

**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**

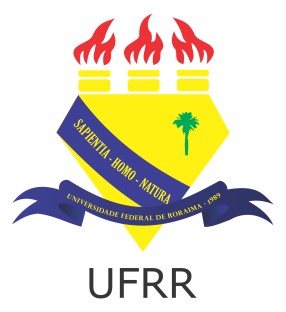
**ALUNOS:**

**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 e mostrará todos os testes realizados durante a implementação. Toda a execução do projeto, incluindo imagens e testes gerados, ocorreu através da utilização da ferramenta Quatus Prime Lite da Intel Corporation e da linguagem de descrição de hardware VHDL.

**Conteúdo**

Lista de Figuras

[**Figura 1 Especificações Quartus 6**](file:///C:\Users\lucas\Downloads\Relatorio_Aoc_XRL-8_1.docx#_Toc72190989)

Lista de Tabelas

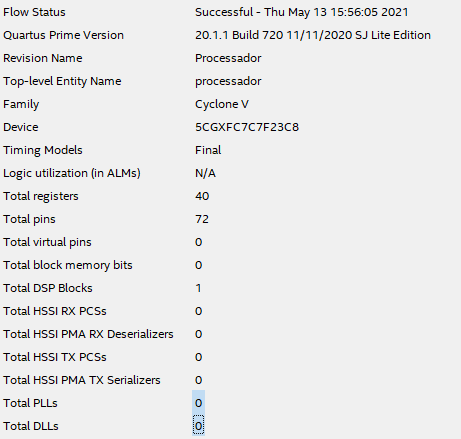
# 

1. 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.

* 1. **Plataforma de desenvolvimento**

Para a implementação do processador XLR-8 foi utilizado a IDE:



**Figura 1 - Especificações do Quartus**

* 1. **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, desta maneira cada bloco realiza uma função específica no barramento do processador:

* **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 |

**Tabela 1 - Divisão de bits para instruções do tipo R**

Formato para escrita em baixo nível:

* **Operação** operando1, operando2.

- **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 |

**Tabela 2 - Divisão de bits para instruções do tipo I**

Formato para escrita em baixo nível:

* **Operação** operando1, endereço de memória.

**- 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 |

**Tabela 3 - Divisão de bits para instruções do tipo J**

Formato para escrita em baixo nível:

* **Operação** endereço de salto.

* 1. **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, porém primordial de instruções para o devido funcionamento do processador. A linguagem suportada pelo processador é chamada de Omni e esta suporta até 2 ^ (número de bits do opcode) - 1 instruções.

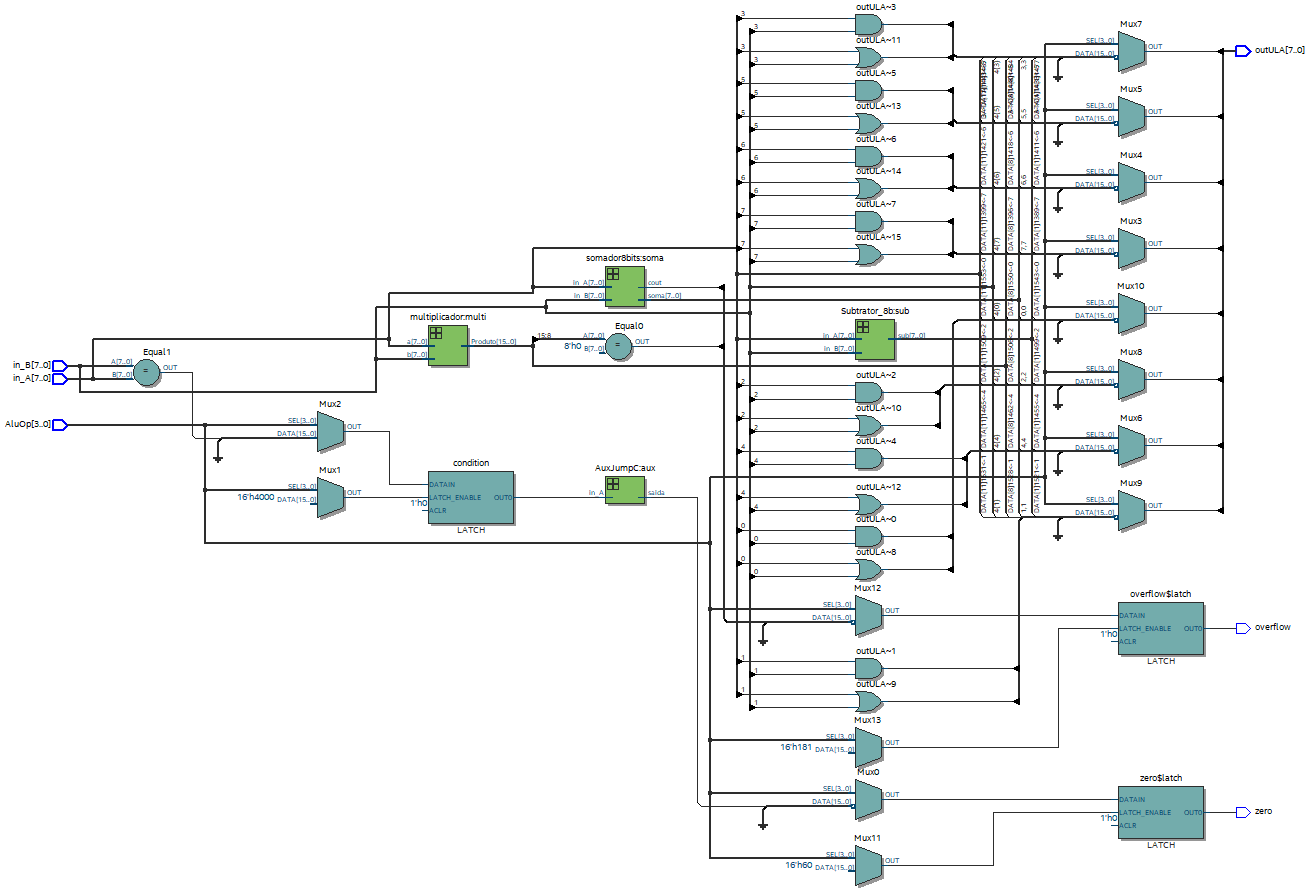
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instrução | Tipo | Opcode | Sintaxe | Registradores |
| Add | R | (0000) | Add $s0,$s1 | 2 |
| Sub | R | (0001) | Sub $s0,$s1 | 2 |
| Lw | I | (0010) | Lw $s0, address | 2 |
| Sw | I | (0011) | Sw $s0, address | 2 |
| J | J | (0100) | J address | 0 |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

**Tabela 4 - Descrição das instruções do processador XRL-8**

* 1. **Descrição do Hardware**
     1. **ULA ou ALU**

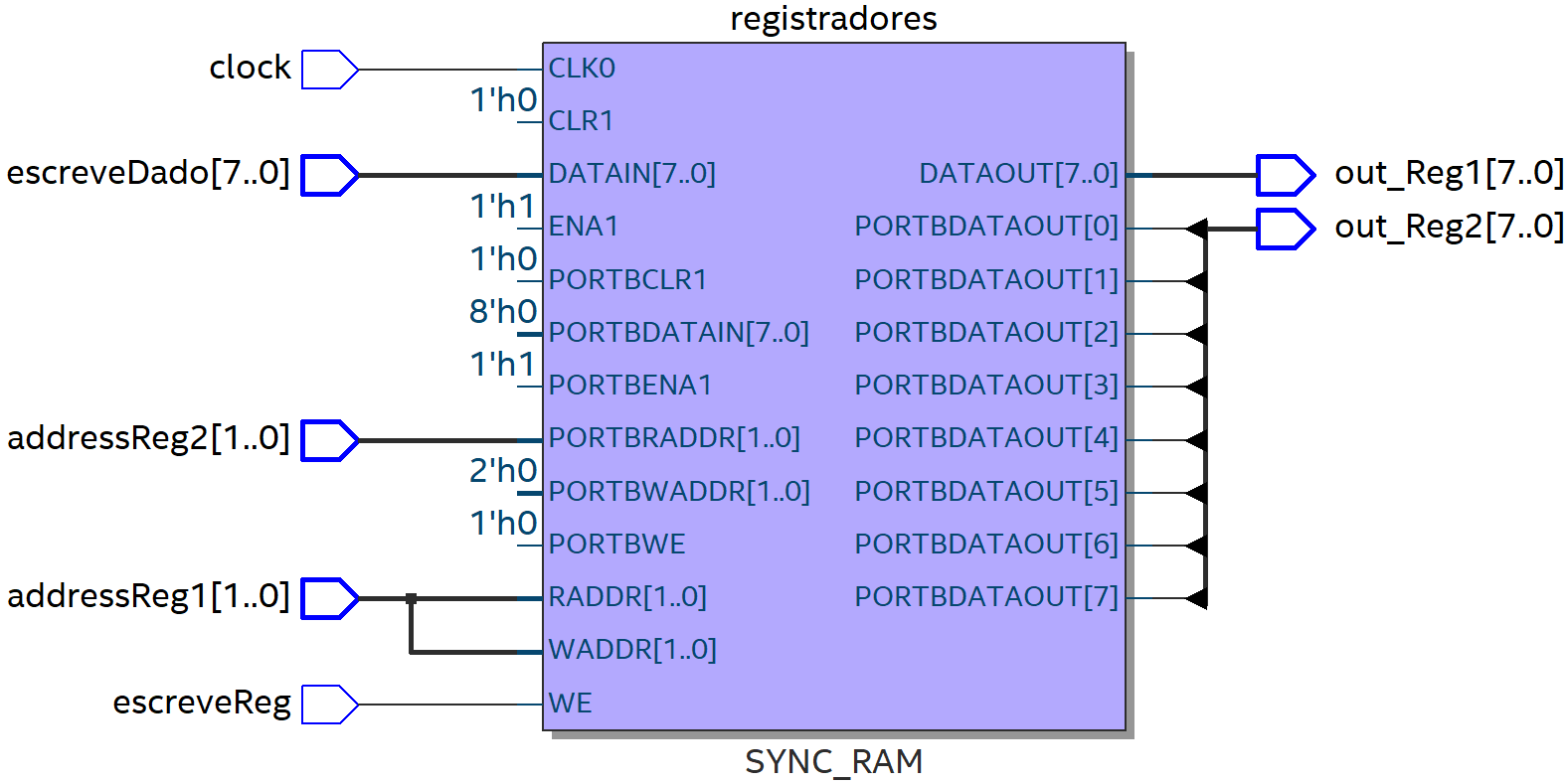
A ULA também conhecida como Unidade Lógica e aritmética (ou em Inglês Arithmetic Logic Unit), é um dispositivo que realiza operações lógicas e aritméticas com os valores passados na entrada. No processador XRL-8, este componente possui 3 entradas e 3 saídas entre elas:

* + Entrada 1: Que recebe uma trilha de 8 bits vinda do banco de registradores.
  + Entrada 2: Recebe uma trilha também de 8 bits vinda de um multiplexador.
  + AluOp: Flag de controle da Unidade de Controle, que indica ao componente qual operação deve ser realizada.
  + Zero: Flag de saída da ULA, que indica se um salto em memória deve ou não ser realizado.
  + Resultado: trilha de 8 bits que passará para o barramento o resultado das operações realizadas na ULA.
  + Overflow: Flag que indica ao barramento se houve ou não um estouro de memória na execução da operação na ULA.



**Figura 2 - RTL VIEWER da ULA**

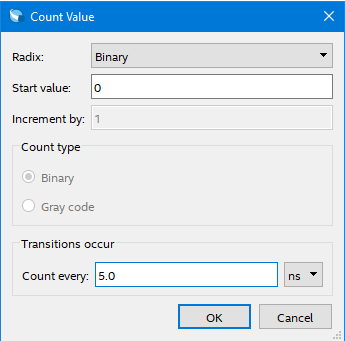
* + 1. **Banco de Registradores**

 É um componente composto por um conjunto de registradores que podem ser acessados de maneira organizada. Podem ser executadas operações de leitura dos dados anteriormente gravados e de escrita de dados para modificar as informações internas. Por ter essas capacidades é um dos componentes mais importantes para o fluxo de dados no processador.

**Figura 3 - RTL VIEWER do Banco de Registradores**

* + 1. **Clock**

É um temporizador que define no barramento quais componentes devem estrar ativos ou inativos. O período de ciclo de clock do processador XRL-8 é de 10 ns e como o processador é monociclo, todas as instruções serão executadas em um único ciclo de clock.



**Figura 4 - Ciclo de Clock da waveform do processador**

* + 1. **Unidade de Controle**

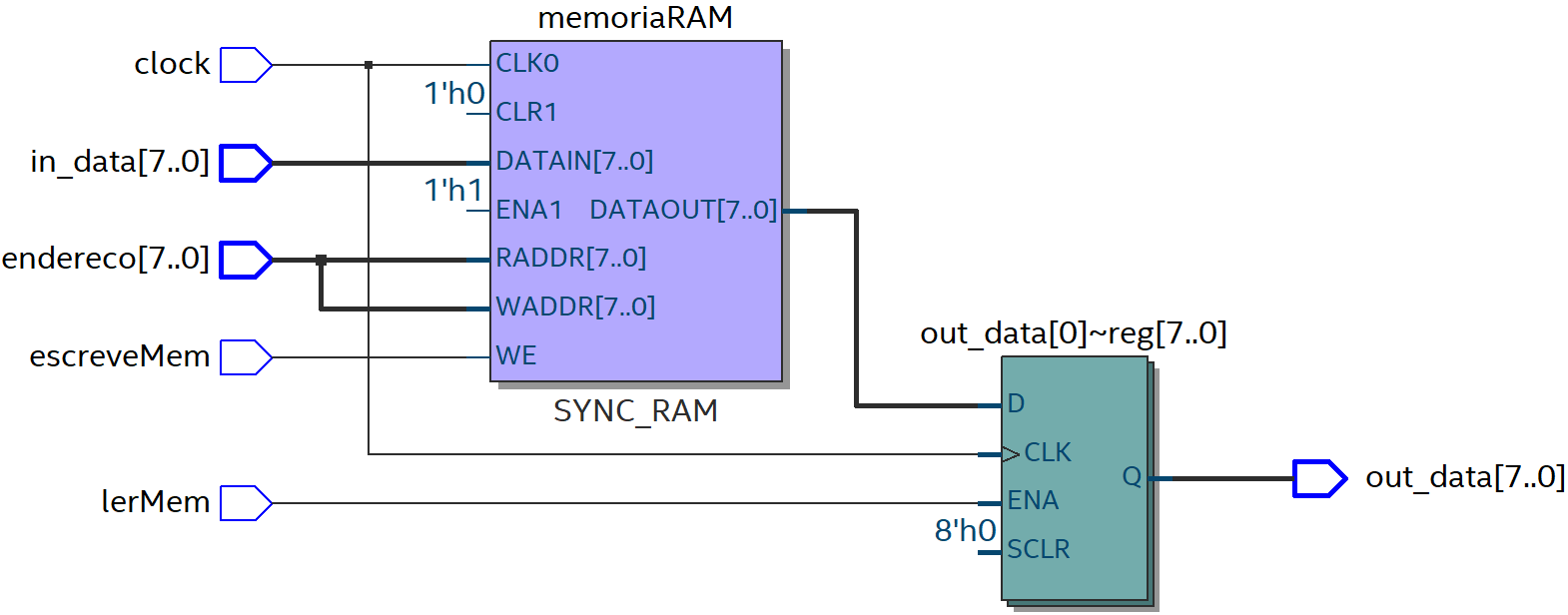
Na Unidade de Controle temos a utilização de 4 bits para o Opcode, onde dependendo do valor utilizado, terá um valor diferente para as flags. São por meio delas que temos as instruções:

* **OrigAlu**: Flag que indica ao multiplexador se o valor da entrada da ULA virá do banco de registradores ou do extensor de sinal.
* **EscreveReg**: Utilizada para a escrita de dados no Banco de registradores. Quando ativa, um determinado dado é guardado em algum dos registradores.
* **EscreveMem**: Indica à memória de dados se um valor de entrada deve ou não ser guardado.
* **AluOp**: Flag vinda da unidade de controle onde será definida qual instrução será executada.
* **MemToReg**: Dependendo do sinal, determina qual valor será passado para guardar no Banco de Registradores.
* **LerMem**: Determina se o valor da memória de dados é passado para o barramento ou não.
* **Branch:** Dependendo da saída da Flag Zero, determina se um salto deverá ou não ser realizado.
* **Jump**: Uso exclusivo para a operação Jump, determina se um salto incondicional deve ou não ser realizado.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | X | 1 | 0 | 0010 | 0 | 1 | 0 | 0 |
| Sw | X | 0 | 1 | 0011 | 1 | 0 | 0 | 0 |
| J | X | 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 5 - Flags de controle da Unidade de Controle**

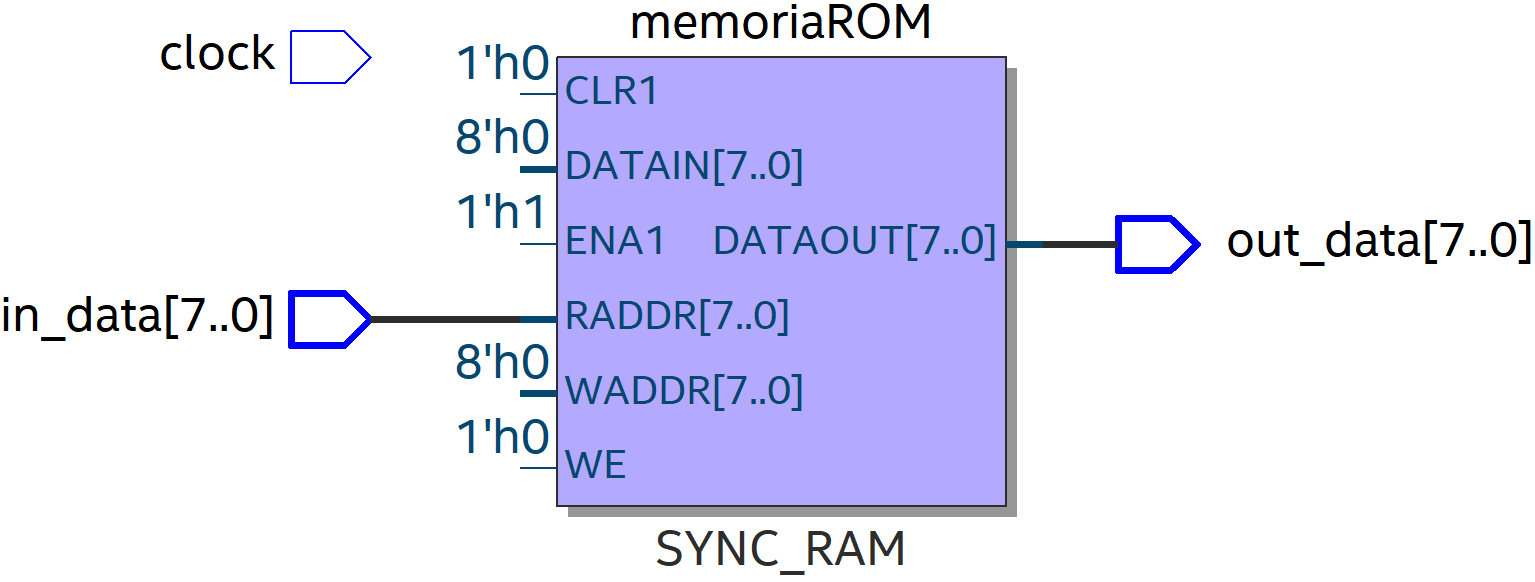
* + **Memória de dados (RAM)**

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

**Figura 5 - RTL VIEWER da memória de dados**

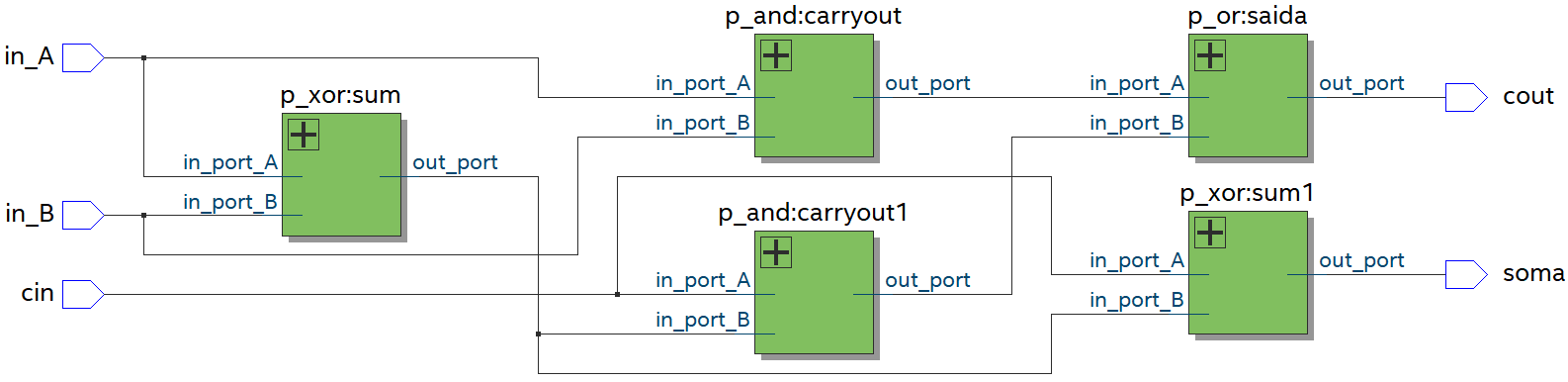
* + 1. **Memória de Instruções (Memória Rom)**

Guarda as instruções a serem executadas no processador.

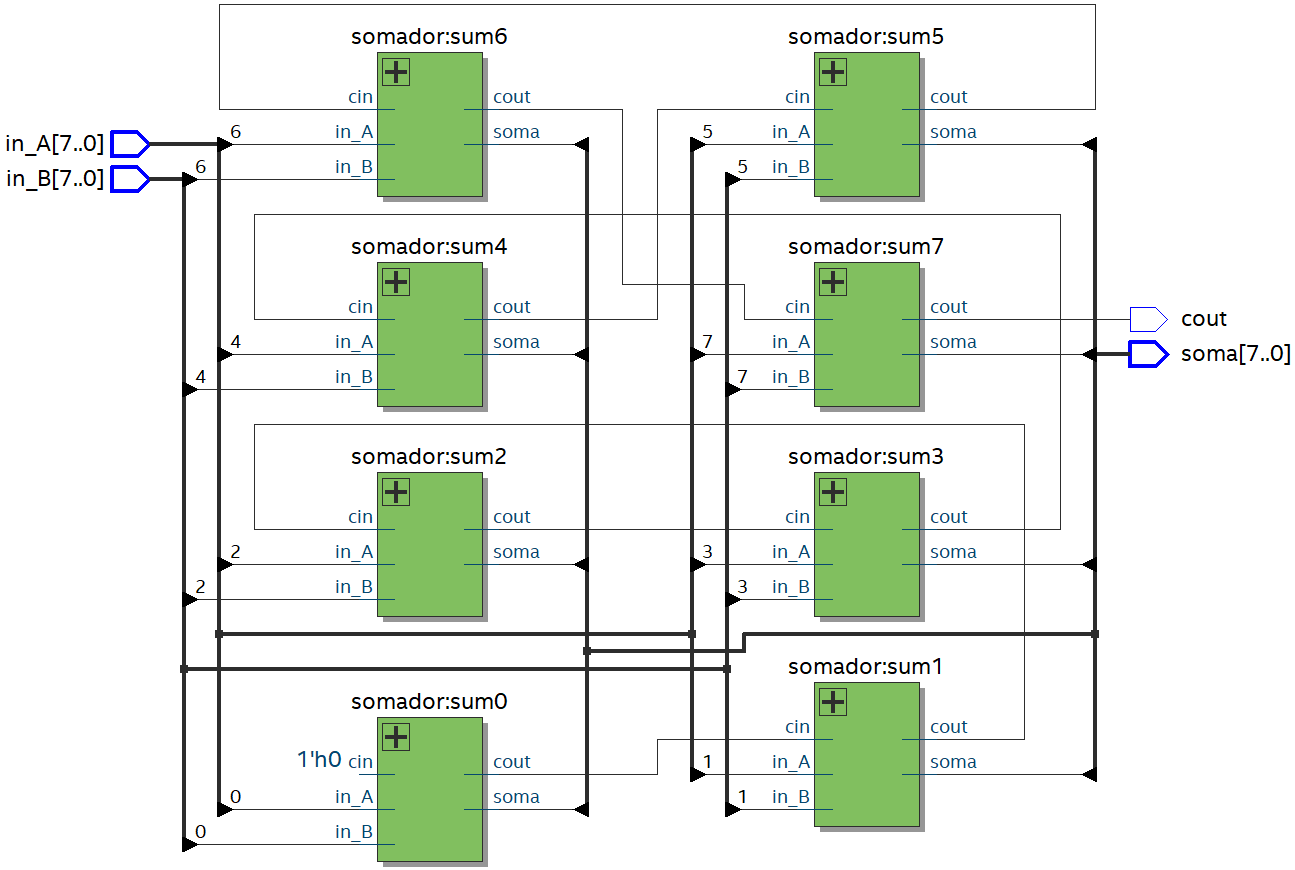


# **Figura 6 - RTL VIEWER da Memória Rom.**

* + 1. **Somador**

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

**Figura 6 - RTL VIEWER do somador**



* + 1. **Subtrator**

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

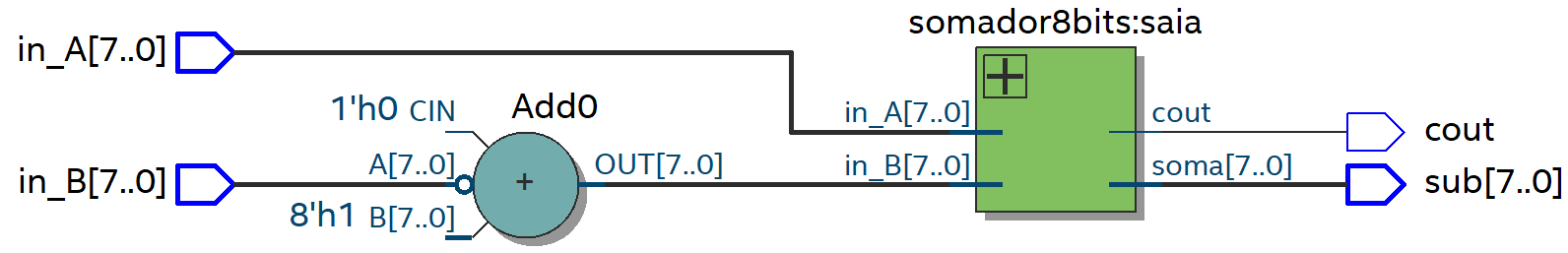
****

Figura 9 - RTL VIEWER do subtrator de 8 bits .

* + 1. **Porta And**

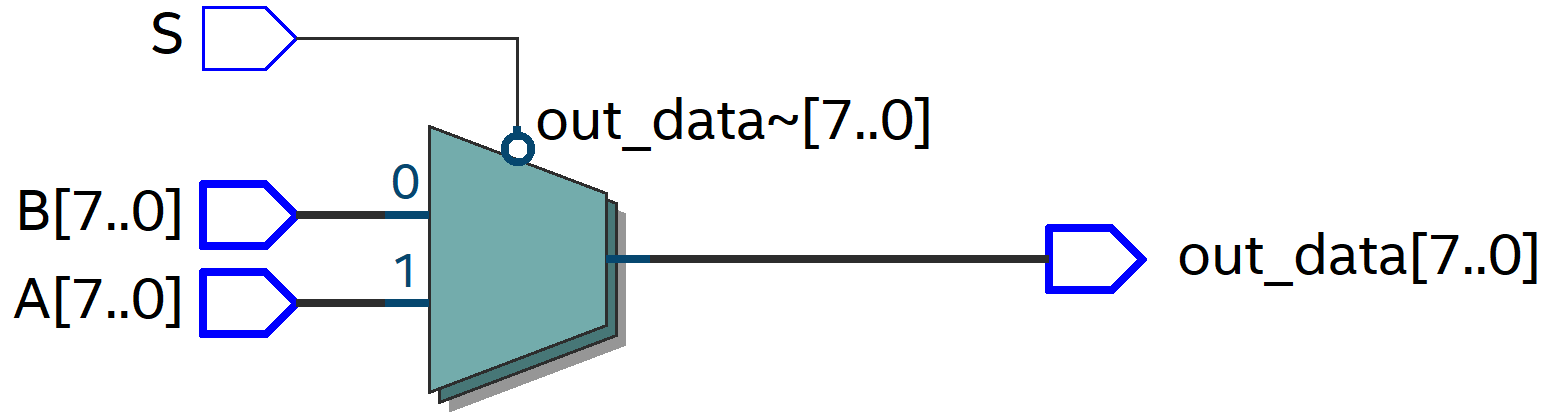
É uma operação lógica que ocorre a entrada de dois operandos que resulta em apenas um valor lógico verdadeiro, mas isso só ocorre se todos os operandos forem verdadeiros.

# 

**Figura 7 - RTL VIEWER porta and**

* + 1. **Muxtiplexador\_2x1**

É um componente onde ocorre uma escolha de trilha para continuar o fluxo.



**Figura 8 - RTL VIEWER do Multiplexador 2x1.**

* + 1. **PC (Program Counter)**

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

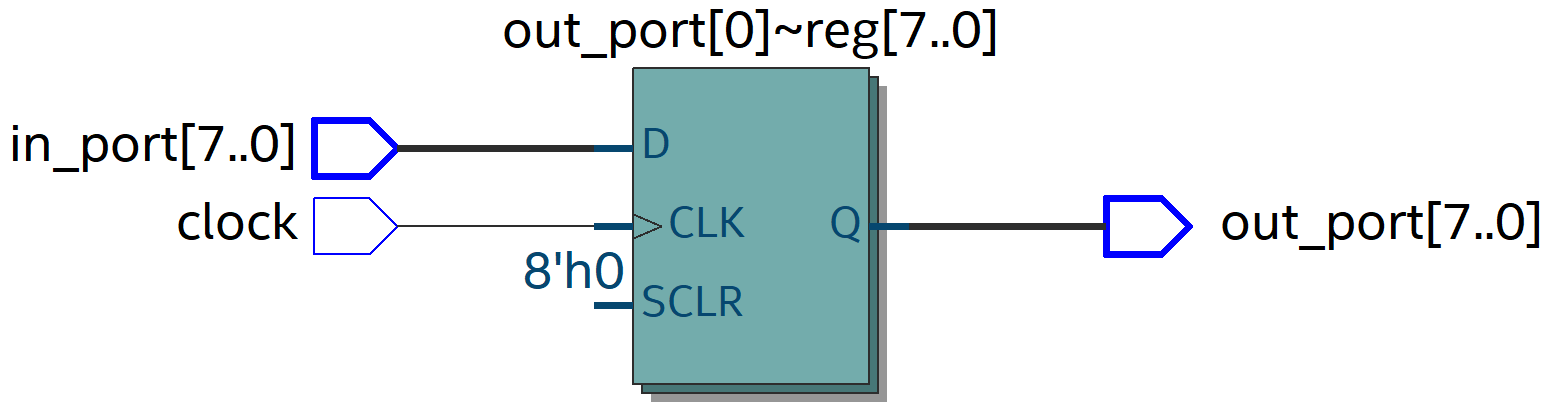


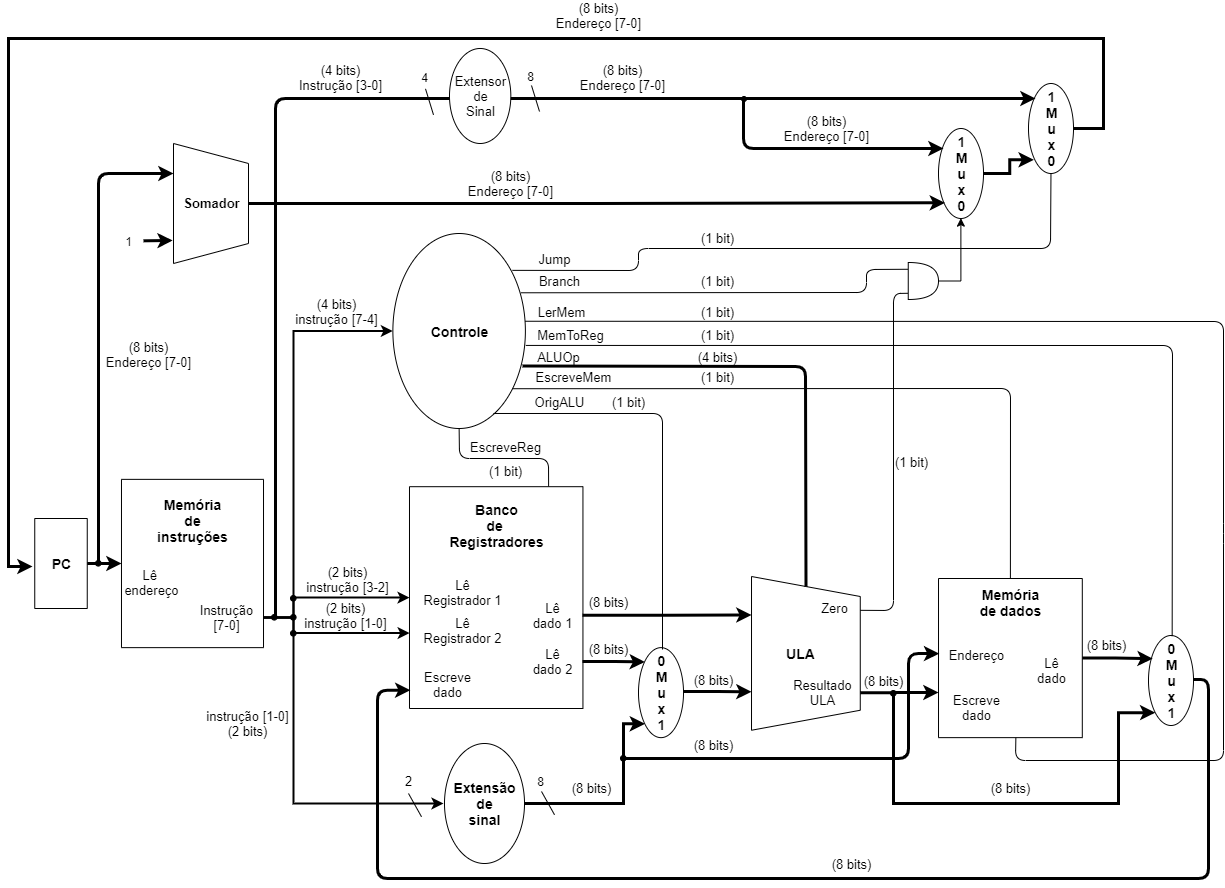
Figura 9 - RTL VIEWER do Program Counter (PC).

* + 1. **Zero**

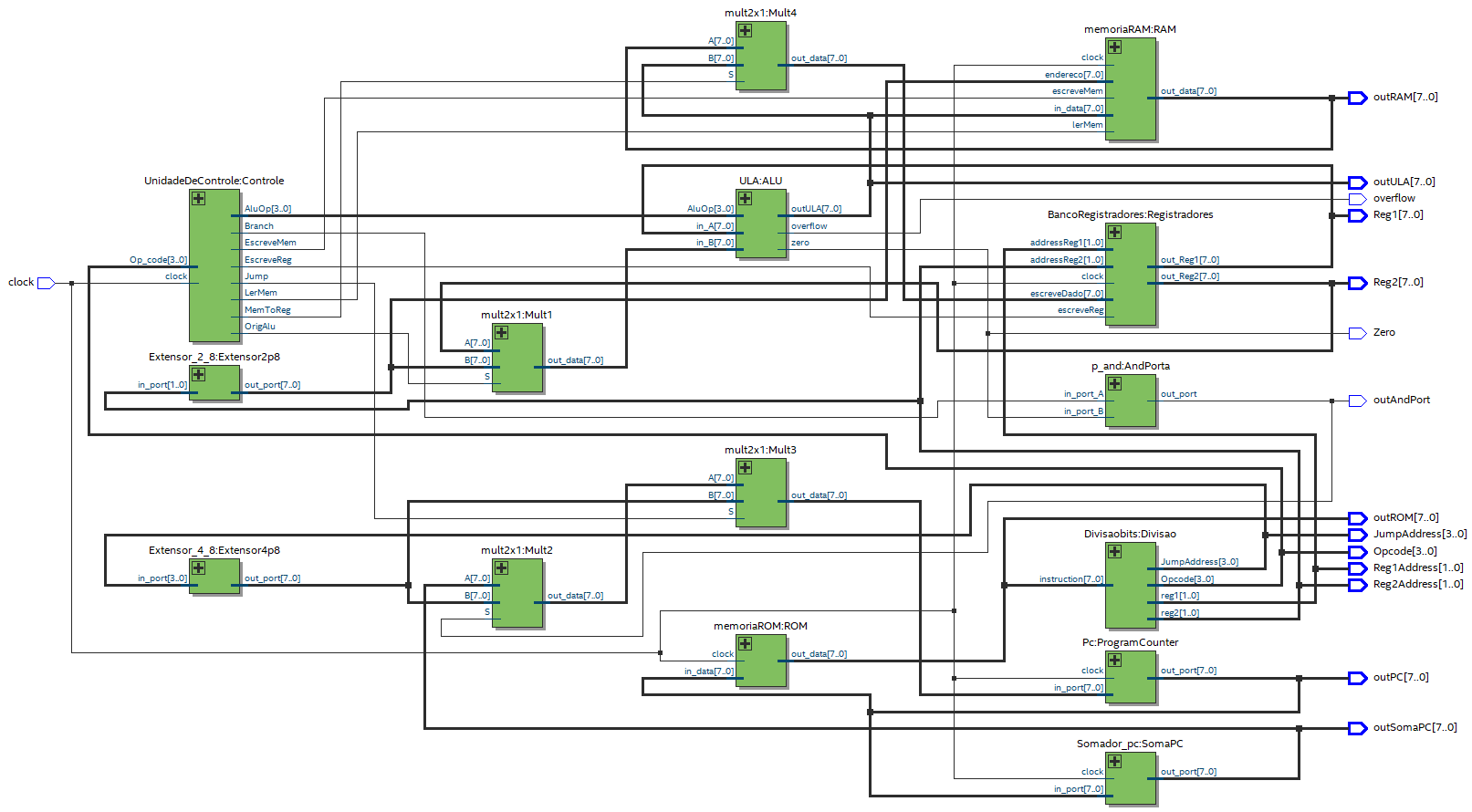
Flag da ULA que retorna para o barramento o sinal alto “1” toda vez que ocorrer um salto condicional. Desta maneira o sinal da flag “ZERO” em conjunto com o sinal da Unidade de controle “Branch” irão selecionar a trilha correta para que ocorra ou não o salto na memória de instruções.

* + 1. **Datapath**

Barramento do processador, hardware que contém todos os componentes e suas respectivas interligações.



**Figura 10 - Representação do datapath**



**Figura 11 - RTL VIEWER do processador**

1. **Simulações e Testes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Endereço** | **Instrução (Omni)** | **Instrução (Binário)** | | |
| **Opcode** | **Reg1** | **Reg2** |
| **Endereço** | |
| 0 | Li $s3, 3 | 1100 | 11 | 11 |
| 1 | Addi $s3, 2 | 0111 | 11 | 10 |
| 2 | Li $s0, 1 | 1100 | 00 | 01 |
| 3 | Move $s1, $s0 | 1101 | 01 | 00 |
| 4 | Mul $s0, $s1 | 1000 | 00 | 01 |
| 5 | JumpC $s1, $s3 | 1110 | 01 | 11 |
| 6 | Addi $s1, 1 | 0111 | 01 | 01 |
| 7 | Bne 4 | 0110 | 0100 | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Endereço** | **Instrução (Omni)** | **Instrução (Binário)** | | |
| **Opcode** | **Reg1** | **Reg2** |
| **Endereço** | |
| 0 | Li $s3, 3 | 1100 | 11 | 11 |
| 1 | Mult $s3, $s3 | 1000 | 11 | 11 |
| 2 | Addi $s3, 1 | 0111 | 11 | 01 |
| 3 | Li $s2, 1 | 1100 | 10 | 01 |
| 4 | Li $s0, 1 | 1100 | 00 | 01 |
| 5 | Move $s1, $s0 | 1101 | 01 | 00 |
| 6 | Sw $s1, 0 | 00110 | 01 | 00 |
| 7 | Add $s1, $s0 | 0000 | 01 | 00 |
| 8 | Lw $s0, 0 | 0010 | 00 | 00 |
| 9 | JumpC $s3, $s2 | 1110 | 11 | 10 |
| 10 | Addi $s2, 1 | 0111 | 10 | 01 |
| 11 | Bne 6 | 0110 | 0110 | |