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

ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

RELATÓRIO DO PROJETO: PROCESSADOR SPLIT

ALUNOS:

João Paulo Ramos Ribeiro Nascimento - 2018022315
Karen Giovanna Furtado Melo - 2018022217

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 SPLIT

Maio de 2021
Boa Vista/Roraima

Resumo

Este trabalho aborda a elaboração e implementação do processador uniciclo de 8 bits denominado Split baseado na arquitetura do processador MIPS. Programado na linguagem VHDL (VHSIC Hardware Description) que é amplamente utilizada para design de circuitos digitais em CPLDs, FPGAs e ASICs e tendo sua implementação feita no programa Quartus Prime Lite, o processador conta com cinco unidades funcionais, tais quais Program Counter, Memória de Instruções (também conhecida como memória ROM), Banco de Registradores, Unidade Lógica Aritmética e Memória de Dados. Em vista disso, durante a utilização destes componentes o processador Split tem capacidade de executar 9 instruções, algumas delas são instruções de comparação (Branch If Equal, Branch If Not Equal), salto incondicional (Jump) e instruções aritméticas(Adição, Subtração, Multiplicação). Os testes de componentes foram feitos utilizando o simulador ModelSim Altera através das Waveform, demonstrando o funcionamento de todas as entradas e saídas de cada componente.

Palavras-chave: MIPS, Instruções, Quartus Prime Lite, Processador, Unidade Funcional

Conteúdo

1. Especificação **7**
    1.1 Plataforma de desenvolvimento 7
    1.2 Conjunto de instruções 7
        1.2.1 Tipo de Instruções 8
    1.3 Descrição do Hardware 9
        1.3.1 Program Counter (PC) 9
        1.3.2 Somador (ADD) 10
        1.3.3 Memória de Instruções 10
        1.3.4 Extensor de Sinal 11
            1.3.4.1 Extensor de Sinal 2 bits para 8 bits 11
            1.3.4.2 Extensor de Sinal 4 bits para 8 bits 12
        1.3.5 Divisor de Instruções 12
        1.3.6 Unidade de Controle 13
        1.3.7 Banco de Registradores 14
        1.3.8 Unidade Lógica Aritmética (ULA) 15
        1.3.9 Multiplexadores 16
            1.3.9.1 Multiplexador ula_src 16
            1.3.9.2 Multiplexador Branch AND 17
            1.3.9.3 Multiplexador Jump 18
        1.3.10 AND 18
        1.3.11 Memória de Dados 19
    1.4 Datapath 21
2. Simulação e Testes 22
    2.1 Algoritmo um 22
    2.2 Algoritmo dois 24
3. Considerações finais 26

Lista de Figuras

Figura 1 - Especificações no Quartus 7
Figura 2 - Bloco simbólico do Program Counter gerado pelo Quartus 10
Figura 3 - Bloco simbólico do somador(add) gerado pelo Quartus 10
Figura 4 - Bloco simbólico do memória de instruções gerado pelo Quartus 11
Figura 5 - Bloco simbólico do extensor de sinal de 2|8 bits gerado pelo Quartus 11
Figura 6 - Bloco simbólico do extensor de sinal de 4|8 bits gerado pelo Quartus 12
Figura 7 - Bloco simbólico do Unidade de Controle gerado pelo Quartus 14
Figura 8 - Bloco simbólico do Banco de Registradores gerado pelo Quartus 15
Figura 9 - Bloco simbólico do Unidade Lógica Aritmética gerado pelo Quartus 16
Figura 10 - Bloco simbólico do Multiplexador ula_src gerado pelo Quartus 17
Figura 11 - Bloco simbólico do Multiplexador Branch and gerado pelo Quartus 17
Figura 12 - Bloco simbólico do Multiplexador Jump gerado pelo Quartus 18
Figura 13 - Bloco simbólico do AND gerado pelo Quartus 18
Figura 14 - Bloco simbólico do Memória de Dados gerado pelo Quartus 20
Figura 15 - Datapath gerado pelo Quartus 21
Figura 16 - Resultado do Algoritmo Um na waveform. 23
Figura 17 - Resultado do Algoritmo Um na waveform. 23
Figura 18 - Resultado do Algoritmo Um na waveform. 24
Figura 19 - Resultado do Algoritmo Dois na waveform. 25
Figura 20 - Resultado do Algoritmo Dois na waveform. 25

Lista de Tabelas

Tabela 1 – Lista de operações do tipo R utilizadas pelo processador Split. 8
Tabela 2 - Lista de operações do tipo I utilizadas pelo processador Split. 8
Tabela 3 - Lista de operações do tipo J utilizadas pelo processador Split. 8
Tabela 4 - Lista de Opcodes utilizadas pelo processador Split. 9
Tabela 5 - Detalhes das flags de controle do processador. 13
Tabela 6 - Código Um para demonstração de instruções do processador Split. 22
Tabela 7 - Código Dois para demonstração de instruções do processador Split. 24

1 Especificação

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

1.1 Plataforma de desenvolvimento

Para a implementação do processador Split foi utilizado a IDE Visual Studio Code que é um editor de código-fonte desenvolvido pela Microsoft. Ele inclui suporte para depuração, controle Git incorporado, realce de sintaxe, complementação inteligente de código, snippets e refatoração de código.

Flow Status	Successful - Mon May 10 17:19:44 2021
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	split
Top-level Entity Name	split
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	80
Total pins	49
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	1
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 - Especificações no Quartus

Para a implementação do processador Split também foi utilizado a IDE: Quartus Prime Lite, versão 20.1 e o simulador ModelSim Altera.

1.2 Conjunto de instruções

O processador Split possui quatro registradores: S0, S1, S2, S3. Assim como três formatos de instruções de 8 bits cada, 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: 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;

1.2.1 Tipo de Instruções:

- Tipo R: este formato aborda as instruções baseadas em operações aritméticas e lógicas, soma, subtração, multiplicação.
- Tipo I: este formato aborda as instruções baseadas em operações com valores imediatos, desvios condicionais e operações relacionadas à memória, BEQ, BNE, Load e Store.
- Tipo J: este formato aborda as instruções de desvios incondicionais, exemplo o Jump.

Operações do Tipo R		
Opcode	Reg1	Reg2
4 bits	2 bits	2 bits
7 - 4	3 - 2	1 - 0

Tabela 1 - Lista de descrição de operações do tipo R utilizadas pelo processador Split

Operações do Tipo I		
Opcode	Reg1	Reg2
4 bits	2 bits	2 bits
7 - 4	3 - 2	1 - 0

Tabela 2 - Lista de descrição de operações do tipo I utilizadas pelo processador Split

Operações do Tipo J	
Opcode	Endereço
4 bits	4 bits
7 - 4	3 - 0

Tabela 3 - Lista de descrição de operações do tipo J utilizadas pelo processador Split

Visão geral das instruções do Processador Split:

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, porém, implementamos apenas nove, assim como é apresentado na Tabela 2.

Opcode	Nome	Formato	Breve Descrição	Exemplo
0000	add	R	Soma	add $s0 $s1
0001	sub	R	Subtração	sub $s0 $s1
0010	mult	R	Multiplicação	mult $s0 $s1
0011	lw	I	Load Word	lw $s0 (memória00)
0100	sw	I	Store Word	sw $s0 (memória00)
0101	li	I	Load Immediate	li $s0 1
0110	beq	J	Desvio Condicional	beq endereço
0111	bne	J	Desvio Condicional	bne endereço
1000	J	J	Desvio Incondicional	J endereço(0000)
1001	If	R	Comparação para salto	if $s0 $s1 if $s0 == $s1

Tabela 4 - Lista de Opcodes utilizadas pelo processador Split

1.3 Descrição do Hardware

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

1.3.1 Program Counter (PC)

O contador de programa, Program Counter, é um registro de propósito especial usado pelo processador para armazenar o endereço de 8 bits da próxima instrução a ser executada. Ele recebe a entrada de um bit chamado clock que indica se a unidade está ligada ou não e a entrada pc que é uma instrução de 8 bits.

Duas entradas:

- clock_pc: 1 bit que indica se o componente está ligado ou desligado
- entrada_pc: indica a instrução a ser executada e possui 8 bits

Uma saída:

- saida_pc: sai uma instrução (ou do tipo R ou J ou I) de 8 bits

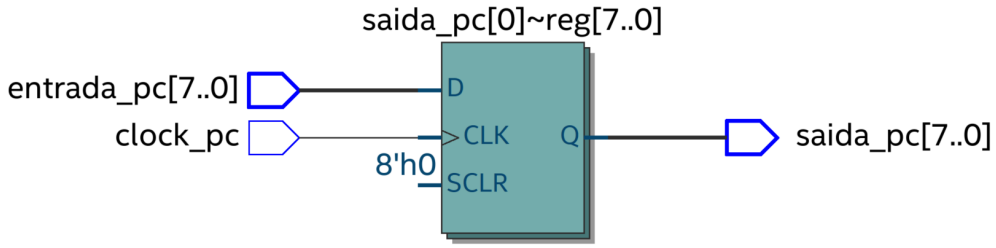

Figura 2 - Componente PC gerado pelo Quartus

1.3.2 Somador (ADD)

O componente somador (ou add) será utilizado em conjunto ao Program Counter pois sua única função é adicionar uma unidade de bit à instrução atual permitindo que o PC receba a instrução seguinte da memória.

Uma entrada:

- saida_pc: a única entrada do componente somador (add) é a instrução de 8 bits que está no program counter

Uma saída:

- saida_add: a única saída do somador é uma instrução de 8 bits, sendo a instrução seguinte da que entrou pelo saída_pc

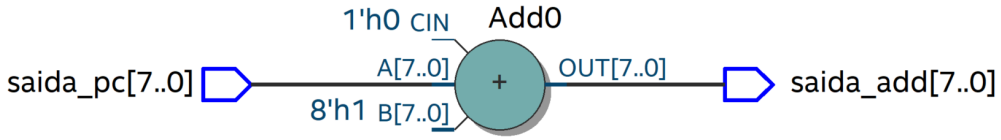

Figura 3 - Componente ADD gerado pelo Quartus

1.3.3 Memória de Instruções

A memória de instruções fornece dados apenas para leitura e é nela onde fica armazenado o programa que será executado pelo processador. No processador Split, existem duas entradas:

Entradas:

- clock_mem: Recebe o sinal com a borda alta para ativar o componente.
- entrada_mem: Recebe um endereço de 8 bits do componente PC (Program Counter) que será enviado para a execução.

Uma saída:

- saida_mem: Uma saída de 8 bits da instrução que será executada.

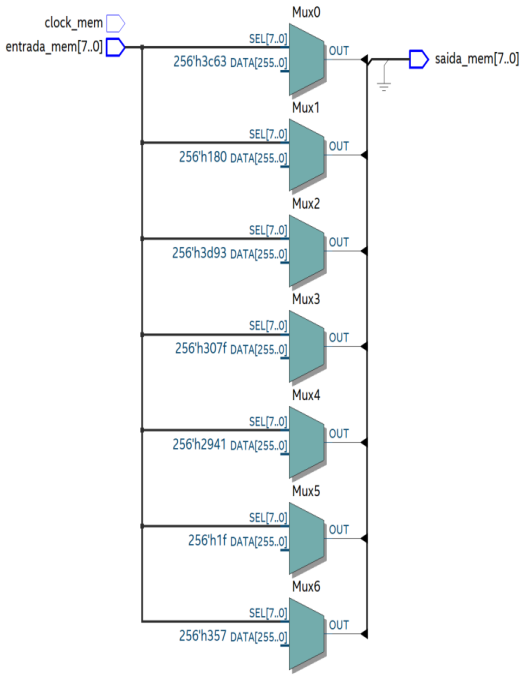

Figura 4 - Componente Memória de Instruções gerado pelo Quartus

1.3.4 Extensor de Sinal

O componente Extensor de Sinal, também conhecido como Extensor de Bits, tem como comportamento transformar um valor em outro de largura diferente. Os bits menos significativos serão preenchidos com zero e os bits de mais alta ordem serão os mesmos.

1.3.4.1 Extensor de Sinal 2 bits para 8 bits

Esse componente é utilizado para estender a palavra do segundo registrador da Memória de Instrução de 2 bits para 8 bits. E o seu resultado (palavra de 8 bits) é utilizado para operações na Unidade Lógica Aritmética(ULA) e como endereço na Memória de Dados(RAM).

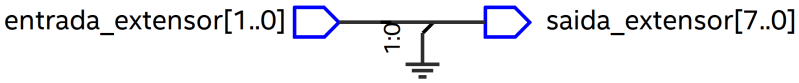

Figura 5 - Componente Extensor de Sinal de 2 bits para 8 bits gerado pelo Quartus

1.3.4.2 Extensor de Sinal 4 bits para 8 bits

Esse componente é responsável por estender o OpCode de 4 bits vindo da Memória de Instrução para 8 bits. Seu resultado será utilizado para saltos condicionais e incondicionais.

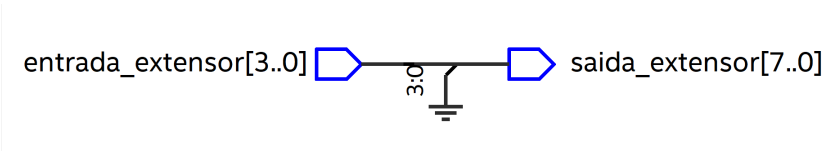

Figura 6 - Componente Extensor de Sinal de 4 bits para 8 bits gerado pelo Quartus

1.3.5 Divisor de Instruções

O componente Divisor de Instruções é responsável por dividir os 8 bits recebidos da Memória de Instrução para cinco saídas.

Após o recebimento da instrução, os primeiros 4 bits à esquerda correspondentes ao OpCode da instrução serão direcionados à Unidade de Controle. Os 2 bits correspondentes ao primeiro registrador serão enviados para primeira entrada do Banco de Registradores e para o Extensor de Sinal 4|8. Os dois bits correspondentes ao segundo registrador serão enviados para a segunda entrada do Banco de Registradores e para o Extensor de Sinal 2|8.

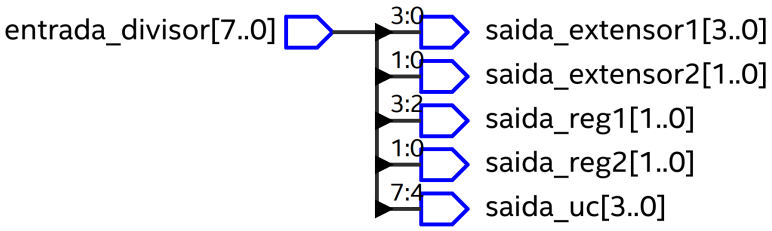

Figura 7 - Componente Divisor de Instruções gerado pelo Quartus

1.3.6 Unidade de Controle

O componente Unidade de Controle tem como objetivo realizar o controle de todos os componentes do processador de acordo com o recebimento de um opcode de 4 bits. A sua entrada (de 4 bits) especifica se a instrução é do

tipo add, sub, mult, lw, sw, li, beq, bne ou J. Para cada instrução, cada flag terá valores específicos. As flags são:

jump: se esta flag estiver ativada (ou seja, passando valor 1) e as demais flags de 1 bit estiverem desligadas, vai ocorrer um desvio incondicional.
branch: se esta flag estiver ativada (ou seja, passando valor 1) e as demais flags de 1 bit estiverem desligadas, vai ocorrer um desvio condicional.
M_read: é um sinal de 1 bit para decidir se será lido um dado da memória RAM.
M_to_reg: sinal de 1 bit para decidir se o dado que será escrito no banco de registradores vai ser enviado pela ULA ou pela memória RAM.
ula_op: sinal de 4 bits que será enviado para a ULA decidir qual operação será realizada
M_write: sinal de 1 bit para decidir se será escrito um dado da memória RAM.
ula_src: sinal de 1 bit para decidir se o dado que entrará na ULA vai ser enviado pelo banco de registradores ou pelo extensor de sinal.
reg_write: sinal de 1 bit para decidir se o banco de registradores vai escrever um dado na posição do registrador de destino.

Abaixo segue a tabela, onde é feita a associação entre os opcodes e as flags de controle:

Comando	Jump	Branch	M_read	M_to_reg	Ula_op	M_write	Ula_src	Reg_write
add	0	0	0	0	0000	0	0	1
sub	0	0	0	0	0001	0	0	1
mult	0	0	0	0	0010	0	0	1
lw	0	0	1	1	0011	0	0	1
sw	0	0	0	0	0100	1	0	0
li	0	0	0	0	0101	0	1	1
beq	0	1	0	0	0110	0	0	0
bne	0	1	0	0	0111	0	0	0
J	1	0	0	0	1000	0	0	0

Tabela 5 - Detalhes das flags de controle do processador.

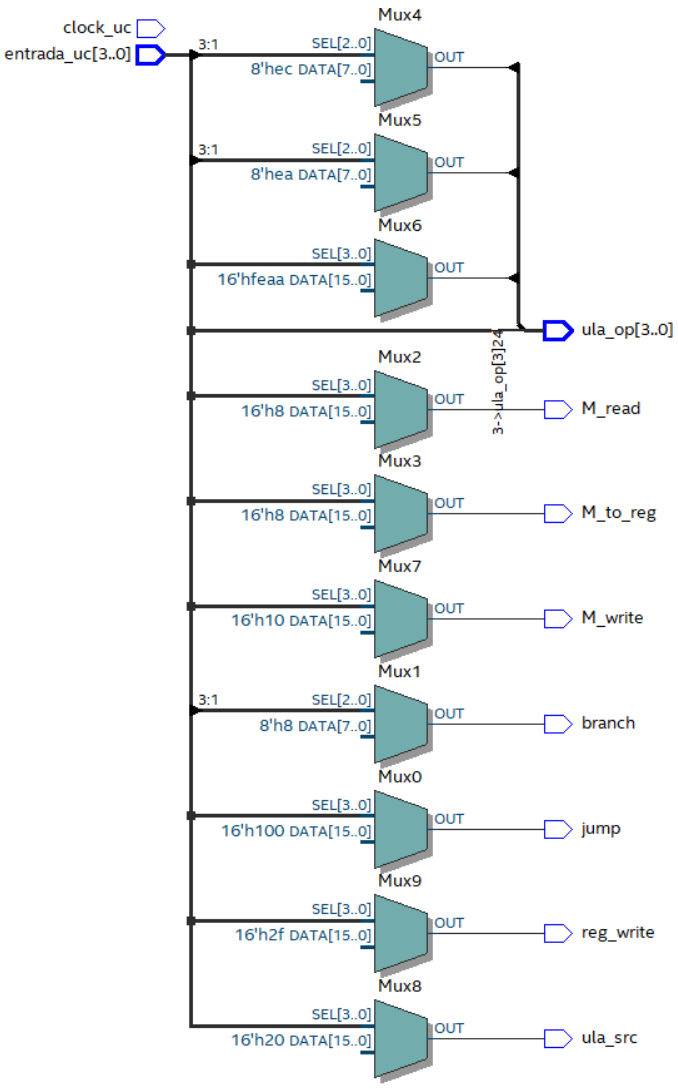

Figura 7 - Componente Unidade de Controle gerado pelo Quartus

1.3.7 Banco de Registradores

A unidade funcional Banco de Registradores é um componente composto por um conjunto de registradores que podem ser acessados de forma organizada. O Banco de Registradores do Split é composto por quatro registradores $S0, $S1, $S2 e $S3 de 2 bits cada, organizados em uma estrutura com três portas de acesso.

O componente Banco de Registradores recebe como entrada quatro valores:

- entrada_reg1 – recebe 2 bits para o endereço do primeiro registrador;
- entrada_reg2 - recebe 2 bits para o endereço do segundo registrador;
- entrada_escrita_dados – dado de 8 bits vindo da memória RAM que será escrito no registrador de destino.

- reg_write - identificador de resultado da flag reg_write de 1 bit. Recebe o valor 1 se a flag estiver ativada e 0 se estiver desativada.

O Banco de Registradores também possui duas saídas:

- saida_dado_lido1 - resulta em uma saída de 8 bits do valor armazenado no entrada_reg1
- saida_dado_lido2 - resulta em uma saída de 8 bits do valor armazenado no entrada_reg2

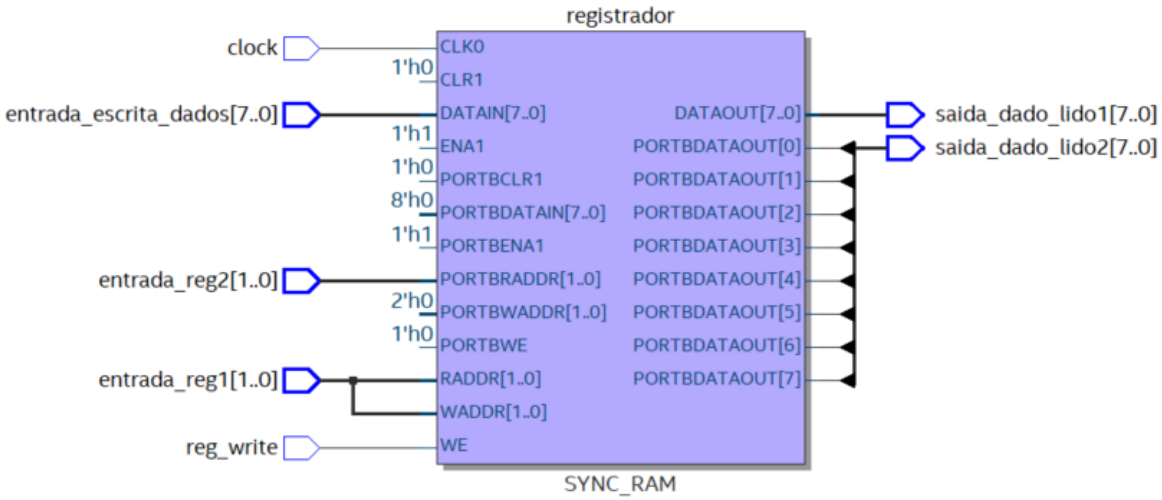

Figura 8 - Bloco simbólico do componente Banco de Registradores gerado pelo Quartus

1.3.8 Unidade Lógica Aritmética (ULA)

O componente ULA (Unidade Lógica Aritmética) tem como objetivo efetuar as principais operações aritméticas, dentre elas: soma, subtração e multiplicação. Adicionalmente a ULA efetua operações de comparação de valor para realizar desvio condicional.

O componente ULA recebe como entrada quatro valores:

- clock_ula: valor de 1 bit que indica se a unidade funcional está ligado(1) ou desligado (0).
- a entrada_dado_lido1 e entrada_dado_lido2 que irão receber os dois dados de 8 bits para realizar as operações, e
- entrada_ula_op que recebe o código da operação que será executada.

A ULA também possui duas saídas, sendo o

- resultado_ula: por onde saíra o resultado da operação que foi executado, e zero, que indica se será executado um desvio condicional.

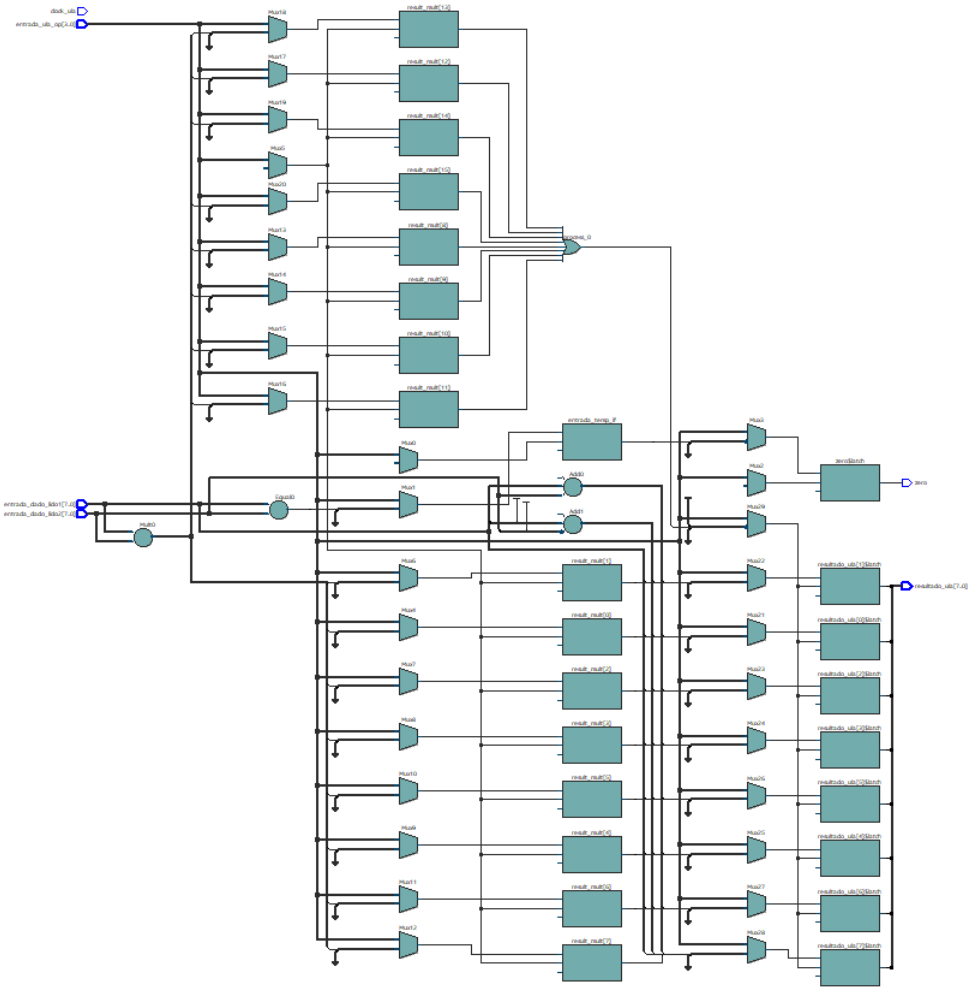

Figura 9 - Bloco simbólico do componente Unidade Lógica Aritmética gerado pelo Quartus

1.3.9 Multiplexadores

1.3.9.1 Multiplexador ula_src

Com este componente é possível selecionar qual entrada vai ser conectada à saída, como um interruptor. A partir do valor da flag ula_src (interruptor) o multiplexador definirá qual entrada será conectada à saída.

Duas entradas:

- ula_src: sinal para decidir se o dado que entrará na ULA vai ser enviado pelo banco de registradores (0) ou pelo extensor de sinal(1) de um bit.
- dado_lido2: 8 bits

- saida_extensor: 8 bits

Uma saída:

- saida_mux_ula_src: 8 bits

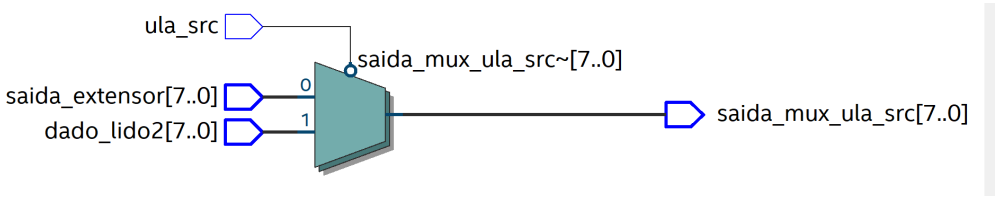

Figura 10 - Bloco simbólico do componente Multiplexador ula_src gerado pelo Quartus

1.3.9.2 Multiplexador Branch and

Com este componente é possível selecionar qual entrada vai ser conectada à saída, como um interruptor. A partir do valor da flag resultado_and (interruptor) o multiplexador definirá qual entrada será conectada à saída.

Entradas:

- resultado_and: 1 bit
- resultado_extensor_2_8: 8 bits
- resultado_add: 8 bits

Saídas:

- saida_mux_branch_and: 8 bits

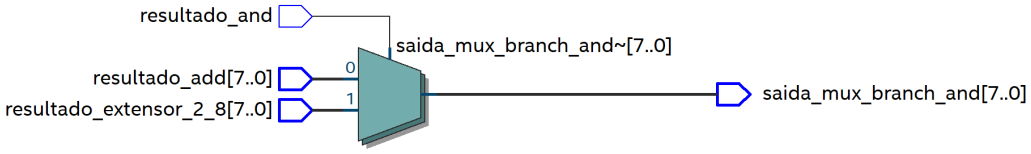

Figura 11 - Bloco simbólico do componente Multiplexador Branch AND gerado pelo Quartus

1.3.9.3 Multiplexador jump

Com este componente é possível selecionar qual entrada vai ser conectada à saída, como um interruptor. A partir do valor da flag jump(interruptor) o multiplexador definirá qual entrada será conectada à saída.

Entradas:

- flag_jump: 1 bit
- resultado_extensor_2_8: 8 bits
- resultado_mux_branch_and: 8 bits

Saídas:

- saida_mux_jump: 1 bit

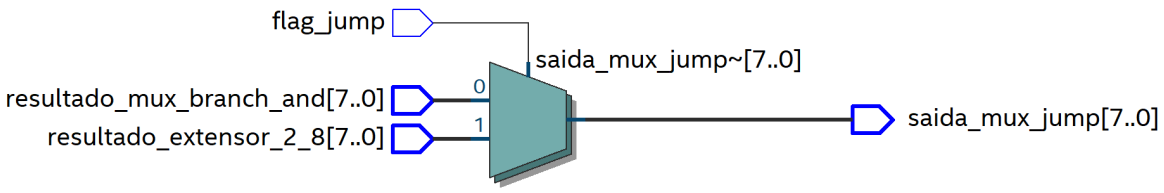

Figura 12 - Bloco simbólico do componente Multiplexador Jump gerado pelo Quartus

1.3.10 AND

O bloco lógico AND assume 1 quando todas as duas entradas forem 1 e assume 0 nos demais casos.

Duas entradas:

- entrada_and_1: a primeira entrada do componente AND diz respeito à flag Branch vinda da unidade de controle e possui 1 bit, se o valor fornecido for 1 significa que a flag de desvio condicional está ativa
- entrada_and_2: a segunda entrada do componente AND possui 1 bit e recebe 1 caso a flag Zero que significa desvio condicional vinda da ULA esteja ativada e 0 caso esteja inativa

Uma saída:

- saida_and: caso ambas as flags de desvio condicional estiverem ativas, saíra 1 bit no valor 1, o que significa que será feita uma instrução de desvio

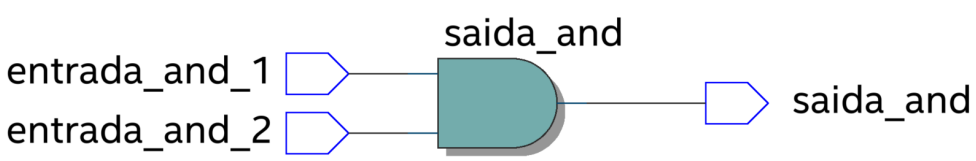

Figura 13 - Bloco simbólico do componente AND gerado pelo Quartus

1.3.11 Memória de dados

A memória RAM é responsável por armazenar temporariamente os dados que são usados durante a execução das instruções

A memória de dados recebe seis entradas:

- clock_mem_dados: que ativa representa a energia do componente, tem valor 1 se o processador está ligado 0 se desligado.
- M_read: recebe 1 bit da flag vinda da unidade de controle informando se será lido algum dado da Memória De Dados - RAM (caso o valor da entrada seja 1) ou não (caso o valor da entrada seja zero)
- M_write: recebe 1 bit da flag vinda Unidade De Controle para saber se vai armazenar dados na Memória de Dados - RAM
- entrada_endereco: recebe 8 bits da posição da memória RAM onde o dado
- deve ser escrito ou lido
- escrita_dados: recebe o dado de 8 bits que será armazenado temporariamente na RAM

E uma saída:

- dado_lido: onde sai um dado de 8 bits da posição que foi recebida no endereço, se a M_read estiver setada em 1.

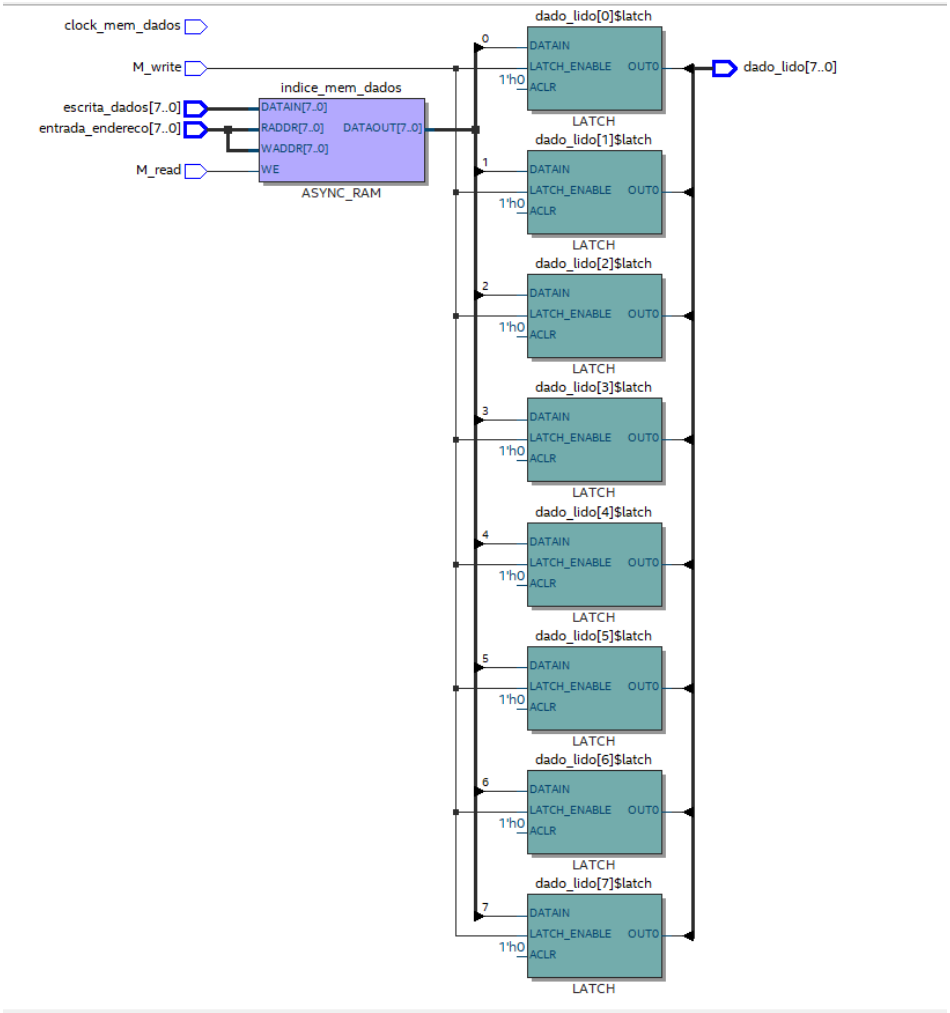

Figura 14 - Bloco simbólico do componente Memória de Dados gerado pelo Quartus

1.4 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.

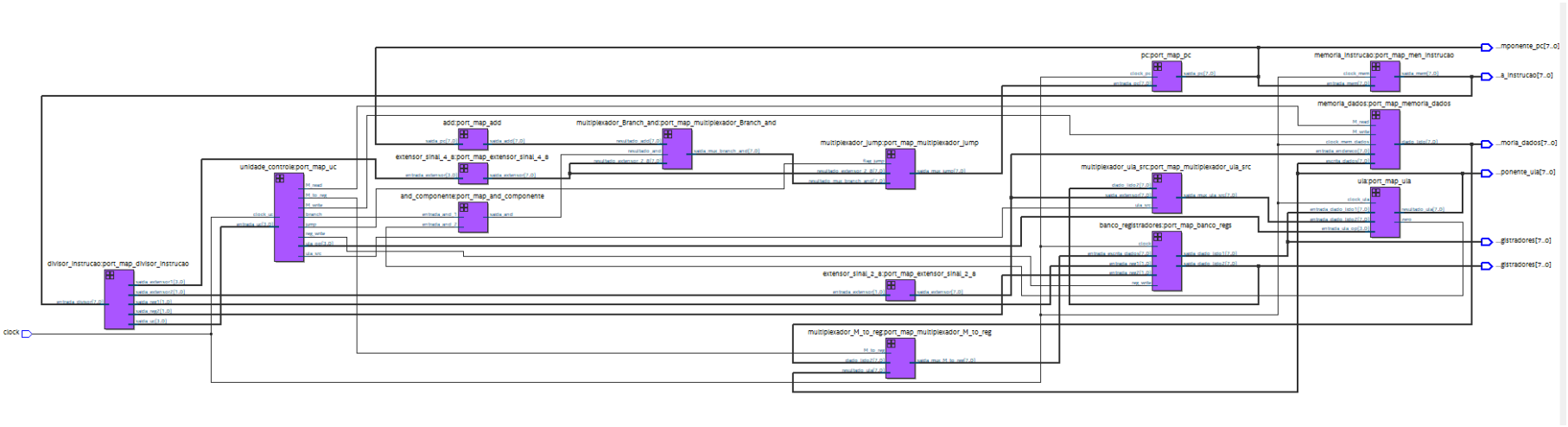

Figura 15 - Figura simbólica do Caminho de Dados gerado pelo Quartus

2 Simulações e Testes

Com o objetivo de 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 Split utilizaremos como exemplo dois códigos.

2.1 Algoritmo um

O primeiro algoritmo feito para testar o processador Split utiliza as instruções LI, IF, BEQ, BNE, MULT, JUMP, SW, LW e LI. Primeiramente o algoritmo guarda o valor 2 no registrador $s0, em seguida guarda o valor 2 no registrador $s1. Faz uma comparação entre os dois registrados e caso sejam iguais é realizado um salto para a linha 7, que é uma sub-rotina. Caso negativo, há um salto para a linha 8.

Endereço	Linguagem de Alto Nível	Binário		
		Opcode	Reg2	Reg2
			Endereço	
			Dado	
0	li $s0 2	0101	00	10
1	li $s1 2	0101	01	10
2	if $s0 == $s1	1001	00	01
3	beq $s0 == $s1 --jump linha 7	0110	01	11
4	bne $s0 != $s1 --jump linha 8	0111	10	00
5	mult $s0 $s1 --s0 vira 4	0010	00	01
6	jump --linha 2	1000	00	10
7	jump linha 5	1000	01	01
8	sw $s0 00	0100	00	00
9	lw $s0 00	0011	00	00
10	li $s0 0	0101	00	00
11	li $s1 0	0101	01	00
12	li $s2 0	0101	10	00
13	li $s3 0	0101	11	00

Tabela 6 - Código Um para demonstração de instruções do processador Split.

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

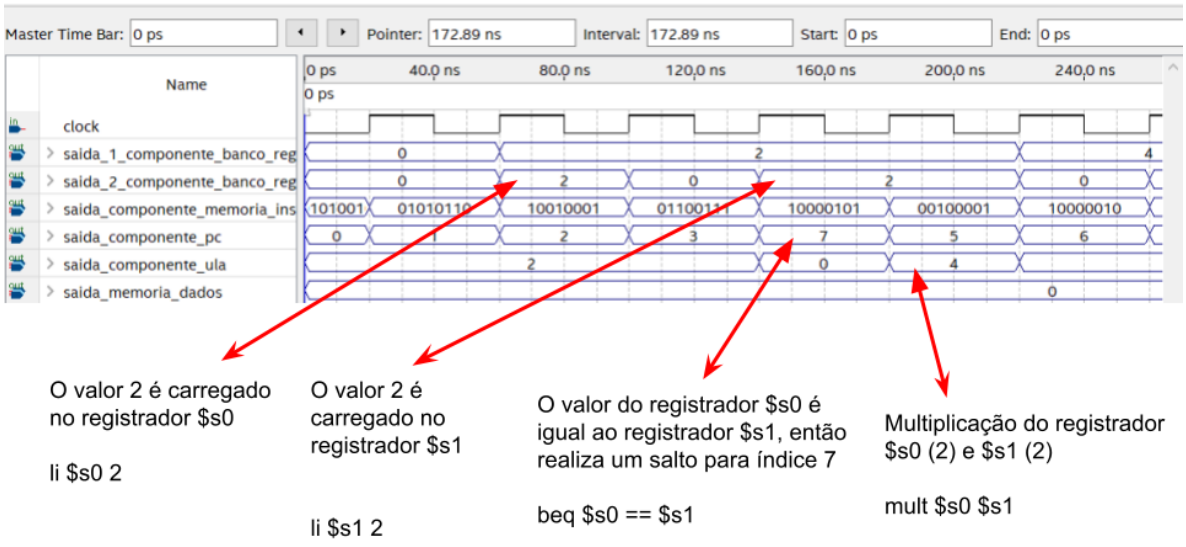

Figura 16 - Resultado do Algoritmo Um na waveform.

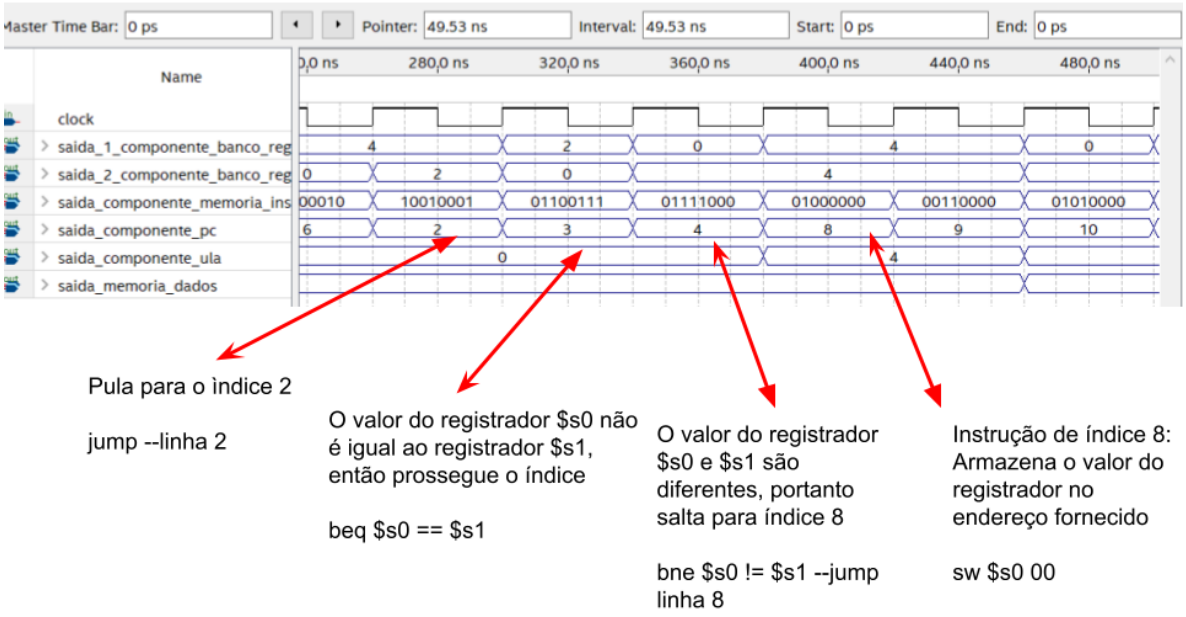

Figura 17- Resultado do Algoritmo Um na waveform.

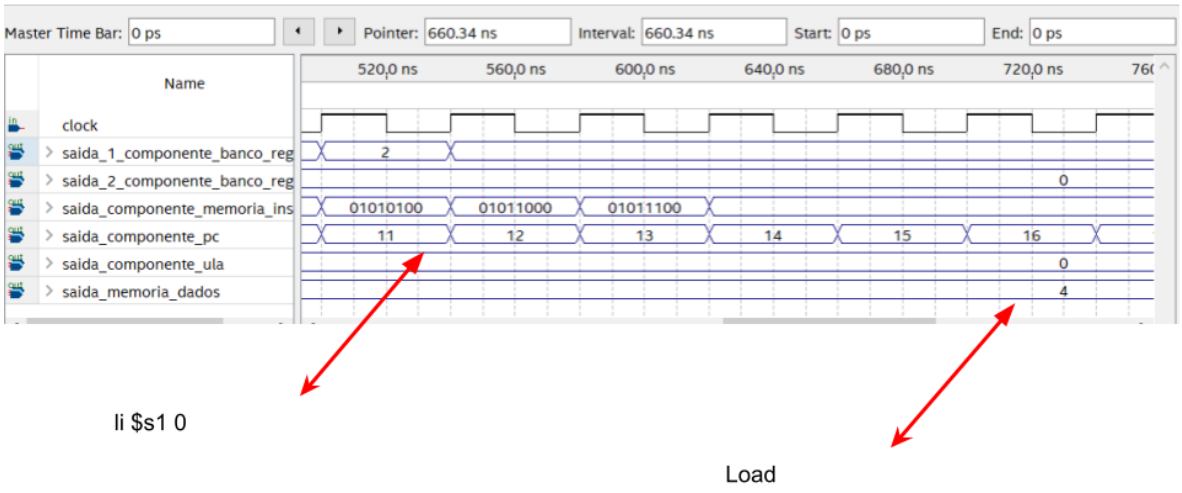

Figura 18- Resultado do Algoritmo Um na waveform.

2.2 Algoritmo dois

O segundo algoritmo feito para testar o processador Split utiliza as instruções ADD, LI, SUB, SW e LW. Primeiramente é feito um carregamento imediato - ela carrega o registrador $s0 com o valor 3 que está imediatamente disponível (sem ir para a memória). Em seguida é feita uma operação de adição, para somar 6 ao registrador $s0 e em seguida guardar o resultado no mesmo registrador.

Após fazer a operação aritmética, fazemos outro load immediate para inserir o número 2 no registrador $s1 e em seguida fazer a subtração do valor 4 em $s1 e inserir o resultado em $S0.

Endereço	Linguagem de Alto Nível	Binário		
		Opcode	Reg2	Reg2
			Endereço	
			Dado	
0	li $s0 3	0000	00	00
1	add $s0 $s0 -- 6	0101	01	10
2	li $s1 2	0101	01	10
3	sub $s0 $s1 -- 6 - 2	0001	00	01
4	sw $s0 endereço(00)	0100	00	00
5	lw $s0 endereço(00)	0011	00	00

Tabela 7 - Código Dois para demonstração de instruções do processador Split.

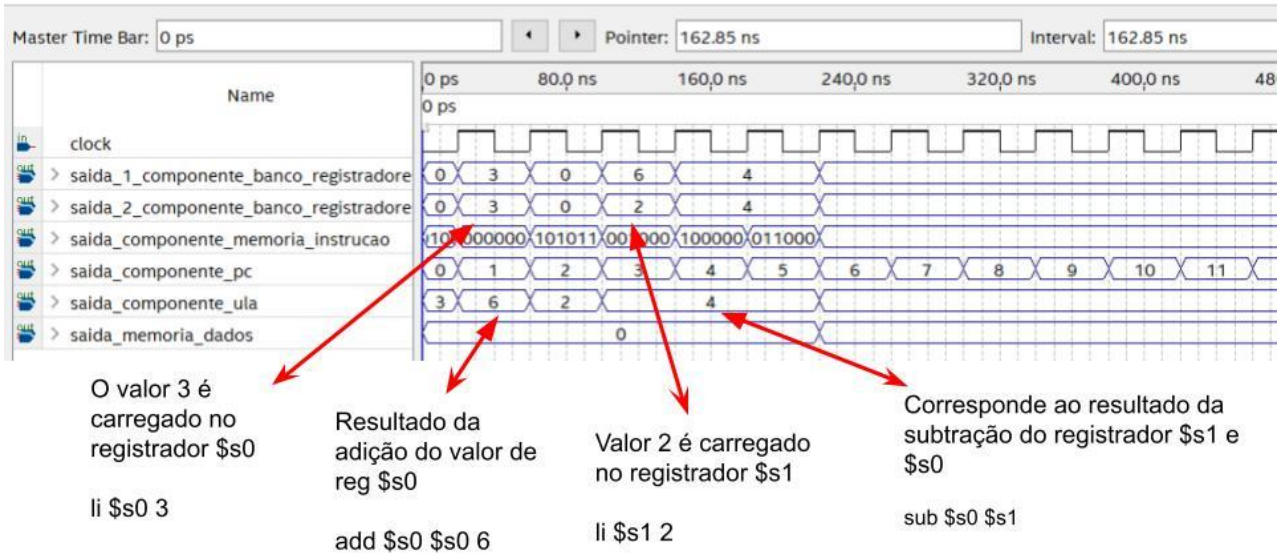

Figura 19 - Resultado do Algoritmo Dois na waveform.

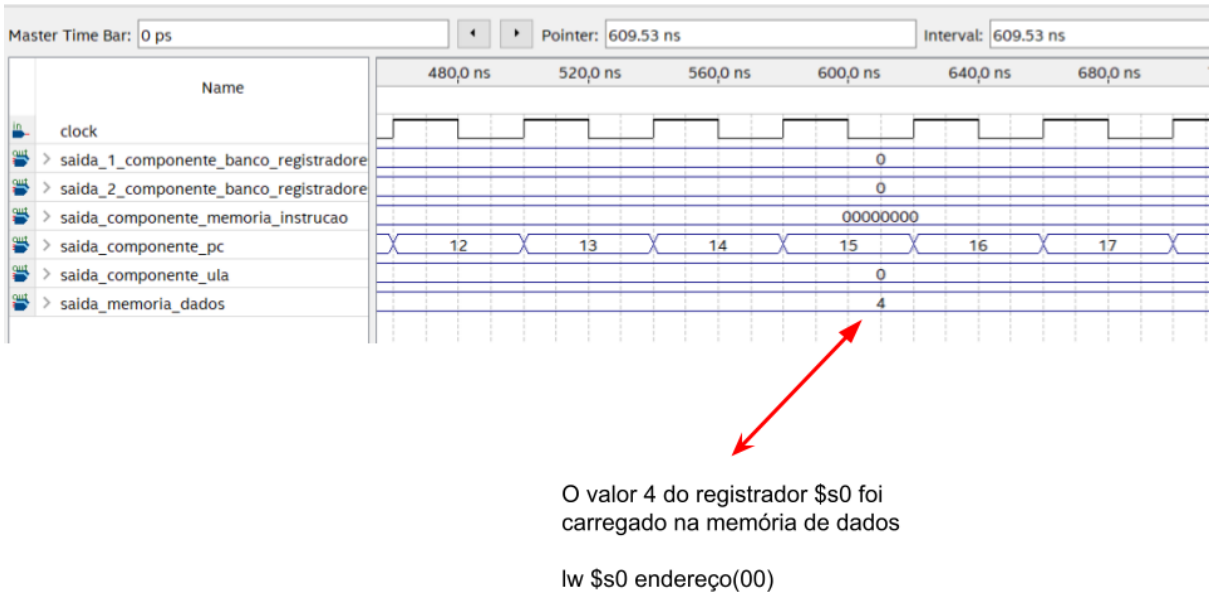

Figura 20 - Resultado do Algoritmo Dois na waveform.

3 Considerações finais

Este trabalho apresentou o projeto e implementação do processador de 8 bits denominado Split, que para sua inicialização foi necessário definir primeiramente o formato das instruções que seriam suportadas pelo processador, a lista de OpCode e as flags da Unidade De Controle. Em seguida, desenhamos o Caminho de Dados (DataPath) baseado no processador MIPS de 32 Bits.

A partir do design do Caminho de Dados podemos visualizar a ligação entre os componentes para iniciar a implementação dos mesmos. Durante a construção dos componentes nos deparamos com dificuldades em relação a Memória de Instruções pois não sabíamos declarar o vetor para armazenar as instruções a serem executadas. Com a ajuda do monitor da disciplina, Leandro Schillreff, conseguimos resolver os problemas que surgiam.

Após a implementação, fizemos a Waveform de cada componente para verificar se os mesmos funcionavam adequadamente. Ao finalizarmos todos os testes, iniciamos a ligação entres eles por meio do Port Map, que é o arquivo principal do processador.

Em vista disso, desenvolvemos dois algoritmos para demonstrar a utilização de todas as instruções do processador Split.