

**PODER EXECUTIVO**

**MINISTÉRIO DA EDUCAÇÃO**

**UNIVERSIDADE FEDERAL DE RORAIMA**

**DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO**

**ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES**

**RELATÓRIO DO PROJETO: PROCESSADOR EK**

**ALUNOS:**

**Eduardo Henrique Freire Machado - 2020001617**

**Kelvin Araújo Ferreira - 2019037653**

**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 EK**

**Março** **de 2022**

**Boa Vista/Roraima**

**Resumo**

Este trabalho aborda o projeto e implementação de um processador EK, utilizando do software Quartus Prime, da linguagem Assembly MIPS e da linguagem VHDL (Very High-Speed Integration Circuit HDL). O processador em questão é um RISC (Reduced Instruction Set Computer) de 8 bits.

O relatório então servirá de instrumento de avaliação dos alunos para a disciplina de AOC (Arquitetura e Organização de Computadores), ministrada pelo professor Herbert Oliveira Rocha.

Sumário

[1. Especificação 7](#_Toc97788805)

[1.1. Plataforma de Desenvolvimento 7](#_Toc97788806)

[1.2. Conjunto de Instruções 7](#_Toc97788807)

[1.3. Descrição do Hardware 9](#_Toc97788808)

[1.3.1. ALU 9](#_Toc97788809)

[1.3.2. BANCO\_REGISTRADORES 10](#_Toc97788810)

[1.3.3. CONTADOR\_SINCRONO 11](#_Toc97788811)

[1.3.4. DIV\_INSTRUCAO 12](#_Toc97788812)

[1.3.5. EXTENSOR\_2X8 12](#_Toc97788813)

[1.3.6. EXTENSOR\_4X8 13](#_Toc97788814)

[1.3.7. MUX\_2X1 13](#_Toc97788815)

[1.3.8. PC 14](#_Toc97788816)

[1.3.9. RAM 14](#_Toc97788817)

[1.3.10. ROM 15](#_Toc97788818)

[1.3.11. SOMADOR\_8BITS 15](#_Toc97788819)

[1.3.12. SUBTRATOR\_8BITS 16](#_Toc97788820)

[1.3.13. UNIDADE\_DE\_CONTROLE 16](#_Toc97788821)

[1.4. Datapath 18](#_Toc97788822)

[2. Simulações e Testes 21](#_Toc97788823)

[2.1. Teste ADDI, SUB e SUBI 21](#_Toc97788824)

[2.2. Teste ADD e ADDI 21](#_Toc97788825)

[2.3. Teste BEQ 21](#_Toc97788826)

[2.4. Teste LI 22](#_Toc97788827)

[2.5. Teste FIBONACCI 22](#_Toc97788828)

[3. Considerações Finais 23](#_Toc97788829)

[4. Repositório 23](#_Toc97788830)

**Lista de Figuras**

|  |  |  |
| --- | --- | --- |
| Figuras | Descrição | Página |
| 1 | Especificações no *Quartus Prime* | 7 |
| 2 | RTL Viewer da ALU | 10 |
| 3 | RTL Viewer do BANCO\_REGISTRADORES | 11 |
| 4 | RTL Viewer do CONTADOR\_SINCRONO | 11 |
| 5 | RTL Viewer da DIV\_INSTRUCAO | 12 |
| 6 | RTL Viewer do EXTENSOR\_2X8 | 12 |
| 7 | RTL Viewer do EXTENSOR\_4X8 | 13 |
| 8 | RTL Viewer do MUX\_2X1 | 13 |
| 9 | RTL Viewer do PC | 14 |
| 10 | RTL Viewer da RAM | 15 |
| 11 | RTL Viewer da ROM | 15 |
| 12 | RTL Viewer do SOMADOR\_8BITS | 16 |
| 13 | RTL Viewer do SUBTRATOR\_8BITS | 16 |
| 14 | RTL Viewer da UNIDADE\_DE\_CONTROLE | 18 |
| 15 | Datapath | 19 |
| 16 | Datapath para instrução tipo R | 19 |
| 17 | Datapath para instrução tipo I | 20 |
| 18 | Datapath para instrução tipo J | 20 |
| 19 e 20 | Teste ADDI, SUB e SUBI | 21 |
| 21 e 22 | Teste ADD e ADDI | 21 |
| 23 e 24 | Teste BEQ | 21 |
| 25 e 26 | Teste LI | 22 |
| 27, 28 e 29 | Teste FIBONACCI | 22 |

**Lista de Tabelas**

|  |  |  |
| --- | --- | --- |
| Tabelas | Descrição | Página |
| 1 | Opcodes suportados pelo processador EK | 9 |
| 2 | Relações entre OPCODES e flags na UNIDADE\_DE\_CONTROLE | 17 e 18 |

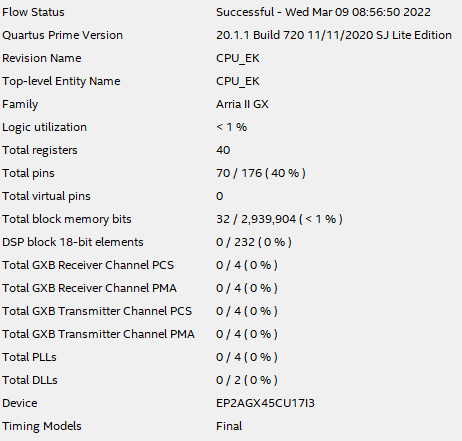
1. Especificação

Nesta seção é apresentado o conjunto de itens para o desenvolvimento do processador EK, bem como a descrição detalhada de cada etapa da construção do processador.

* 1. Plataforma de Desenvolvimento

Para a implementacação do processador EK foi utilizada a IDE *Quartus Prime*:

# Figura 1 – Especificações no *Quartus Prime*

****

Fonte: Elaborada pelos autores.

* 1. Conjunto de Instruções

O processador EK possui 2 registradores: $s0 e $s1. Assim como 11 formatos de instruções de 8 bits cada, instruções do tipo R, I e J. Seguem algumas considerações sobre as estruturas contidas nas instruções:

* Opcode: indica ao processador qual a instrução a ser executada;
* 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.

**Tipos de Instruções:**

* Tipo R: Este tipo de instrução trata de operações aritméticas.
  + Formato para escrita de código na linguagem MIPS:

|  |  |  |
| --- | --- | --- |
| Opcode | rs | rt |

* + Formato para escrita em código binário:

|  |  |  |
| --- | --- | --- |
| **Instrução do tipo R** | | |
| Opcode | rs | rt |
| 4bits | 2bits | 2bits |
| 7-4 | 3-2 | 1-0 |

* Tipo I: Este tipo de instrução aborda carregamentos diretos na memória.
  + Formato para escrita de código na linguagem MIPS:

|  |  |  |
| --- | --- | --- |
| Opcode | rs | Imediato |

* + Formato para escrita em código binário:

|  |  |  |
| --- | --- | --- |
| **Instrução do tipo I** | | |
| Opcode | rs | Imediato |
| 4bits | 2bits | 2bits |
| 7-4 | 3-2 | 1-0 |

* Tipo J: Este tipo de instrução é responsável por desvios condicionais e incondicionais.
  + Formato para escrita de código na linguagem MIPS:

|  |  |
| --- | --- |
| Opcode | Endereço |

* + Formato para escrita em código binário:

|  |  |
| --- | --- |
| **Instrução do tipo J** | |
| Opcode | Endereço |
| 4bits | 4bits |
| 7-4 | 3-0 |

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

O número de bits do campo Opcode das instruções é igual a quatro, sendo assim obtemos um total de 16 Opcodes (0-15) que são distribuídos entre as instruções, assim como é apresentado na Tabela 1.

## Tabela 1 – Opcodes suportados pelo Processador EK

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opcode | Sintaxe | Formato | Significado | Exemplos |
| 0000 | add | R | Soma | **add** $s0, $s1 |
| 0001 | addi | I | Soma Imediata | **addi** $s0, 3 |
| 0010 | sub | R | Subtração | **sub** $s0, $s1 |
| 0011 | subi | I | Subtração Imediata | **subi** $s0, 6 |
| 0100 | lw | I | Load Word | **lw** $s0 ram (00) |
| 0101 | sw | I | Store Word | **sw** $s0 ram (00) |
| 0110 | li | I | Load Imediato | **li** $s0 2 |
| 0111 | beq | J | Branch Equal | **beq** endereço |
| 1000 | if | J | If Equal | **If** $s0 $s1 |
| 1001 | J | J | Jump | **j** endereço(0000) |

Fonte: Elaborada pelos autores.

* 1. Descrição do Hardware

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

* + 1. ALU

O componente ALU (Unidade Lógica Aritmética) tem como principal objetivo efetuar as principais operações aritméticas (considerando apenas resultados inteiros), dentre elas: soma e subtração. Adicionalmente, ela efetua operações de comparação de valor como BEQ.

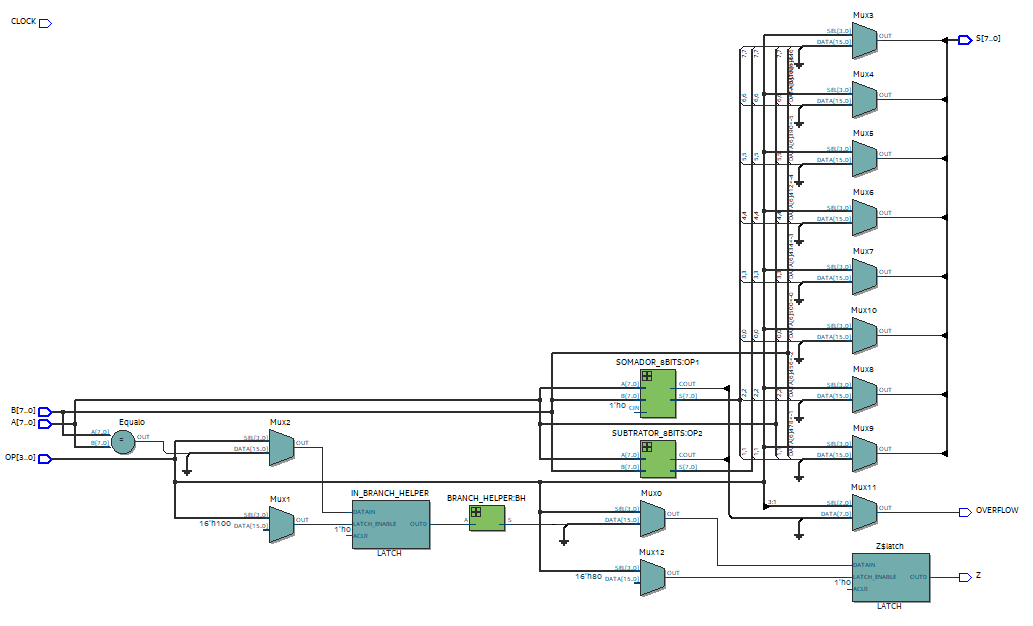
O componente ALU recebe como entrada quatro valores:

* CLOCK: dado de 1 bit;
* A e B: dados de 1 byte;
* OP: opcode de 4 bits.

A ALU possui três saídas:

* S: resultado de 1 byte;
* Z: resultado de 1 bit para verificar se o valor retornado é zero;
* OVERFLOW: resultado de 1 bit para verificar se a operação resulta num overflow.

# Figura 2 – RTL Viewer da ALU



Fonte: Elaborada pelos autores.

* + 1. BANCO\_REGISTRADORES

O componente BANCO\_REGISTRADORES tem como principal objetivo escrever, ler e armazenar valores nos registradores.

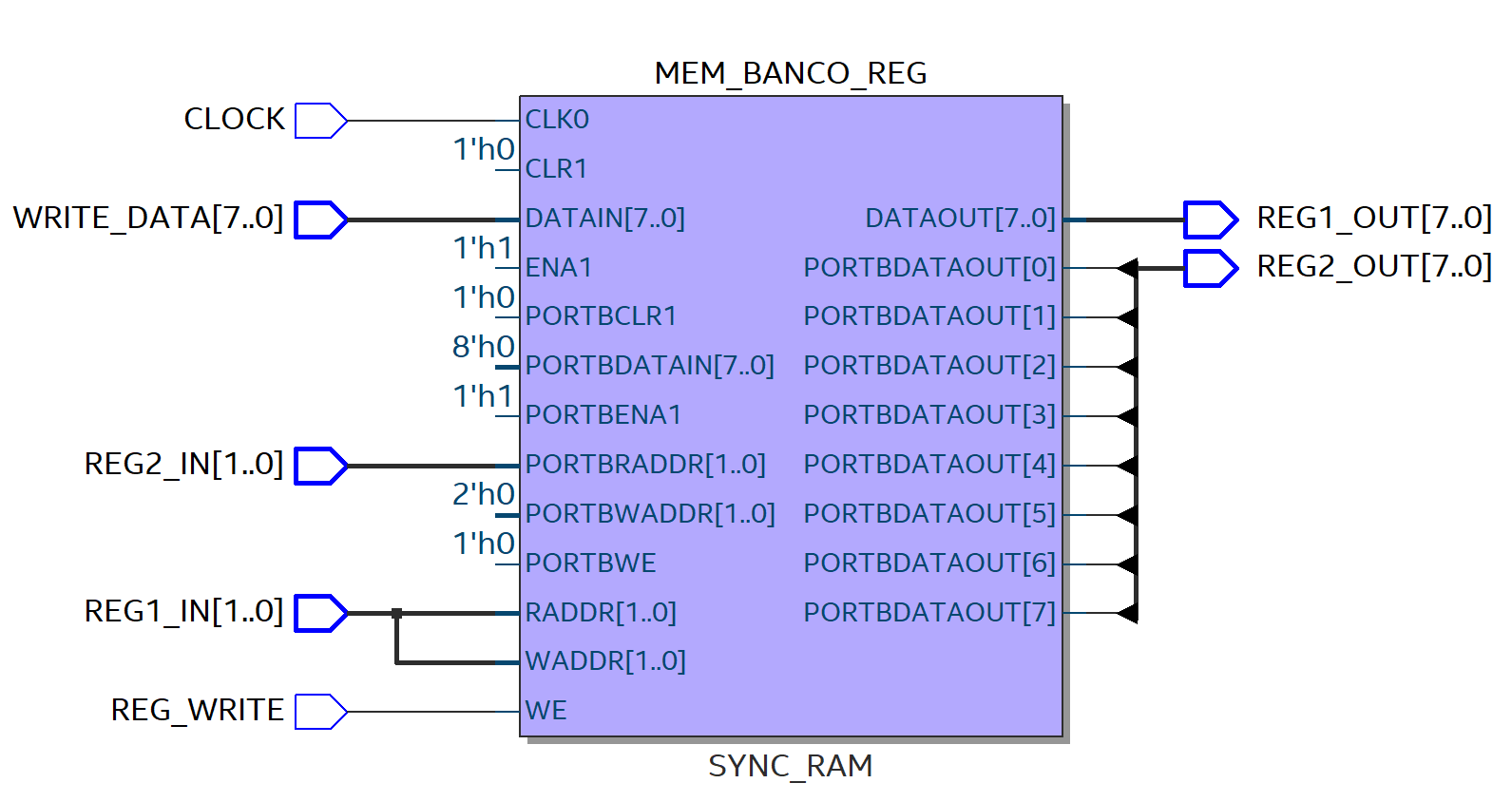
O componente BANCO\_REGISTRADORES recebe como entrada quatro valores:

* CLOCK: dado de 1 bit;
* REG\_WRITE: dado de 1 bit que indica se há escrita em registradores;
* REG1\_IN e REG2\_IN: dados de dois bits que indicam sobre quais registradores devem ser executados as operações;
* WRITE\_DATA: dado de 1 byte a ser escrito em registrador.

O componente BANCO\_REGISTRADORES tem duas saídas:

* REG1\_OUT e REG2\_OUT: dados de 1 byte armazenados nos registradores.

# Figura 3 – RTL Viewer do BANCO\_REGISTRADORES



Fonte: Elaborada pelos autores.

* + 1. CONTADOR\_SINCRONO

O componente CONTADOR\_SINCRONO tem como objetivo somar 1 ao PC, avançando assim para a próxima linha de código do programa na memória ROM.

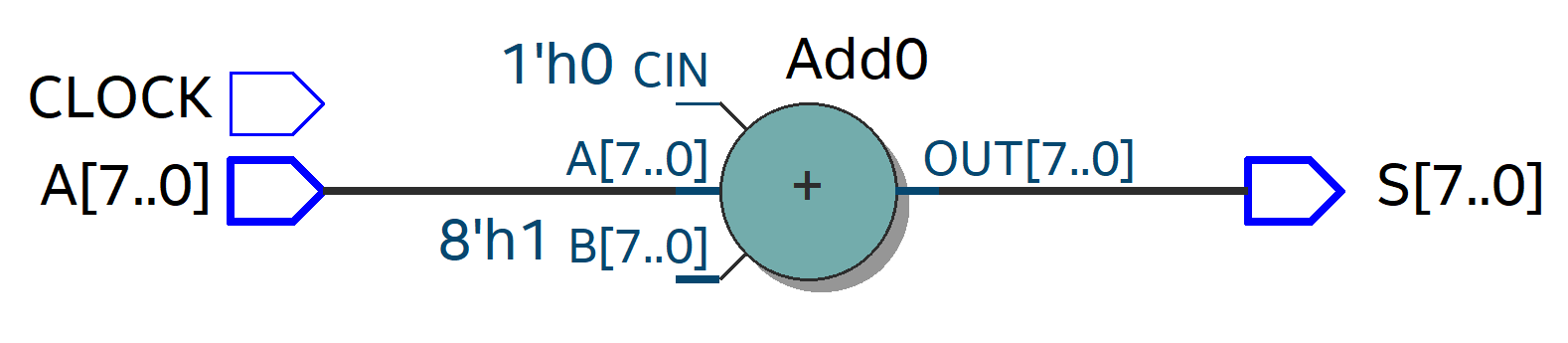
O componente CONTADOR\_SINCRONO recebe como entrada dois valores:

* CLOCK: dado de 1 bit;
* A: dado de 1 byte equivalente ao endereço no PC.

O componente CONTADOR\_SINCRONO tem como saída o valor:

* S: PC + 1.

# Figura 4 – RTL Viewer do CONTADOR\_SINCRONO



Fonte: Elaborada pelos autores.

* + 1. DIV\_INSTRUCAO

O componente DIV\_INSTRUCAO divide a instrução recebida em 4 trilhas, que são utilizadas para definir qual o tipo de instrução a ser executado e quais seus operandos.

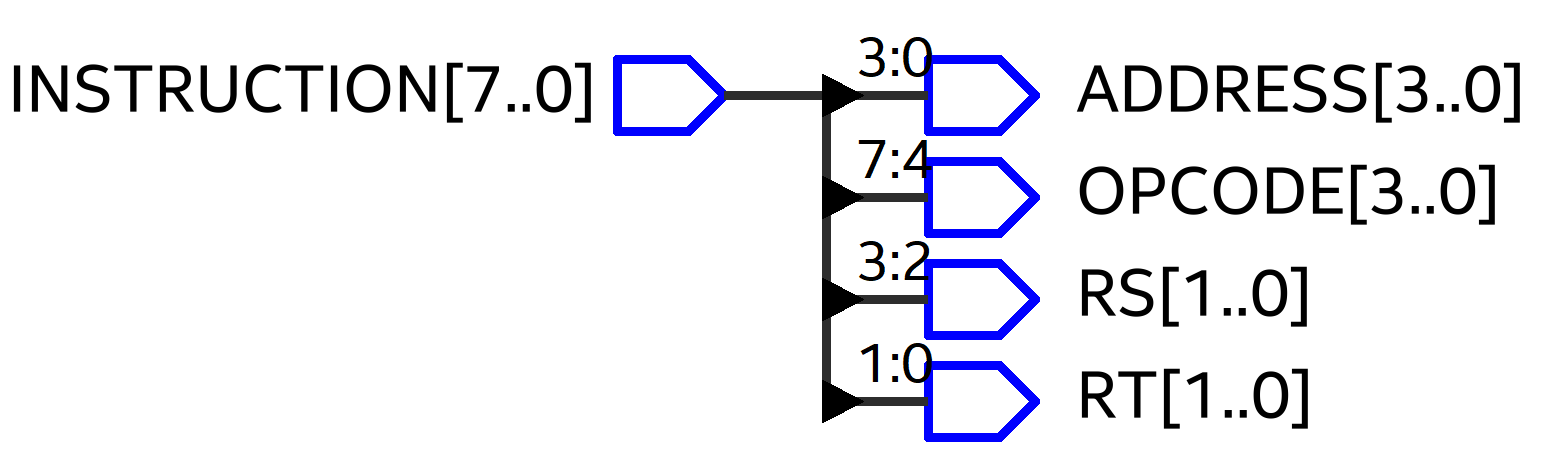
O componente DIV\_INSTRUCAO recebe como entrada:

* INSTRUCTION: dado de 1 byte com a instrução.

O componente DIV\_INSTRUCAO tem como saída:

* ADDRESS: dado de 4 bits com endereço;
* OPCODE: dado de 4 bits com o opcode;
* RS: dado de 2 bits com o primeiro operando;
* RT: dado de 2 bits com o segundo operando.

# Figura 5 – RTL Viewer do DIV\_INSTRUCAO



Fonte: Elaborada pelos autores.

* + 1. EXTENSOR\_2X8

O componente EXTENSOR\_2X8 extende um sinal de 2 bits para um de 1 byte.

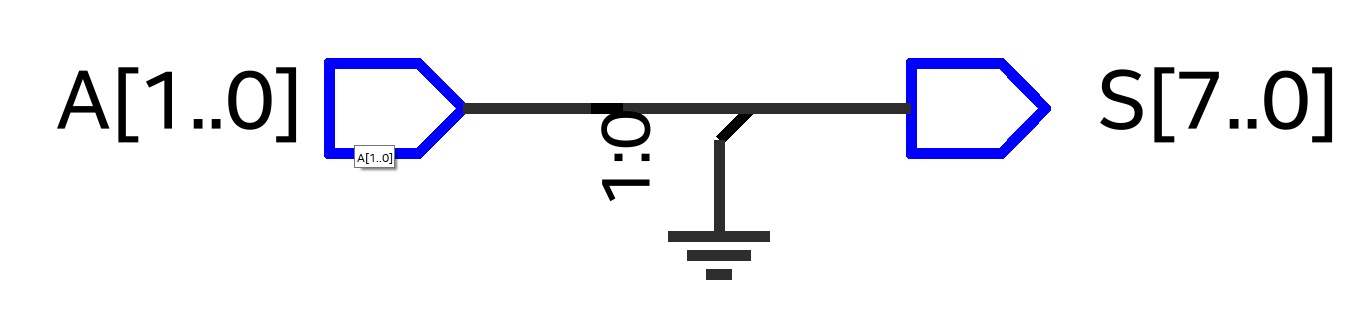
O componente EXTENSOR\_2X8 recebe como entrada:

* A: dado de 2 bits a ser extendido.

O componente EXTENSOR\_2X8 tem como saída:

* S: dado de 1 byte extendido.

# Figura 6 – RTL Viewer do EXTENSOR\_2X8



Fonte: Elaborada pelos autores.

* + 1. EXTENSOR\_4X8

O componente EXTENSOR\_4X8 extende um sinal de 4 bits para um de 1 byte.

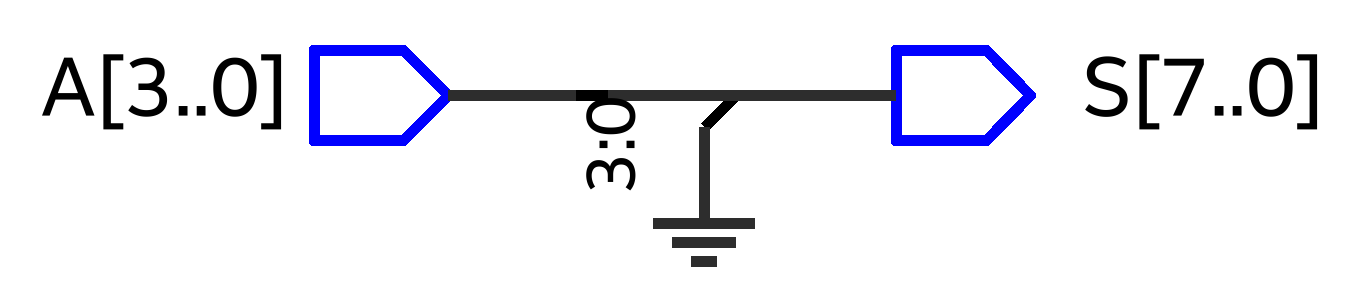
O componente EXTENSOR\_4X8 recebe como entrada:

* A: dado de 4 bits a ser extendido.

O componente EXTENSOR\_4X8 tem como saída:

* S: dado de 1 byte extendido.

# Figura 7 – RTL Viewer do EXTENSOR\_4X8



Fonte: Elaborada pelos autores.

* + 1. MUX\_2X1

O componente MUX\_2X1 seleciona um entre dois dados de 1 byte com base num seletor.

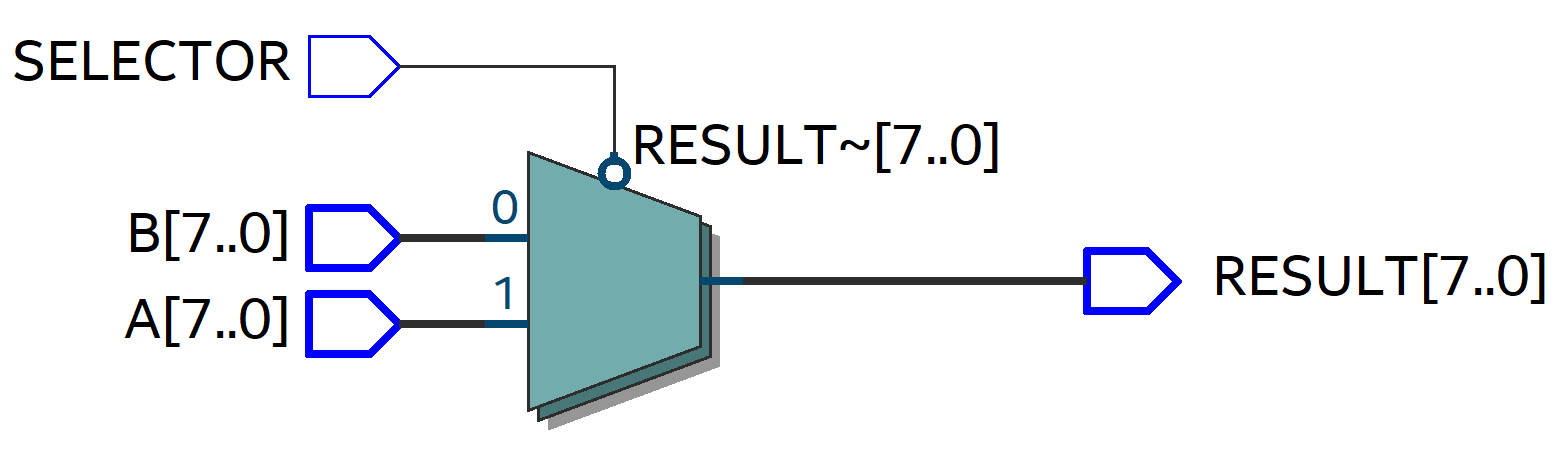
O componente MUX\_2X1 recebe como entrada:

* A e B: dados de 1 byte;
* SELECTOR: dado de 1 bit.

O componente MUX\_2X1 tem como saída:

* RESULT: dado de 1 byte que foi selecionado.

# Figura 8 – RTL Viewer do MUX\_2X1



Fonte: Elaborada pelos autores.

* + 1. PC

O componente PC é responsável por passar a linha de código do programa que deve ser executada.

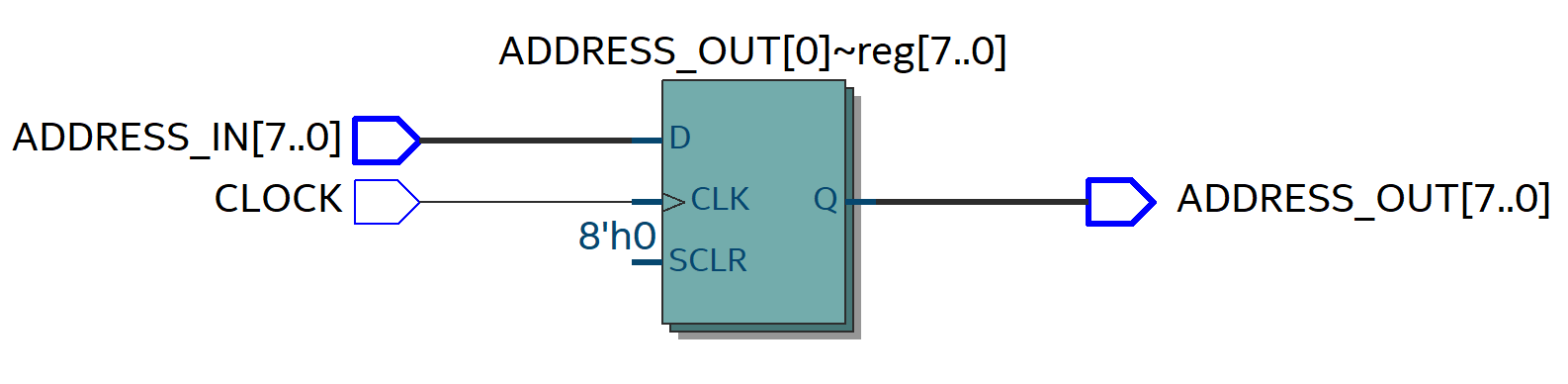
O componente PC recebe como entrada:

* CLOCK: dado de 1 bit;
* ADDRESS\_IN: dado de 1 byte com a linha atualizada.

O componente PC tem como saída:

* ADDRESS\_OUT: dado de 1 byte com a linha atual.

# Figura 9 – RTL Viewer do PC



Fonte: Elaborada pelos autores.

* + 1. RAM

O componente RAM é responsável por armazenar e ler até 8 dados de 1 byte por meio de instruções load e store.

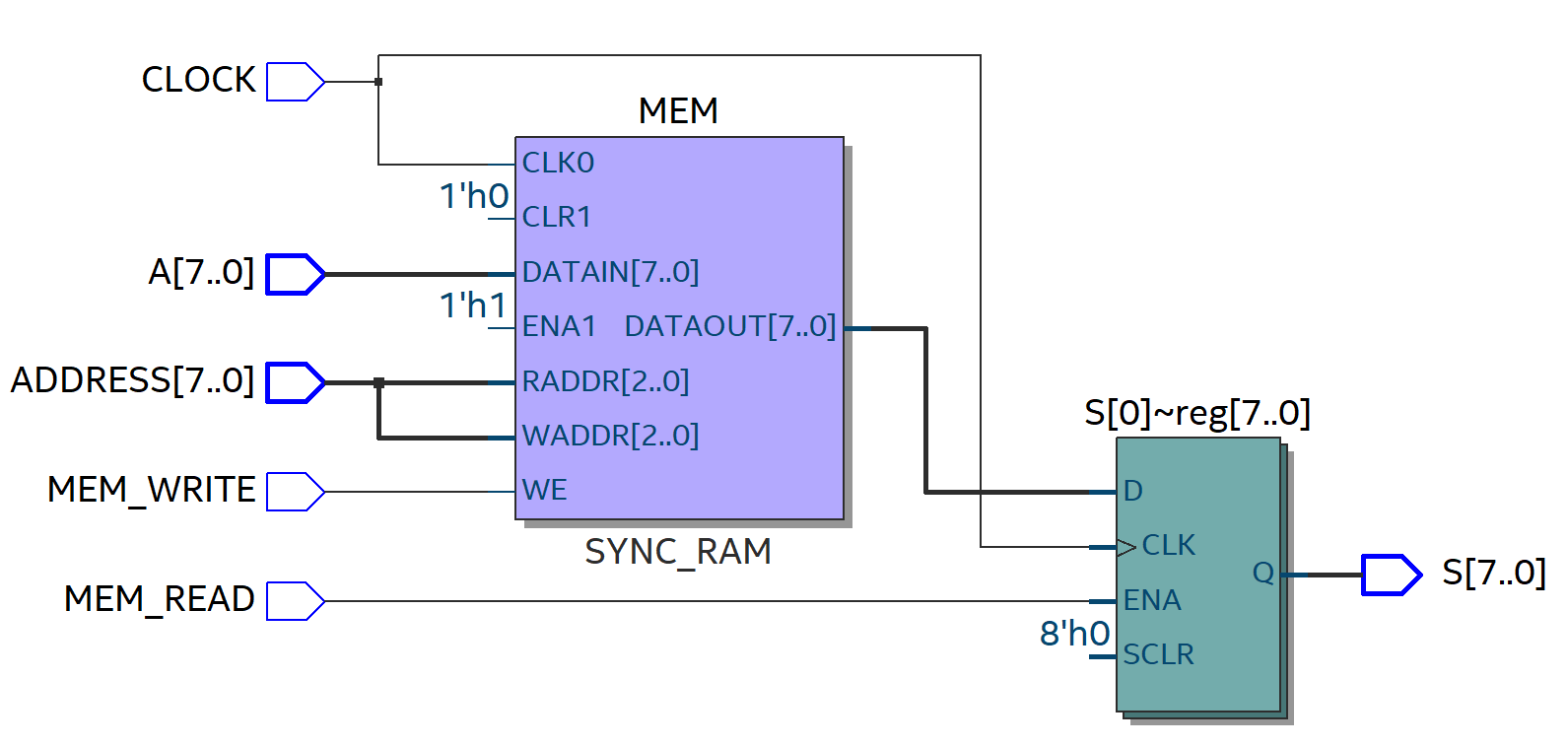
O componente RAM recebe como entrada:

* CLOCK: dado de 1 bit;
* A: dado de 1 byte a ser escrito;
* ADDRESS: dado de 1 byte com endereço;
* MEM\_WRITE: dado de 1 bit que serve como flag;
* MEM\_READ: dado de 1 bit que serve como flag.

O componente RAM tem como saída:

* S: dado de 1 byte com o resultado.

# Figura 10 – RTL Viewer da RAM



Fonte: Elaborada pelos autores.

* + 1. ROM

O componente ROM é responsável por armazenar o programa, com até 256 linhas de código.

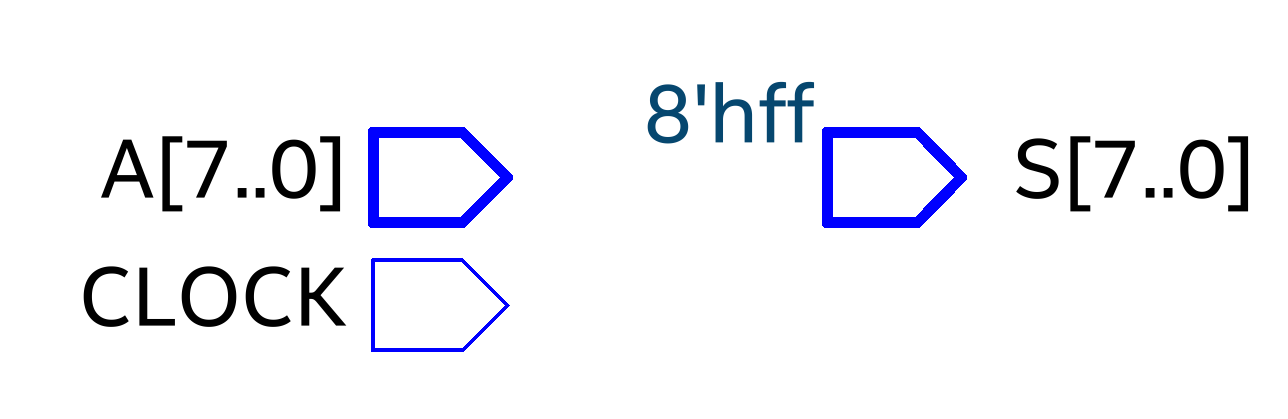
O componente ROM recebe como entrada:

* CLOCK: dado de 1 bit;
* A: dado de 1 byte com o endereço da linha a ser lida.

O componente ROM tem como saída:

* S: dado de 1 byte armazenado naquela linha.

# Figura 11 – RTL Viewer da ROM



Fonte: Elaborada pelos autores.

* + 1. SOMADOR\_8BITS

O componente SOMADOR\_8BITS tem como principal objetivo efetuar operações de soma entre dois dados de 1 byte.

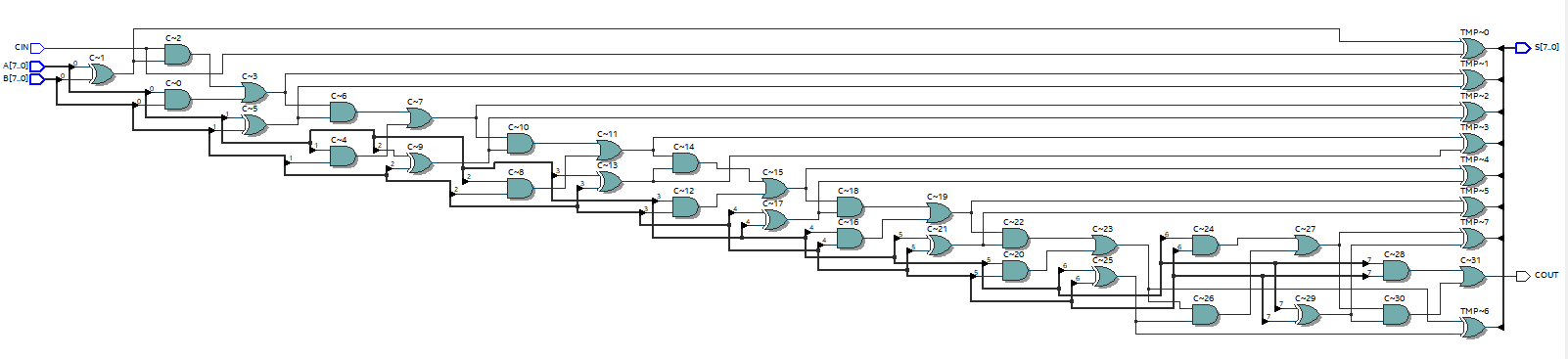
O componente SOMADOR\_8BITS recebe como entrada:

* A e B: dados de 1 byte com valores a serem somados;
* CIN: dado de 1 byte com o *carry in*.

O componente SOMADOR\_8BITS tem como saída:

* S: dado de 1 byte com o resultado;
* COUT: dado de 1 bit com o *carry out.*

# Figura 12 – RTL Viewer do SOMADOR\_8BITS



Fonte: Elaborada pelos autores.

* + 1. SUBTRATOR\_8BITS

O componente SUBTRATOR\_8BITS tem como principal objetivo efetuar operações de subtração entre dois dados de 1 byte.

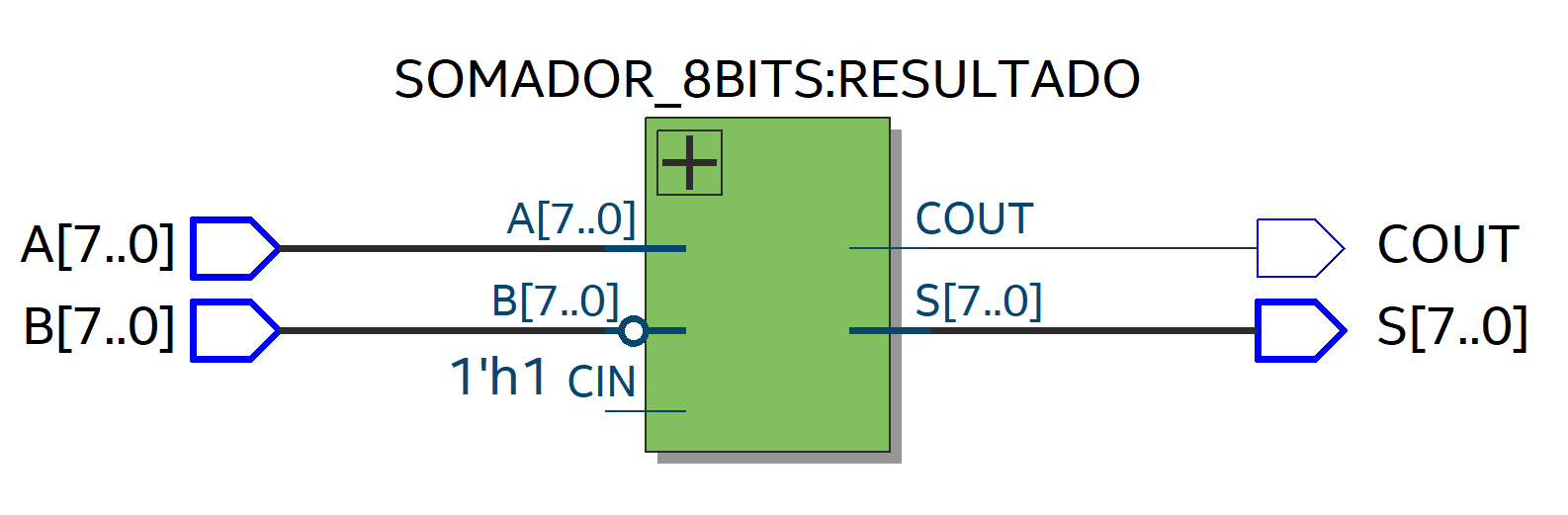
O componente SUBTRATOR\_8BITS recebe como entrada:

* A e B: dados de 1 byte com valores a se subtrair;

O componente SUBTRATOR\_8BITS tem como saída:

* S: dado de 1 byte com o resultado;
* COUT: dado de 1 bit com o *carry out.*

# Figura 13 – RTL Viewer do SUBTRATOR\_8BITS



Fonte: Elaborada pelos autores.

* + 1. UNIDADE\_DE\_CONTROLE

O componente UNIDADE\_DE\_CONTROLE tem como principal objetivo administrar as flags necessárias para cada tipo de instrução.

O componente UNIDADE\_DE\_CONTROLE recebe como entrada:

* CLOCK: dado de 1 bit;
* OPCODE: dado de 3 bits, indicando a operação a ser realizada;

O componente UNIDADE\_DE\_CONTROLE tem como saída:

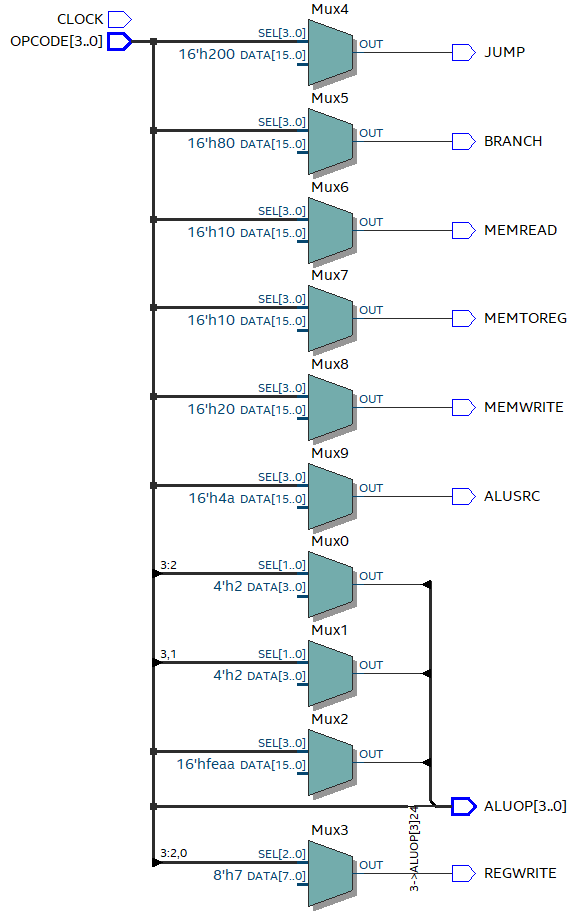
* JUMP: dado de 1 bit que serve de flag se há uma operação do tipo jump;
* BRANCH: dado de 1 bit que serve de flag se há uma operação característica de instrução tipo J;
* MEMREAD: dado de 1 bit que server de flag se há leitura de memória;
* MEMTOREG: dado de 1 bit que serve de flag se há escrita de um dado da memória num registrador;
* ALUOP: dado de 3 bits que indica a operação a ser realizada na ALU;
* MEMWRITE: dado de 1 bit que serve de flag se há escrita na memória;
* ALUSRC: dado de 1 bit que serve de flag se há necessidade de executar operação com registrador ou imediato;
* REGWRITE: dado de 1 bit que serve de flag se há escrita de dados no banco de registradores.

## Tabela 2 – Relações entre OPCODES e flags na UNIDADE\_DE\_CONTROLE

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| OPCODE | JUMP | BRANCH | MEMREAD | MEMTOREG | ALUOP | MEMWRITE | ALUSRC | REGWRITE |
| 0000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0001 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0010 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0011 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0100 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0101 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0110 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0111 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1001 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Fonte: Elaborada pelos autores.

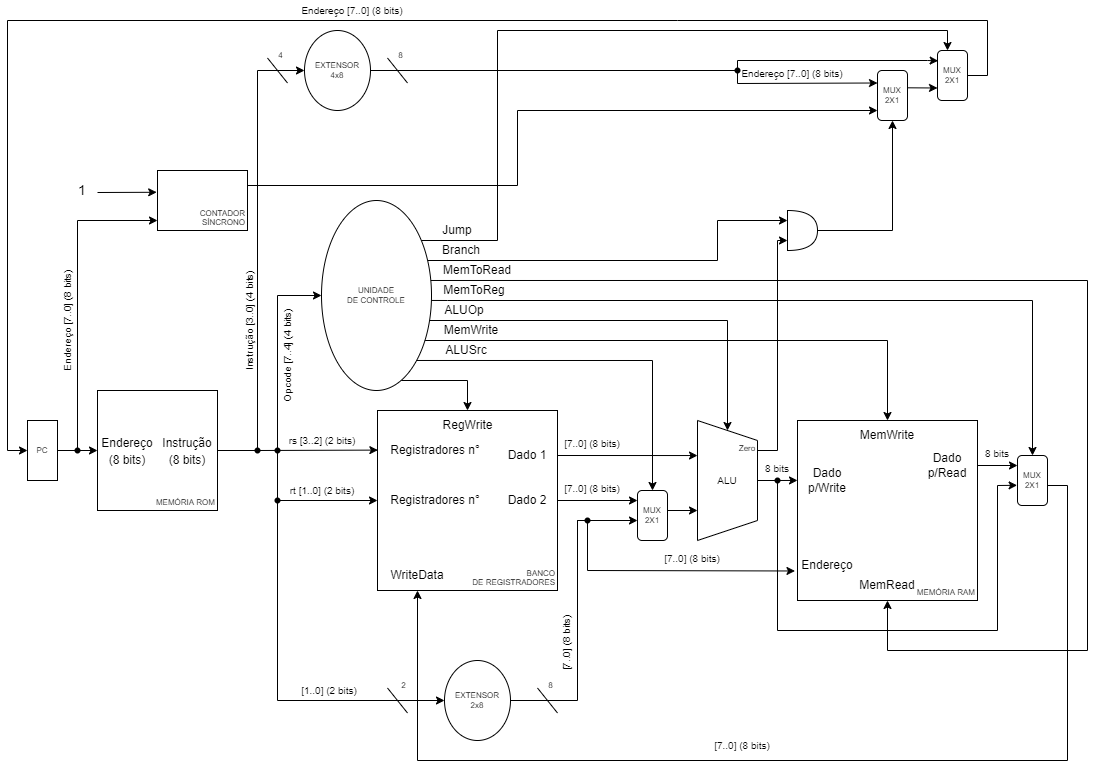
# Figura 14 – RTL Viewer da UNIDADE\_DE\_CONTROLE



Fonte: Elaborada pelos autores.

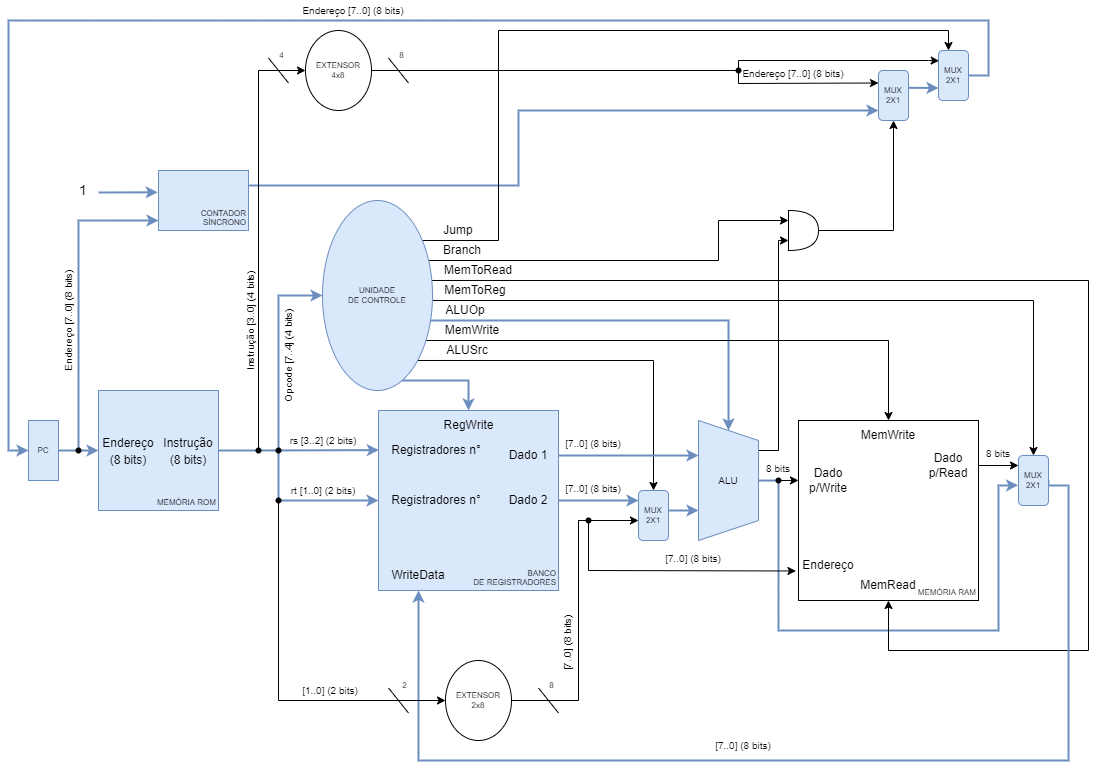
* 1. Datapath

# Figura 15 – Datapath



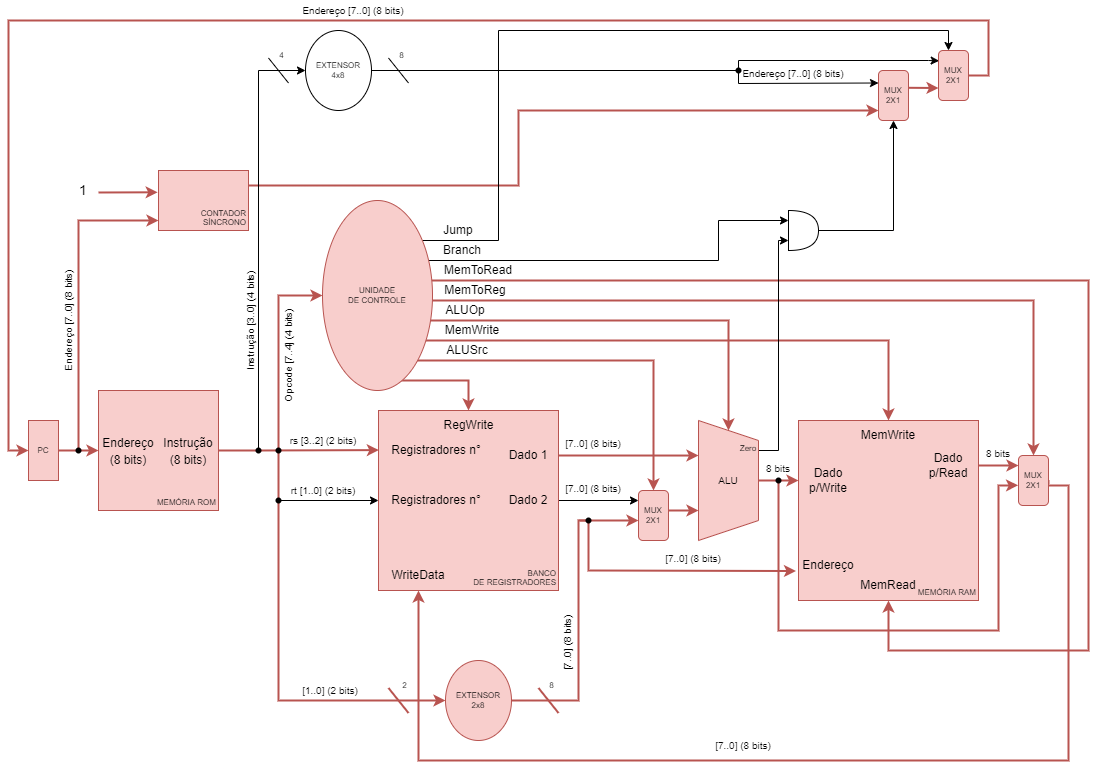
Fonte: Elaborada pelos autores.

# Figura 16 – Datapath para instrução tipo R



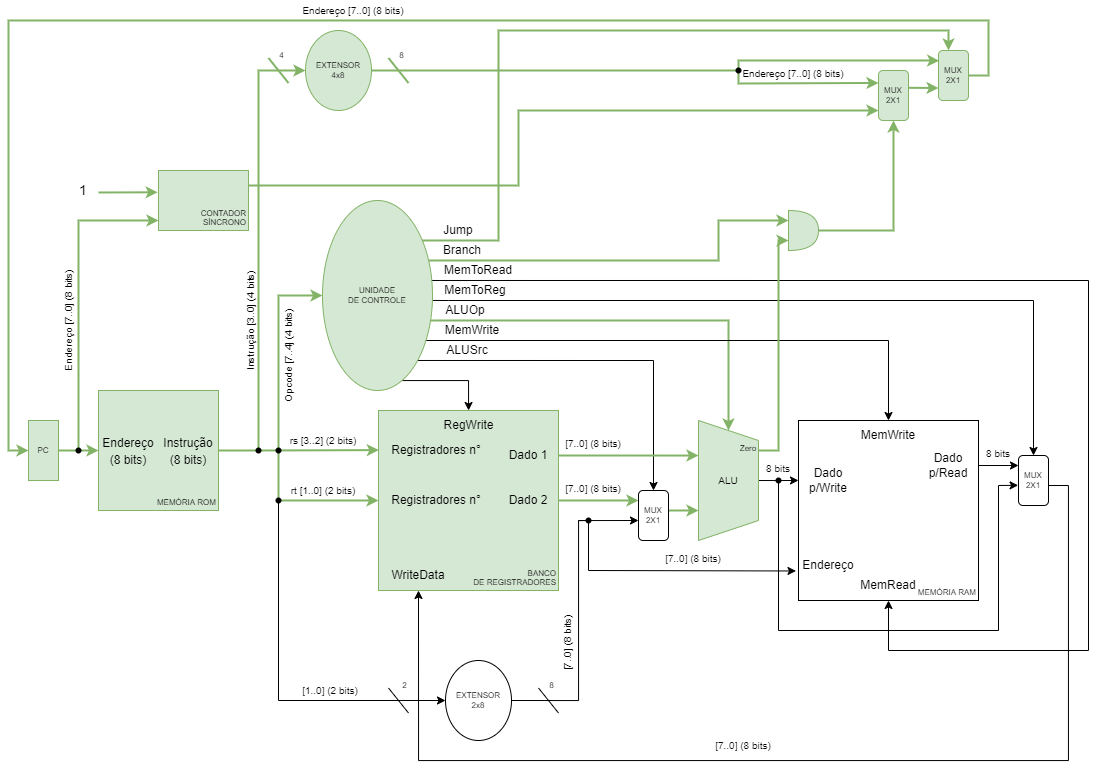
Fonte: Elaborada pelos autores.

# Figura 17 – Datapath para instrução tipo I



Fonte: Elaborada pelos autores.

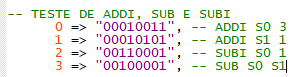
# Figura 18 – Datapath para instrução tipo J

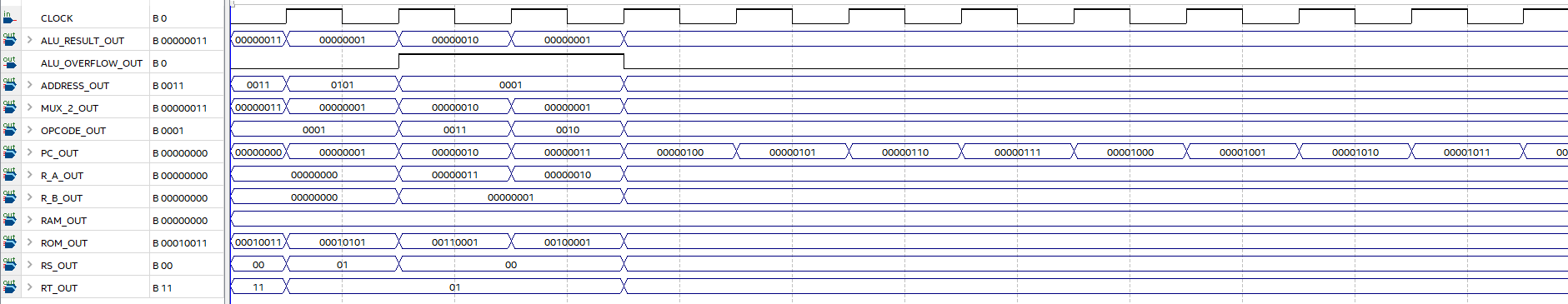


Fonte: Elaborada pelos autores.

1. Simulações e Testes
   1. Teste ADDI, SUB e SUBI

# Figuras 19 e 20 – Teste ADDI, SUB e SUBI



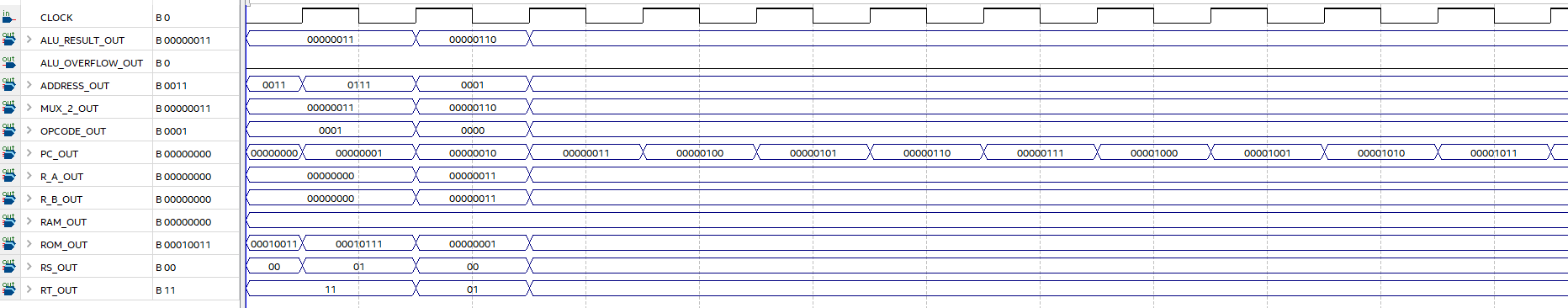


Fonte: Elaborada pelos autores.

* 1. Teste ADD e ADDI

# Figuras 21 e 22 – Teste ADD e ADDI

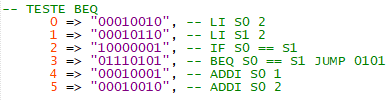
Teste de ADD e ADDI

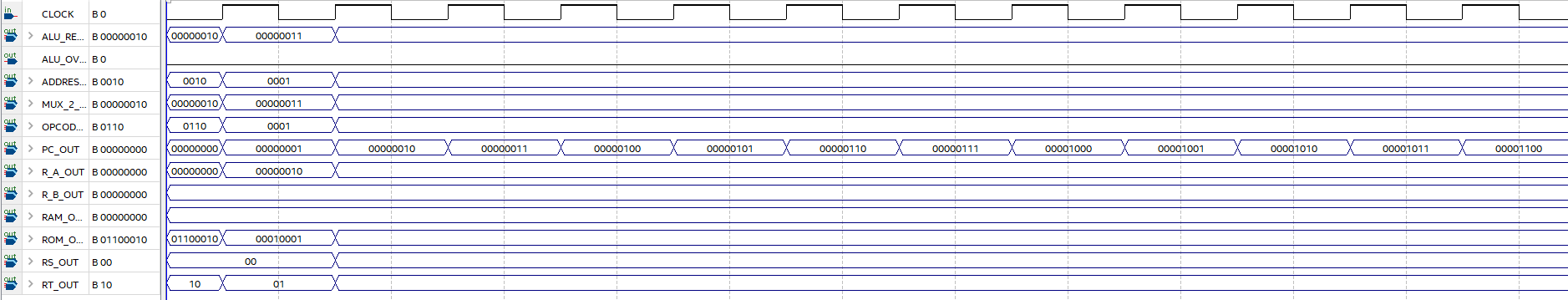


Fonte: Elaborada pelos autores.

* 1. Teste BEQ

# Figuras 23 e 24 – Teste BEQ



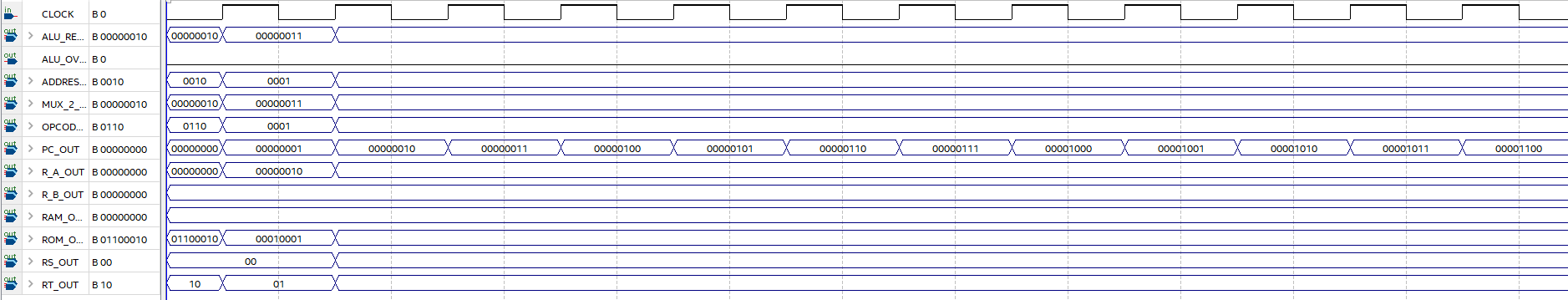


Fonte: Elaborada pelos autores.

* 1. Teste LI

# Figuras 25 e 26 – Teste LI

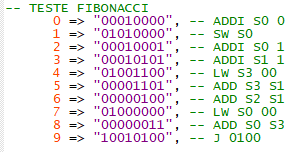
Teste de LI

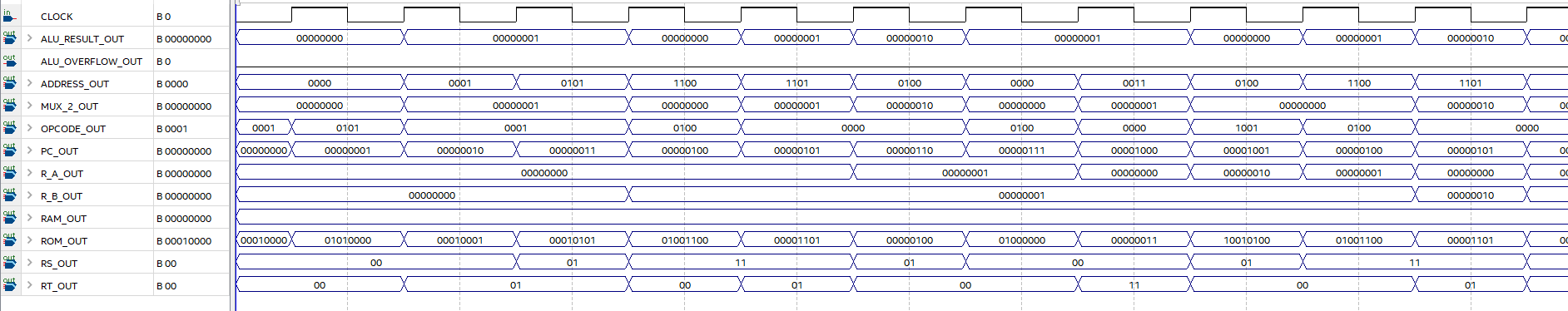


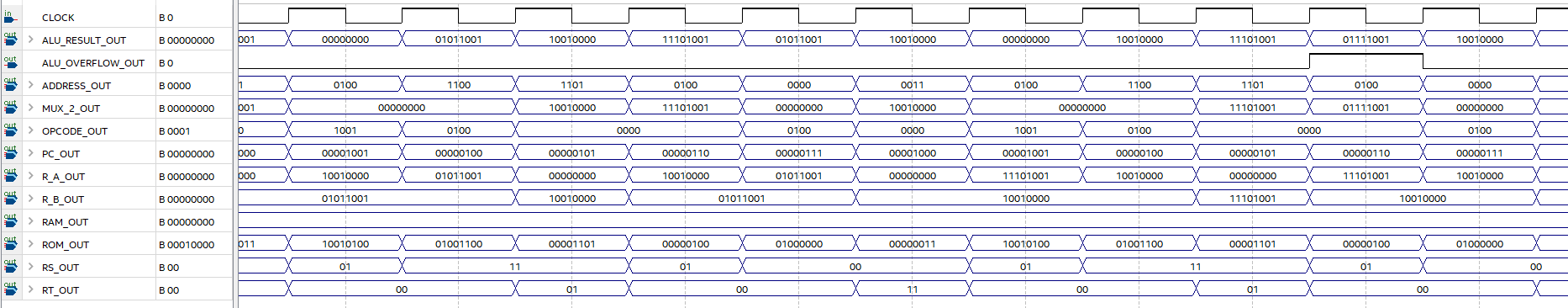
Fonte: Elaborada pelos autores.

* 1. Teste FIBONACCI

# Figuras 27, 28 e 29 – Teste LI







Fonte: Elaborada pelos autores.

1. Considerações Finais

Este trabalho apresentou o projeto e implementação do processador de 8 bits denominado de EK, que foi uma rica oportunidade para pôr em prática o que nos foi ensinado na disciplina de AOC, e esclarecer diversos pontos que antes estavam difíceis de se entender. Uma das maiores dificuldades encontradas foi justamente a de ter que lidar com um baixo número de bits.

No entanto, houveram limitações impostas pelas restrições definidas, tais como: jump apenas consegue pular entre as 16 primeiras linhas de código do programa; um programa pode ter no máximo 256 linhas de código; o processador dispões de apenas 4 registradores.

1. Repositório

<https://github.com/ed-henrique/AOC_Eduardo_Kelvin_UFRR_2022>