

**PODER EXECUTIVO**

**MINISTÉRIO DA EDUCAÇÃO**

**UNIVERSIDADE FEDERAL DE RORAIMA**

**DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO**

**ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES**

**RELATÓRIO DO PROJETO: PROCESSADOR CHUMUSUKE**

**ALUNOS:**

**JOSHUA KOOK HO PEREIRA – 2017009284**

**LUIS HENRIQUE MATOS SALES - 2017020657**

**Dezembro de 2018**

**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 CHUMUSUKE**

**Dezembro de 2018**

**Boa Vista/Roraima**

**Resumo**

Este projeto aborda a elaboração e implementação de um processador uniciclo de 16 bits, baseado na arquitetura do processador MIPS Uniciclo.

**Conteúdo**

[1 Especificação 7](#_Toc531605264)

[1.1 Plataforma de desenvolvimento 7](#_Toc531605265)

[1.2 Conjunto de instruções 8](#_Toc531605266)

[1.3 Descrição do Hardware 9](#_Toc531605267)

[1.3.1 ALU ou ULA 9](#_Toc531605268)

[1.3.2 Banco de Registradores 10](#_Toc531605269)

[1.3.3 PC/Program Counter 11](#_Toc531605270)

[1.3.4 Unidade de Controle 11](#_Toc531605271)

[1.3.5 Memória de dados 12](#_Toc531605272)

[1.3.6 Memória de Instruções 13](#_Toc531605273)

[1.3.7 Somador 13](#_Toc531605274)

[1.3.8 And 14](#_Toc531605275)

[1.3.9 Multiplexadores 14](#_Toc531605276)

[1.3.10 DataSplit 15](#_Toc531605277)

[1.3.11 Extensor de Sinal 16](#_Toc531605278)

[1.3.12 Shift 16](#_Toc531605279)

[1.4 Datapath 17](#_Toc531605280)

[2 Simulações e Testes 19](#_Toc531605281)

[3 Considerações finais 20](#_Toc531605282)

**Lista de Figuras**

Figura 1 - Especificações no Quartus 7

Figura 2 - Bloco simbólico do componente ULA gerado pelo Quartus 10

Figura 3 – Bloco Simbólico do componente BancoRegistradores gerado pelo Quartus 10

Figura 4 - Bloco Simbólico do componente PC gerado pelo Quartus 11

Figura 5 - Bloco Simbólico do componente ControlUnit gerado pelo Quartus 12

Figura 6 - Bloco Simbólico do componente MemoriaRAM gerado pelo Quartus 13

Figura 7- Bloco simbólico do componente ROMMem gerado pelo Quartus 13

Figura 8 - Bloco simbólico do componente BAdder gerado pelo Quartus 14

Figura 9- Bloco simbólico do componente SomadorPC gerado pelo Quartus 14

Figura 10 - Bloco simbólico do componente ANDGate gerado pelo Quartus 14

Figura 11 - Bloco simbólico do componente Mulitplexador2x1 gerado pelo Quartus 15

Figura 12 - Bloco simbólico do componente Mulitplexador2x1\_16bits gerado pelo Quartus 15

Figura 13 - Bloco simbólico do componente DataSplit gerado pelo Quartus 16

Figura 15 - Bloco simbólico do componente DataSplit gerado pelo Quartus 16

Figura 16 - Resultado na waveform. 20

**Lista de Tabelas**

Tabela 1 – Possíveis operações do processador Chumusuke e seus respectivos OPCodes. 9

Tabela 2 - Associação entre os OPCodes e as flags de controle. 12

Tabela 3 - Código Fibonacci para o processador Quantum/EXEMPLO. 19

# Especificação

A seguir serão apresentados os itens utilizados no desenvolvimento do processador Chumusuke, assim como uma descrição detalhada de cada etapa da construção do processador.

## Plataforma de desenvolvimento

Para a implementação do processador Chumusuke foi utilizado a IDE: Quartus Prime Lite Edition, versão 18.0.0.

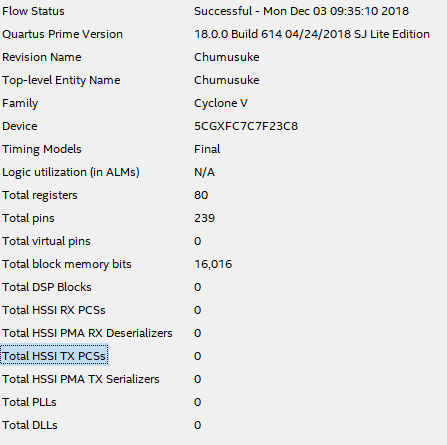


Figura 1 - Especificações no Quartus

## Conjunto de instruções

O processador Chumusuke possui 16 (dezesseis) registradores: S0, ... e SN. Assim como 3 (três) formatos de instruções de 16 bits cada: **Instruções do tipo R, Instruções do tipo I** e **Instruções do tipo J.** Seguem algumas considerações respectivas às estruturas contidas nas instruções:

* **OPCode**: a operação básica a ser executada pelo processador;
* **RegA**: endereço do registrador contendo o primeiro operando fonte;
* **RegB**: endereço do registrador contendo o segundo operando fonte, também possuí a função de representar o registrador de destino para Instruções do tipo I;
* **RegDst**: registrador de destino para operações do tipo R e adicionalmente também representa uma constante ou endereço de memória para instruções do tipo I.

Vale lembrar que para as instruções do tipo J os três registradores (RegA, RegB e RegDst) representam o endereço para onde o processo será desviado.

Tipo de Instruções:

**Formato das instruções:**

1. **Tipo R:** aborda instruções baseadas em operações aritméticas e lógicas: Add, Sub e Multiplicação;
2. **Tipo I:** aborda instruções baseadas em operações com valores imediatos, desvios condicionais ou operações relacionadas à memória: Add Imediato, Sub Imediato, BEQ, BNE, Load e Store;
3. **Tipo J:** aborda instruções de desvios incondicionais: Jump;

Formato para escrita de código na linguagem Chumusuke:

* **Operações do Tipo R:**

|  |  |  |  |
| --- | --- | --- | --- |
| Operação | RegA | RegB | RegDst |

* **Operações do Tipo I:**

|  |  |  |  |
| --- | --- | --- | --- |
| Operação | RegA | RegDst | Constante |

* **Operações do tipo J:**

|  |  |
| --- | --- |
| Operação | Endereço |

Formato para escrita em código binário:

|  |  |  |  |
| --- | --- | --- | --- |
| 4 bits | 2 bits | 2 bits | 4 bits |
| 15-12 | 11-8 | 7-4 | 3-0 |
| OPCode | RegA | RegB | RegDst |

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

A quantidade de bits do campo **OPCode** das instruções é igual a quatro, dessa maneira podemos ter até um total de 16 OPCodes () **OPCodes (0-15)**, divididos entre as possíveis operações do processador. A tabela a seguir demostra as possíveis operações e seus respectivos OPCodes:

Tabela 1 – Possíveis operações do processador Chumusuke e seus respectivos OPCodes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OPCode** | **Nome** | **Formato** | **Breve Descrição** | **Exemplo** |
| 0000 | ADD | R | Soma | **add** $R0, $R1, $R2 ( $R2 := $R0 + $R1 ) |
| 0001 | SUB | R | Subtração | **sub** $R0, $R1, $R2 ( $R2 := $R0 - $R1 ) |
| 0010 | MUL | R | Multiplicação | **mul** $R0, $R1, $R2 ( $R2 := $R0 \* $R1 ) |
| 0011 | BEQ | I | Desvio Condicional | **beq** $R0, $R1, Endereço |
| 1011 | BNE | I | Desvio Condicional | **bne** $R0, $R1, Endereço |
| 0100 | LOAD | I | Load | **lw** $R0, $R1, Endereço |
| 0101 | STORE | I | Store | **sw** $R0, $R1, Endereço |
| 0110 | ADDI | I | Soma Imediata | **addi** $R0, $R1, Const ( $R1 := $R0 + Const ) |
| 0111 | SUBI | I | Subtração Imediata | **subi** $R0, $R1, Const ( $R1 := $R0 - Const ) |
| 1000 | AND | R | AND (bit a bit) | **and** $R0, $R1, $R2 ($R2 := $R0 and $R1) |
| 1001 | OR | R | OR (bit a biit) | **or** $R0, $R1, $R2 ($R2 := $R0 or $R1) |
| 1111 | JUMP | J | Desvio Incondicional | **jump** Endereço |

## Descrição do Hardware

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

### ALU ou ULA

O componente ULA (Unidade Lógica Aritmética) tem como o objetivo de efetuar as principais operações aritméticas e lógicas (ex.: soma, subtração, multiplicação, and, or), operando apenas com valores inteiros. O componente também efetua operações de comparação de valores iguais. O componente ULA recebe três valores de entrada: **A** – dado de 16 bits para operação; **B** - dado de 16bits para operação e **OPULA** – identificador de três bits que defina a operação a ser realizada. Em relação às saídas, temos: **ULAZero** – flag representando o resultado de comparações (1 caso for verdade, 0 caso contrário); **ULAOut** – resultado da operação realizada pelo componente; e **Prod\_HIGH** – saída auxiliar para operações de multiplicação, representando a metade mais significativa do resultado.

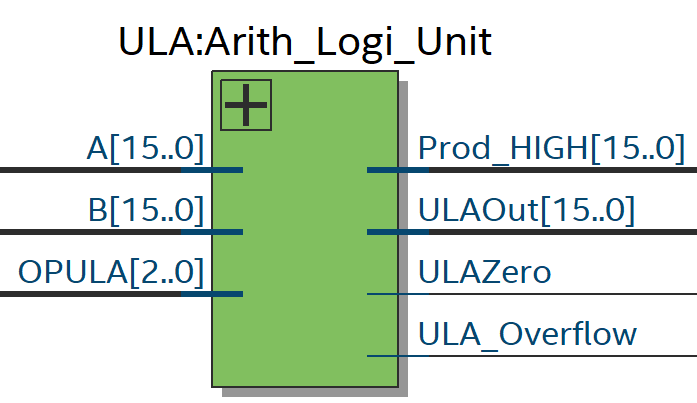


Figura 2 - Bloco simbólico do componente ULA gerado pelo Quartus

### Banco de Registradores

O componente BancoRegistradores tem como papel armazenar e enviar informações a serem utilizadas ao decorrer da execução do programa. Possuí 16 registradores que armazenam até 16 bits de informações. Tem como entrada os seguintes valores: **Clock** – sinal que ativa o componente; **EscReg** – sinal que ativa/desativa a função de escrita de dados no registrador de destino; **Data** – dado a ser armazenado no registrador de destino; **RegDst** – endereço do registrador de destino ao qual o dado de escrita será armazenado; **LeReg1** – endereço do registrador que contém a primeira informação a ser enviada para o componente ULA; **LeReg2** – endereço do registrador que contém a segunda informação a ser enviada para o componente ULA. Como saída temos: **RegA** – conteúdo do registrador especificado pela entrada LeReg1; **RegB** – conteúdo do registrador especificado pela entrada LeReg2.

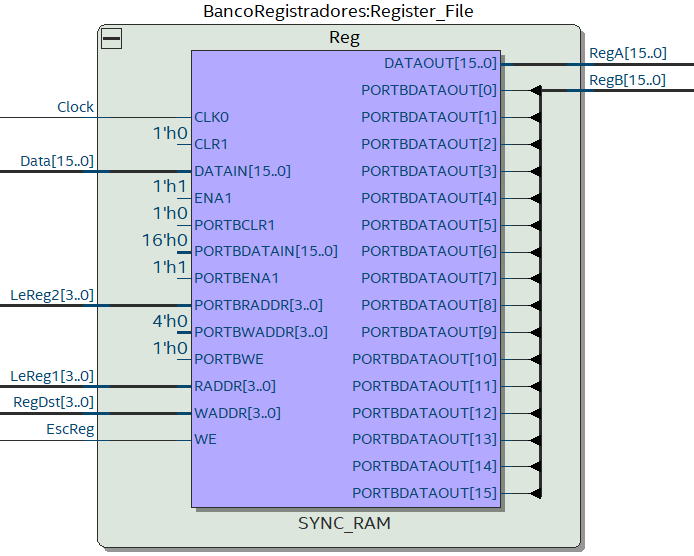


Figura 3 – Bloco Simbólico do componente BancoRegistradores gerado pelo Quartus

### PC/Program Counter

O componente PC tem como função armazenar o endereço de 16 bits da instrução que está sendo executada atualmente. Possuí dois valores de entrada: **clk** – sinal que ativa o componente; **pin** - -endereço da instrução a ser executada; e uma saída: **pout** – endereço da instrução que será executada.

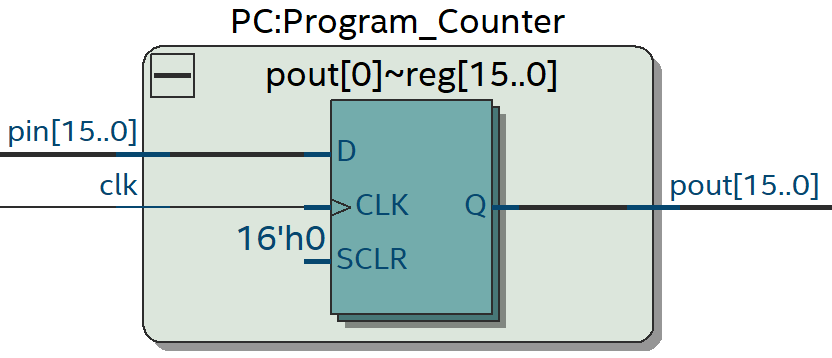


Figura 4 - Bloco Simbólico do componente PC gerado pelo Quartus

### Unidade de Controle

O componente ControlUnit tem como objetivo realizar o controle de todos os componentes do processador baseado na entrada OPCode, através das flags de saída a seguir:

* **Branch**: sinal enviado à um multiplexador para decidir se haverá um desvio condicional ou não.
* **RegDestino**: sinal enviado ao multiplexador que irá decidir qual será o registrador de destino.
* **EscMem**: sinal enviado ao componente MemoriaRAM que definirá se os dados serão ou não armazenados na memória RAM.
* **MemParaReg**: sinal enviado para um multiplexador para decidir qual dado será enviado ao banco de registradores.
* **OPULA**: código que defina a operação a ser executada pelo componente ULA.
* **Jump**: sinal que decide se haverá um desvio incondicional ou não.
* **EscReg:** sinal que decide se o dado de escrita de volta será armazenado ou não.
* **LeMem**: sinal que decide se será retirado algum dado do componente MemoriaRAM.
* **ULAFonte**: sinal enviado para o multiplexador que decide qual será o segundo dado a entrar no componente ULA.

A seguir é apresentado uma tabela onde é feita a associação entre os OPCodes e as flags de controle do componente ControlUnit:

Tabela 2 - Associação entre os OPCodes e as flags de controle.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Comando | Branch | RegDestino | LeMem | Esc  Mem | Mem  ParaReg | OPULA | ULAFonte | EscReg | Jump |
| add | 0 | 1 | 0 | 0 | 0 | 000 | 0 | 1 | 0 |
| sub | 0 | 1 | 0 | 0 | 0 | 001 | 0 | 1 | 0 |
| mul | 0 | 1 | 0 | 0 | 0 | 010 | 0 | 1 | 0 |
| BEQ | 1 | X | 0 | 0 | 0 | 101 | 0 | 0 | 0 |
| BNE | 1 | X | 0 | 0 | 0 | 110 | 0 | 0 | 0 |
| load | 0 | 0 | 1 | 0 | 1 | 000 | 0 | 1 | 0 |
| store | 0 | X | 0 | 1 | 0 | 000 | 0 | 0 | 0 |
| addi | 0 | 0 | 0 | 0 | 0 | 000 | 1 | 1 | 0 |
| subi | 0 | 0 | 0 | 0 | 0 | 001 | 1 | 1 | 0 |
| and | 0 | 1 | 0 | 0 | 0 | 011 | 0 | 1 | 0 |
| or | 0 | 1 | 0 | 0 | 0 | 100 | 0 | 1 | 0 |
| jump | 0 | 0 | 0 | 0 | 0 | 111 | 0 | 0 | 1 |

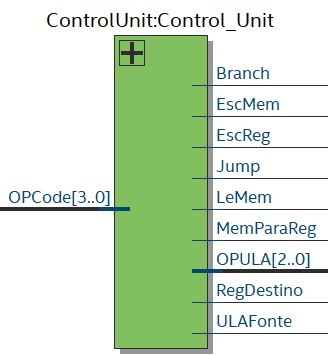


Figura 5 - Bloco Simbólico do componente ControlUnit gerado pelo Quartus

### Memória de dados

O componente MemoriaRAM tem como papel armazenar dados temporariamente em endereços de memória voláteis para serem usados no decorrer da execução do programa. Possuí as entradas: **Clock, LeMem, EscMem**  - sinais utilizados para ativar o componente, decidir se haverá leitura dos dados da memória e decidir se será escrito algum dado na memória, respectivamente; **EscData** – dado a ser escrito no endereço especificado; **Endereco** – endereço da memória onde o dado de escrita será escrito. Tem apenas uma saída: **SaiData**  - dado contido no endereço especificado pela entrada Endereco.

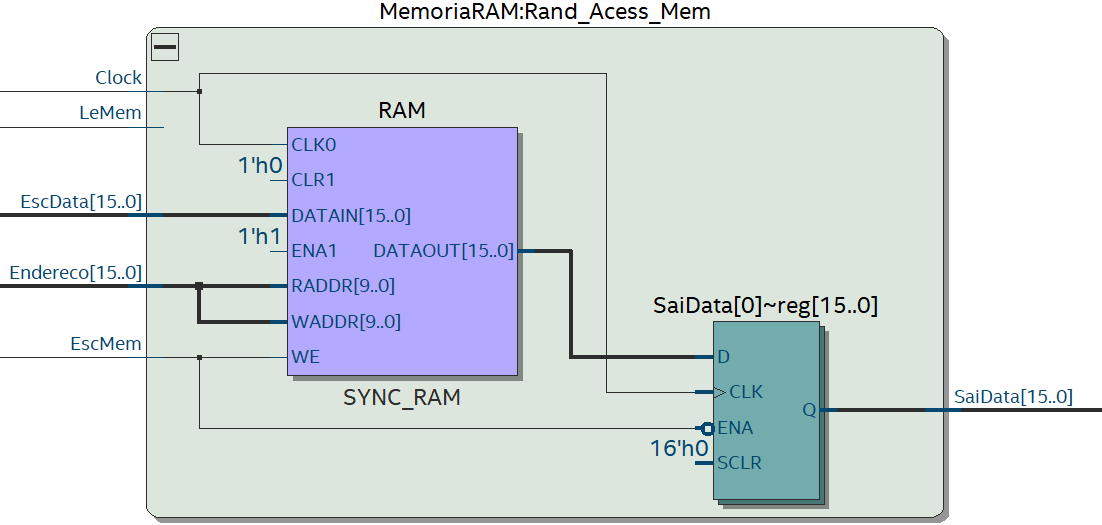


Figura 6 - Bloco Simbólico do componente MemoriaRAM gerado pelo Quartus

### Memória de Instruções

O componente ROMMem tem o papel de armazenar as instruções que serão executadas pelo processador. Possuí apenas uma entrada: **PC\_In –** endereço de 16 bits da instrução à ser enviada para a execução. E apenas uma saída: **ROM\_Out** – instrução de 16 bits a ser executada pelo processador.

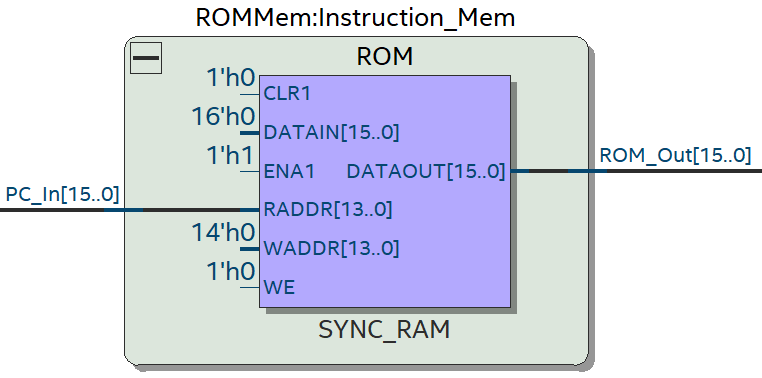


Figura 7- Bloco simbólico do componente ROMMem gerado pelo Quartus

### Somador

O processador possuí dois componentes somadores de 16 bits: SomadorPC e BAdder. O SomadorPC tem a função de somar o valor da instrução atual com o valor 1, onde o resultado será o endereço da próxima instrução. Contém apenas uma entrada de 16 bits: **AddIn** – endereço da instrução atual. Para saídas, também teremos apenas uma: **AddOut** – endereço da próxima instrução. Já o componente BAdder tem o papel de fazer a soma do endereço da próxima instrução com o endereço especificado pelos comandos BEQ e BNE, resultando no endereço de salto da instrução. O componente BAdder tem duas entradas de 16 bits: **A** – Endereço da próxima instrução; **B** - endereço especificado pela instrução BEQ/BNE. E na saída temos: **Result** – endereço da instrução ao qual o processo será desviado.

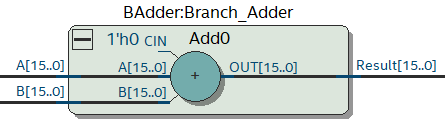


Figura 8 - Bloco simbólico do componente BAdder gerado pelo Quartus

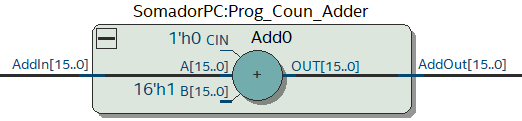


Figura 9- Bloco simbólico do componente SomadorPC gerado pelo Quartus

### And

O componente ANDGate tem o papel de decidir a flag que define se haverá um desvio condicional ou não. As entradas são: **A** – sinal vindo do componente ControlUnit; **B –** sinal vindo do componente ULA. Já a saída será: **S** – sinal que vai para o multiplexador que decide qual será o próximo endereço de instrução.

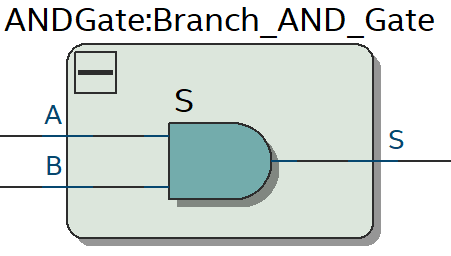


Figura 10 - Bloco simbólico do componente ANDGate gerado pelo Quartus

### Multiplexadores

O processador possuí um total de 4 (quatro) componentes Multiplexador2x1\_16bits e 1 (um) Multiplexador2x1. Os quatro primeiros mencionados são multiplexadores com entrada e saída de 16 bits, utilizados para as seguintes decisões: qual será o dado de escrita de volta que irá para o componente BancoRegistradores; qual será o segundo dado de entrada do componente ULA; se haverá desvio condicional; e por fim se haverá desvio incondicional. Já o componente Multiplexador2x1 tem como função decidir qual será o registrador de destino. Começando pelos componentes Multiplexador2x1\_16bits, temos como entrada: **A, B** – entradas de 16 bits vindos de diferentes componentes (dependendo da instrução); **S –** sinal seletor que decidirá a saída. Na saída temos: **SAIDA** – saída de 16 bits que será enviada ao respectivo componente ao qual o multiplexador está ligado.

Similarmente, o Multiplexador2x1 irá decidir qual o endereço do registrador de destino da operação. Entradas: **A, B** – entradas de 4 bits que representam o endereço do registrador de destino; **S** – sinal seletor vindo do componente ControlUnit que decidirá qual será o endereço do registrador de destino. Saída: **SAIDA** – endereço do registrador de destino.

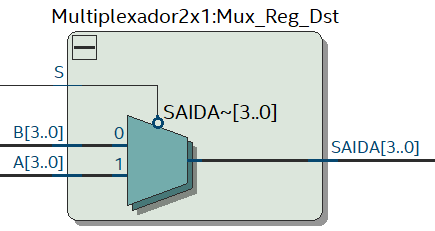


Figura 11 - Bloco simbólico do componente Mulitplexador2x1 gerado pelo Quartus

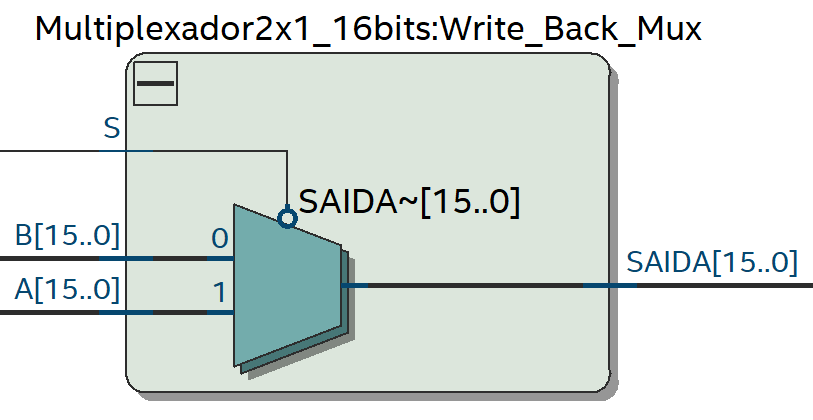


Figura 12 - Bloco simbólico do componente Mulitplexador2x1\_16bits gerado pelo Quartus

### DataSplit

O componente DataSplit tem como única função a divisão dos bits da instrução que saem do componente ROMMem. Possuí uma única entrada: **DataSplit\_In** – dado de 16 bits que vem do componente ROMMem que será divido. Em relação às saídas teremos quatro: **Jump** – saída de 12 bits que representa o endereço de salto para a instrução jump; **OP\_Code** – saída de 4 bits que representa a operação que o processador irá executar; **rd** – saída de 4 bits que pode representar tanto um endereço do registrador de destino (instruções tipo R) quanto uma constante/endereço de instrução (tipo I); **rs** – saída de 4 bits endereço do primeiro registrador da operação; **rt** - saída de 4 bits que pode representar ou o endereço do segundo registrador da operação (instruções tipo R) ou o registrador de destino (instruções tipo I).

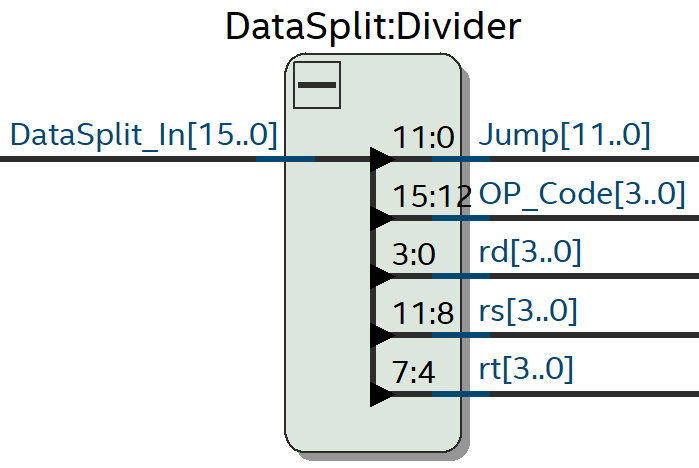


Figura 13 - Bloco simbólico do componente DataSplit gerado pelo Quartus

### Extensor de Sinal

O componente ExtensordeSinal4To16bits tem a função de aumentar o número de bits do sinal de entrada para 16 bits, de forma a torna-lo apropriado para cálculos nos outros componentes (Ex.: ULA). Possuí uma entrada: **ENTRADA** – sinal de 4 bits que representam um valor constante. Saída: **SAIDA** – sinal de entrada, porém no formato de 16 bits.

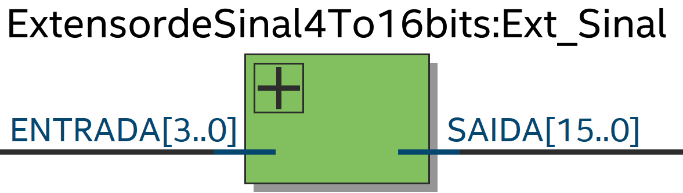


Figura 14 - Bloco simbólico do componente DataSplit gerado pelo Quartus

### Shift

O componente JumpShift serve para realizar o shift de dois bits para a esquerda do endereço de entrada que representa o endereço especificado pela instrução jump, para o cálculo do endereço de destino. Adicionalmente, ele também concatena o resultado da soma com os 2 bits mais significativos do endereço da próxima instrução. Possuí duas entradas: **JumpAddress** – endereço especificado na instrução jump; **PC** – endereço da próxima instrução. A saída será: **Saida** – endereço de destino da operação.

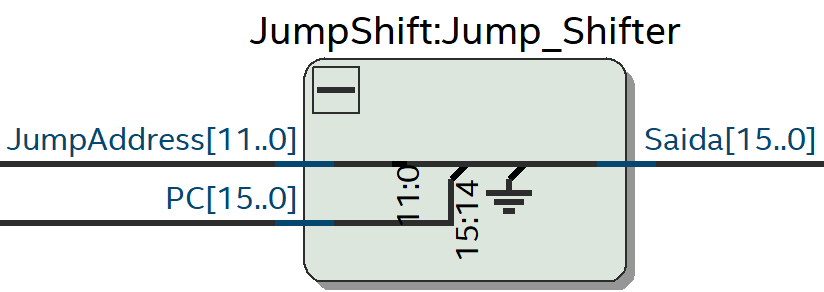


Figura 15 - Bloco simbólico do componente DataSplit gerado pelo Quartus

## Datapath

É a conexão entre as unidades funcionais formando um único caminho de dados e acrescentando uma unidade de controle responsável pelo gerenciamento das ações que serão realizadas para diferentes classes de instruções...

**[Todo] Figura RTL**

# 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 16 - Resultado na waveform.

# Considerações finais

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