

**PODER EXECUTIVO**

**MINISTÉRIO DA EDUCAÇÃO**

**UNIVERSIDADE FEDERAL DE RORAIMA**

**DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO**

**ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES**

**RELATÓRIO DO PROJETO:**

**PROCESSADOR KARR**

**ALUNOS:**

**Angelo Almeida Ferro – 1201524424**

**Kaio Guilherme Ferraz De Sousa Silva - 2020014670**

**Março** **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 KARR**

**Março** **de 2022**

**Boa Vista/Roraima**

**Resumo**

Este trabalho aborda o projeto e implementação de um processador de 8 bits baseado na arquitetura mips uniciclo. Nele haverá um total de 6 instruções devido a limitação de 8bits por instrução e 4 registradores, pois ao tentar fazer contas mais complexas do que essas pode haver problemas de overflow ou overheat.

**Conteúdo**

[1 Especificação 7](#_Toc444681789)

[1.1 Plataforma de desenvolvimento 7](#_Toc444681790)

[1.2 Conjunto de instruções 8](#_Toc444681791)

[1.3 Descrição do Hardware 9](#_Toc444681792)

[1.3.1 ALU ou ULA 9](#_Toc444681793)

[1.3.2 BDRegister 9](#_Toc444681794)

[1.3.3 Clock 9](#_Toc444681795)

[1.3.4 Controle 9](#_Toc444681796)

[1.3.5 Memória de dados 10](#_Toc444681797)

[1.3.6 Memória de Instruções 10](#_Toc444681798)

[1.3.7 Somador 10](#_Toc444681799)

[1.3.8 And 10](#_Toc444681800)

[1.3.9 Mux\_2x1 10](#_Toc444681801)

[1.3.10 PC 10](#_Toc444681802)

[1.3.11 ZERO 11](#_Toc444681803)

[1.4 Datapath 11](#_Toc444681804)

[2 Simulações e Testes 13](#_Toc444681805)

[3 Considerações finais 14](#_Toc444681806)

**Lista de Figuras**

[Figura 1 - Especificações no Quartus 6](#_Toc444681815)

[Figura 2 - Bloco simbólico do componente QALU gerado pelo Quartus 8](#_Toc444681816)

[Figura 19 - Resultado na waveform. 13](#_Toc444681817)

**Lista de Tabelas**

[Tabela 1 – Tabela que mostra a lista de Opcodes utilizadas pelo processador XXXX. 7](#_Toc444681822)

[Tabela 2 - Detalhes das flags de controle do processador. 9](#_Toc444681823)

[Tabela 3 - Código Fibonacci para o processador Quantum/EXEMPLO. 12](#_Toc444681824)

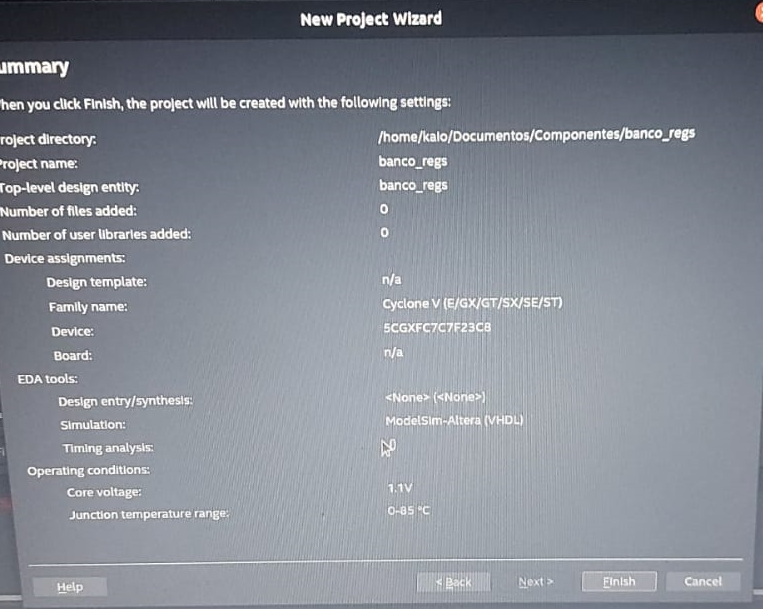
# Especificação

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

## Plataforma de desenvolvimento

Para a implementação do processador KARR foi utilizado a IDE: Intel Quartus Lite edition 20.1 como mostra o print das especificações a baixo:

Print Parecido(falta checar):



Exemplo de print:



Figura 1 - Especificações no Quartus

## Conjunto de instruções

O processador KARR possui 4 registradores: Reg1, Reg2, Reg3 e Reg4. Assim como 02 formatos de instruções de 8 bits cada, Instruções do **tipo R** e do **tipo** **I**, seguem algumas considerações sobre as estruturas contidas nas instruções:

* **Opcode**: a operação básica a ser executada pelo processador, tradicionalmente chamado de código de operação;
* **Reg1**: O registrador contendo o primeiro operando fonte e adicionalmente para alguns tipos de instruções (ex. instruções do tipo R) é o registrador de destino;
* **Reg2**: O registrador contendo o segundo operando fonte;
* **Func**: Sub operação do Opcode

Tipo de Instruções:

**- Formato do tipo R:** Este formatado aborda instruções de Load (exceto *load Immediately*), Store e instruções baseadas em operações aritméticas.

**-Formato do tipo I: Este formatado aborda instruções de modo imediato.**

Formato para escrita do processador:

|  |  |  |  |
| --- | --- | --- | --- |
| Tipo da Instrução | Reg1 | Reg2 | Func |

Formato para escrita em código binário:

|  |  |  |  |
| --- | --- | --- | --- |
| 2 bits | 2 bits | 2 bits | 2 bits |
| 7-6 | 5-4 | 3-2 | 1-0 |
| Opcode | Reg2 | Reg1 | Func |

**Visão geral das instruções do Processador KARR:**

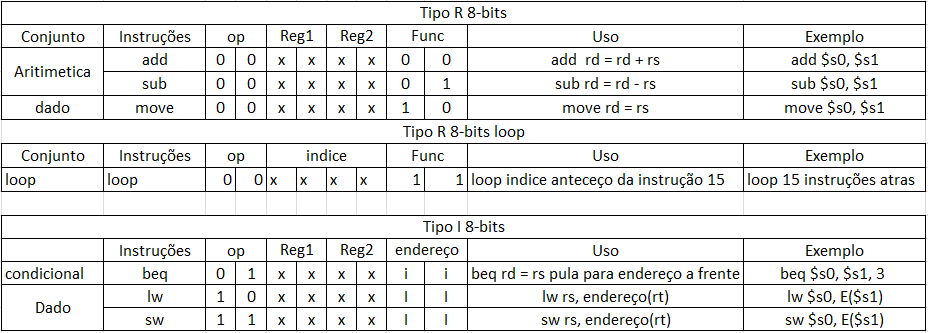
O número de bits do campo **Opcode** das instruções é igual a dois, sendo assim obtemos um total () de 04 **Opcodes (0-3)** que para funções de Opcode 00 existem 4 subfunções destinadas pelo campo Func de 0 a 3 são distribuídos entre as instruções, assim como é apresentado na Tabela 1.

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

## Descrição do Hardware

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

### ALU ou ULA

O componente ULA (Unidade Logica Aritmética) tem como principal objetivo efetuar as principais operações aritméticas, dentre elas: Soma e Subtração (utilizando apenas valores inteiros). Também possui operação de comparação de valor como igualdade onde se for igual executa a ação e se for diferente será direcionado pro zero. A ULA possui três valores de entrada que são: **A -** dado de 8 bits para operação; **B -** dado de 8 bits para operação; **OP -** Identificador da operação que será realizada 2bits. Também possui dois valores de saída que são: **zero -** identificador de resultado de seleção onde se for 0 soma, se for 1 subtrai e se for 2 redireciona para o clock; **Result –** Saída com o resultado das operações aritméticas.

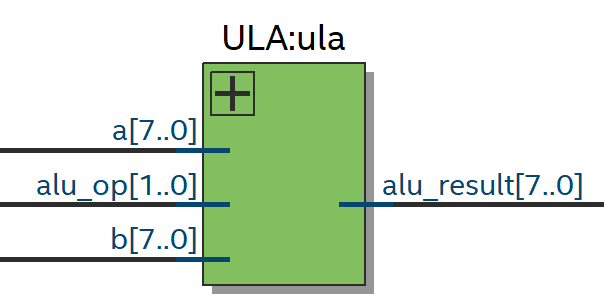
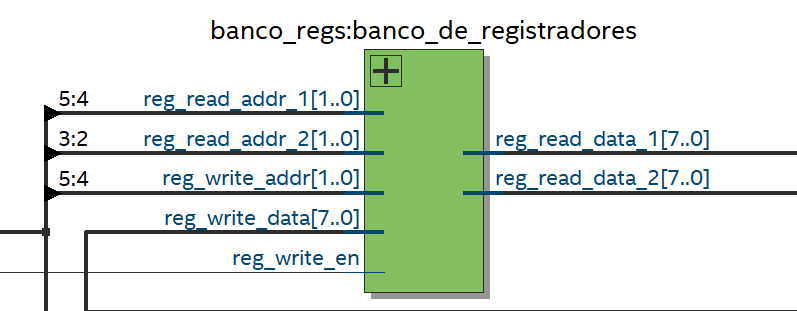


Figura 2 - Bloco simbólico do componente QALU gerado pelo Quartus

### BDRegister

O componente BDRegister (Banco de Registradores ) tem como principal objetivo “escrever” e “ler” dados que podem ou não ser armazenados momentaneamente durante o clock. Esse componente possui 5 valores de entrada que são: **Reg\_Write\_En –** Dado que define se a função vai ser de leitura ou escrita; **Reg\_Write\_Addr –** Endereço do registrador onde o dado será escrito; **Reg\_Write\_Data –** valor do dado que será escrito no endereço selecionado; **Reg\_Read\_Addr1 –** local do endereço de acesso desejado do Registrador; **Reg\_Read\_Addr2 –** local do endereço de acesso desejado do Registrador. Também possui 2 valores de saída que são: **Reg\_Read\_Data1 –** Leitura do dado armazenado no registrador; **Reg\_Read\_Data1 –** Leitura do dado armazenado no registrador.

### Clock

O componente Clock tem como objetivo realizar o controle do andamento do programa do processador, sendo responsável pelo tempo de execução.

### Controle

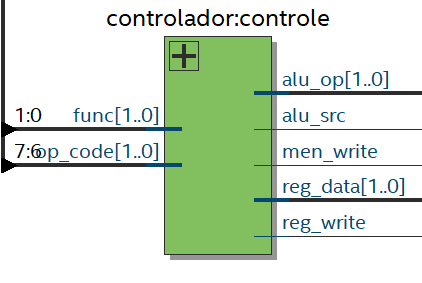
O componente Control(Controle) tem como objetivo realizar o controle de todos os componentes do processador de acordo com o OP Code(2 bits) e Func(2 bits). Esse controle é feito através das flags de saída abaixo:

* **Reg Data**: 2 bits.
* **Reg Write**: 1 bit.
* **ALU Src**: 1 bit.
* **ALU Op**: 2 bits.
* **MEM Write**: 1 bit.
* **Loop Func**: 2 bits.

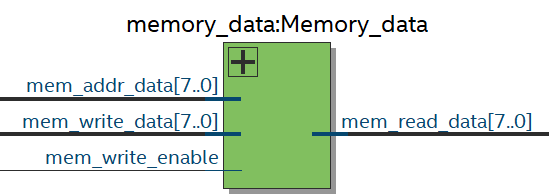
Abaixo segue a tabela, onde é feita a associação entre os opcodes e as flags de controle:

Tabela 2 - Detalhes das flags de controle do processador.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Controlador | | | | | | | | | | | | | |
| Instruções | Op Code | | Func | | Reg Data | | Reg Write | ALU src | ALU op | | MEMwrite | loop func | |
| add | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| sub | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| move | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| loop | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| beq | 0 | 1 | x | x | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| lw | 1 | 0 | x | x | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| sw | 1 | 1 | x | x | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |

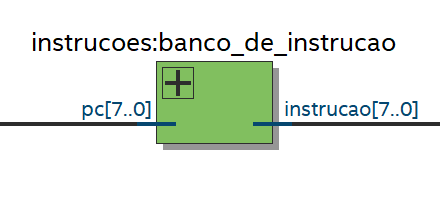


### Memória de dados

O componente Memory Data (Memoria de Dados ) tem como principal objetivo “escrever” e “ler” dados que podem ou não ser armazenados momentaneamente durante o clock. Esse componente possui 3 valores de entradas que são: **Mem\_Write\_Enable –** Determina a função realizada onde 0 “ler” e 1 “escreve”; **Mem\_Addr\_Data –** Endereço de memoria onde o dado será “Escrito” ou “Lido”; **Mem\_Write\_Data –** Dado que sera alocado no endereço de memoria selecionado. Também possui 1 valor de saída que é: **Mem\_Read\_Data –** Dado que será “Lido”.

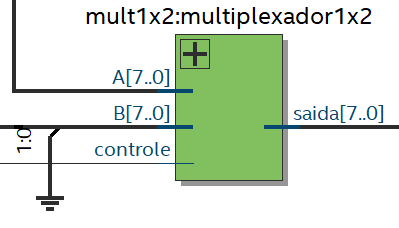
### Memória de Instruções

O componente Memoria de Instruções tem como principal objetivo receber o indice que define qual operação o processador executara no momento sendo 7 instruções no total as quais são: **Add –** Soma de valores **;Sub –** Subtração de valores **;Move –** Encaminha o dado de um Reg para outro e reescrever se necessário**; Loop –** Retorna quantas “casas” foram selecionadas no array **; Beq –** Comparação **;Lw –** “Leitura” de dados **;Sw –** Aramazena dados.

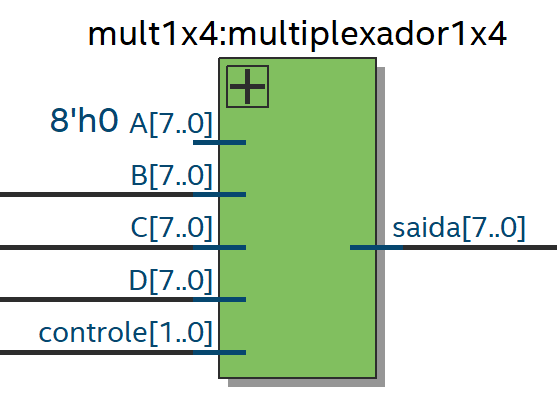


### Mux\_1x2

O componente Mux\_1x2 (Multiplexador 2 pra 1) tem como principal objetivo receber vários dados ao mesmo tempo e através de um controle decidir qual desses dados recebidos será utilizado. Possui 3 valores de entrada que são: **A –** Dado de 8 bits para operação **;B –** Dado de 8 bits para operação **; Controle –** Dado de 2 bits que define qual das outras entradas será utilizada. Também possui 1 valor de saída: **Saida –** dado de 8 bits selecionado.

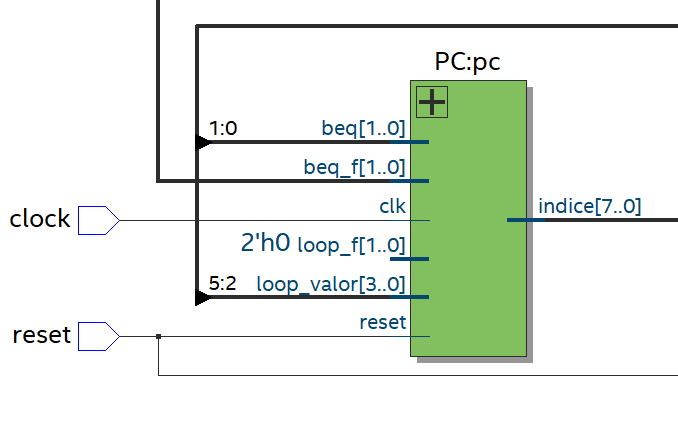


### Mux\_1x4

O componente Mux\_1x4 (Multiplexador 4 pra 1) tem como principal objetivo receber vários dados ao mesmo tempo e através de um controle decidir qual desses dados recebidos será utilizado. Possui 5 valores de entrada que são: **A –** Dado de 8 bits para operação **;B –** Dado de 8 bits para operação **;C –** Dado de 8 bits para operação **;D -** Dado de 8 bits para operação **;Controle –** Dado de 2 bits que define qual das outras entradas será utilizada. Também possui 1 valor de saída: **Saida –** dado de 8 bits selecionado.

### PC

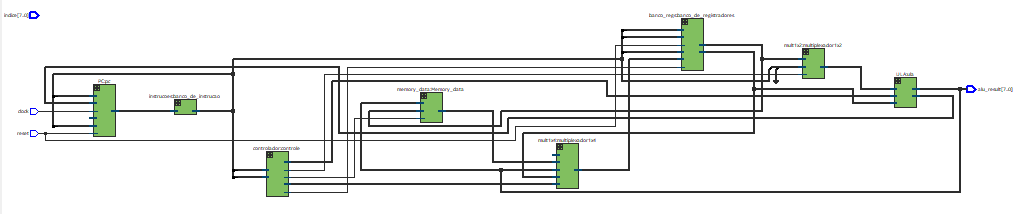
O componente PC tem como principal objetivo de enviar o índice para o banco de instruções definindo o que será executado e enquanto isso também controla as funções de beq e loop onde se alternam para executar o programa sem problemas.



## Datapath

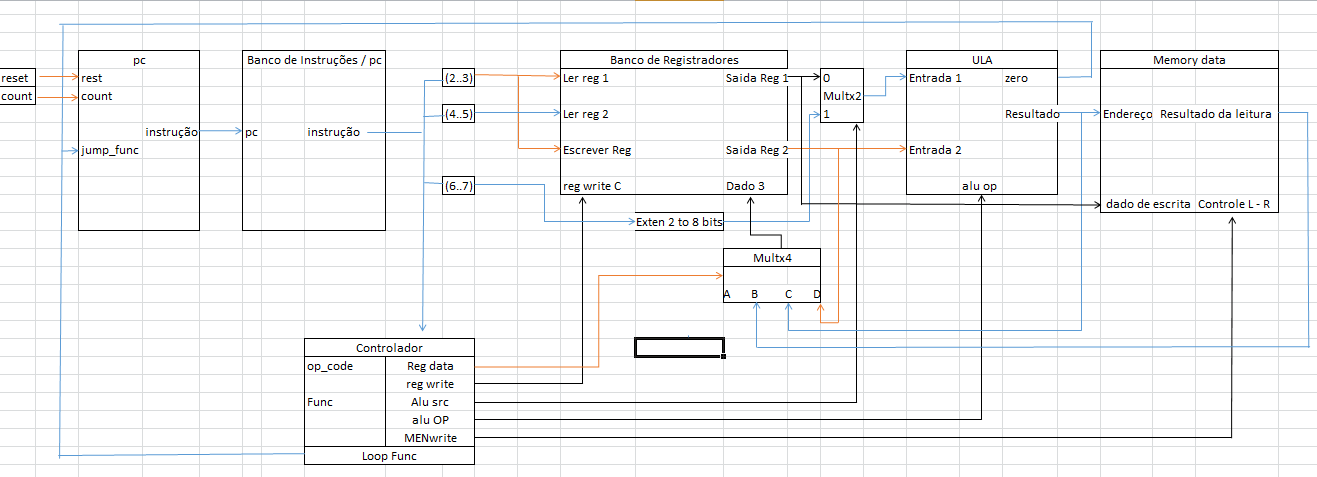
É a ligação entre os componentes que executam de forma única os dados e adiciona uma forma de controle que organiza as ações que serão executadas por classes diferentes de instrução.

**[Todo] Figura RTL**



**Versão RTL:**

**Versão Rascunho:**



# 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.



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.

Neste ponto o processador inicia a execução das instruções, são

esperados dois ciclos de clock para que o sistema estabilize.

Figura 3 - Resultado na waveform.

# Considerações finais

ISSO AQ E UMA POHA !!