

**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**

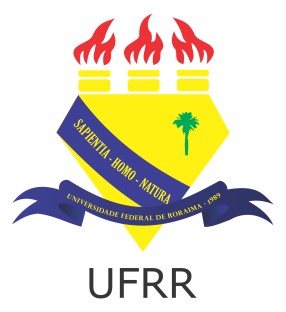
**ALUNOS:**

**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 7](#_gjdgxs)

[1.1 Plataforma de desenvolvimento 7](#_30j0zll)

[1.2 Conjunto de instruções 8](#_3znysh7)

[1.3 Descrição do Hardware 9](#_tyjcwt)

[1.3.1 ALU ou ULA 9](#_3dy6vkm)

[1.3.2 BDRegister 9](#_4d34og8)

[1.3.3 Clock 9](#_2s8eyo1)

[1.3.4 Controle 9](#_17dp8vu)

[1.3.5 Memória de dados 10](#_26in1rg)

[1.3.6 Memória de Instruções 10](#_35nkun2)

[1.3.7 Somador 10](#_1ksv4uv)

[1.3.8 And 10](#_44sinio)

[1.3.9 Mux\_2x1 10](#_2jxsxqh)

[1.3.10 PC 10](#_z337ya)

[1.3.11 ZERO 11](#_3j2qqm3)

[1.4 Datapath 11](#_1y810tw)

[2 Simulações e Testes 13](#_4i7ojhp)

[3 Considerações finais 14](#_3whwml4)

**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)

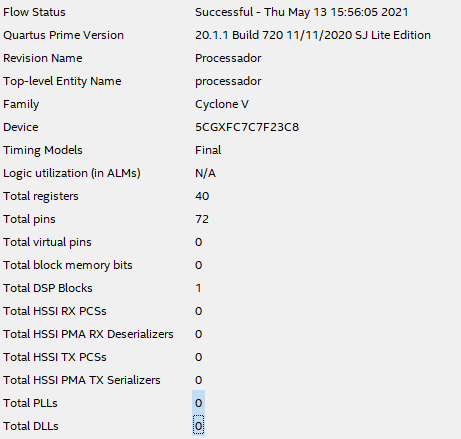
[Tabela 2 - Detalhes das flags de controle do processador. 9](#_3rdcrjn)

[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 do tipo J:**

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

O número de bits do campo **Opcode** das instruções é igual a quatro, sendo assim obtemos um total () de XX **Opcodes (0-XX)** que são distribuídos entre as instruções, assim como é apresentado na Tabela 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Opcode** | **Nome** | **Formato** | **Breve Descrição** | **Exemplo** |
| 1111 | LI | I | Load Immediately | **li** $S0, 31 |
| 0010 | ADD | R | Soma | **add** $S0, $S1 ,ou seja, $S0 := $S0+$S1 |
| 0011 | SUB | R | Subtração | **sub** $S0, $S1 ,ou seja, $S0 := $S0 - $S1 |
| 0100 | DIV\_INT | R | Divisão | **div** $S0, $S1 ,ou seja, $S0 := $S0 / $S1 |

**Tabela 1 – Tabela que mostra a lista de Opcodes utilizadas pelo processador XXXX.**

## Descrição do Hardware

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

### ALU ou ULA

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

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

### BDRegister

**[Todo] Descrição**

### Clock

**[Todo] Descrição**

### Controle

O componente Control tem como objetivo realizar o controle de todos os componentes do processador de acordo com o opcode ... Esse controle é feito através das flags de saída abaixo:

* **DvC**: XXXX.
* **en\_data**: XXXX.
* **EscMem**: XXXX.
* **MemParaReg**: XXXX.
* **UlaOp**: XXXX.
* **LwSwOp**: XXXX.
* **EscReg:** XXXX.
* **Wrt\_LRT**: XXXX.
* **FlagPC**: XXXX.

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

**Tabela 2 - Detalhes das flags de controle do processador.**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Comando | DvC | off\_instruction | en\_data | Esc  Mem | Mem  ParaReg | UlaOp | LwSw  Op | EscReg | Wrt\_LRT | FlagPc | enJmp |
| add | 0 | 1 | 1 | 0 | 1 | 0000 | 0 | 1 | 0 | 1 | 0 |
| sub | 0 | 1 | 1 | 0 | 1 | 0001 | 0 | 1 | 0 | 1 | 0 |
| div | 0 | 1 | 1 | 0 | 1 | 0111 | 0 | 1 | 0 | 1 | 0 |
| Inicialização | 0 | 1 | 1 | Z | Z | ZZZZ | Z | Z | Z | 1 | 0 |

### Memória de dados

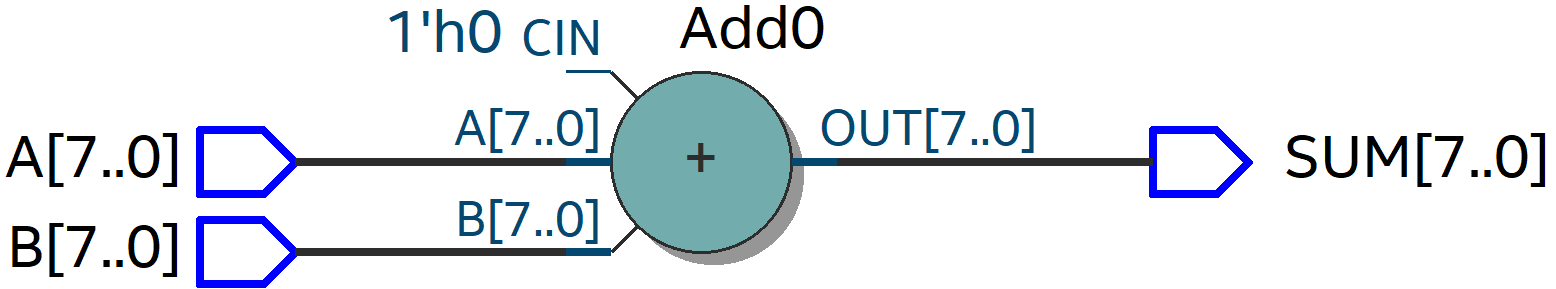
Componente Funcional utilizado para armazenar dados gerais através do endereçamento. Este componente pode armazenar até 2^(número de bits do processador) - 1 dados, valor equivalente a 255 espaços de 8 bits de armazenamento

### Memória de Instruções

Guarda as instruções a serem executadas no processador.

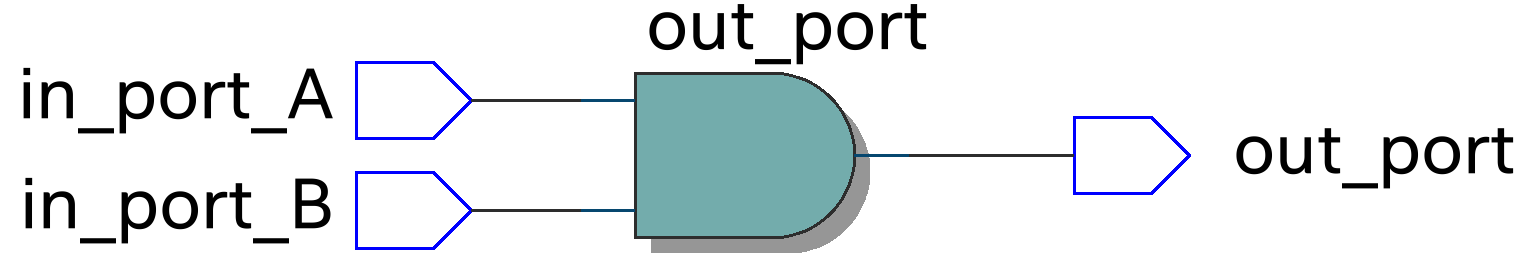
### Somador

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



### And

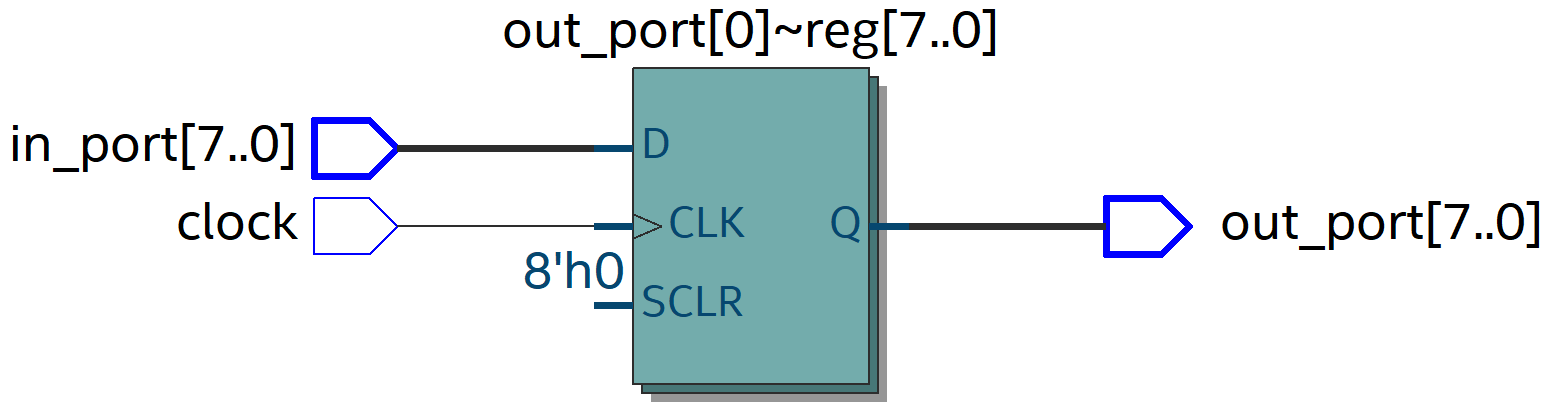
É uma operação lógica que ocorre a entrada de dois operandos que resulta em apenas um valor lógico verdadeiro,mas isso só ocorre se todos os operandos forem verdadeiros.



### Mux\_2x1

### PC

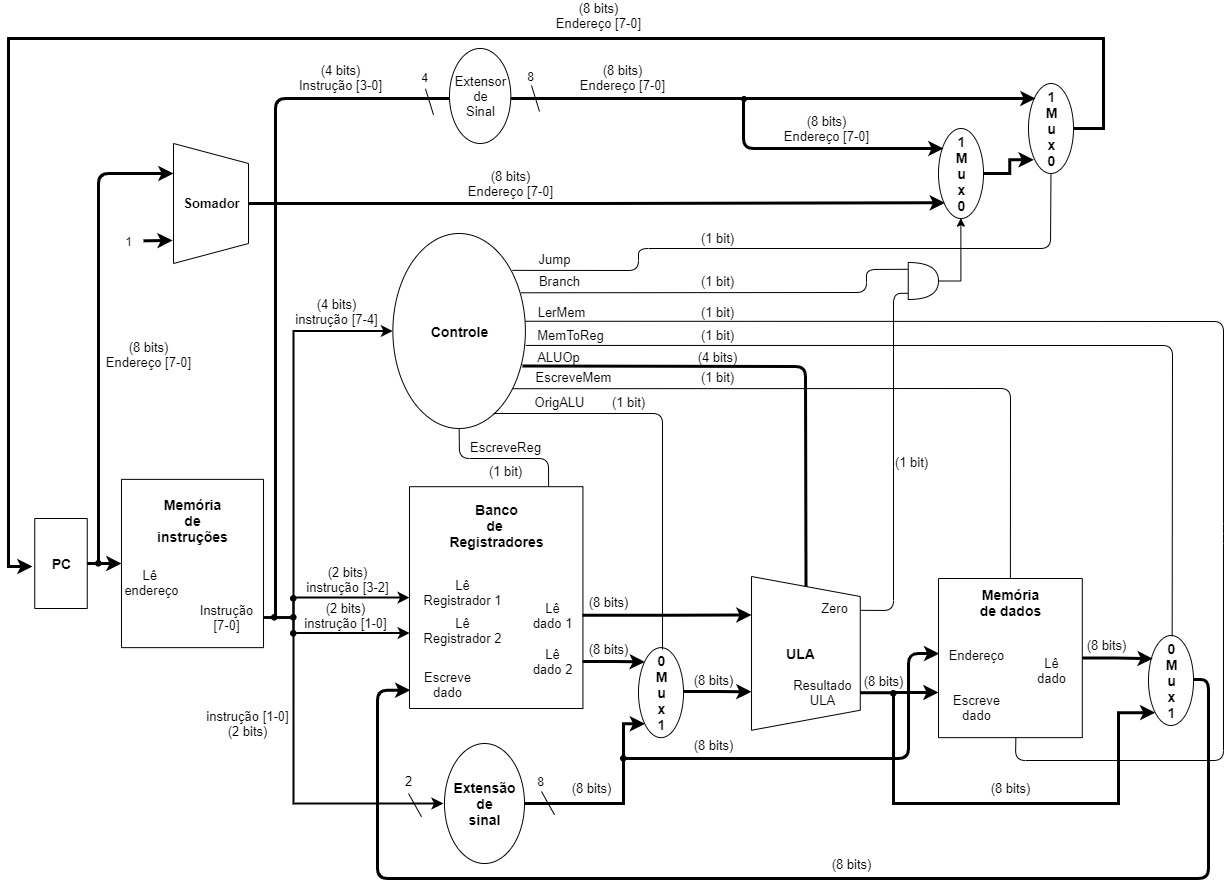
Indexa as instruções a serem executadas no processador, e envia a trilha de bits correspondente às instruções individuais para as próximas unidades funcionais.



### 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

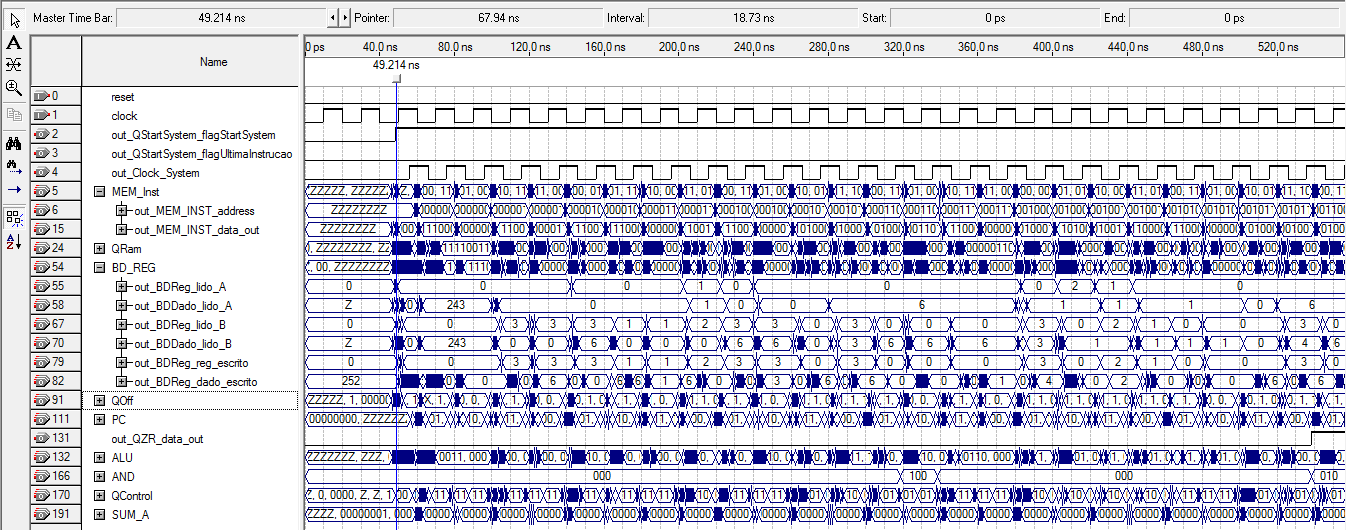
Objetivando analisar e verificar o funcionamento do processador, efetuamos alguns testes analisando cada componente do processador em especifico, em seguida efetuamos testes de cada instrução que o processador implementa. Para demonstrar o funcionamento do processador XXXX utilizaremos como exemplo o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