****

**PODER EXECUTIVO**

**MINISTÉRIO DA EDUCAÇÃO**

**UNIVERSIDADE FEDERAL DE RORAIMA**

**DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO**

**ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES**

**RELATÓRIO DO PROJETO: PROCESSADOR INTELIGENTE**

**DISCENTES:**

**Victor Deluca Almirante Gomes – 2201524401**

**Allan Cordeiro Rocha de Araújo – 2201524427**

**Março de 2017**

**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 INTELIGENTE**

**Março de 2017**

**Boa Vista/Roraima**

**Resumo**

Este trabalho aborda o projeto e a implementação do processador IntelIgente, desenvolvido por Victor Deluca e Allan Cordeiro. Serão descritos aqui os componentes do processador e a forma como eles se conectam no datapath, o formato das instruções a serem executadas no processador e os testes que foram realizados para garantir que todas as unidades funcionam corretamente.

**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)

# Especificações

Nesta seção são apresentados e descritos cada um dos componentes do processador e suas funcionalidades, além de outros itens relativos ao desenvolvimento do projeto.

## 1.1 Plataforma de desenvolvimento

Para a implementação do processador IntelIgente foi utilizada a IDE Quartus Prime 16.1 Lite, e a linguagem utilizada foi VHDL.

**1.2 Propriedades e limitações**

O processador IntelIgente possui 16 registradores REG0, REG1 ... e REG16, dos quais nenhum é um registrador auxiliar ou reservado: É possível ler e escrever em todos eles livremente. A memória ROM é limitada a 16 instruções, cada uma das quais consiste em 16 bits. O sistema opera apenas com inteiros entre -8 e 7, portanto qualquer operação

## 1.3 Conjunto de instruções

O processador IntelIgente possui 12 tipos de instruções diferentes dividias em várias categorias descritas abaixo. Os 4 bits mais significativos representam o opcode da instrução, enquanto os 12 outros assumem funções diferentes de acordo com o tipo de instrução.

* **Instruções de três registradores:** São as instruções mais simples, que realizam operações entre dois registradores e armazenam o resultado em um terceiro registrador. Consistem na maior parte das instruções do processador. São instruções de três registradores: ADD, SUB, AND, OR e XOR.
* **Instrução de dois registradores:** Instrução de formato similar ao das instruções de três registradores, porém utiliza apenas o valor de um registrador na entrada. A única instrução que utiliza este formato é o NOT. O segundo registrador de entrada é sempre “0000” por padrão, mas o valor lido nesse registrador nunca é utilizado.
* **Instruções de operador constante:** Instruções que operam utilizando um ou dois registradores, e uma constante de 4 bits. As instruções LW e SW utilizam esse formato, com um registrador para a leitura do endereço a partir do offset e um registrador para receber a entrada (SW) ou armazenar o resultado (LW), dependendo da operação. A instrução LI também utiliza esse formato, porém apenas um registrador é utilizado, para armazenar o valor lido. O outro registrador recebe “0000” por padrão, mas nunca é utilizado.
* **Instruções de salto:** Instruções que podem alterar o valor do PC, saltando a execução para uma instrução específica desejada. Existem dois tipos de salto: O salto incondicional (J) e o salto condicional (BEQ, BNE). No caso do salto condicional, a instrução possui dois registradores e os 4 bits menos significativos representam a “distância” do salto, ou seja, quantas instruções serão puladas a partir da próxima instrução, e o salto será realizado apenas quando a condição para o salto for cumprida. No caso do salto incondicional, o salto possui 12 bits. É importante notar que, como o salto é maior que o número total de bits, o endereço da nova instrução é calculado da forma: (PC + 1 + distância do salto) mod 16.

Tipo de Instruções:

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

Formato para escrita de código na linguagem Quantum:

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

Formato para escrita em código binário:

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

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

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.

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

## 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

**[Todo] Descrição**

### Memória de Instruções

**[Todo] Descrição**

### Somador

**[Todo] Descrição**

### And

**[Todo] Descrição**

### Mux\_2x1

**[Todo] Descrição**

### PC

**[Todo] Descrição**

### ZERO

**[Todo] Descrição**

## 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 19 - Resultado na waveform.

# Considerações finais

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