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

ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

RELATÓRIO DO PROJETO: PROCESSADOR KARR

ALUNOS:
Angelo Almeida Ferro – 1201524424
Kaio Guilherme Ferraz De Sousa Silva - 2020014670

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 KARR

Março de 2022
Boa Vista/Roraima

Resumo

Este trabalho aborda o projeto e implementação de um processador de 8 bits baseado na arquitetura mips uniciclo. Nele haverá um total de 6 instruções devido a limitação de 8bits por instrução e 4 registradores, pois ao tentar fazer contas mais complexas do que essas pode haver problemas de overflow ou overheat.

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Especificação

Nesta seção será apresentado o conjunto de itens utilizados para o desenvolvimento do processador KARR de 8 bits, bem como a descrição detalhada de cada etapa da construção do processador, tendo o mesmo quatro registradores e e 255 espaços de memoria para trabalhar podendo trabalhar somente com valores entre 0 255.

1.1 Plataforma de desenvolvimento

Para a implementação do processador KARR foi utilizado a IDE: Intel Quartus Lite edition 20.1 como mostra o print das especificações a baixo:

Flow Status	Successful - Mon Mar 14 10:44:13 2022
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	processador_mips_8_bits
Top-level Entity Name	processador_mips_8_bits
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	32 / 56,480 (< 1 %)
Total registers	0
Total pins	10 / 268 (4 %)
Total virtual pins	0
Total block memory bits	0 / 7,024,640 (0 %)
Total DSP Blocks	0 / 156 (0 %)
Total HSSI RX PCSs	0 / 6 (0 %)
Total HSSI PMA RX Deserializers	0 / 6 (0 %)
Total HSSI TX PCSs	0 / 6 (0 %)

Figura 1 - Especificações no Quartus

1.2 Conjunto de instruções

O processador KARR possui 4 registradores: Reg1, Reg2, Reg3 e Reg4. Assim como 02 formatos de instruções de 8 bits cada, Instruções do **tipo R** e do **tipo I**, 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;
Func: Sub operação do Opcode

Tipo de Instruções:

- Formato do tipo R: Este formatado aborda instruções aritimeticas como suma subtração e ifs condicionais como beq e bne.

-Formato do tipo I: Este formato aborda instruções de caracter imediato principalmente instruções de dados como o load imediate , lw e sw.

Formato para escrita do processador:

Tipo da Instrução	Reg1	Reg2	Func

Formato para escrita em código binário:

2 bits	2 bits	2 bits	2 bits
7-6	5-4	3-2	1-0
Opcode	Reg2	Reg1	Func

Visão geral das instruções do Processador KARR:

O número de bits do campo **Opcode** das instruções é igual a dois, sendo assim obtemos um total ($Bit(0e1)^{NumeroTodaldeBitsdoOpcode} \therefore 2^X = X$) de 04 **Opcodes (0-3)** que para funções de Opcode 00 existem 4 subfunções destinadas pelo campo Func de 0 a 3 são distribuídos entre as instruções, assim como é apresentado na Tabela 1.

Tipo R 8-bits											
Conjunto	Instruções	op code		rd		rs		func		Uso	Exemplo
Aritimetica	add	0	0	x	x	x	x	0	0	add rd = rd + rs	add $s0, $s1
	sub	0	0	x	x	x	x	0	1	sub rd = rd - rs	sub $s0, $s1
Tipo R 8-bits loop											
Conjunto	Instruções	op code		indice				func		Uso	Exemplo
loop	loop	0	0	x	x	x	x	1	0	loop indice anteceço da instrução 15	loop 15 instruções atras
Tipo I 8-bits											
Conjunto	Instruções	op code		rd		rs		func		Uso	Exemplo
Dado	lw	0	1	x	x	x	x	0	0	lw rs, endereço	lw $s0, 0 - 255
	sw	0	1	x	x	x	x	0	1	sw rs, endereço	sw $s0, 0 - 255
	li	0	1	x	x	x	x	1	1	li rd = rs	move $s0, $s1
Tipo R 8-bits											
	Instruções	op		rd		rs		endereço		Uso	Exemplo
condicional	beq	1	0	x	x	x	x	i	i	beq rd = rs pula para endereço a frente	beq $s0, $s1, 3
condicional	bne	1	1	x	x	x	x	i	i	bne rd != rs pula para endereço a frente	bne $s0, $s1, 3

Tabela 1 – Tabela que mostra a lista de Opcodes utilizadas pelo processador KARR.

1.3 Descrição do Hardware

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

1.3.1 ALU ou ULA

O componente ULA (Unidade Logica Aritmética) tem como principal objetivo efetuar as principais operações aritméticas, dentre elas: Soma e Subtração (utilizando apenas valores inteiros). Também possui operação de comparação de valor como igualdade onde se for igual executa a ação e se for diferente será direcionado pro zero. A ULA possui três valores de entrada que são: **A -** dado de 8 bits para operação; **B -** dado de 8 bits para operação; **OP -** Identificador da operação que será realizada 2bits. Também possui dois valores de saída que são: **zero -** identificador de resultado de seleção onde se for 0 soma, se for 1 subtrai e se for 2 redireciona para o clock; **Result –** Saída com o resultado das operações aritméticas.

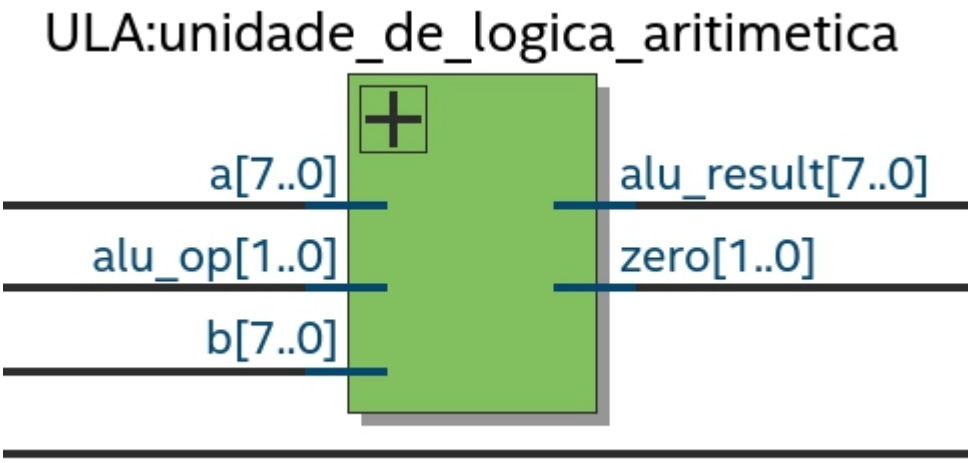
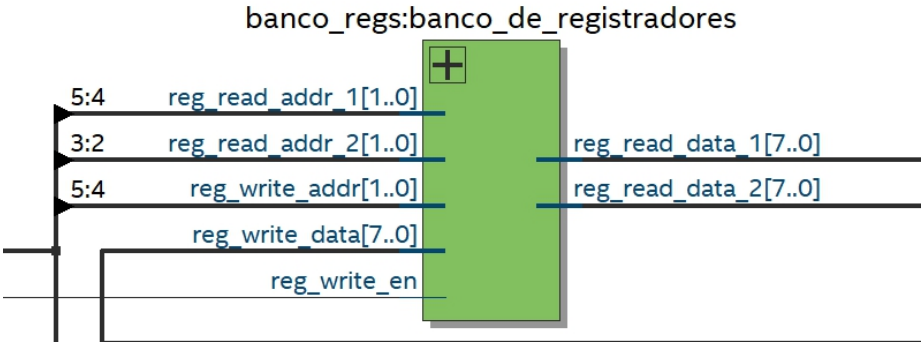


Figura 2 - Bloco simbólico do componente QALU gerado pelo Quartus

1.3.2 BDRegister

O componente BDRegister (Banco de Registradores) tem como principal objetivo “escrever” e “ler” dados que podem ou não ser armazenados momentaneamente durante o clock. Esse componente possui 5 valores de entrada que são: **Reg_Write_En –** Dado que define se a função vai ser de leitura ou escrita;
Reg_Write_Addr – Endereço do registrador onde o dado será escrito;
Reg_Write_Data – valor do dado que será escrito no endereço selecionado;
Reg_Read_Addr1 – local do endereço de acesso desejado do Registrador;



Reg_Read_Addr2 – local do endereço de acesso desejado do Registrador. Também possui 2 valores de saída que são: **Reg_Read_Data1 –** Leitura do dado armazenado no registrador; **Reg_Read_Data1 –** Leitura do dado armazenado no registrador.

1.3.3 Clock

O componente Clock tem como objetivo realizar o controle do andamento do programa do processador, sendo responsável pelo tempo de execução.

1.3.4 Controle

O componente Control(Controle) tem como objetivo realizar o controle de todos os componentes do processador de acordo com o OP Code(2 bits) e Func(2 bits). Esse controle é feito através das flags de saída abaixo:

Reg Data: 2 bits.
Reg Write: 1 bit.
Seletor: 1 bit.
ALU Op: 2 bits.
MEM Write: 1 bit.
Loop Func: 2 bits.

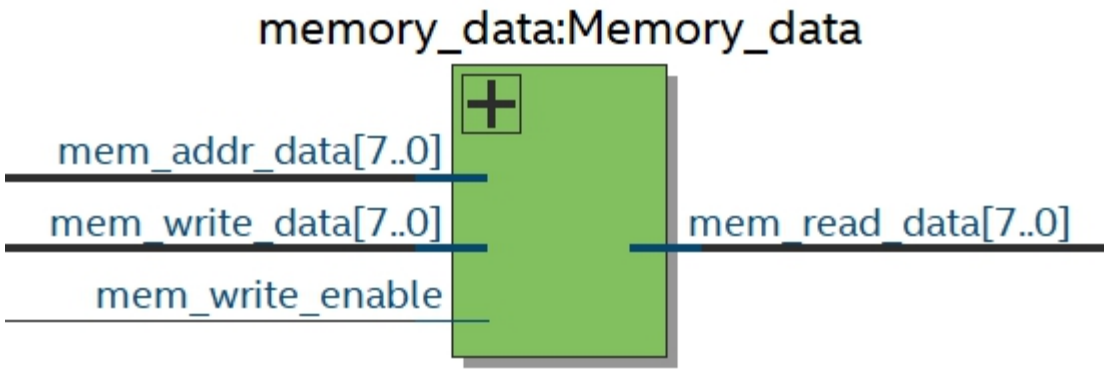
Abaixo segue a tabela, onde é feita a associação entre os opcodes e as flags de controle:

Controlador												
Instruções	Op Code		Func		Reg Data		Reg Write	seletor	ALU op		MEMwrite	loop func	
add	0	0	0	0	1	0	1	0	0	0	0	0	0
sub	0	0	0	1	1	0	1	0	0	1	0	0	0
loop	0	0	1	0	0	0	0	0	0	0	0	1	0
lw	0	1	0	0	0	1	1	1	0	0	0	1	1
sw	0	1	0	1	0	0	0	1	0	0	1	1	1
LI	0	1	1	0	0	0	1	0	0	0	0	1	1
beq	1	0	x	x	0	0	0	0	1	0	0	0	0
bne	1	1	x	x	0	0	0	0	1	1	0	0	0

Tabela 2 - Detalhes das flags de controle do processador.

1.3.5 Memória de dados

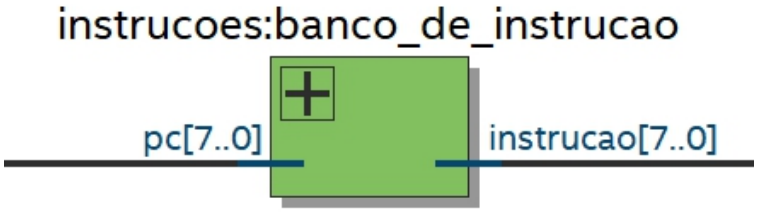
O componente Memory Data (Memoria de Dados) tem como principal objetivo “escrever” e “ler” dados que podem ou não ser armazenados



momentaneamente durante o clock. Esse componente possui 3 valores de entradas que são: **Mem_Write_Enable –** Determina a função realizada onde 0 “ler” e 1 “escreve”; **Mem_Addr_Data –** Endereço de memoria onde o dado será “Escrito” ou “Lido”; **Mem_Write_Data –** Dado que sera alocado no endereço de memoria selecionado. Também possui 1 valor de saída que é: **Mem_Read_Data –** Dado que será “Lido”.

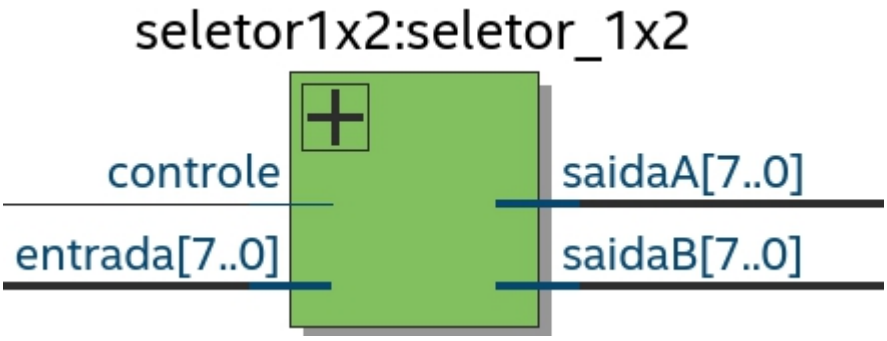
1.3.6 Memória de Instruções

O componente Memoria de Instruções tem como principal objetivo receber o indice que define qual operação o processador executara no momento sendo 7 instruções no total as quais são: **Add –** Soma de valores **;Sub –** Subtração de valores **;Move –** Encaminha o dado de um Reg para outro e reescrever se necessário**; Loop –** Retorna quantas “casas” foram selecionadas no array **; Beq –** Comparação **;Lw –** “Leitura” de dados **;Sw –** Aramazena dados.



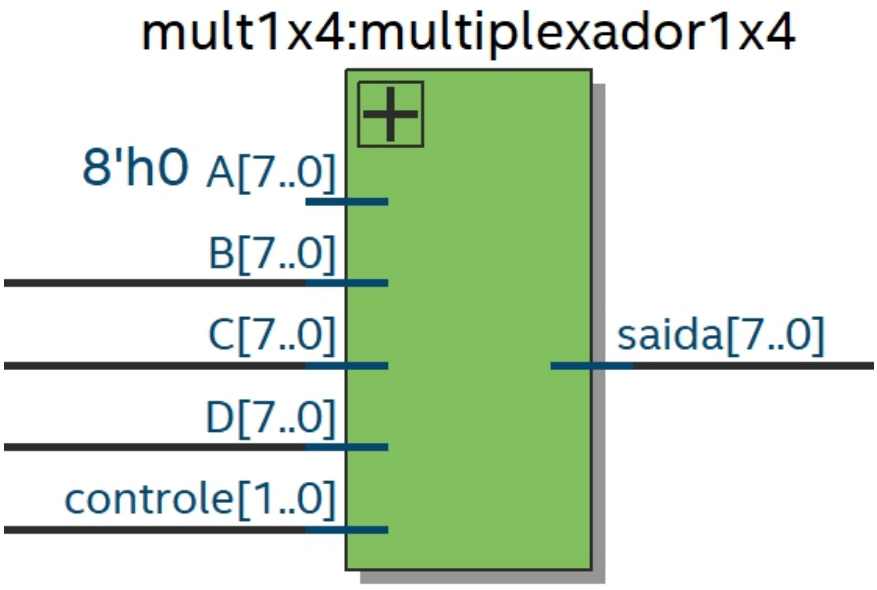
1.3.7 Seletor 1x2

O componente seletor tem como o objetivo pegar um entrada e controlar para qual barramento esses dados deverão ir, sendo assim um sinal entra pela entrada e irar sair pela saida A ou B dependendo da entrada do controlador sendo 0 para sair na A e 1 para sair na B.



1.3.8 Mux_1x4

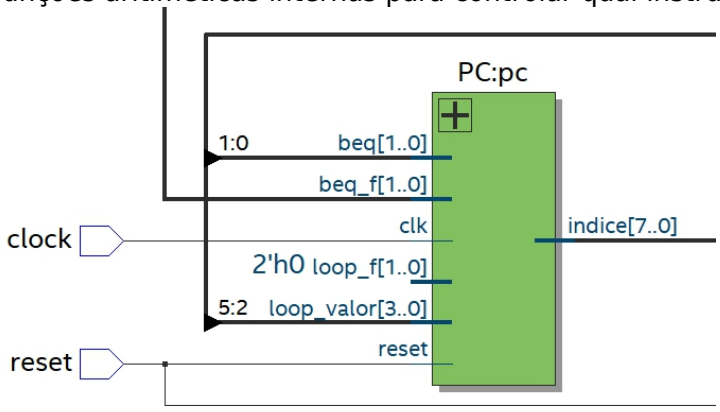
O componente Mux_1x4 (Multiplexador 4 pra 1) tem como principal objetivo receber vários dados ao mesmo tempo e através de um controle decidir qual desses dados recebidos será utilizado. Possui 5 valores de entrada que são: **A –** Dado de 8 bits para operação **;B –** Dado de 8 bits para operação **;C –** Dado de 8 bits para operação **;D -** Dado de 8 bits para operação **;Controle –** Dado de 2 bits que define qual das outras entradas será utilizada. Também possui 1 valor de saída: **Saida –** dado



de 8 bits selecionado.

1.3.9 PC

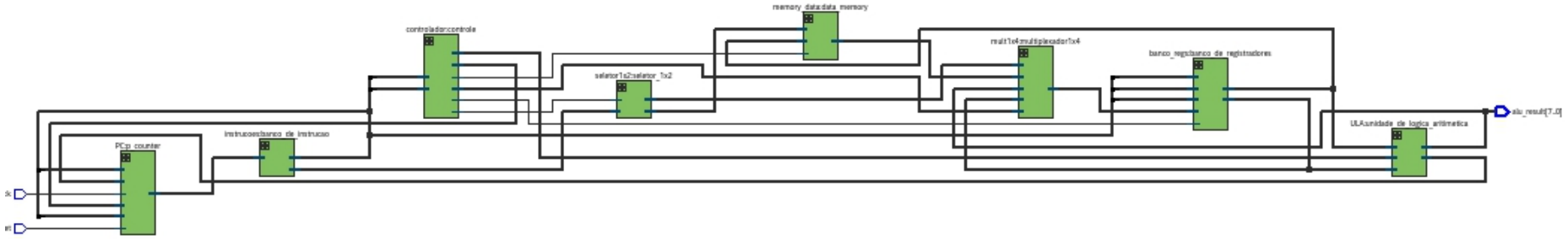
O componente PC tem como principal objetivo de enviar o índice para o banco de instruções definindo o que será executado e enquanto isso também controla as funções de beq e loop onde se alternam para executar o programa sem problemas usando de funções aritimeticas internas para controlar qual instrução sera executada.



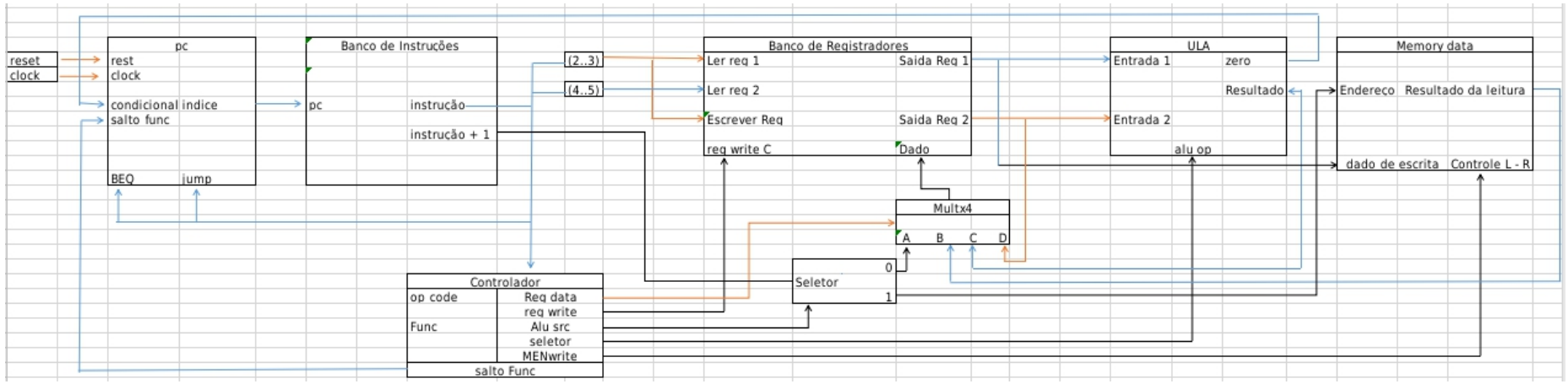
1.4 Datapath

É a ligação entre os componentes que executam de forma única os dados e adiciona uma forma de controle que organiza as ações que serão executadas por classes diferentes de instrução.

Versão RTL:



Versão Rascunho:



2 Simulações e Testes

Objetivando analisar e verificar o funcionamento do processador, efetuamos alguns testes analisando cada componente do processador em especifico, em seguida efetuamos testes de cada instrução que o processador implementa. Para demonstrar o funcionamento do processador KARR utilizaremos como exemplo o código para calcular todas as instruções possíveis.

Instruções de teste									
Endereço	op code		rd		rs		func		assembly
0	1	0	0	0	0	0	1	1	LI $s0 = 10
1	0	0	0	0	1	0	1	0	valor passado
2	1	0	0	1	0	0	1	1	LI $s1 = 5
3	0	0	0	0	0	1	0	1	valor passado
4	0	0	0	0	0	1	0	0	add $s0 ,$s1
5	0	0	0	0	0	1	0	1	sub$s0, $s1
6	0	1	0	0	0	0	0	1	sw $0, 234
7	1	1	1	0	1	0	1	0	endereço passado
8	0	1	1	1	0	0	0	0	lw $3, 234
9	1	1	1	0	1	0	1	0	endereço passado
10	1	0	0	0	0	1	1	0	beq $s0, $s1, 2
11	1	1	0	0	0	1	1	0	bne $s0, $s1, 2
12	0	0	1	1	1	0	1	0	loop 14

Tabela 3 - Código teste para o processador KARR.

Verificação dos resultados no relatório da simulação: Não conseguimos executar o teste principal pois algum bug impedia de o clock proceguir, porem os teste de todos os componentes individuais estão disponiveis no repositorio do processador sendo acessiveis neste link:

3 Considerações finais

Com a finalização do processador podemos notar a dificuldade que é trabalhar com uma quantidade minima de bits, logo tentamos de tudo para contornar essa limitações, como exemplo a ultilização de uma segunda instrução destinada ao armazenamento de valores brutos designado ao endereçamento e armazenamento de dados principalmente nas instruções que lidavam com dados como o load imediate e o lw e sw, sendo assim o processador karrs mesmo não termos consiguido testa ele, creio que com um pouco mais de tempo e foco poderiamos melhora-lo muito.