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

ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

RELATÓRIO DO PROJETO: PROCESSADOR MKdusk

ALUNO:
Markus Kaul Monteiro Gerrits - 2017024370

Novembro de 2019
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 MKdusk

Novembro de 2019
Boa Vista/Roraima

Resumo

Este trabalho aborda o projeto e implementação de um processador uniciclo de 8 bits com base na arquitetura do MIPS e ao longo deste relatório serão abordados os componentes utilizado e todas suas funcionalidades.

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Especificação

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

1.1 Plataforma de desenvolvimento

Para a implementação do processador MKdusk foi utilizado a IDE: Quartus (Quartus Prime 18.1) Lite Edition.

Flow Status	Successful - Wed Nov 27 09:29:49 2019
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	MKdusk8BitsProcess
Top-level Entity Name	MKdusk8BitsProcess
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	48
Total pins	110
Total virtual pins	0
Total block memory bits	0
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

Figura 1 - Especificações no Quartus

1.2 Conjunto de instruções

O processador MKdusk possui 2 registradores: S0 e S1. Assim como 3 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;
Funct:

Tipo de Instruções:

- Formato do tipo R: Este formatado aborda instruções de operações aritméticas.

Formato para escrita de código na linguagem Quantum:

Tipo da Instrução Reg1 Reg2 funct

Formato para escrita em código binário:

3 bits	1 bits	1 bits	3 bits
7-5	4	3	2-0
Opcode	Reg2	Reg1	funct

- Formato do tipo I: Este formatado aborda instruções de Load, Store, Load Immediately e Branch.

Formato para escrita de código na linguagem Quantum:

Tipo da Instrução Reg1 funct

Formato para escrita em código binário:

3 bits	1 bits	4 bits
7-5	4	3-0
Opcode	Reg1	funct

- Formato do tipo J: Este formatado aborda instruções de Jumps.

Formato para escrita de código na linguagem Quantum:

Tipo da Instrução Endereço

Formato para escrita em código binário:

3 bits	3 bits
7-5	4-0
Opcode	Endereço

Visão geral das instruções do Processador MKdusk:

O número de bits do campo **Opcode** das instruções é igual a quatro, sendo assim obtemos um total ($Bit(0e1)^{NumeroTodaldeBitsdoOpcode}$ **∴** $2^X = X$) de 8 **Opcodes (0-7)** 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 MKdusk.

Opcode	Nome	Formato	Breve Descrição	Exemplo

000000	ADD	R	Soma	**add** $S0,$S1, ou seja, $S0 := $S0+$S1
000001	SUB	R	Subtração	**sub** $S0,$S1, ou seja, $S0 := $S0-$S1
000100	MULT	R	Multiplicação	**mult** $S0, $S1 ,ou seja, $S0 := $S0 * $S1
001	LW	I	Load	**lw** $S0, 8
110	LI	I	Load Immediately	**li** $S0, 10
010	SW	I	Store	**sw** $S0, 8
011	BNE	I	Branch Not Equal	**bne** $S0,$S1,TESTE
100	BEQ	I	Branch Equal	**beq** $S0,$S1,TESTE
111	JUMP	J	Jump	**jump** Endereço

1.3 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.

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 e multiplicação (Considerando apenas números inteiros). A ULA também efetua operações de comparação de valor como igual ou diferente. O componente ULA recebe três valores: **a** - dado de 8bits para operação, **b** - dado de 8bits para operação e **ula_controle** - identificador da operação que será realizada de 3bits. A ULA também possui duas saídas: **zero** - identificador de resultado (1bits) para comparações (1 se verdade e 0 caso contrário); e **ula_resultado** - saída com o resultado das operações aritméticas.

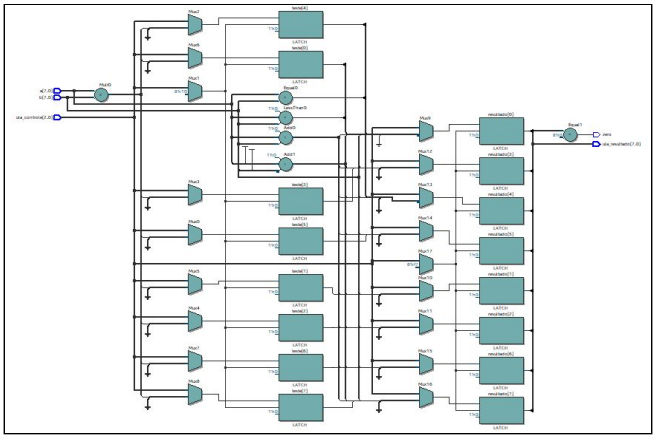

Figura 2 - Bloco simbólico do componente ULA gerado pelo Quartus

1.3.2 Banco de Registradores

O componente de Banco de Registradores funciona para guardar valores e/ou resultados de operações que foram realizadas na ULA.

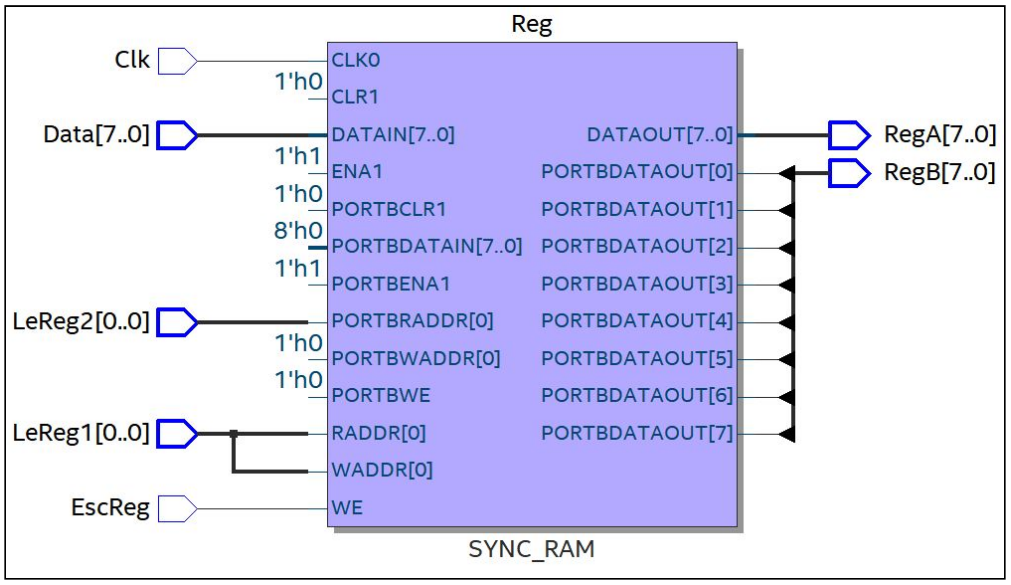

Figura 3 - Bloco simbólico do componente Banco de Registradores gerado pelo Quartus

1.3.3 Clock

O objetivo do clock é "alimentar" os componentes para funcionarem apenas quando o clock estiver ligado, ou seja, o valor do clock for um.

1.3.4 Unidade de 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:

mem_to_reg: XXXX.

ula_op: XXXX.

branch: XXXX.

mem_read: XXXX.

mem_write: XXXX.

ula_src: XXXX.

reg_write: XXXX.

jump: XXXX.

reg_for_write: 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	mem_to_reg	ula_op	branch	mem_read	mem_write	ula_src	reg_write	jump	reg_for_write
add	0	000	0	0	0	1	1	0	0
sub	0	001	0	0	0	1	1	0	0

mult	0	100	0	0	0	1	1	0	0
lw	1	110	0	1	0	0	1	0	0
li	0	110	0	0	0	0	1	0	0
sw	0	110	0	0	1	0	0	0	1
bne	0	101	1	0	0	1	0	0	0
beq	0	010	1	0	0	1	0	0	0
jump	0	111	0	0	0	0	0	1	0

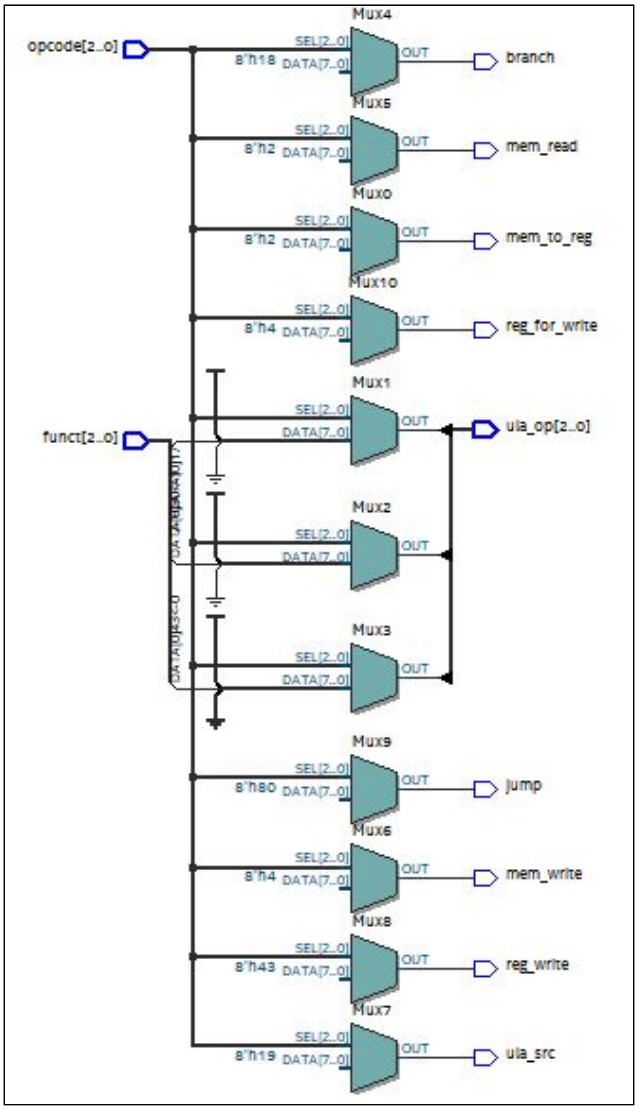

Figura 4 - Bloco simbólico do componente Unidade de Controle gerado pelo Quartus

1.3.5 Memória de dados (RAM)

O a unidades de dados é responsável por armazenar os valores na memória e em disponibilizar esses valores para serem salvos nos registradores. Tendo como entrada **dado_entrada** - que define o que vai ser salvo na memória; **endereco** - o endereço onde o valor será salvo na memória; **EscMem** - flag que define se algo vai ser escrito na memória ou não; **LeMem** - flag que define se algo vai ser lido da memória e passada para o registrador; **clk** - clock. E ele retorna com

saída **dado_saida** - que retorna o valor que foi lido da memória.

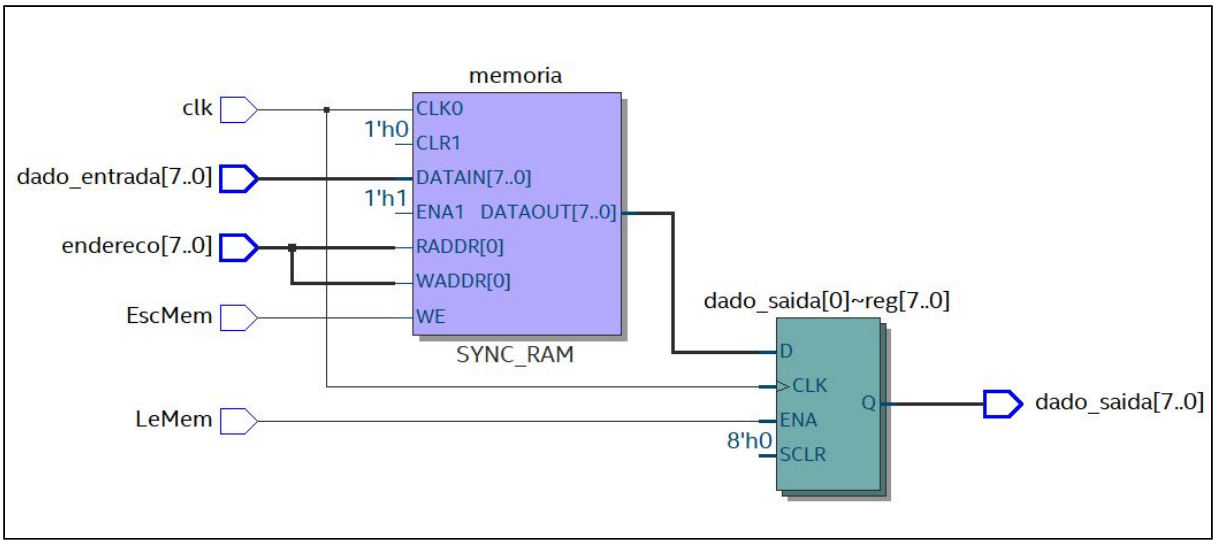

Figura 5 - Bloco simbólico do componente Memória de Dados gerado pelo Quartus

1.3.6 Memória de Instruções (ROM)

É na memória de instruções que todas as instruções serão executadas e é a partir dela que o opcode vai para a unidade de controle e que sabemos quais registradores usar. A memória de instrução recebe um endereço que vem do PC e executa a instrução que está nesta instrução.

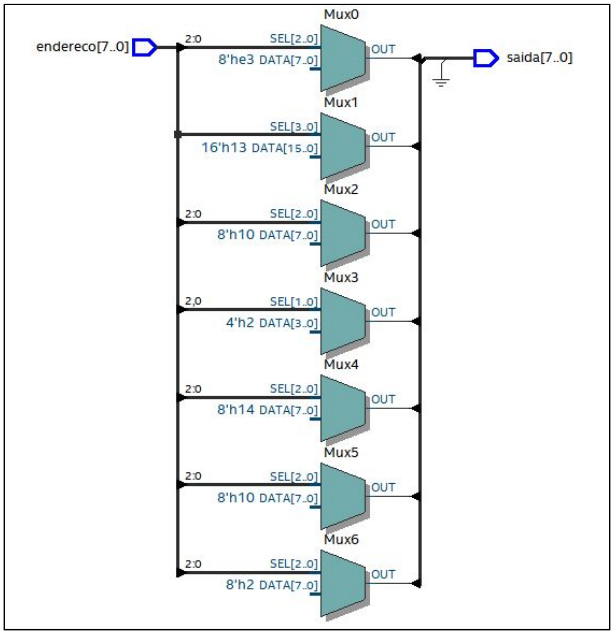

Figura 6 - Bloco simbólico do componente Memória de Instrução gerado pelo Quartus

1.3.7 Somador

Responsável por somar de um em um o PC para que ele execute as instruções.

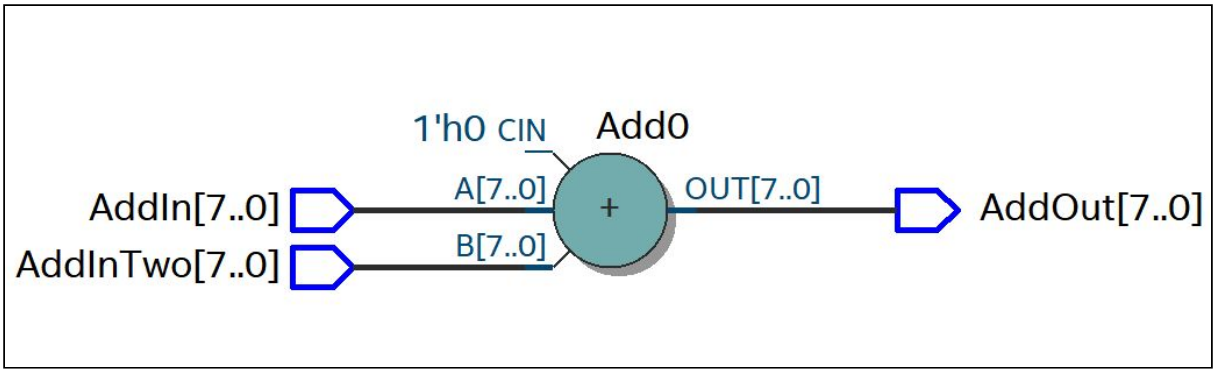

Figura 7 - Bloco simbólico do componente Somador gerado pelo Quartus

1.3.8 And

Usado para verificar se vai ocorrer um branch ou não.

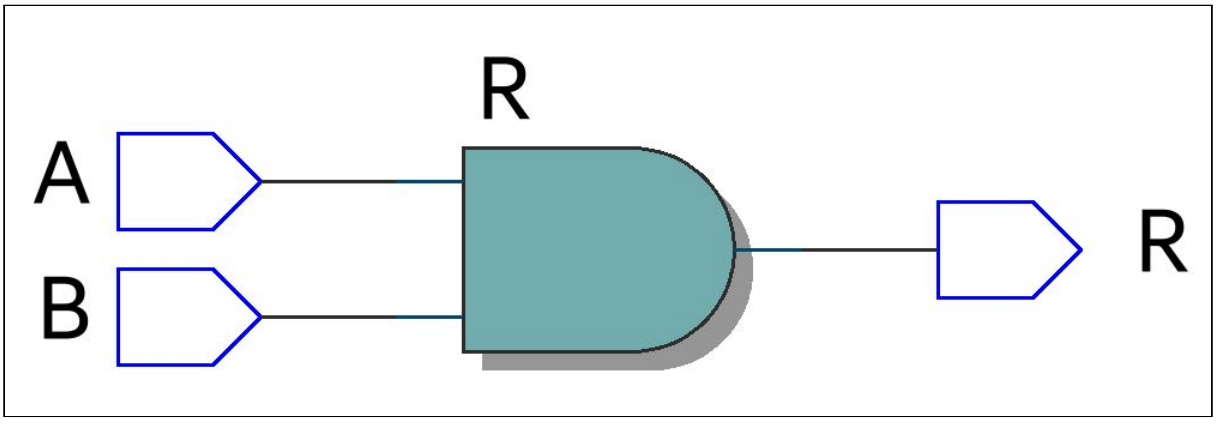

Figura 8 - Bloco simbólico do componente And gerado pelo Quartus

1.3.9 Mux_2x1

O multiplexador auxilia a decidir que trilha será usada dependendo do valor da flag que o multiplexador está recebendo.

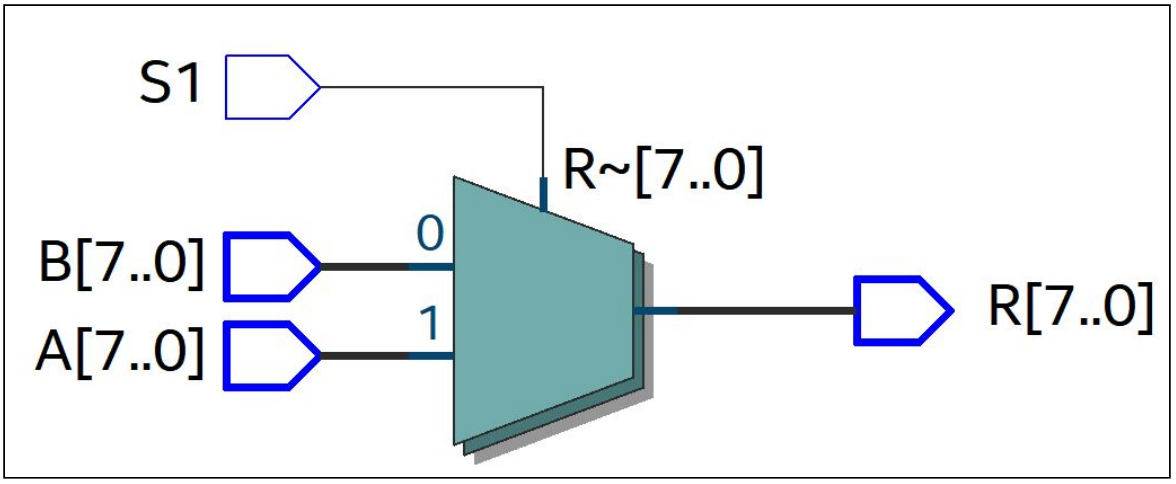

Figura 9 - Bloco simbólico do componente Multiplexador gerado pelo Quartus

1.3.10 PC

Responsável em mandar o endereço para a próxima instrução para a memória de instruções.

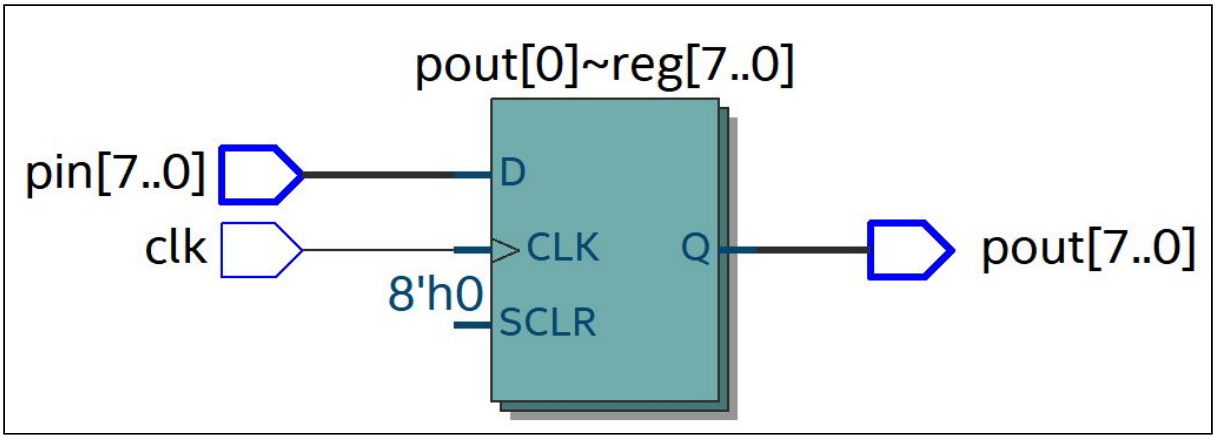

Figura 10 - Bloco simbólico do componente PC gerado pelo Quartus

1.3.11 Zero

identificador de resultado (1bits) para comparações (1 se verdade e 0 caso contrário).

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.

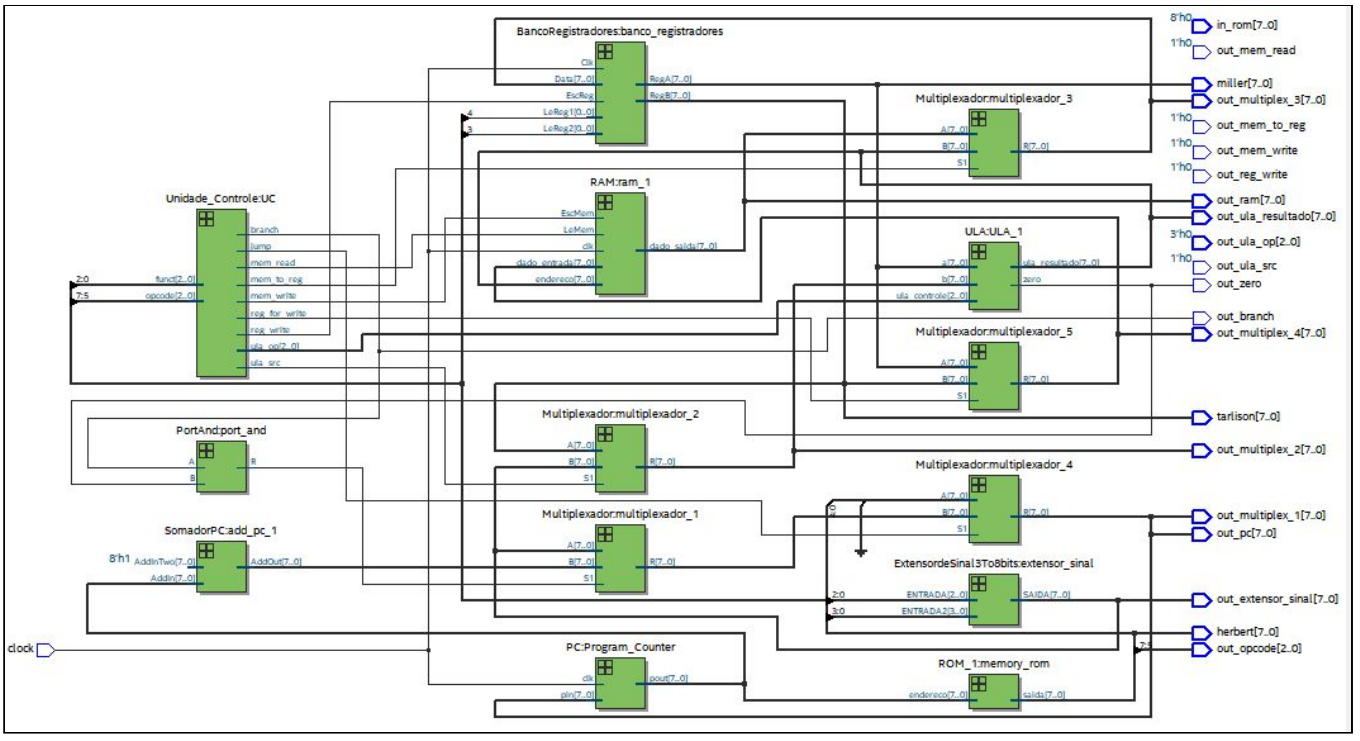

Figura 11 - Datapath gerado pelo Quartus

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 MKdusk 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	Funct
			Endereço		
		Dado			
0	**LI** $S0, 0	110	0	0000	
		00000000			
1	**LI** $S1, 1	110	1	0001	
		00000001			
2	**ADD** $S0, $S1	000	0	1	000
3	**ADD** $S0, $S1	000	1	0	000
4	**BNE** $S0, SS1	011	0	1	010

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

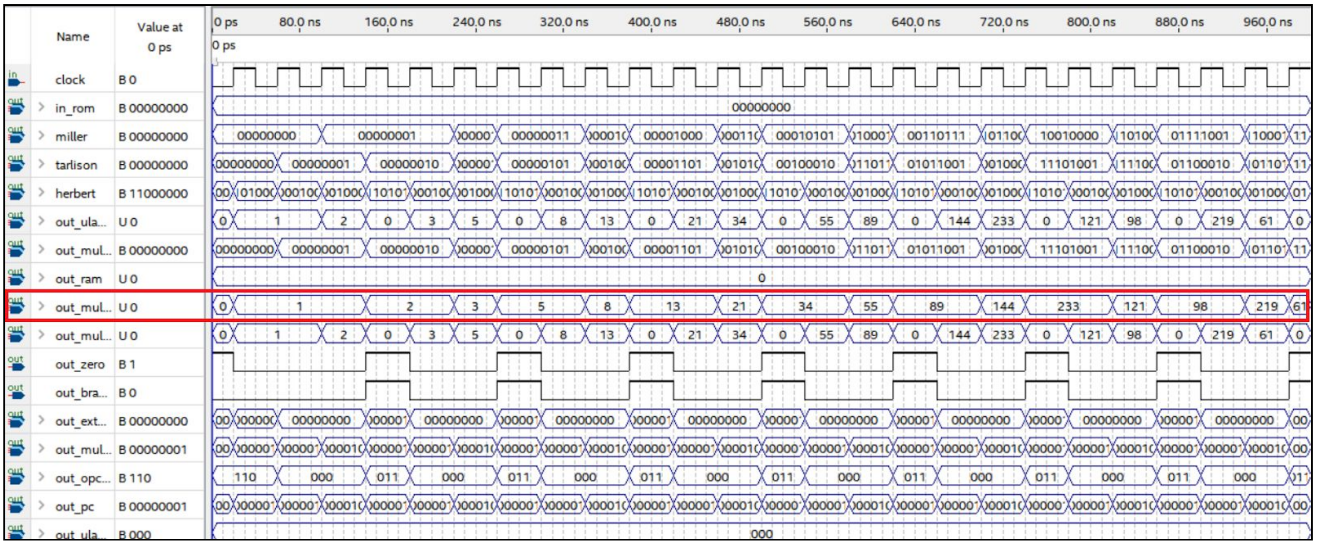

Figura 12 - Resultado na Wafeform para o Fibonacci.

3 Considerações finais

Este trabalho apresentou o projeto e implementação do processador uniciclo de 8 bits como projeto final de disciplina de Arquitetura e Organização de computadores, tendo o objetivo de colocar todo aprendizado obtido em prática. O projeto requereu amplos assuntos da disciplina, a fim de que tenhamos um desenvolvimento mais aprimorado do assunto, foram seguindo vários passos começando desde criação das instruções, do desenho do datapath, até a programação dos componentes do processador partindo assim para a sua montagem. O processador contém todos os requisitos para ser assim denominado, por exemplo, contém não só operações aritméticas, como memória de dados e instruções, desvios condicionais, controle de dados, etc.