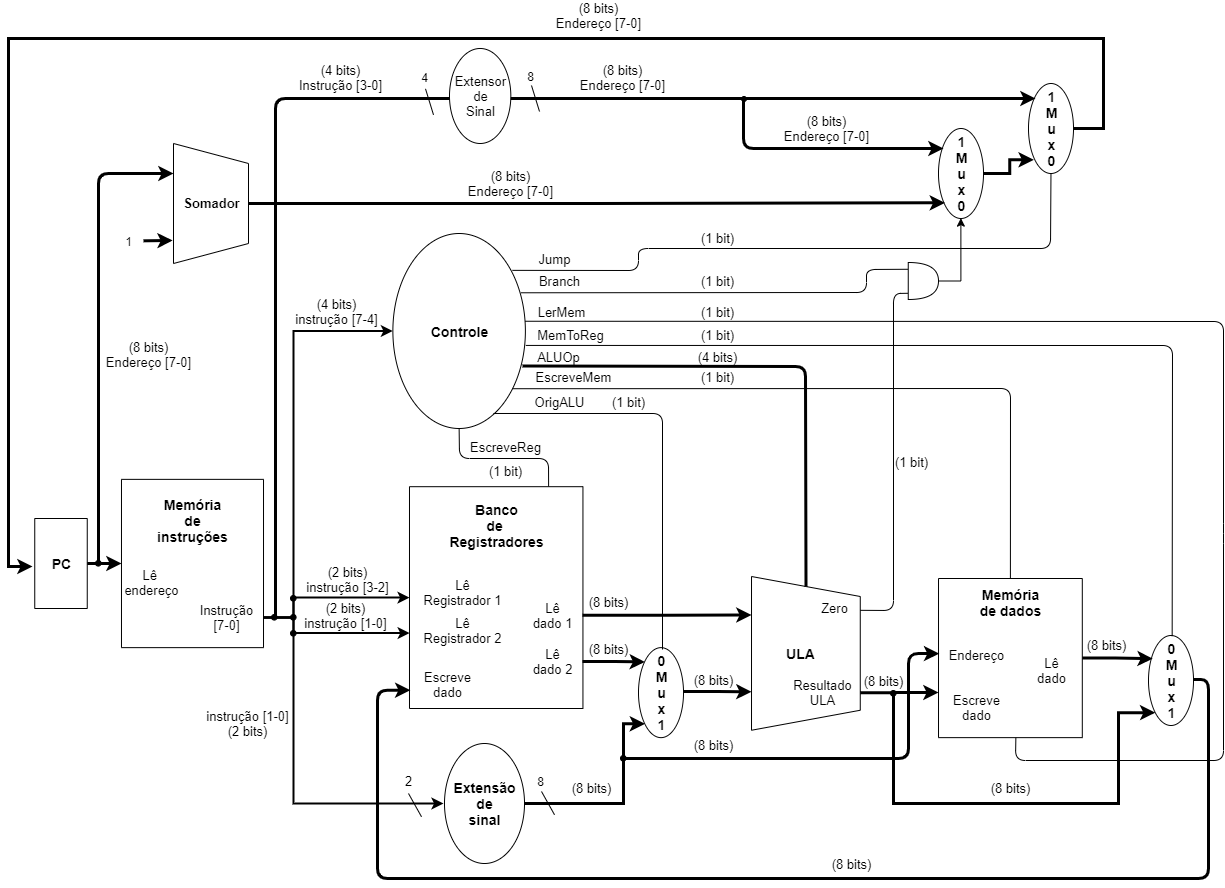
Porta lógica que retorna valor alto somente quando as duas entradas possuem valor alto. No caso do processador XLR-8, as duas entradas correspondem ao valor da flag “branch” e a saída Zero da ULA. Já a saída servirá como seletor de um multiplexador, assim quando a flag “branch” e a saída Zero da ULA possuírem valor alto, a porta and selecionará uma trilha de bits correspondente ao endereço de salto.



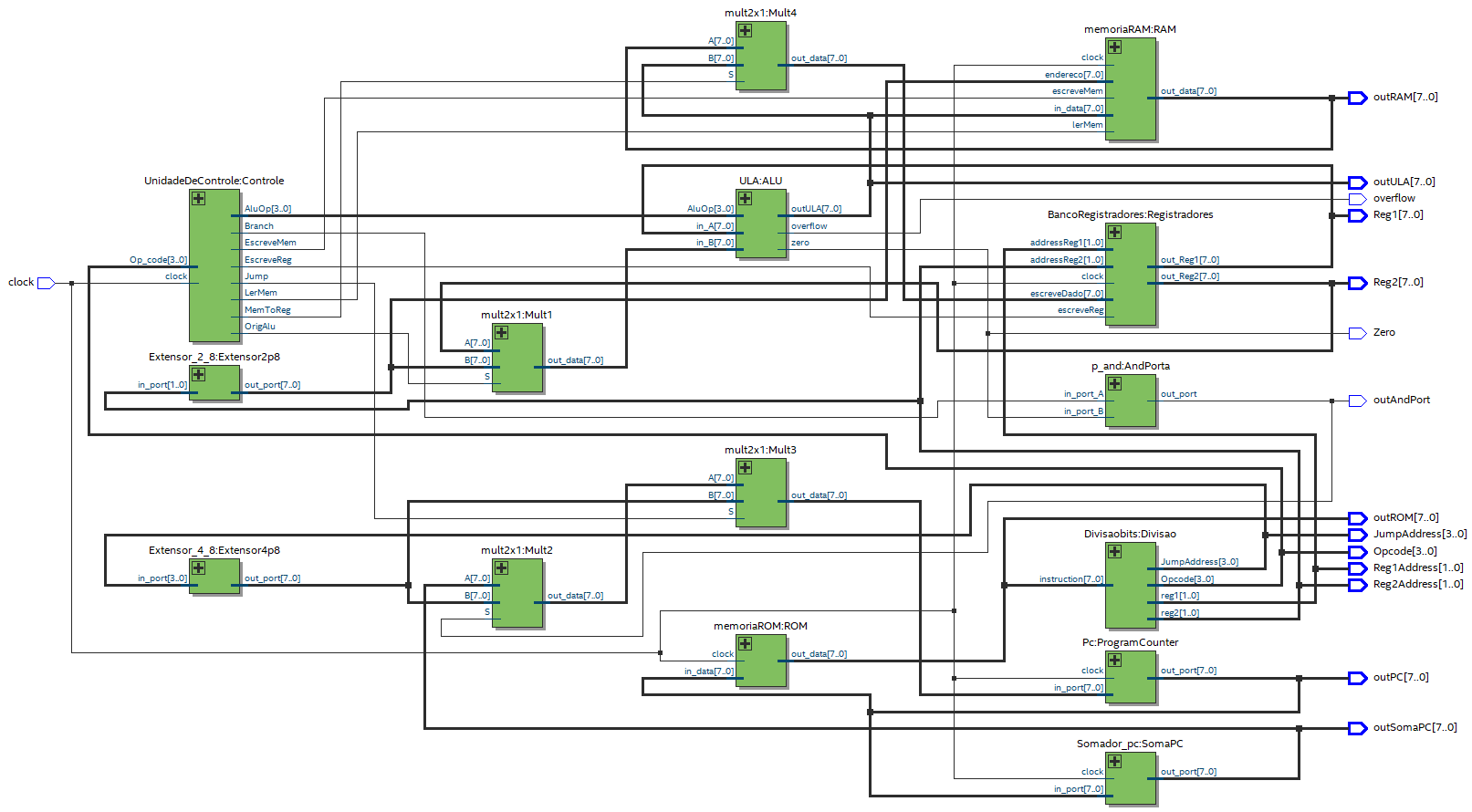
**Figura 10 - RTL VIEWER da porta and**

## Datapath

Barramento do processador, circuito lógico que contém todos os componentes que fazem parte do processador XLR-8 e suas respectivas interligações.



**Figura 11 - Representação do datapath feito com draw.io**



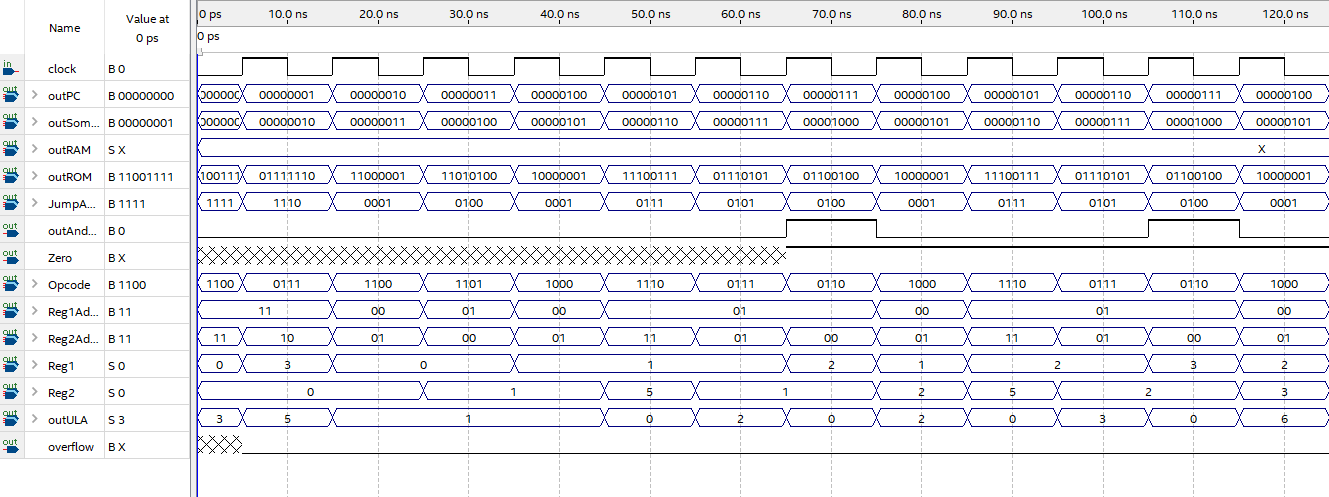
**Figura 12 - RTL VIEWER do processador**

## Simulações e Testes

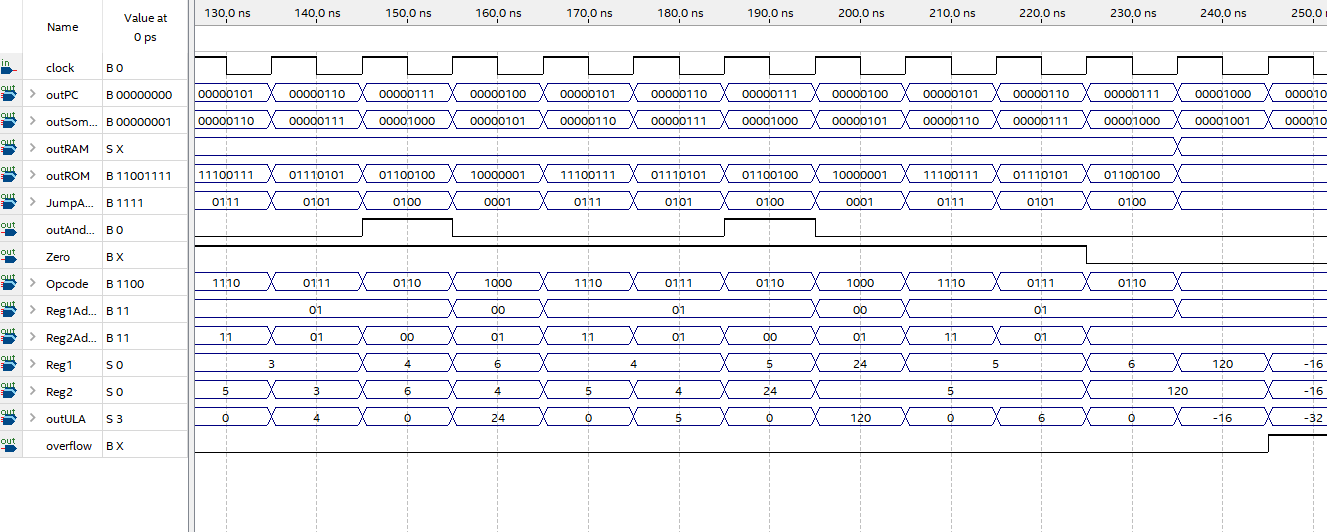
Aqui apresentamos, alguns testes realizados no Quartus Prime Lite, com o objetivo de testar o processador e seu comportamento através das instruções.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Endereço** | **Instrução (Omni)** | **Instrução (Binário)** | | |
| **Opcode** | **Reg1** | **Reg2** |
| **Endereço** | |
| 0 | Li $s3, 3 | 1100 | 11 | 11 |
| 1 | Addi $s3, 2 | 0111 | 11 | 10 |
| 2 | Li $s0, 1 | 1100 | 00 | 01 |
| 3 | Move $s1, $s0 | 1101 | 01 | 00 |
| 4 | Mul $s0, $s1 | 1000 | 00 | 01 |
| 5 | JumpC $s1, $s3 | 1110 | 01 | 11 |
| 6 | Addi $s1, 1 | 0111 | 01 | 01 |
| 7 | Bne 4 | 0110 | 0100 | |

**Tabela 9 - Teste de cálculo do Fatorial**

****

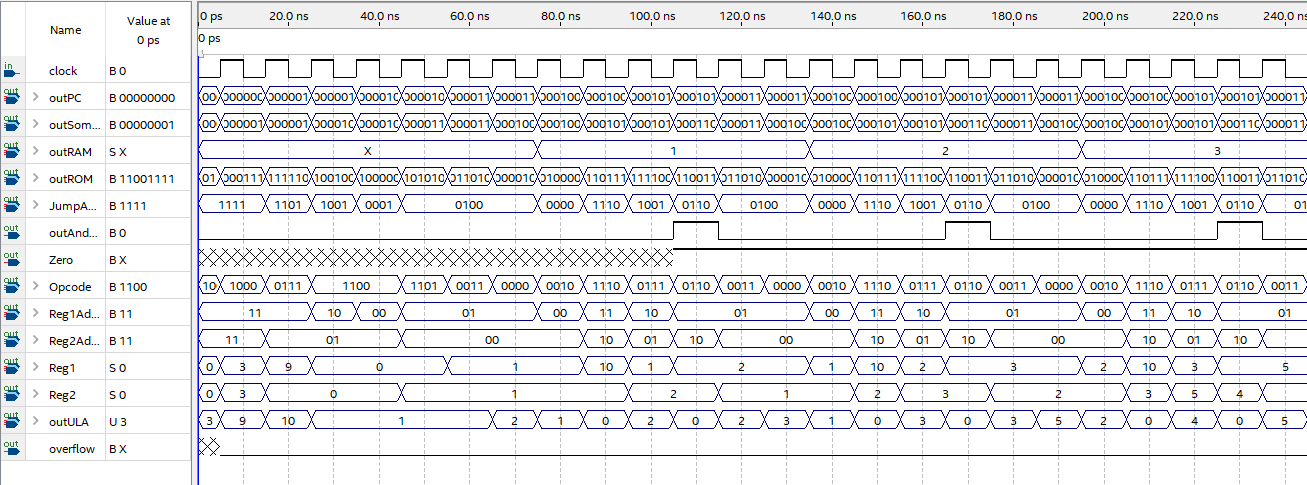
**Figura 13 - Teste de Fatorial 1 utilizando waveforms**

****

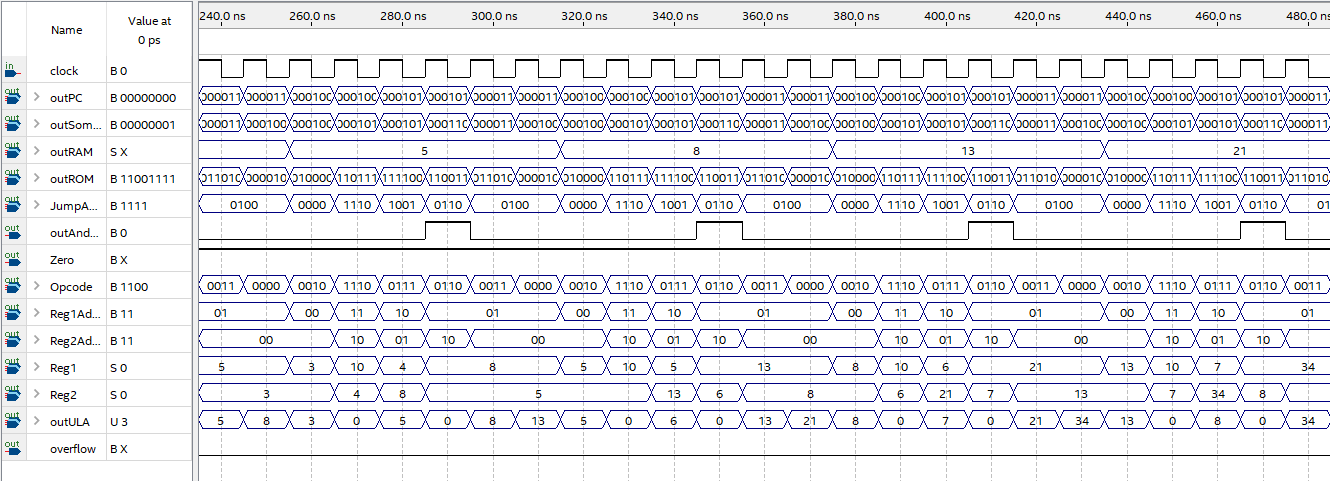
**Figura 14 - Teste de Fatorial 2 utilizando waveforms**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Endereço** | **Instrução (Omni)** | **Instrução (Binário)** | | |
| **Opcode** | **Reg1** | **Reg2** |
| **Endereço** | |
| 0 | Li $s3, 3 | 1100 | 11 | 11 |
| 1 | Mult $s3, $s3 | 1000 | 11 | 11 |
| 2 | Addi $s3, 1 | 0111 | 11 | 01 |
| 3 | Li $s2, 1 | 1100 | 10 | 01 |
| 4 | Li $s0, 1 | 1100 | 00 | 01 |
| 5 | Move $s1, $s0 | 1101 | 01 | 00 |
| 6 | Sw $s1, 0 | 00110 | 01 | 00 |
| 7 | Add $s1, $s0 | 0000 | 01 | 00 |
| 8 | Lw $s0, 0 | 0010 | 00 | 00 |
| 9 | JumpC $s3, $s2 | 1110 | 11 | 10 |
| 10 | Addi $s2, 1 | 0111 | 10 | 01 |
| 11 | Bne 6 | 0110 | 0110 | |

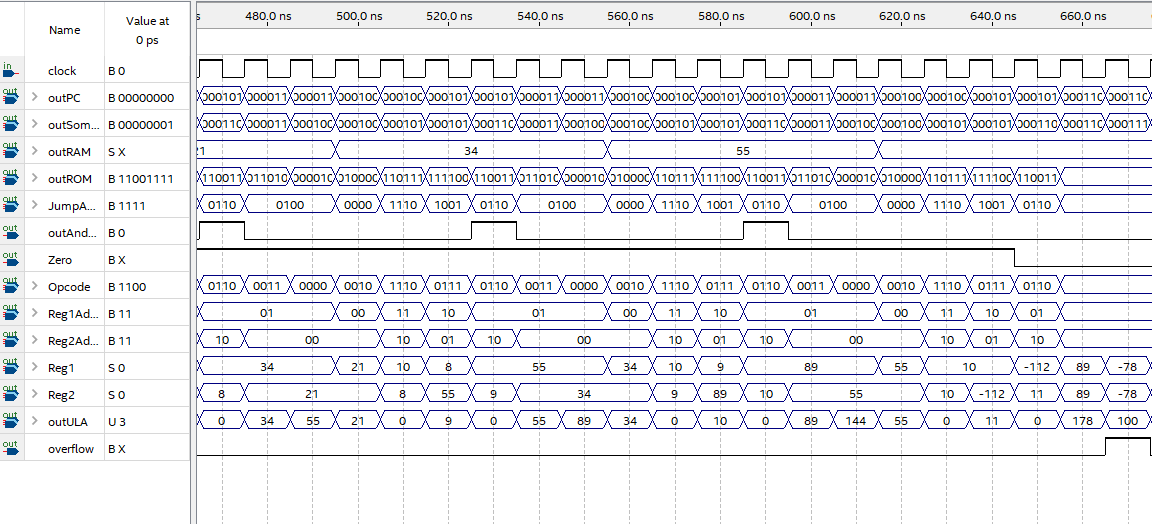
**Tabela 10 - Teste de cálculo dos valores da Sequência Fibonacci**

****

**Figura 15 - Teste de Fibonacci 1 utilizando waveforms**

****

**Figura 16 - Teste de Fibonacci 2 utilizando waveforms**

****

**Figura 17 - Teste de Fibonacci 3 utilizando waveforms**

## Considerações Finais

Esta foi a apresentação do funcionamento e das especificações do processador XLR-8 de 8 bits baseado na arquitetura do processador MIPS.

O projeto se desenvolveu com várias limitações devido ao tipo de arquitetura, porém ainda é capaz de realizar variadas operações e funções.