



**PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE RORAIMA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO**

ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

RELATÓRIO DO PROJETO: PROCESSADOR TRINITY

ALUNOS:

Rafael da Silva – 2021022827
Vicente Sampaio – 2021007894
William Faray – 2021005577

**Novembro de 2023
Boa Vista/Roraima**



UFRR

**PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE RORAIMA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO**

ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

RELATÓRIO DO PROJETO: PROCESSADOR TRINITY

**Novembro de 2023
Boa Vista/Roraima**

Resumo

Este trabalho aborda o projeto e implementação de um processador de 16 bits, em arquitetura MIPS e RISC (Reduced Instruction Set Computer), denominado Trinity. O desenvolvimento do processador foi por meio da ferramenta Intel Quartus Prime Lite, para utilização dessa ferramenta foi necessária a programação de cada componente na linguagem VHDL (Very High-Speed Integrated Circuit Hardware Description Language), os testes do processador foram realizados por meio de WaveForms, permitindo a visibilidade das entradas e das saídas, e conhecimento de Assembly MIPS para serem gerados os testes corretamente.

Palavras-chave: MIPS, RISC, Quartus Prime, Processador, 16 Bits, VHDL.

Conteúdo

1 Especificação.....	7
1.1 Plataforma de desenvolvimento.....	7
1.2 Conjunto de instruções.....	8
1.3 Descrição do Hardware.....	10
1.3.1 ALU ou ULA.....	10
1.3.2 Banco de Registradores.....	11
1.3.3 Clock.....	11
1.3.4 Unidade de Controle.....	11
1.3.5 Memória de dados.....	14
1.3.6 Memória de Instruções.....	15
1.3.7 Somador.....	15
1.3.8 Somador 16 Bits.....	15
1.3.9 Subtrator 16 Bits.....	16
1.3.10 And.....	17
1.3.11 Mux_2x1.....	17
1.3.12 PC.....	17
1.3.13 Divisor de Instrução.....	18
1.3.14 Branch Helper.....	19
1.3.15 Extensor de Bits 4x16.....	19
1.3.16 ZERO.....	19
1.4 Data Path.....	20
1.5 RTL VIEW.....	21
2 Simulações e Testes.....	22
3 Considerações finais.....	26

Lista de Figuras

Figura 1 - Especificação no Quartus.....	7
Figura 2 - RTL View do componente ULA gerado pelo Quartus.....	10
Figura 3 - RTL View do componente Banco de Registradores gerado pelo Quartus.....	11
Figura 4 - RTL View do componente Unidade de Controle gerado pelo Quartus.....	13
Figura 5 - RTL View do componente Memória de Dados gerado pelo Quartus.....	14
Figura 6 - RTL View do componente Memória de Dados gerado pelo Quartus.....	15
Figura 7 - RTL View do componente Somador gerado pelo Quartus.....	15
Figura 8 - RTL View do componente Somador de 16 Bits gerado pelo Quartus.....	16
Figura 9 - RTL View do componente Subtrator de 16 Bits gerado pelo Quartus.....	16
Figura 10 - RTL View do componente da porta lógica AND gerado pelo Quartus.....	17
Figura 11 - RTL View do componente Multiplexador gerado pelo Quartus.....	17
Figura 12 - RTL View do componente Program Counter gerado pelo Quartus.....	18
Figura 13 - RTL View do componente Divisor de Instrução gerado pelo Quartus.....	18
Figura 14 - RTL View do componente Branch Helper gerado pelo Quartus.....	19
Figura 15 - RTL View do componente Extensor de Bits gerado pelo Quartus.....	19
Figura 16 - RTL View do componente ZERO gerado pelo Quartus.....	20
Figura 17 - Datapath do Processador Trinity de 16 Bits feito em aplicações externas.....	20
Figura 18 - RTL View do Processador Trinity e seus componentes gerado pelo Quartus.	21

Lista de Tabelas

Tabela 1 – Formatação de Instruções Tipo R.....	8
Tabela 2 – Formatação de bits de Instruções Tipo R.....	8
Tabela 3 – Formatação de Instruções Tipo I.....	8
Tabela 4 – Formatação de bits de Instruções Tipo I.....	8
Tabela 5 – Formatação de Instruções Tipo J.....	9
Tabela 6 – Formatação de bits de Instruções Tipo J.....	9
Tabela 7 – Opcode das Instruções Suportadas.....	9
Tabela 8 – Detalhes das flags de controle do processador.....	14
Tabela 9 – Código usando ADD, SUB, LW, e J no processador.....	22
Tabela 10 – Código usando BEQ com teste condicional dando True.....	23
Tabela 11 – Código usando BEQ com teste condicional dando False.....	25

1 Especificação

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

1.1 Plataforma de desenvolvimento

Para a implementação do processador Trinity foi utilizado a IDE:

Flow Status	Successful - Wed Nov 29 19:07:30 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	processor_16bits
Top-level Entity Name	Trinity
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	48
Total pins	146
Total virtual pins	0
Total block memory bits	64
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

Figura 1 - Especificação no Quartus

1.2 Conjunto de instruções

O processador Trinity possui 4 registradores: \$s0, \$s1, \$s2 e \$s3. Assim como 3 formatos de instruções de 16 bits cada, que seriam as instruções do **tipo R, I e J**, 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:** Para alguns tipos de instruções (ex. instruções do tipo R) é o registrador de destino;
- **Reg2:** o registrador contendo o primeiro operando fonte;
- **Reg3:** o registrador contendo o segundo operando fonte;

Tipo de Instruções:

– **Formato do tipo R:** Este formatado aborda instruções baseadas em operações aritméticas.

Formato para escrita de código na linguagem Quantum:

Tipo da Instrução	Reg1	Reg2	Reg3
-------------------	------	------	------

Tabela 1 – Formatação de Instruções Tipo R

Formato para escrita em código binário:

4 bits	4 bits	4 bits	4 bits
15-12	11-8	7-4	3-0
Opcode	Reg3	Reg2	Reg1

Tabela 2 – Formatação de bits de Instruções Tipo R

– **Formato do tipo I:** Este formatado aborda instruções que envolvem valores imediatos, como addi, ou subi.

Formato para escrita de código na linguagem Quantum:

Tipo da Instrução	Reg1	Reg2	Reg3
-------------------	------	------	------

Tabela 3 – Formatação de Instruções Tipo I

Formato para escrita em código binário:

4 bits	4 bits	4 bits	4 bits
15-12	11-8	7-4	3-0
Opcode	Reg3	Reg2	Reg1

Tabela 4 – Formatação de bits de Instruções Tipo I

– **Formato do tipo J:** Este formatado aborda instruções que envolvem saltos condicionais e desvios.

Formato para escrita de código na linguagem Quantum:

Tipo da Instrução	Endereço
-------------------	----------

Tabela 5 – Formatação de Instruções Tipo J

Formato para escrita em código binário:

4 bits	12 bits
15-12	11-0
Opcode	Endereço

Tabela 6 – Formatação de bits de Instruções Tipo J

Visão geral das instruções do Processador Trinity:

O número de bits do campo **Opcode** das instruções é igual a quatro, sendo assim obtemos um total ($Bit (0 e 1)^4 \therefore 2^4 = 16$) de 16 **Opcode (0-15)** que são distribuídos entre as instruções, assim como é apresentado na Tabela 1.

Opcode	Nome	Formato	Breve Descrição	Exemplo
0000	Add	R	Soma de dois operandos	add \$s0, \$s1, \$s2, ou seja, \$s0 <= \$s1 + \$s2
0001	Addi	R	Soma Imediata	Addi \$s0, 1, ou seja, \$s0 <= \$s0 + 1
0010	Sub	R	Subtração de dois operandos	Sub \$s0, \$s1, \$s2, ou seja, \$s0 <= \$s1 - \$s2
0011	Addi	R	Subtração Imediata	Subi \$s0, 1, ou seja, \$s0 <= \$s0 - 1
0100	Lw	I	Carrega valor na memória	Lw \$s0, 0(\$s1)
0101	Sw	I	Armazena valor na memória	Sw \$s0, 0(\$s1)
0110	Li	I	Carregar valor imediato	Li \$s0, 2
0111	BEQ	J	Desvio condicional se operandos iguais	BEQ \$s3, \$s4, L
1000	IF	J	Flag de condição se operandos iguais	IF \$s3, \$s4
1001	J	J	Desvio condicional para endereço especificado	j Exit

Tabela 7 – Opcode das Instruções Suportadas

1.3 Descrição do Hardware

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

1.3.1 ALU ou ULA

O componente ULA (Unidade Lógica Aritmética) tem como principal objetivo efetuar as principais operações aritméticas, dentre elas: soma, subtração, soma imediata e subtração imediata (considerando apenas resultados inteiros). Adicionalmente a ULA efetua operações de comparação de valor como maior ou igual, menor ou igual, igual ou diferente. O componente ULA recebe como entrada três valores: **inA** – dado de 16 bits para operação; **inB** - dado de 16 bits para operação e **AluOp**– identificador da operação que será realizada de 4bits. A ULA também possui três saídas: **zero** – identificador de resultado (2 bits) para comparações (1 se verdade e 0 caso contrário); **overflow** – identificador de overflow caso a operação exceda os 16 bits; e **Sout**– saída com o resultado das operações aritméticas.

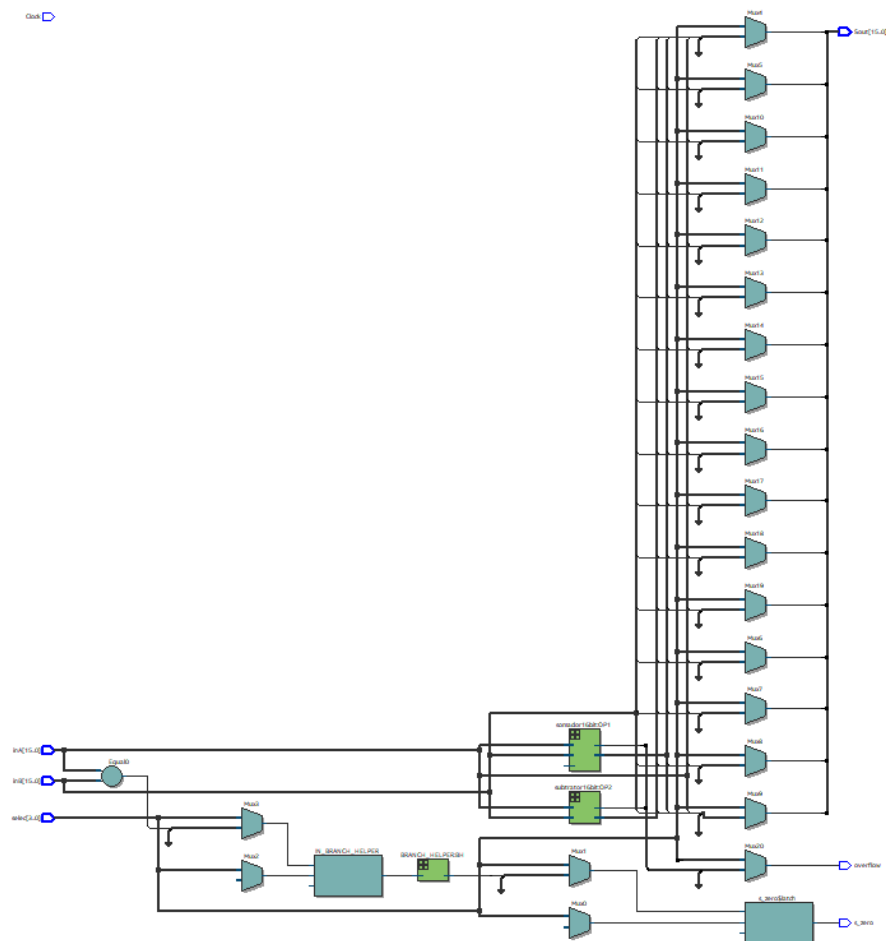


Figura 2 - RTL View do componente ULA gerado pelo Quartus.

1.3.2 Banco de Registradores

O Banco de Registradores é um componente composto por um conjunto de registradores de tamanho adequado para armazenar dados de 16 bits. As entradas incluem um sinal de clock (clock) para sincronização, um sinal de escrita (escreveReg) para indicar a operação de escrita, endereços de registradores (addressReg1, addressReg2, addressReg3) para selecionar os registradores envolvidos nas operações e dados de escrita (escreveDado) para especificar o valor a ser armazenado. As saídas compreendem os valores lidos dos registradores selecionados (out_Reg2, out_Reg3).

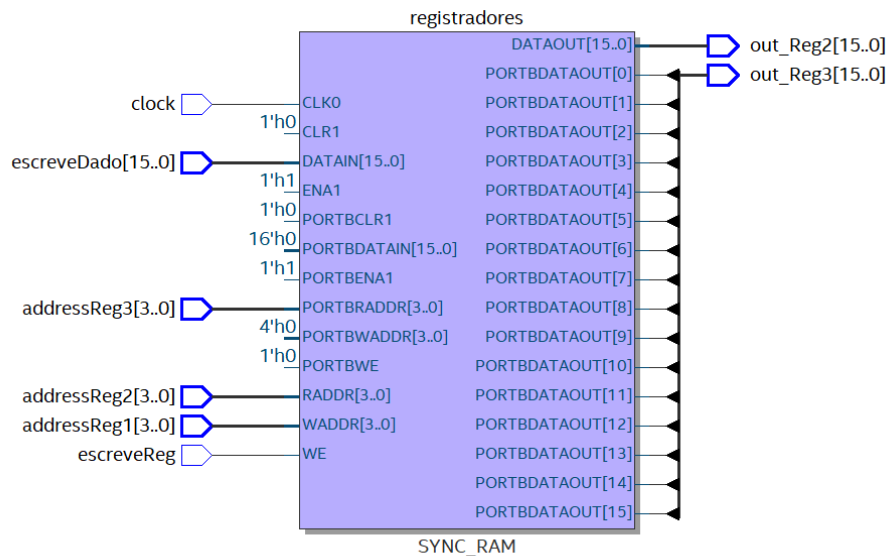


Figura 3 - RTL View do componente Banco de Registradores gerado pelo Quartus.

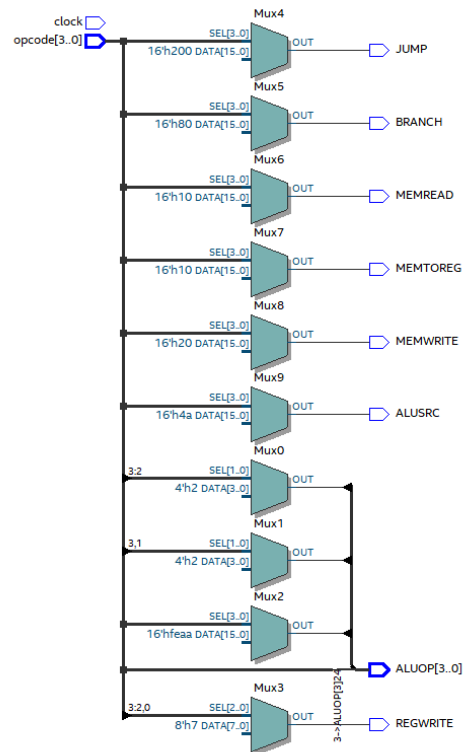
1.3.3 Clock

Um pulso de sincronização que regula o fluxo de operações sequenciais. É utilizado para coordenar e sincronizar eventos dentro do processador, garantindo a execução ordenada de instruções e a correta sincronização de operações entre diferentes componentes do sistema. O flanco de subida do sinal de clock é frequentemente utilizado como referência para desencadear operações, como a leitura ou escrita de dados em registradores, garantindo uma execução controlada e determinística.

1.3.4 Unidade de Controle

A Unidade de Controle é responsável por coordenar e controlar as operações de todos os componentes do processador, com base no opcode da instrução atual. Recebe o sinal de clock para sincronização e o opcode que identifica a natureza da operação a ser executada. Suas saídas incluem sinais de controle essenciais para dirigir o fluxo de dados e operações dentro do processador.

- **JUMP:**
Descrição: Indica se a instrução atual é uma operação de salto (JUMP).
Valores Possíveis: 0 (sem salto) ou 1 (com salto).
- **BRANCH:**
Descrição: Sinaliza se a instrução atual é uma operação de desvio condicional (BRANCH).
Valores Possíveis: 0 (sem desvio) ou 1 (com desvio).
- **MEMREAD:**
Descrição: Indica se a operação atual envolve uma leitura de memória.
Valores Possíveis: 0 (sem leitura) ou 1 (com leitura).
- **MEMTOREG:**
Descrição: Sinaliza se o dado a ser escrito no registrador vem da memória (MEMTOREG) em operações de load.
Valores Possíveis: 0 (não vem da memória) ou 1 (vem da memória).
- **ALUOP:**
Descrição: Especifica a operação a ser realizada pela Unidade Lógica Aritmética (ALU).
Valores Possíveis: Dependem do conjunto de operações suportadas pela ALU.
- **MEMWRITE:**
Descrição: Indica se a operação atual envolve uma escrita na memória.
Valores Possíveis: 0 (sem escrita) ou 1 (com escrita).
- **ALUSRC:**
Descrição: Determina a fonte dos dados de entrada para a ALU (ALUSRC).
Valores Possíveis: 0 (dados do registrador) ou 1 (constante/imediato).
- **REGWRITE:**
Descrição: Indica se a operação atual envolve a escrita de dados em um registrador.
Valores Possíveis: 0 (sem escrita) ou 1 (com escrita).



*Figura 4 - RTL View do componente
Unidade de Controle gerado pelo
Quartus.*

Comando	AluOp	Reg Write	JumP	Branch	Mem Read	MemTo Reg	Mem Write	AluSRC
add	0000	1	0	0	0	0	0	0
addi	0001	1	0	0	0	0	0	1
sub	0010	1	0	0	0	0	0	0
subi	0011	1	0	0	1	1	0	0
lw	0100	1	0	0	1	1	0	0

sw	0101	0	0	0	0	0	1	0
Li	0110	1	0	0	0	0	0	1
BEQ	0111	0	0	1	0	0	0	0
IF_OP	1000	0	0	0	0	0	0	0
J	1001	0	1	0	0	0	0	0

Tabela 8 – Detalhes das flags de controle do processador.

1.3.5 Memória de dados

Essa memória é responsável por armazenar e recuperar dados durante as operações do processador. As entradas incluem um sinal de clock (clock) para sincronização, flags de escrita (mem_write) e leitura (mem_read) para controlar operações de escrita e leitura, respectivamente. Além disso, há um barramento de entrada (in_A) para o valor a ser escrito na memória, um barramento de endereço (addr) para especificar a localização da memória, e as saídas (Sout) que fornecem os dados lidos da memória. Essa memória desempenha um papel crucial na manipulação eficiente de dados durante a execução das instruções do processador.

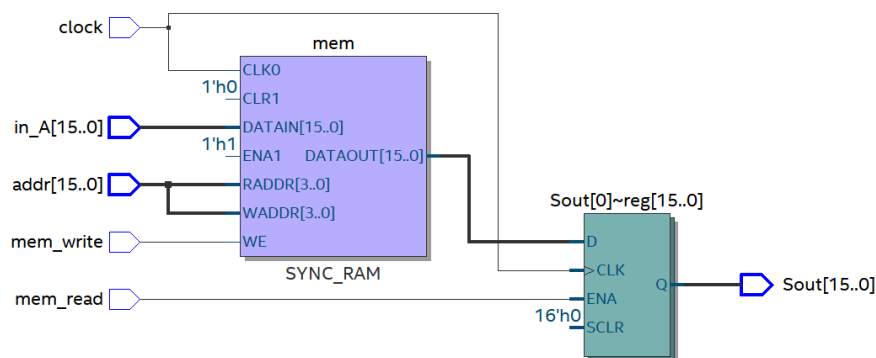


Figura 5 - RTL View do componente Memória de Dados gerado pelo Quartus.

1.3.6 Memória de Instruções

Sua função é armazenar as instruções do programa e disponibilizá-las para a execução pelo processador. As entradas incluem um barramento de entrada (in_data) para dados de configuração, um sinal de clock (clock) para sincronização, e as saídas (out_data) que fornecem as instruções lidas da memória. A Memória de Instruções desempenha um papel fundamental ao disponibilizar as instruções sequenciais necessárias para a execução correta do programa no processador.

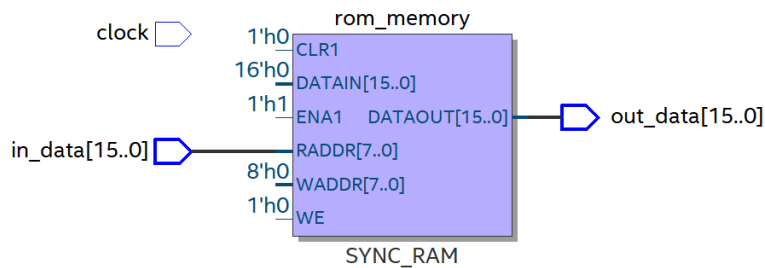


Figura 6 - RTL View do componente Memória de Dados gerado pelo Quartus.

1.3.7 Somador

Sua função é realizar a adição de dois operandos de 16 bits. As entradas incluem um sinal de clock (clock) para sincronização, e um barramento de entrada (inPort) que contém os operandos a serem somados. A saída (outPort) fornece o resultado da adição, também representado por um vetor de 16 bits. O somador é crucial para a execução de operações aritméticas no processador, contribuindo para a realização de cálculos e manipulação de dados.

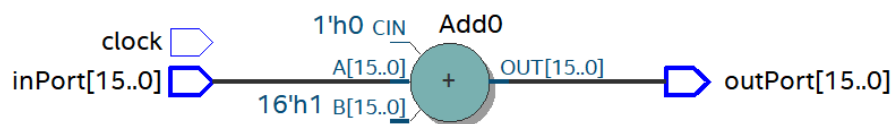


Figura 7 - RTL View do componente Somador gerado pelo Quartus.

1.3.8 Somador 16 Bits

O somador de 16 bits integrado à Unidade Lógica Aritmética (ULA) possui a função de realizar a adição de dois operandos de 16 bits. As entradas incluem dois barramentos de entrada (A e B) que representam os operandos a serem somados, um sinal de entrada (CIN) que indica a entrada de transporte inicial, e as saídas que

consistem em um sinal de saída de transporte (COUT) e o resultado da adição (S). Este somador dentro da ULA é essencial para a execução de operações aritméticas e lógicas, contribuindo para a manipulação de dados durante a execução de instruções no processador.

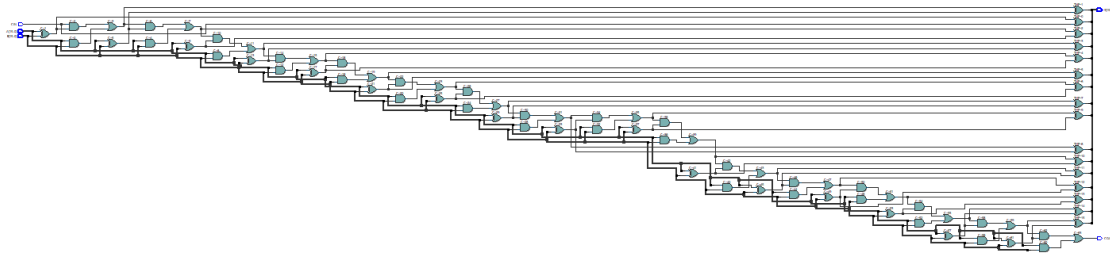


Figura 8 - RTL View do componente Somador de 16 Bits gerado pelo Quartus.

1.3.9 Subtrator 16 Bits

Sua função é realizar a operação de subtração entre dois operandos de 16 bits (A e B). As entradas incluem os operandos A e B, e a saída (S) fornece o resultado da subtração. Além disso, há uma saída de empréstimo (COUT), indicando se houve um empréstimo durante a operação de subtração. O subtrator utiliza um somador de 16 bits interno (somador16bit) para realizar a operação de subtração, onde o operando B é complementado e um bit de empréstimo é adicionado. O complemento de B é calculado de forma bit a bit utilizando portas XOR. O resultado da subtração é obtido através do somador interno, e o empréstimo é indicado pelo sinal COUT.

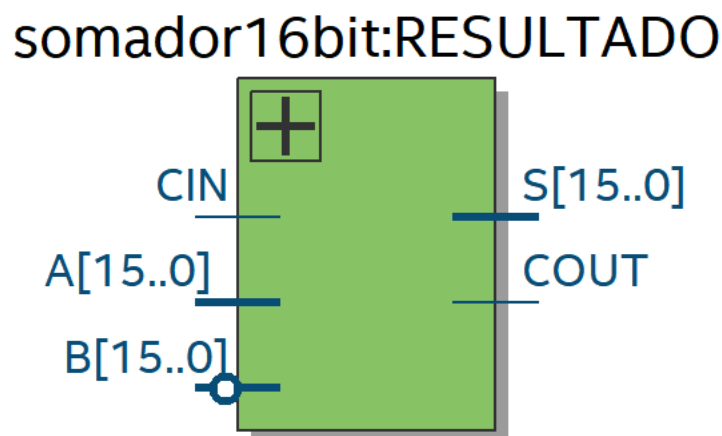


Figura 9 - RTL View do componente Subtrator de 16 Bits gerado pelo Quartus.

1.3.10 And

Esta porta lógica realiza a operação lógica AND entre dois sinais de entrada (inand1 e inand2). O resultado da operação AND é fornecido na saída (saida).

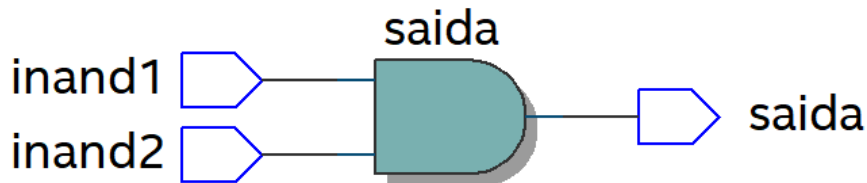


Figura 10 - RTL View do componente da porta lógica AND gerado pelo Quartus.

1.3.11 Mux_2x1

A entidade multiplexador representa um multiplexador 2x1 em um circuito digital. Sua função é seleccionar um dos dois sinais de entrada, A ou B, com base no sinal de controle SEL. Quando SEL é '0', o multiplexador direciona o sinal de entrada A para a saída Sout; quando SEL é '1', direciona o sinal de entrada B.

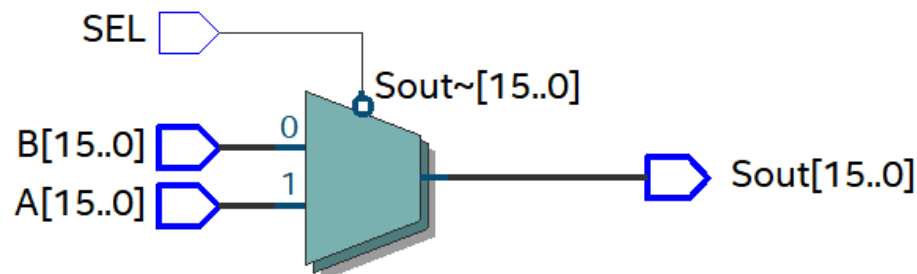


Figura 11 - RTL View do componente Multiplexador gerado pelo Quartus.

1.3.12 PC

Sua função é armazenar o endereço da próxima instrução a ser executada. As entradas incluem um sinal de clock (clock), utilizado para sincronização, e um barramento de entrada (inPort) que pode conter um novo valor para o Program Counter, por exemplo, durante operações de desvio. A saída (outPort) fornece o endereço atual do Program Counter, permitindo que seja utilizado para buscar a próxima instrução na memória de instruções.

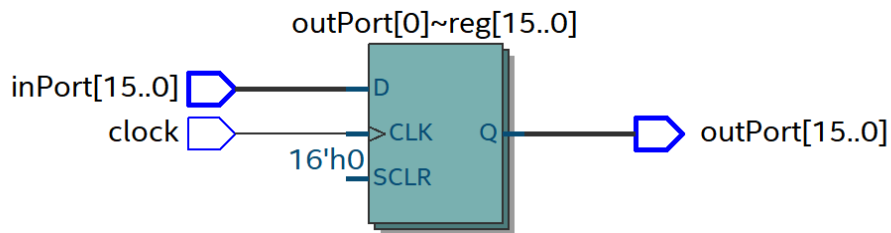


Figura 12 - RTL View do componente Program Counter gerado pelo Quartus.

1.3.13 Divisor de Instrução

Sua função é dividir uma instrução representada por um vetor de bits (in_port) em partes distintas, tais como opcode, registradores (rs, rt, rd), e, possivelmente, um endereço de salto (jump). As saídas incluem:

out_op_code: Um vetor de 4 bits que representa o código de operação (opcode) da instrução.

out_rs: Um vetor de 4 bits que representa um dos registradores de origem (rs) da instrução.

out_rt: Um vetor de 4 bits que representa um dos registradores de origem (rt) da instrução.

out_rd: Um vetor de 4 bits que representa o registrador de destino (rd) da instrução.

out_jump: Um vetor de 16 bits que representa o endereço de salto (jump) se aplicável.

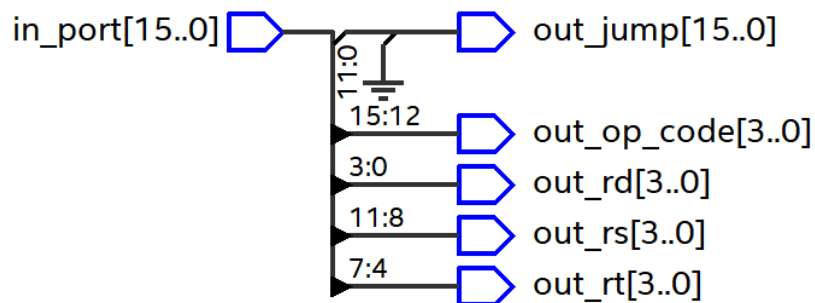


Figura 13 - RTL View do componente Divisor de Instrução gerado pelo Quartus.

1.3.14 Branch Helper

Sua função é ajudar na avaliação de condições de desvio, geralmente utilizadas em instruções de branch. As entradas e saídas incluem:

A: Um sinal de entrada que pode representar uma condição de desvio.

S: Um sinal de saída que indica se a condição de desvio foi satisfeita.

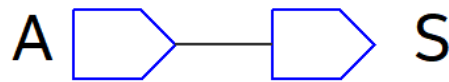


Figura 14 - RTL View do componente Branch Helper gerado pelo Quartus.

1.3.15 Extensor de Bits 4x16

Sua função é estender o valor de entrada de 4 bits (`in_data`) para um vetor de 16 bits (`s`). O componente realiza essa extensão adicionando zeros à esquerda do valor original.

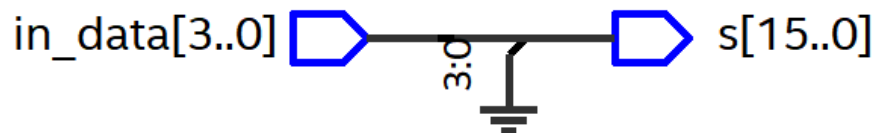


Figura 15 - RTL View do componente Extensor de Bits gerado pelo Quartus.

1.3.16 ZERO

A entidade zero representa um componente que avalia se um sinal de entrada (`in_port`) é igual a zero. O objetivo desse componente é gerar um sinal de saída (`out_port`) que indica se o valor de entrada é zero ou não. Se o valor de entrada for zero, o sinal de saída será '1'; caso contrário, será '0'. Este componente é útil em lógica digital para condições de igualdade, como nos `beq` ou `bne`.

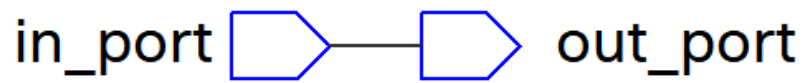


Figura 16 - RTL View do componente ZERO gerado pelo Quartus.

1.4 Data Path

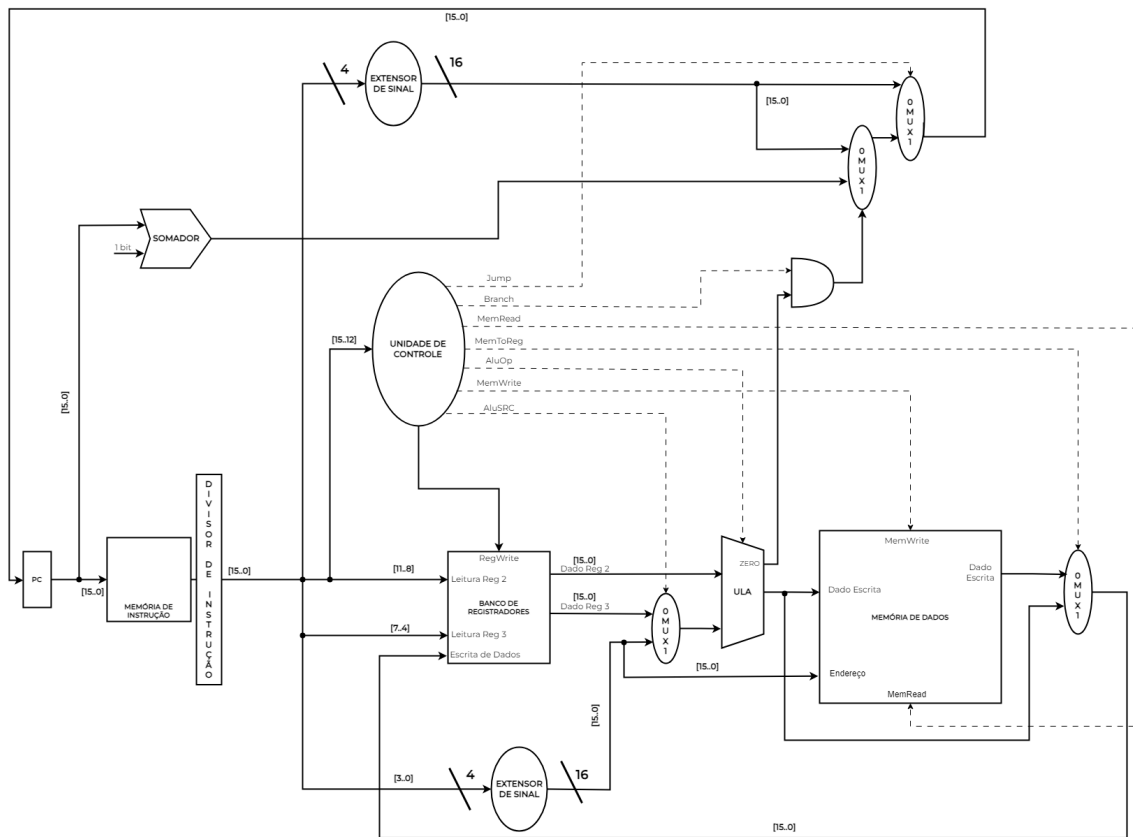


Figura 17 - Datapath do Processador Trinity de 16 Bits feito em aplicações externas

1.5 RTL VIEW

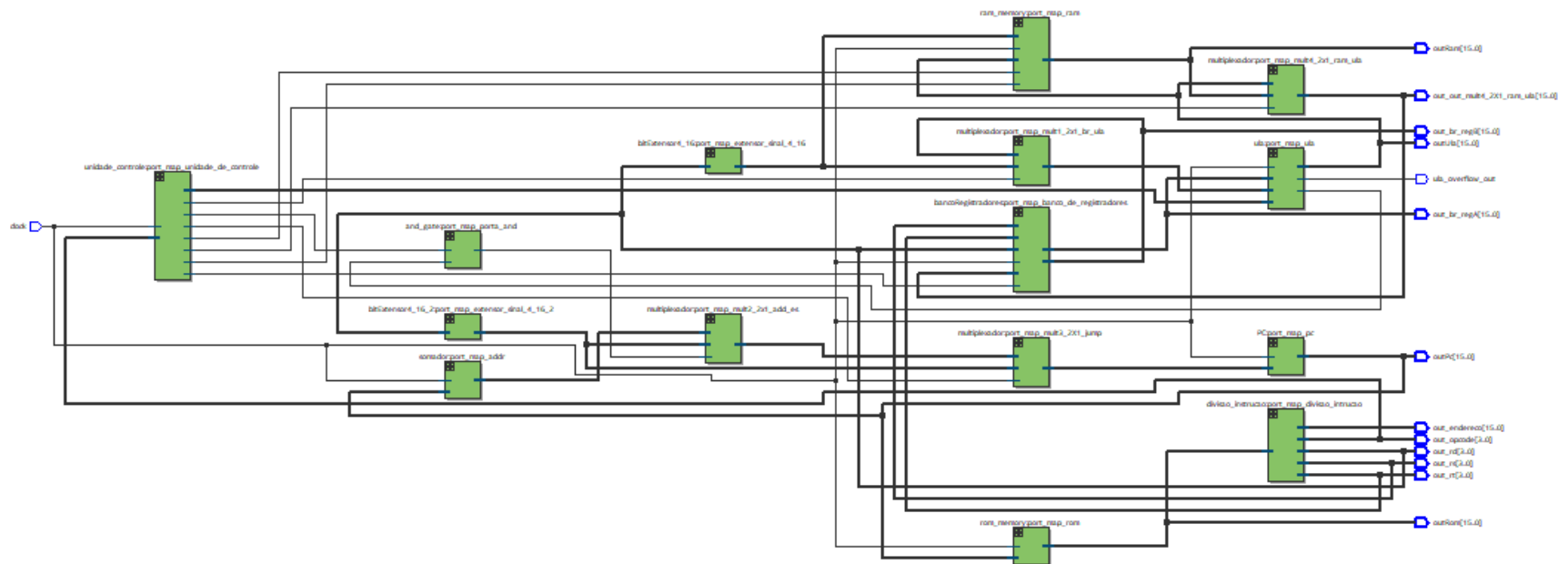


Figura 18 - RTL View do Processador Trinity e seus componentes gerado pelo Quartus.

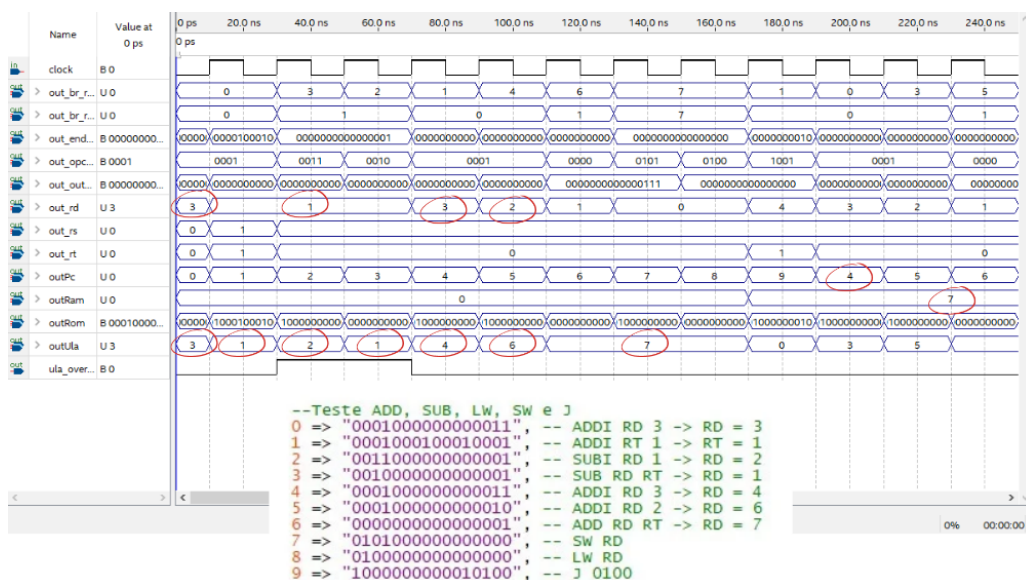
2 Simulações e Testes

Objetivando analisar e verificar o funcionamento do processador, efetuamos alguns testes analisando cada componente do processador em específico, em seguida efetuamos testes de cada instrução que o processador implementa. Para demonstrar o funcionamento do processador Trinity utilizaremos como exemplo o código que usasse várias instruções e diversos componentes. Além de dois códigos usando BEQ tendo diferentes finais, se verdadeiro e se falso.

Tabela 9 – Código usando ADD, SUB, LW, e J no processador.

Teste ADD, SUB, LW, e J					
Endereço	Linguagem de alto nível	Binario			
		opcode	Reg 3	Reg 2	Reg1
			Endereço		
		Dados			
0	ADDI \$s0, 3	0001	0000	0000	0011
1	ADDI \$s1, 1	0001	0001	0001	0001
2	SUBI \$s0, 1	0011	0000	0000	0001
3	SUB \$s0, 1	0010	0000	0000	0011
4	ADDI \$s0, 3	0001	0000	0000	0010
5	ADDI \$s0, 2	0001	0000	0000	0010
6	ADDI \$s0, \$s1	0000	0000	0000	0001
7	SW \$s0	0101	0000	0000	0000
8	LW \$s0	0100	0000	0000	0000
9	J 0100	1000	0000	0001	0100

Verificação dos resultados no relatório da simulação: Após a compilação e execução da simulação, o seguinte relatório é exibido.

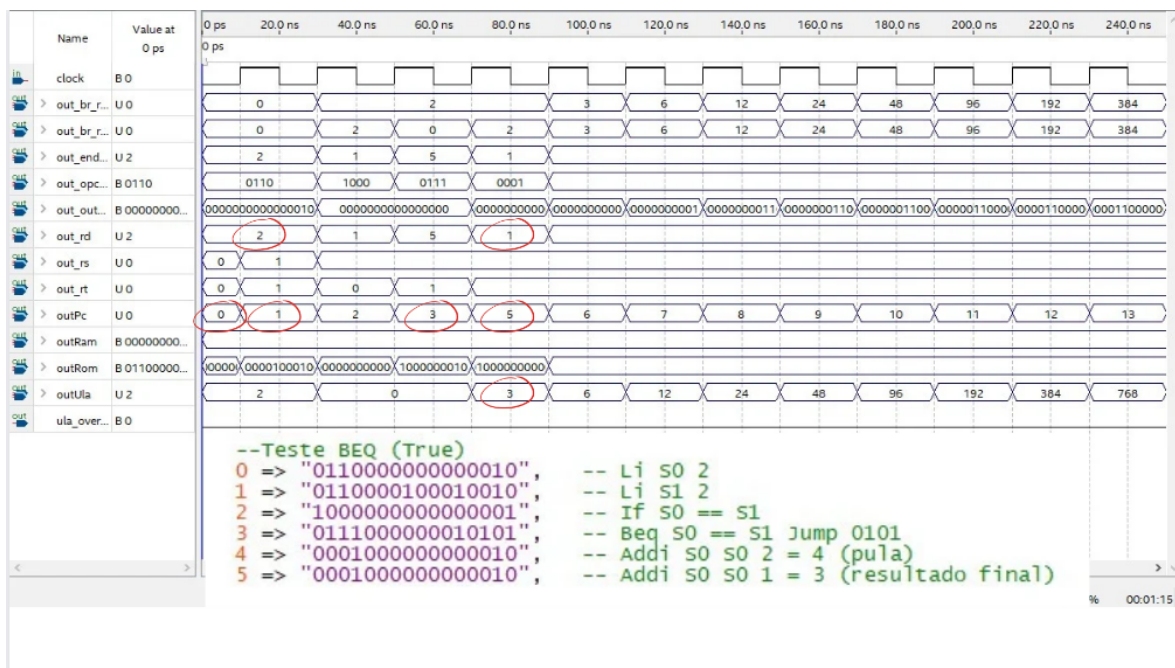


Podemos verificar nessa WaveForm os valores atribuídos nos registradores fonte que são depois colocados no registrador no destino, assim como os resultados das operações na saída da ULA (Unidade Lógica e Aritmética), e o Jump reiniciando o PC (Program Counter) de 9 para o valor de 4, e realizando um Loop a partir dessa instrução.

Tabela 10 – Código usando BEQ com teste condicional dando True.

Teste BEQ (True)					
Endereço	Linguagem de alto nível	Binario			
		opcode	Reg 3	Reg 2	Reg1
			Endereço		
Dados					
0	Li S0 2	0110	0000	0000	0010
1	Li S1 2	0110	0001	0001	0010
2	If S0 == S1	1000	0000	0000	0001
3	Beq S0 == S1 jump 0101	0111	0000	0001	0101
4	Addi \$s0, 2 = 4 # pula	0001	0000	0000	0010
5	Addi \$s0, 1 = 33 # resultado final	0001	0000	0000	0010

Verificação dos resultados no relatório da simulação: Após a compilação e execução da simulação, o seguinte relatório é exibido.

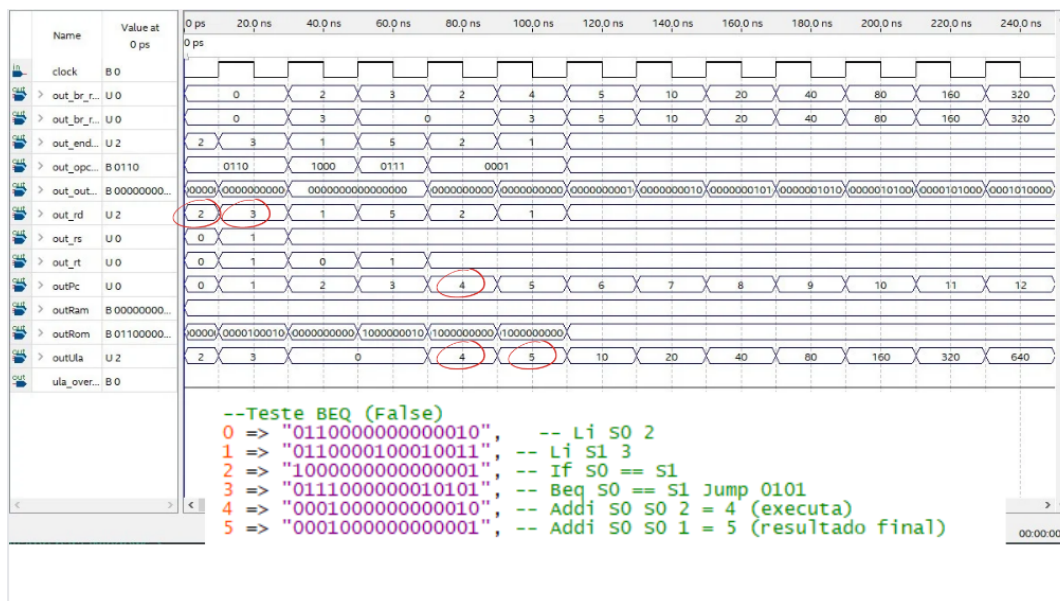


Podemos verificar nessa WaveForm os valores atribuídos nos registradores por meio do Load Immediate, e o teste de comparação usando a instrução If e a instrução BEQ, assim como o PC (Program Counter) pulando a instrução de valor 4, de acordo com a instrução de Jump. Por fim temos o valor final na ULA (Unidade Lógica e Aritmética) de acordo com a nova instrução no PC.

Tabela 11 – Código usando BEQ com teste condicional dando False.

Teste BEQ (False)					
Endereço	Linguagem de alto nível	Binario			
		opcode	Reg 3	Reg 2	Reg1
			Endereço		
		Dados			
0	Li \$s0, 2	0110	0000	0000	0010
1	Li \$s1, 3	0110	0001	0001	0010
2	If S0 == S1	1000	0000	0000	0001
3	Beq S0 == S1 jump 0101	0111	0000	0001	0101
4	Addi \$s0, 2 = 4 # executa	0001	0000	0000	0010
5	Addi \$s0, 1 = 5 # resultado final	0001	0000	0000	0010

Verificação dos resultados no relatório da simulação: Após a compilação e execução da simulação, o seguinte relatório é exibido.



Assim como o último teste, temos os valores atribuídos nos registradores por meio do Load Immediate, e o teste de comparação usando a instrução If e a instrução BEQ, só que dessa vez o teste retorna False, e assim o PC (Program Counter) executa a instrução de valor 4. Por fim temos o valor final na ULA (Unidade Lógica e Aritmética) de acordo com a sequência das instruções 4 e 5.

3 Considerações finais

Neste projeto, desenvolvemos e implementamos o processador Trinity, de 16 bits, e destacamos nesse relatório os principais componentes que o compõem e seu funcionamento primário. Cada bloco funcional, como o Banco de Registradores, a Unidade Lógica Aritmética (ULA), as Memórias (RAM e ROM), e as unidades de controle, foi projetado para garantir um funcionamento eficiente e coerente com aquilo que definimos como a arquitetura do nosso processador.

Ao longo do desenvolvimento, aplicamos conceitos teóricos aprendidos em sala de aula para criar um processador funcional e capaz de executar instruções em linguagem de máquina. O processo envolveu o projeto detalhado de cada componente, a escolha de arquiteturas apropriadas, e a implementação utilizando a linguagem de descrição de hardware VHDL.

Por fim, este trabalho serviu como uma plataforma para explorar e aplicar os conhecimentos teóricos da disciplina, destacando a importância da arquitetura de computadores no desenvolvimento de sistemas computacionais eficientes e funcionais.