

**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**

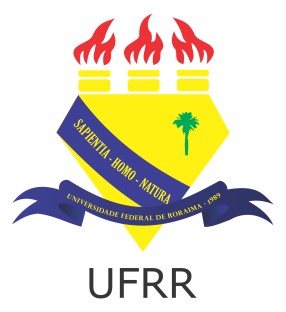
**ALUNOS:**

**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 pelo processador assim como mostrará todos os testes realizados durante a implementação.

**Conteúdo**

[1 Especificação 6](#_Toc71838609)

[1.1 Plataforma de desenvolvimento 6](#_Toc71838610)

[1.2 Conjunto de instruções 7](#_Toc71838611)

[1.3 Descrição do Hardware 8](#_Toc71838612)

[1.3.1 ALU ou ULA 8](#_Toc71838613)

[1.3.2 BDRegister 9](#_Toc71838614)

[1.3.3 Clock 9](#_Toc71838615)

[1.3.4 Controle 9](#_Toc71838616)

[1.3.5 Memória de dados 10](#_Toc71838617)

[1.3.6 Memória de Instruções 10](#_Toc71838618)

[1.3.7 Somador 10](#_Toc71838619)

[1.3.8 And 11](#_Toc71838620)

[1.3.9 Mux\_2x1 11](#_Toc71838621)

[1.3.10 PC 11](#_Toc71838622)

[1.3.11 ZERO 11](#_Toc71838623)

[1.4 Datapath 12](#_Toc71838624)

[2 Simulações e Testes 13](#_Toc71838625)

[3 Considerações finais 14](#_Toc71838626)

**Lista de Figuras**

[Figura 1 - Especificações no Quartus 6](#_1fob9te)

[Figura 2 - Bloco simbólico do componente QALU gerado pelo Quartus 8](#_1t3h5sf)

[Figura 19 - Resultado na waveform. 13](#_1ci93xb)

**Lista de Tabelas**

[Tabela 1 – Tabela que mostra a lista de Opcodes utilizadas pelo processador XXXX. 7](#_2et92p0)

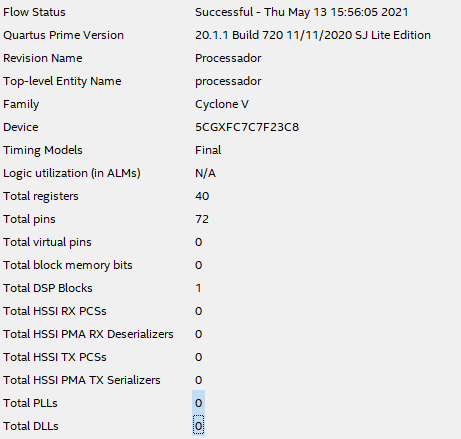
[Tabela 2 - Detalhes das flags de controle do processador. 9](#_3rdcrjn)

[Tabela 3 - Código Fibonacci para o processador Quantum/EXEMPLO. 12](#_2xcytpi)

# 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 no 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:

* **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 |

-**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 |

**-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 |

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

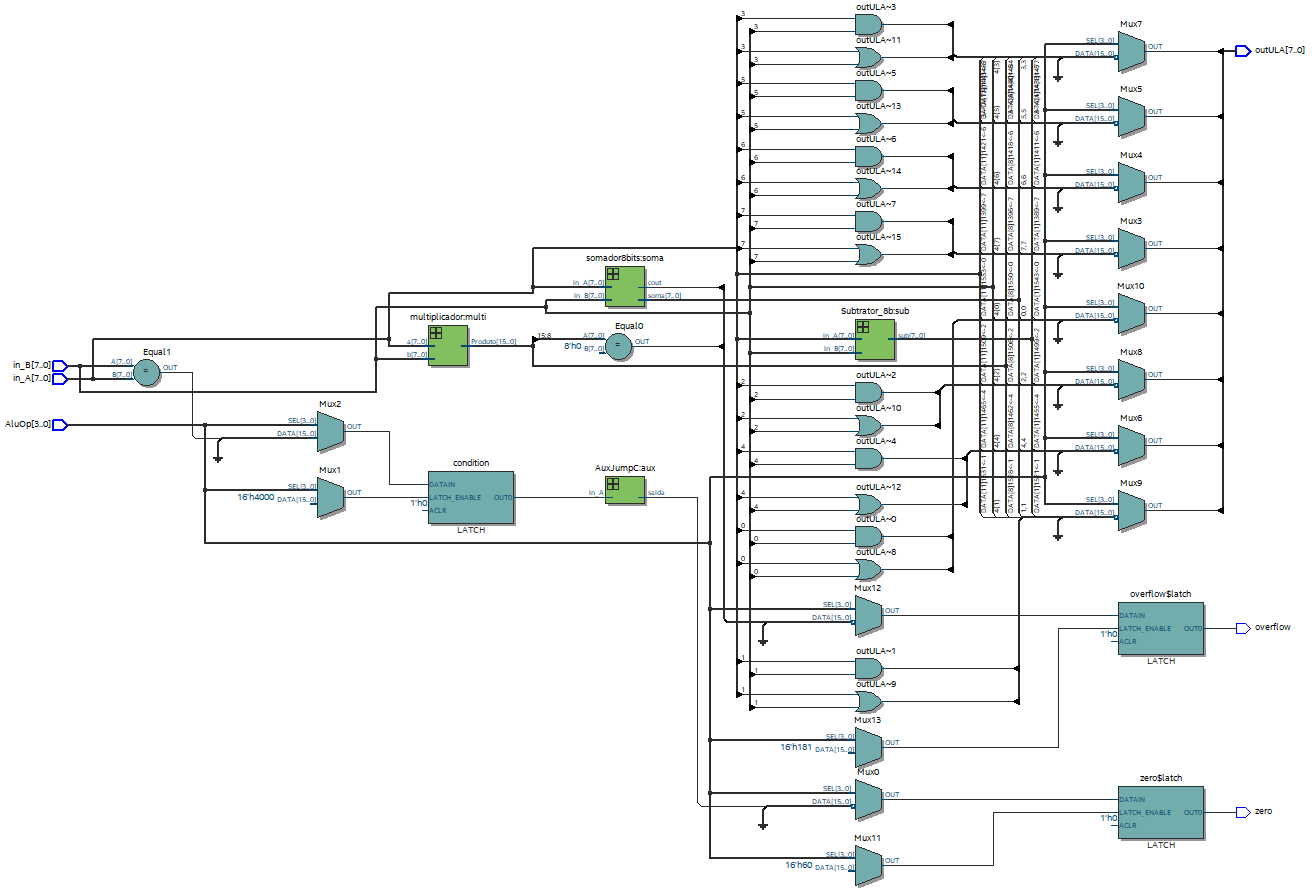
Como o processador XLR-8 é do tipo RISC 8 bits a seleção de bits foi realizada estrategicamente e de forma abranger uma quantidade reduzida porem primordiais para o devido funcionamento do processador.

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

**Tabela 1 – Tabela que mostra a lista de Opcodes utilizadas pelo processador XRL-8.**

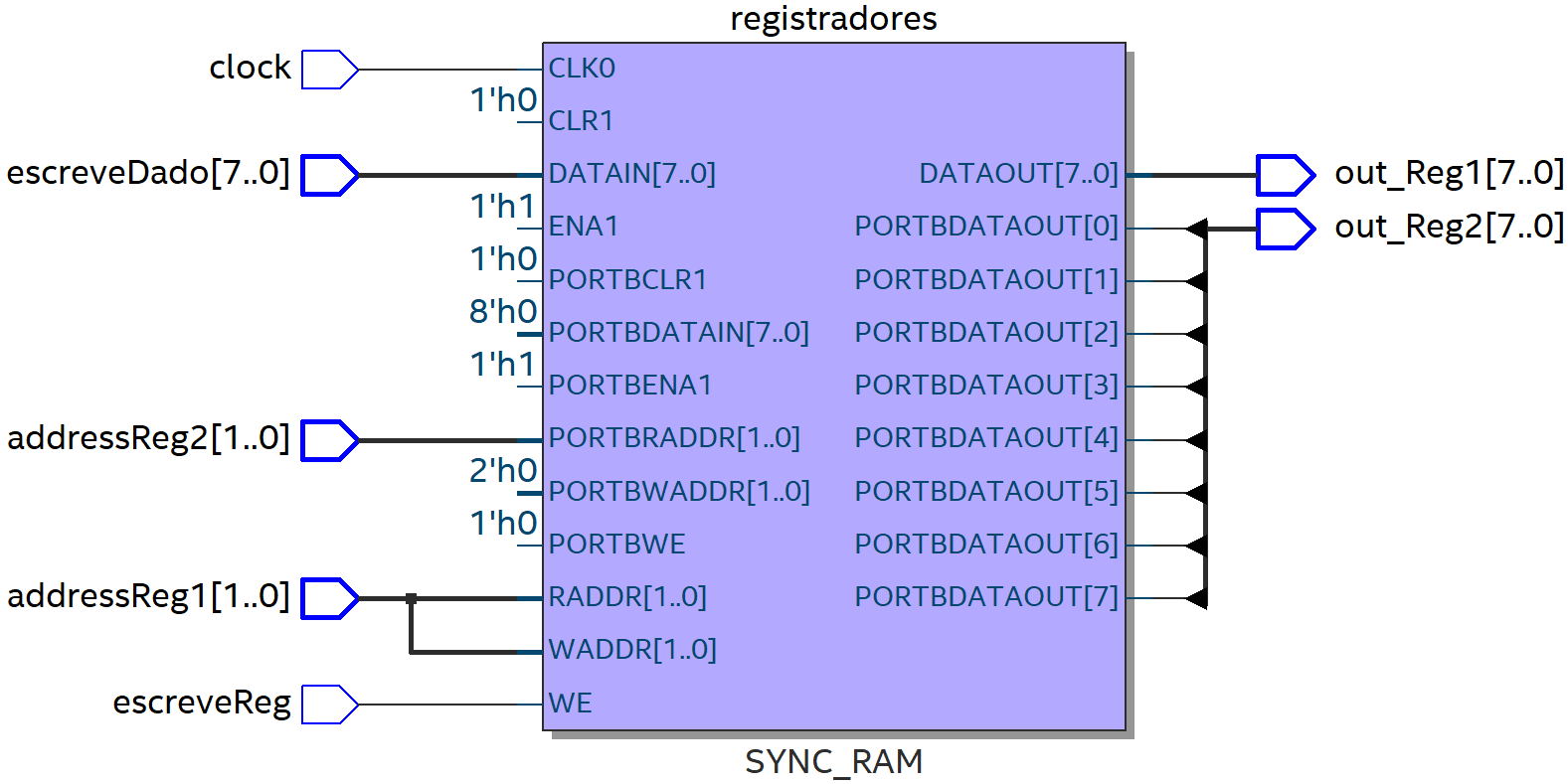
## Descrição do Hardware

### ALU ou ULA



**Figura 2 - Bloco simbólico do componente QALU gerado pelo Quartus**

### BDRegister



### Clock

**É o número de ciclos por segundo de um sinal de sincronismo usado no processador**

### Controle

* **OrigAlu**: XXXX.
* **EscreveReg**: XXXX.
* **EscreveMem**: XXXX.
* **AluOp**: XXXX.
* **MemToReg**: XXXX.
* **LerMem**: XXXX.
* **Branch:** XXXX.
* **Jump**: XXXX.

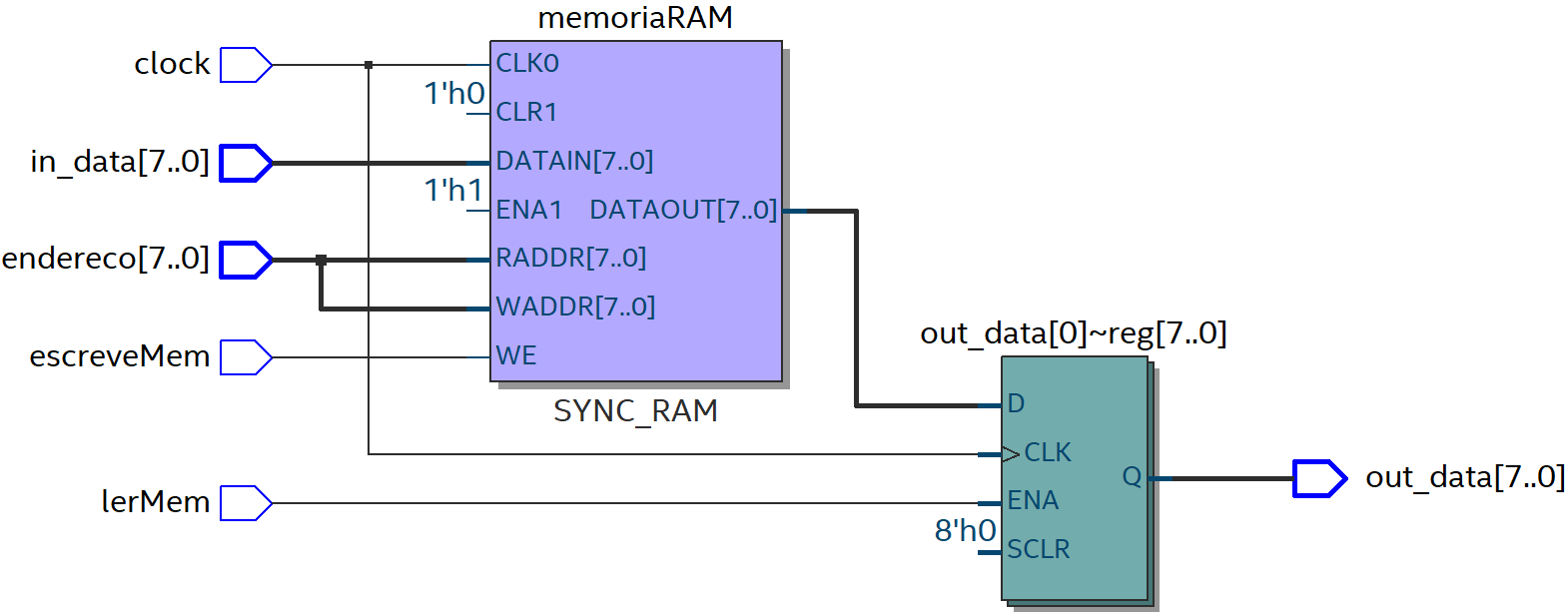
Abaixo segue a tabela, onde é feita a associação entre os opcodes e as flags de controle:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | 1 | 1 | 0 | 0010 | 0 | 1 | 0 | 0 |
| Sw | 1 | 0 | 1 | 0011 | 1 | 0 | 0 | 0 |
| J | 0 | 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 2 - Detalhes das flags de controle do processador.**

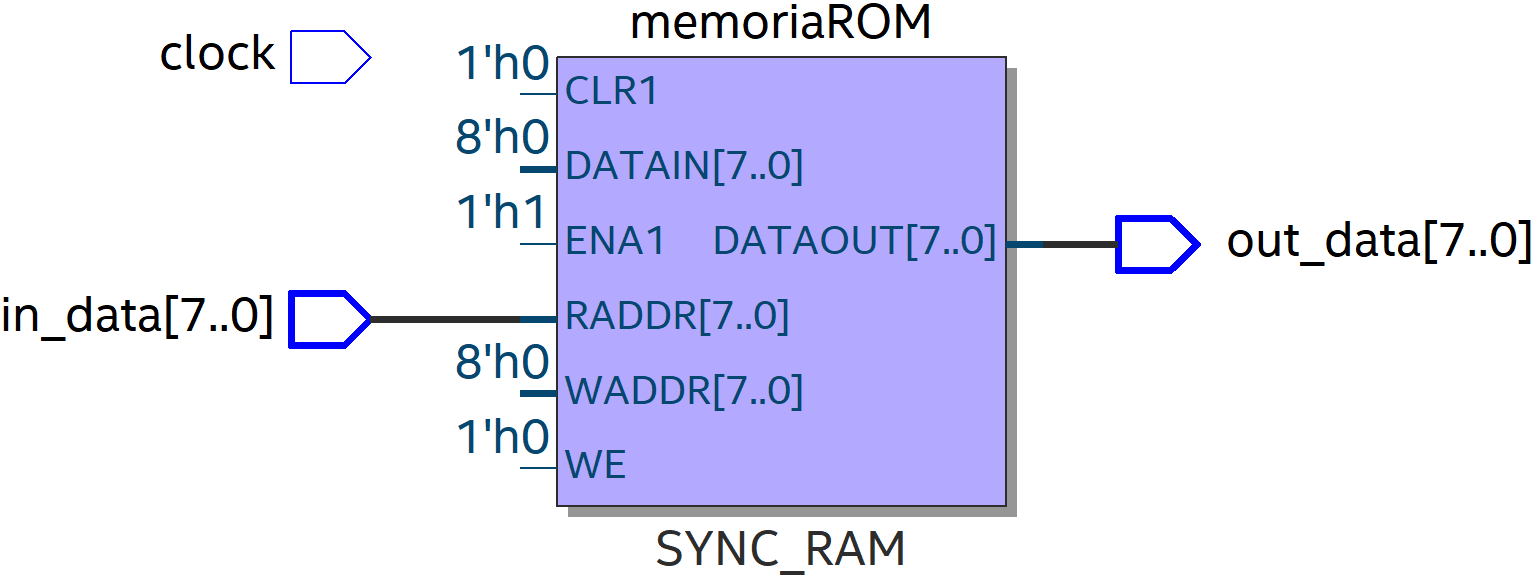
### Memória de dados

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



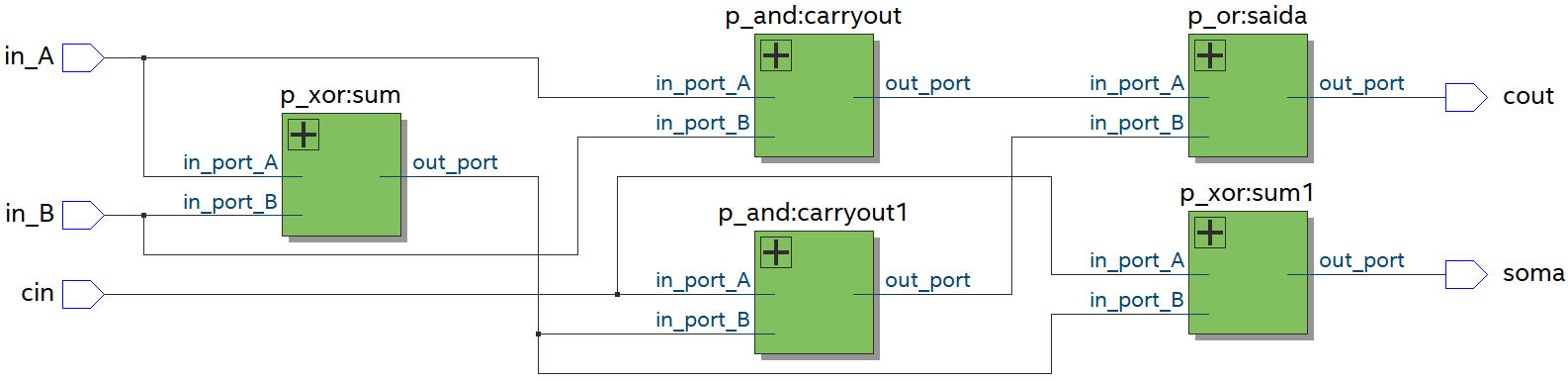
### Memória de Instruções

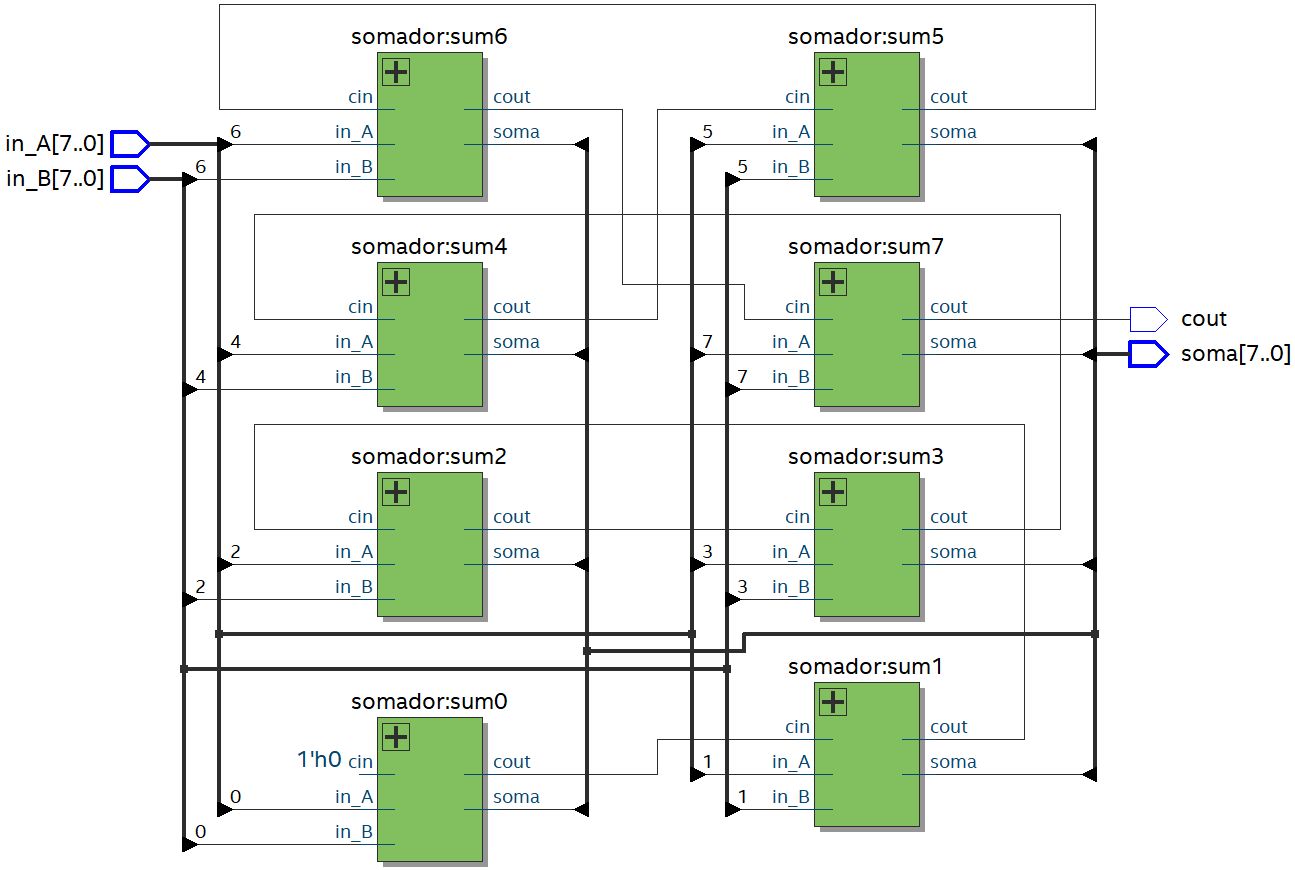
Guarda as instruções a serem executadas no processador.



### Somador

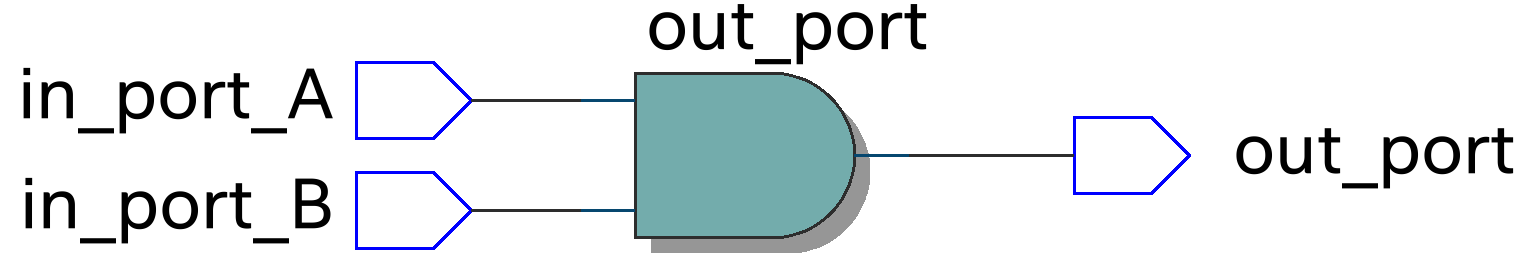
Ele realiza a função de soma no processador, ele faz a soma bit a bit e depois mostra o resultado.



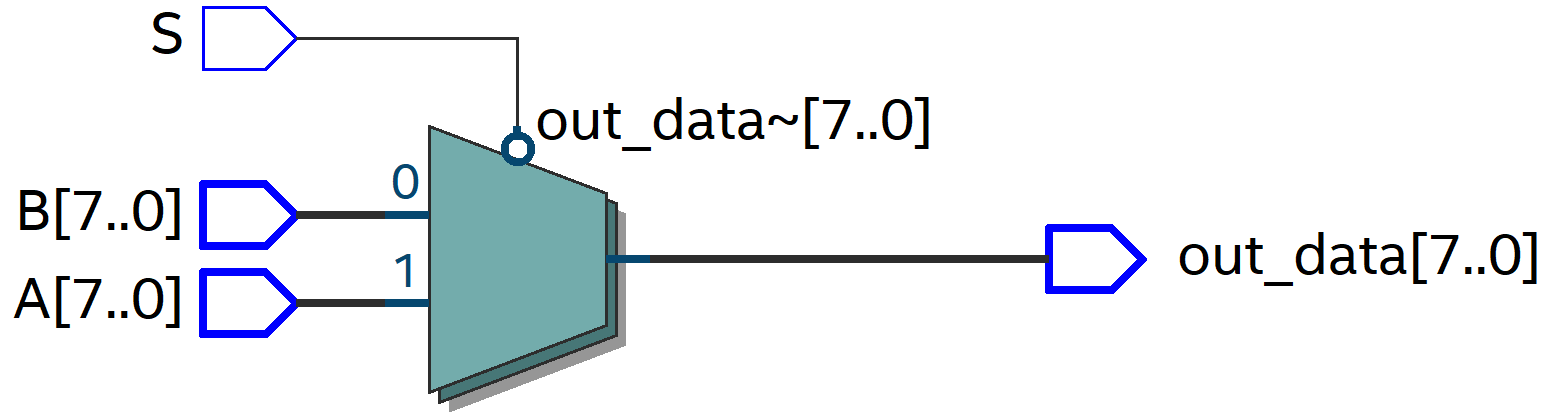


### And

É uma operação lógica que ocorre a entrada de dois operandos que resulta em apenas um valor lógico verdadeiro, mas isso só ocorre se todos os operandos forem verdadeiros.

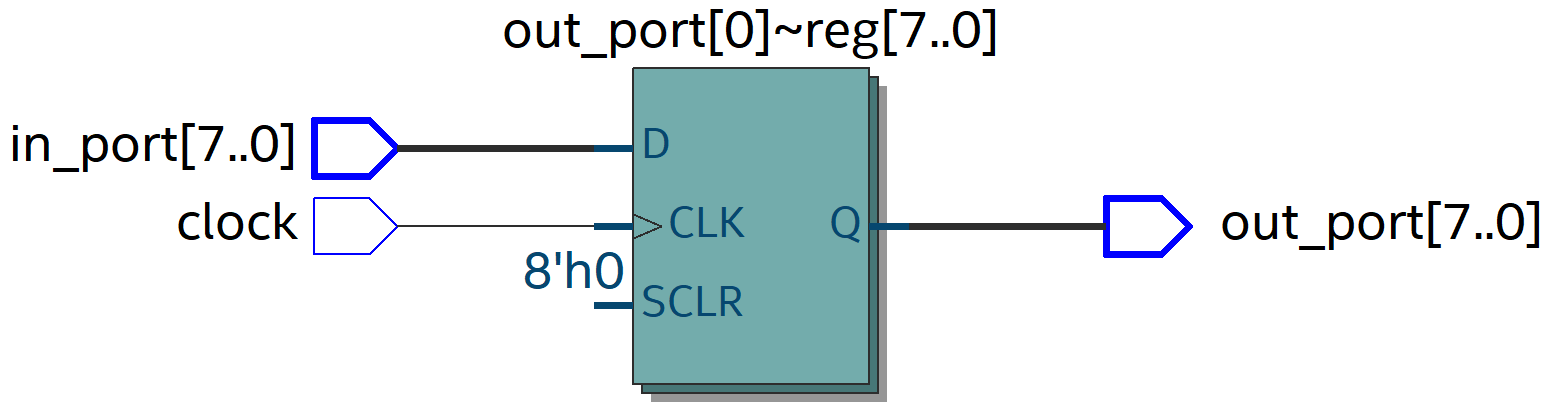


### Mux\_2x1



### PC

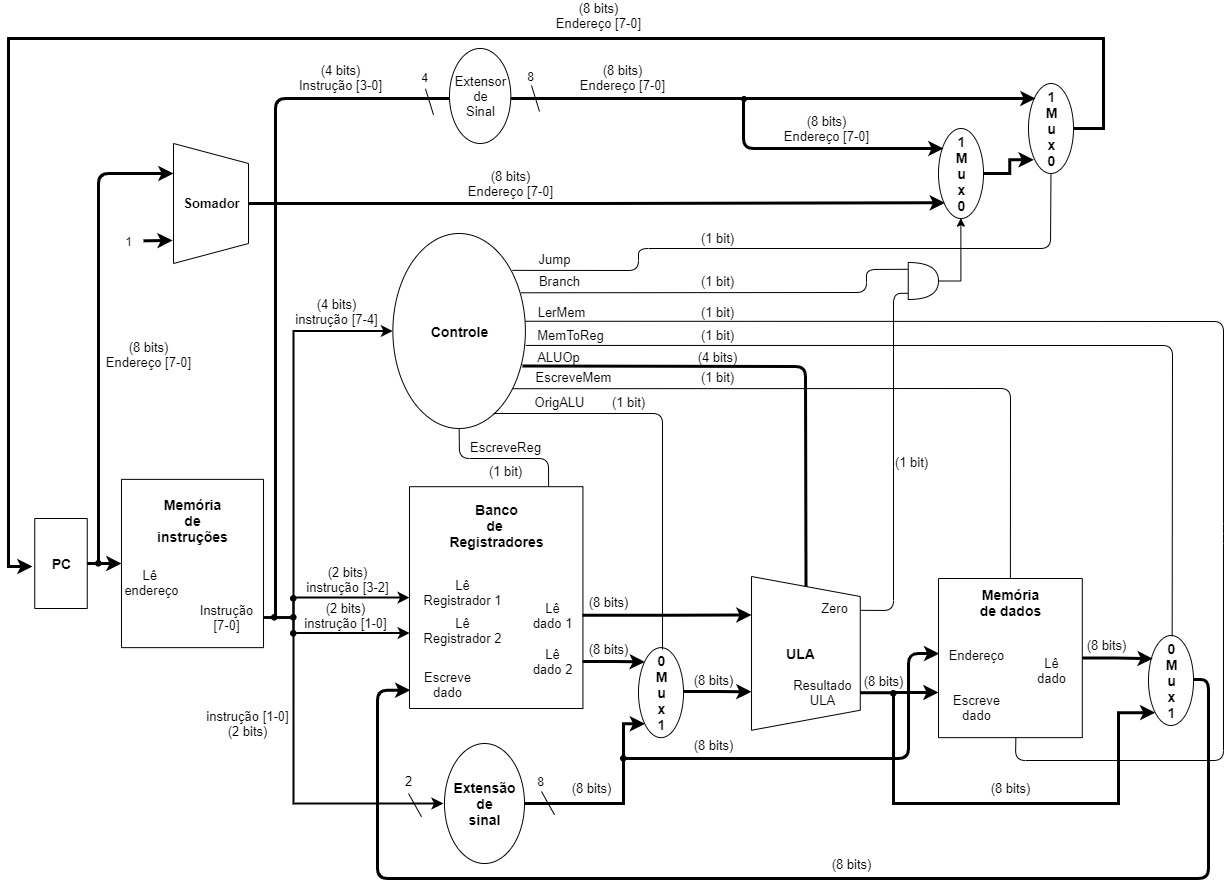
Indexa as instruções a serem executadas no processador, e envia a trilha de bits correspondente às instruções individuais para as próximas unidades funcionais.



### 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.

## Datapath



# Simulações e Testes

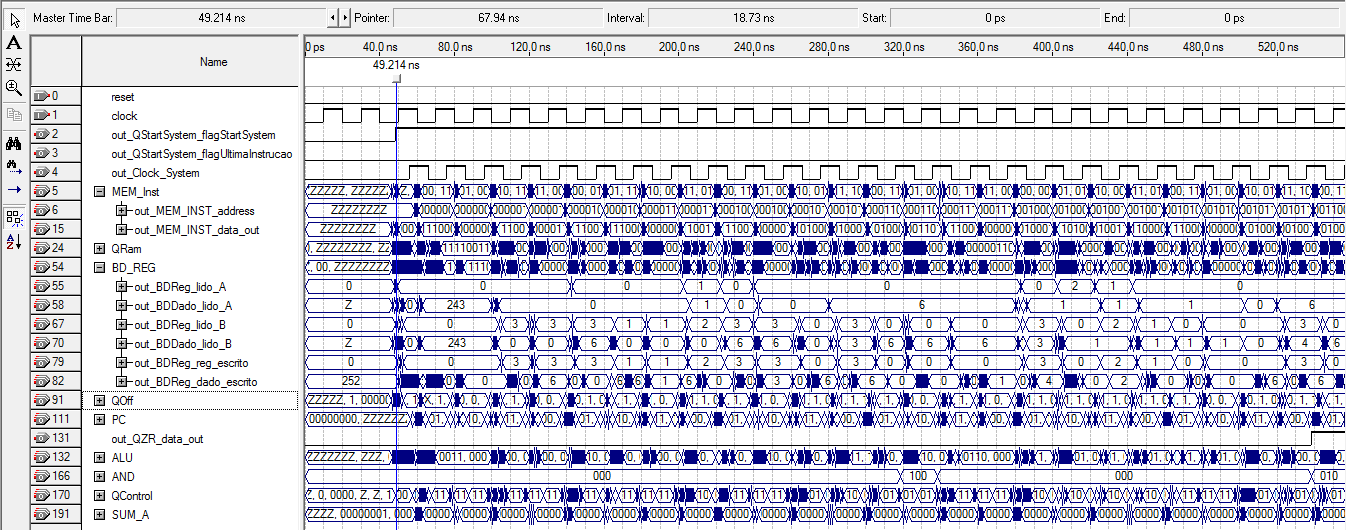
Objetivando analisar e verificar o funcionamento do processador, efetuamos alguns testes analisando cada componente do processador em especifico, em seguida efetuamos testes de cada instrução que o processador implementa. Para demonstrar o funcionamento do processador XXXX utilizaremos como exemplo o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

**Tabela 3 - Código Fibonacci para o processador Quantum/EXEMPLO.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Endereço** | **Linguagem de Alto Nível** | **Binário** | | |
| Opcode | Reg2 | Reg1 |
| Endereço | |
| Dado | | |
| 0 | **LI** $S0, 0 | 1111 | 00 | 00 |
| 1 | 00000000 | | |
| 2 | **LI** $S3, 6 | 1111 | 00 | 11 |
| 3 | 00000110 | | |
| 4 | **SW** $S3, $S0 | 0111 | 00 | 11 |
| 5 | **LI** $S1, 1 | 1111 | 00 | 01 |
| 6 | 00000001 | | |
| 7 | **LRT** $S2, $S1 | 0110 | 01 | 10 |
| 8 | **LI** $S3, 3 | 1111 | 00 | 11 |
| 9 | 00000011 | | |
| 10 | **LW** $S0, $S0 | 0101 | 00 | 00 |
| 11 | **CMPG** $S3,$S0 | 1010 | 00 | 11 |
| 12 | **JMP fim** | 1101 | 0000 | |
| 13 | 00011010 | | |
| 14 | **loop\_fib:** **LI** $S0, 1 | 1111 | 00 | 00 |
| 15 | 00000001 | | |
| 16 | **ADD** $S3, $S0 | 0010 | 00 | 11 |
| 17 | **LRT** $S0, $S2 | 0110 | 10 | 00 |
| 18 | **ADD** $S2, $S1 | 0010 | 01 | 10 |
| 19 | **LRT** $S1, $S0 | 0110 | 00 | 01 |
| 20 | **LI** $S0, 0 | 1111 | 00 | 00 |
| 21 | 00000000 | | |
| 22 | **LW** $S0, $S0 | 0101 | 00 | 00 |
| 23 | **CMPLE** $S3,$S0 | 1001 | 00 | 11 |
| 24 | **JMP loop\_fib** | 1101 | 0000 | |
| 25 | 00001110 | | |
| 26 | **Fim: DEBUG** $S2, $S2 | 0001 | 10 | 10 |

**[Todo] Descrição dos testes**

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



Neste ponto o processador inicia a execução das instruções, são

esperados dois ciclos de clock para que o sistema estabilize.

Estes são os pinos de saída para observação dos resultados, entre eles nós podemos citar: PC,

Memória de Instruções, ULA, Controladora e assim por diante.

**Figura 3 - Resultado na waveform.**

# Considerações finais

Este trabalho apresentou o projeto e implementação do processador de 8 bits denominado de XXXX....