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

ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

RELATÓRIO DO PROJETO: Prolag

ALUNOS: Leonam de Sousa Silva – 2022007093
Ângelo Garcia Fernandez 2022010034
Guilherme Almeida da Luz- 2022010043

Novembro de 2023
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: Prolag

Novembro de 2023
Boa Vista/Roraima

Resumo

Este projeto aborda a elaboração e implementação do processador uniciclo de 16 bits denominado “Prolag” baseado na arquitetura do processador MIPS. Programado na linguagem VHDL, que é utilizada de forma extensiva para o design de circuitos digitais. A implementação do projeto foi realizada na IDE Quartus Prime Lite Edition v18.1. O processador conta com Program Counter, Memória de Instruções (ROM), Banco de Registradores, Unidade Lógica Aritmética (ULA), Memória de Dados (RAM), Extensores de 6 para 16 bits e Extensor de 12 para 16 bits. Os testes

Dos componentes do processador “Prolag” foram realizados com o simulador Modelsim, tentando demonstrar o funcionamento das entradas e dos componentes.

Palavras-Chave: MIPS, VHDL, Processador, 16 Bits, Quartus Prime Lite.

Conteúdo

Lista de Figuras

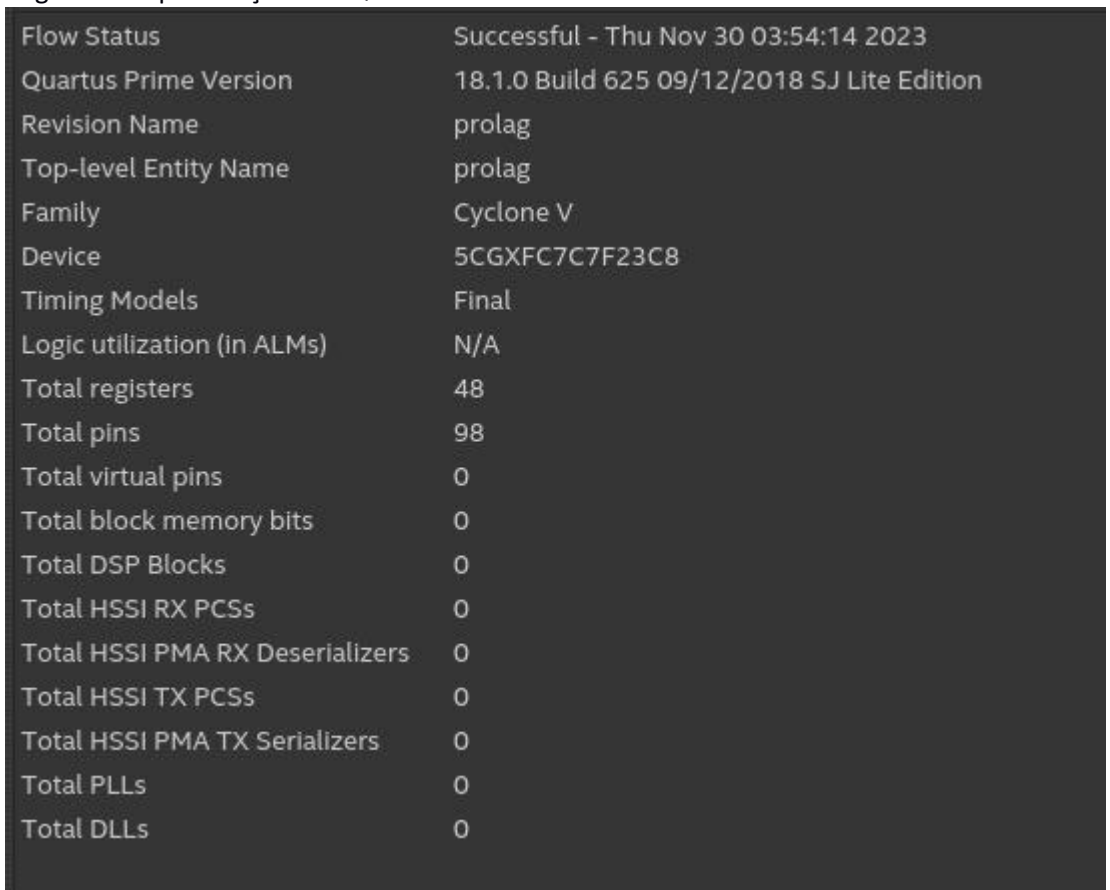
1 Especificação

Nesta seção é apresentado o conjunto de itens para o desenvolvimento do processador Prolag e a descrição detalhada de cada etapa da construção do processador.

1.1 Plataforma de desenvolvimento

Para a implementação do processador Prolag foi utilizado a IDE: Quartis Prime 18.1 Lite Edition

Figura 1 - Especificações no Quartus



Flow Status	Successful - Thu Nov 30 03:54:14 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	prolag
Top-level Entity Name	prolag
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	48
Total pins	98
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
Total PLLs	0
Total DLLs	0

1.2 Conjunto de instruções

O processador Prolag possui registradores: S0 e S1. Que combinam com formatos de instruções do **tipo R, I e J** de 16 bits:
Opcode: a operação básica a ser executada pelo processador

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

Tipo de Instruções:

- Formato do tipo R: (Add, Sub) Instruções de operações aritméticas.

4 bits	3 bits	3 bits	3 bits	0 bits	3 bits
Opcode	**RS**	**RD**	**RT**	**SHAMT**	**FUNCT**

- Formato do tipo I : (Lw, Sw, Beq) Instruções de Load, Store e Branch.

4 bits	3bits	3bits	6bits
OP	**RS**	**RD**	**ENDEREÇO**

- Formato do tipo J : (jump) Instrução do tipo jump.

4 bits	12bits
OP	**SALTO**

Visão geral das instruções do Processador Prolag:

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 3 **Opcodes (0-1)** que são distribuídos entre as instruções, assim como é apresentado na Tabela 1.

1.2.1 ULA

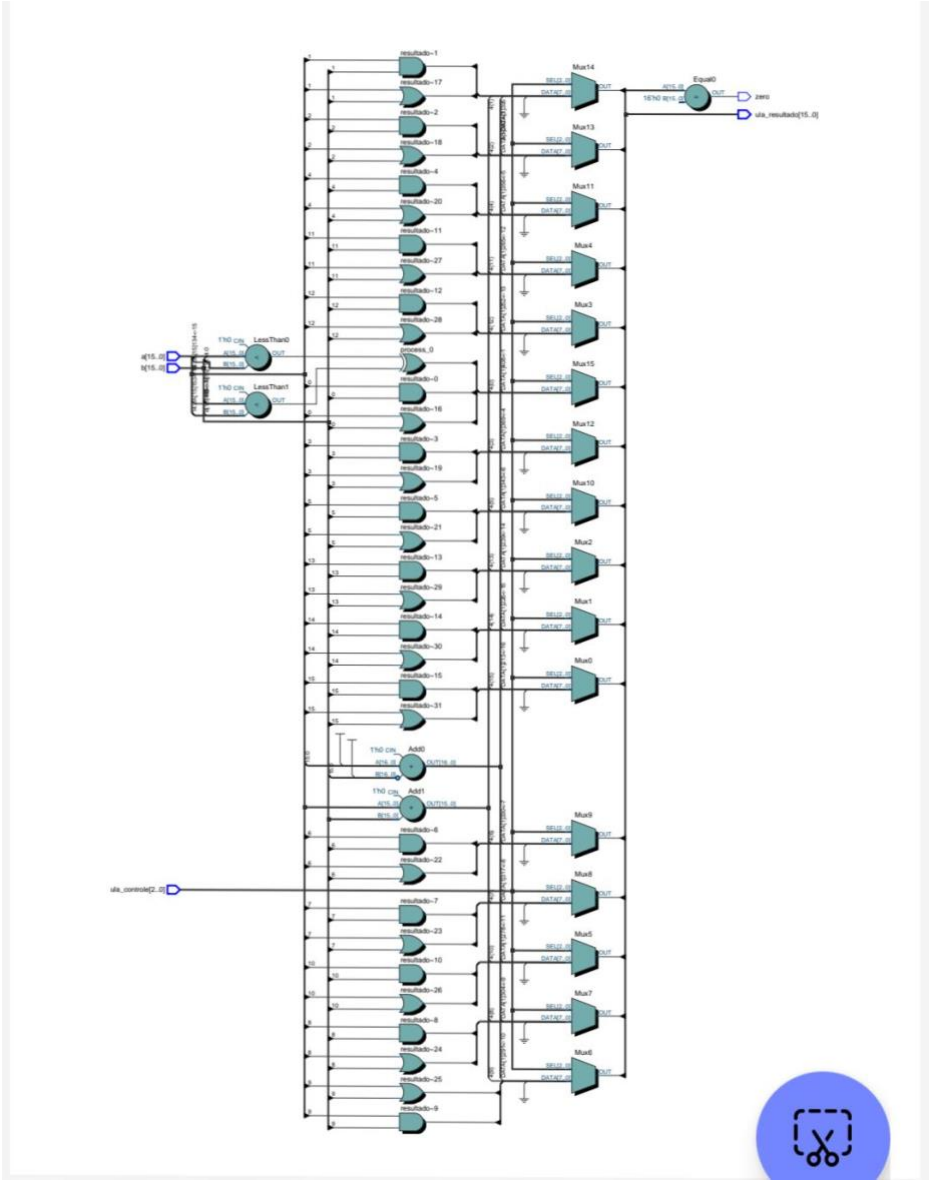


Figura 2 – ULA

Branco de Registradores
BDRegisterO BDRegister guarda valores e/ou resultados das operações realizadas pela ULA

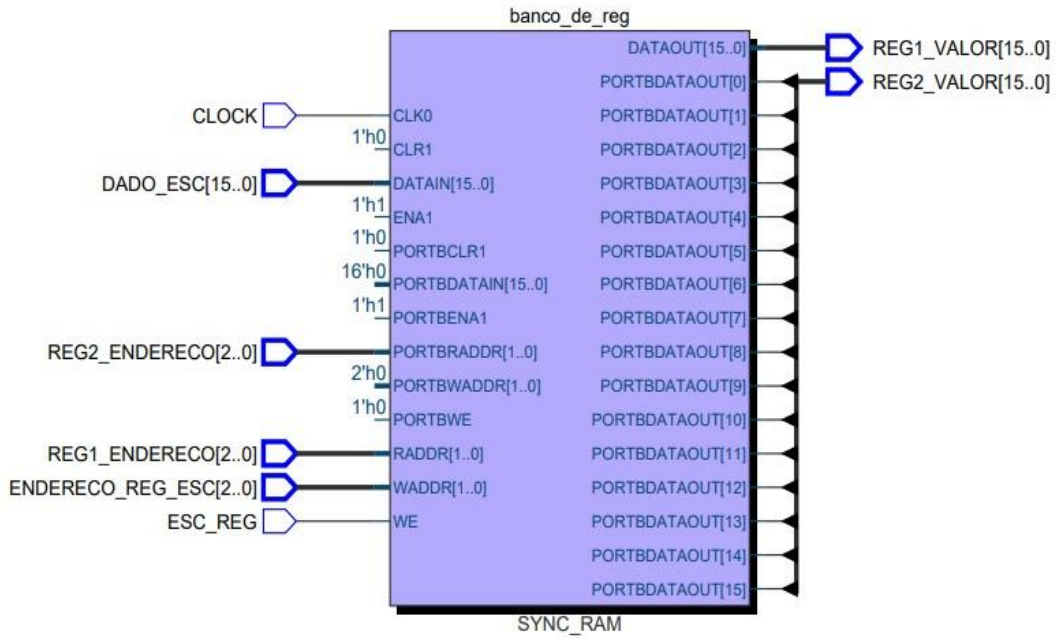


Figura 3- Banco De Registradores

Clock

Serve para manter os componentes funcionando apenas quando estiver ligado, portanto, Se for True/1.

1.2.2 Controle

O componente Control tem como objetivo realizar o controle de todos os componentes do processador de acordo com o opcode.

1.2.3 Memória de dados

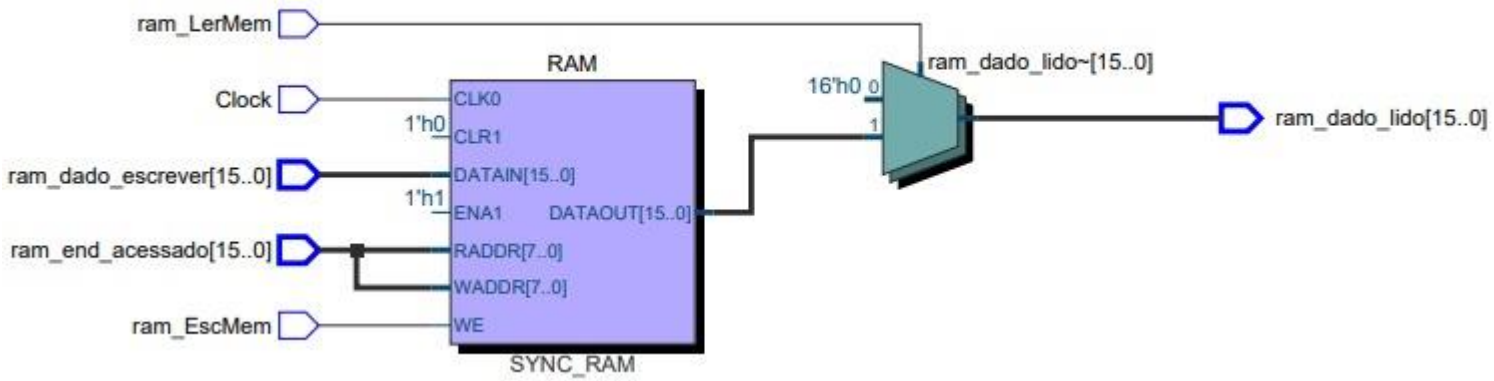


Figura 4 - RAM

Armazena os valores na memória e os disponibiliza para serem salvos nos registradores.

Tendo como entrada dado_escrever (que define o que vai ser salvo na memória);

Endereco – acaessado (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 o que foi lido na

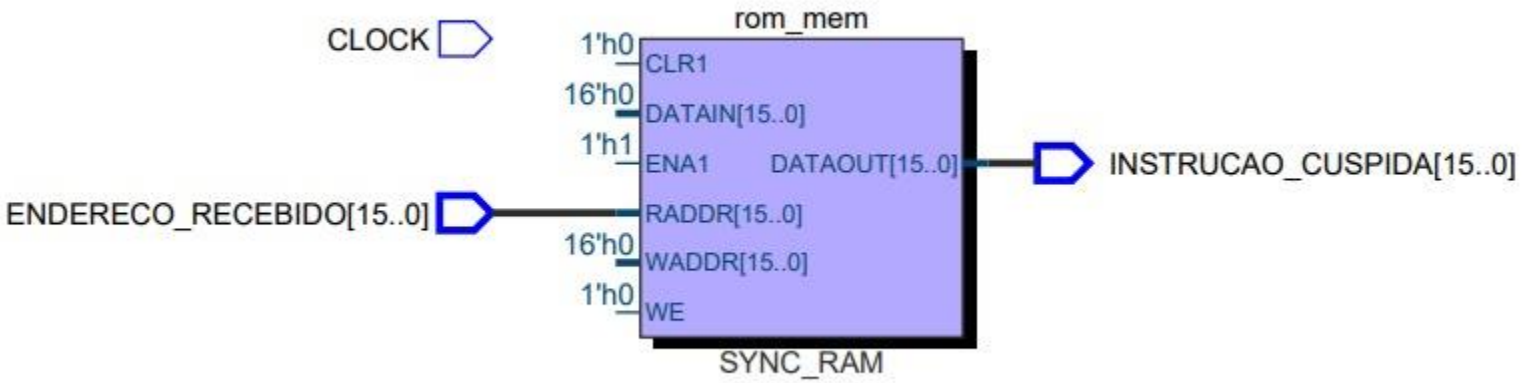
Memória – dado_lido.

1.2.4 Memória de Instruções

Onde 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 ROM recebe um

Endereço que vem do PC e executa o que está nesta instrução.



. Figura 5 -ROM

1.2.5 Somador

Soma cada PC para que ele execute as instruções

1.2.6 And

Verifica se vai ou não ocorrer um branc

1.2.7 Mux_2x1

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

1.2.8 PC

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

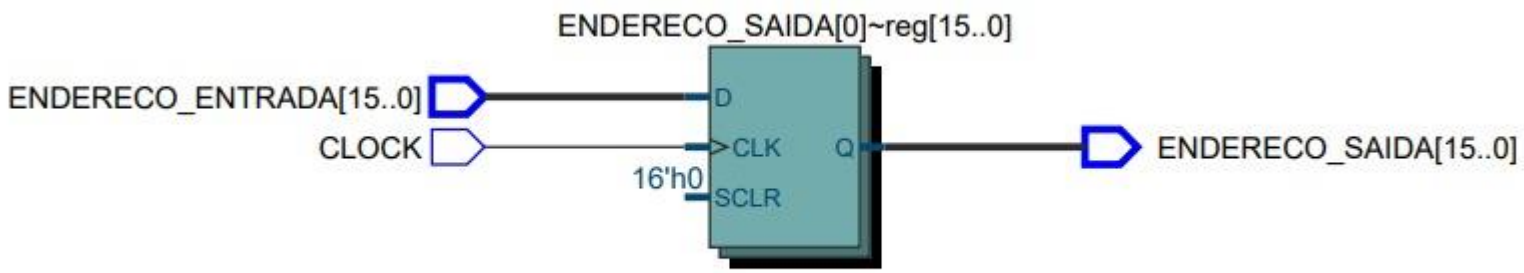


Figura 6 – Pc

1.2.9 ZERO

Identificador de resultado (1bit) para comparações (1 se verdade e 0 se falso)

1.3 Datapath

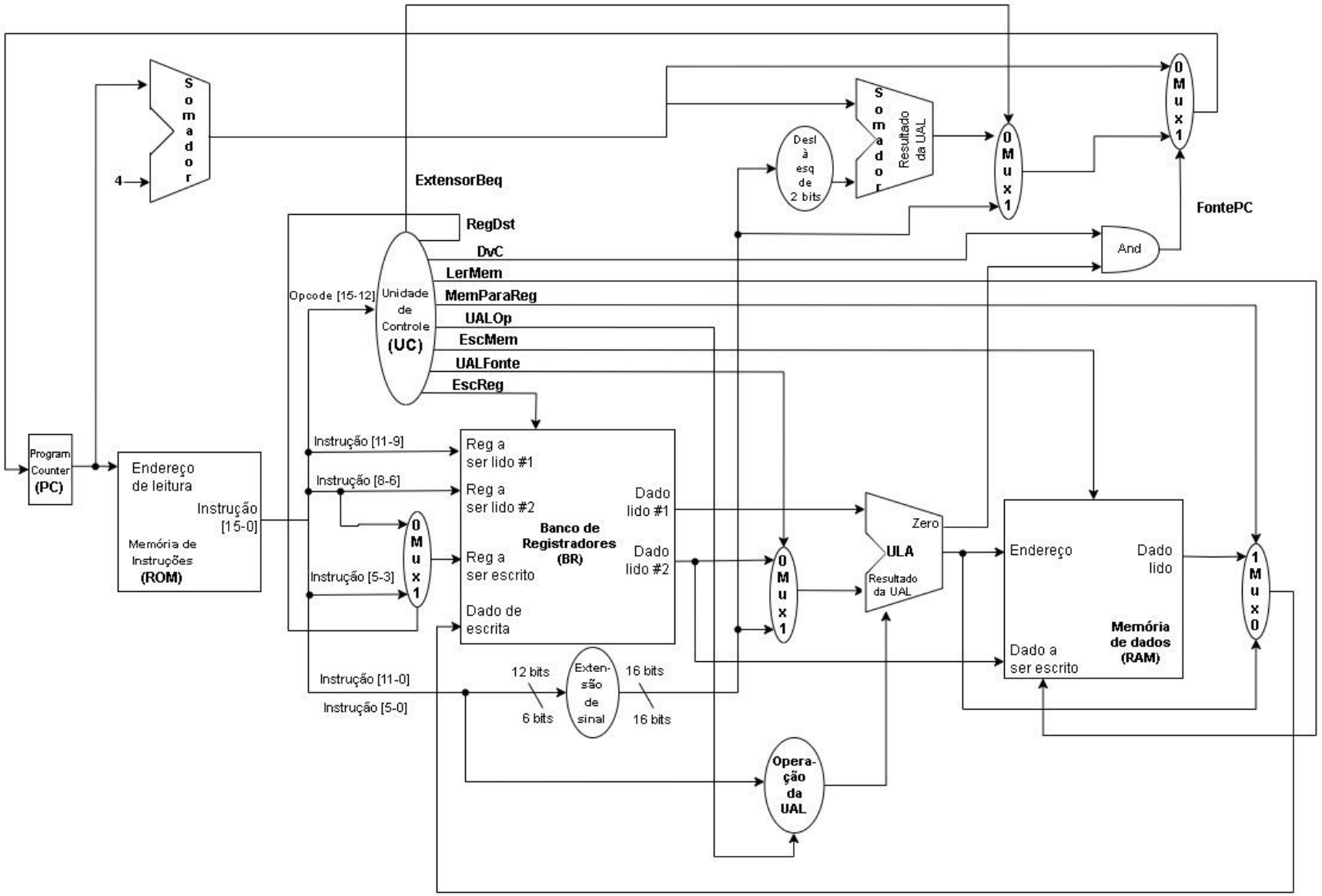

Somador
4
ExtensorBeq
Desl à esq de 2 bits
Somador
Resultado da UAL
0 M u x 1
0 M u x 1
FontePC
And
RegDst
DvC
LerMem
MemParaReg
UALOp
EscMem
UALFonte
EscReg
Opcode [15-12]
Unidade de Controle (UC)
Program Counter (PC)
Endereço de leitura
Instrução [15-0]
Memória de Instruções (ROM)
Instrução [11-9]
Instrução [8-6]
Instrução [5-3]
Instrução [11-0]
Instrução [5-0]
Reg a ser lido #1
Reg a ser lido #2
Reg a ser escrito
Dado de escrita
Banco de Registradores (BR)
Dado lido #1
Dado lido #2
0 M u x 1
0 M u x 1
Zero
ULA
Resultado da UAL
12 bits
6 bits
Exten-são de sinal
16 bits
16 bits
Opera-ção da UAL
Endereço
Dado lido
Dado a ser escrito
Memória de dados (RAM)
1 M u x 0

Figura 7 - Datapaht

1.4.Prolag. RLT View

RTL Viewer

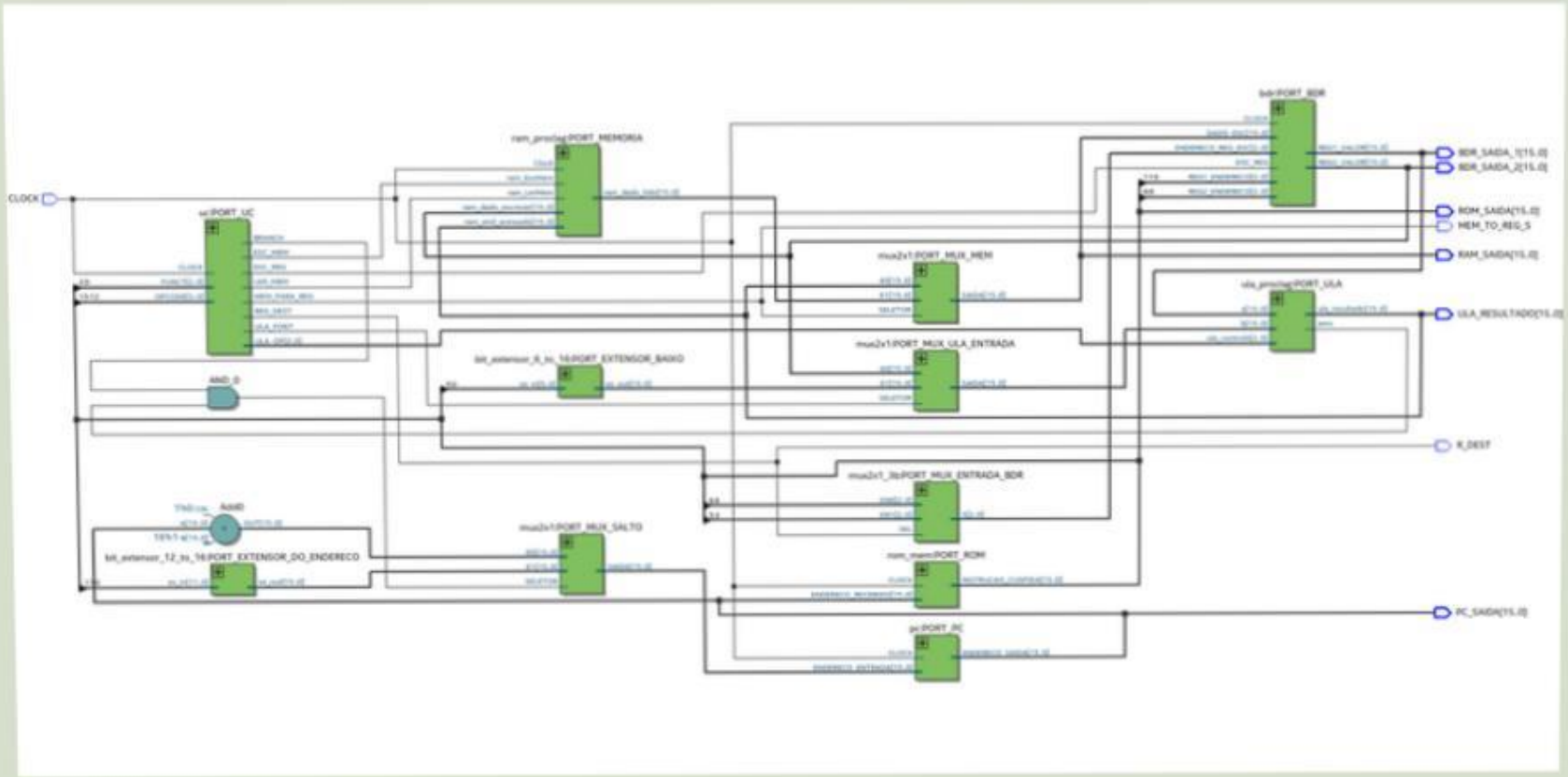

CLOCK
uc:PORT_UC
ram_princ:PORT_MEMORIA
mux2x1:PORT_MUX_MEM
mux2x1:PORT_MUX_ULA_ENTRADA
bit_extensor_6_to_16:PORT_EXTENSOR_BAIXO
mux2x1_3b:PORT_MUX_ENTRADA_BDR
mux2x1:PORT_MUX_SALTO
bit_extensor_12_to_16:PORT_EXTENSOR_DO_ENDERECO
rom_inst:PORT_ROM
pc:PORT_PC
bdr:PORT_BDR
ula_princ:PORT_ULA
BDR_SAIDA_1[15..0]
BDR_SAIDA_2[15..0]
ROM_SAIDA[15..0]
MEM_TO_REG_S
RAM_SAIDA[15..0]
ULA_RESULTADO[15..0]
K_DEST
PC_SAIDA[15..0]

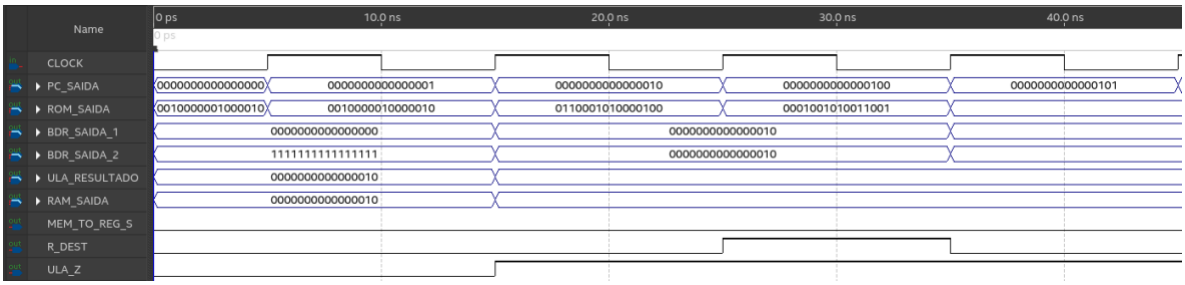
Figura 8 - Prolag (RLT View)

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.

Figura 9 – Instrução Tipo R

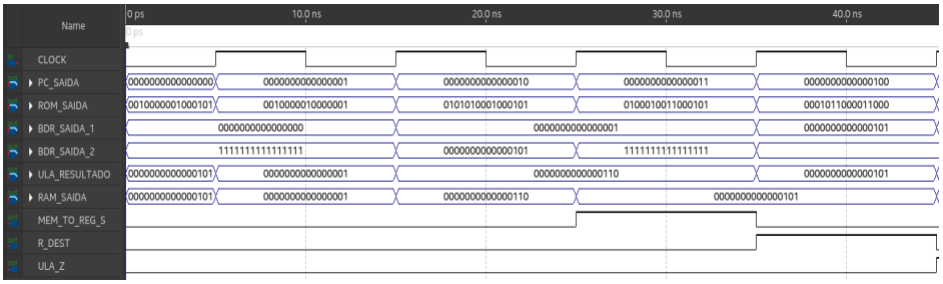
```
-- Teste do beq
0 => "0010000001000010", -- addi $t0, $zero, 2
1 => "0010000010000010", -- addi $t1, $zero, 2
2 => "0110001010000100", -- beq $t0, $t1, 100 (pula para a instrução 4)
3 => "0111000000000000", -- j 0 (ignorado)
4 => "0001001010011001", -- sub $t2, $t0, $t1
others => "0000000000000000"
```



No teste acima, vemos a inicialização de ambos os registradores $t0 e $t1 com a constante 2. Depois, é realizado um beq (branch equal) comparando os valores dos registradores inicializados, para, se forem iguais, realizar o salto para a instrução no endereço 4 da memória de instruções, o que nesse caso é verdadeiro. No final, temos a subtração do resultado de $t0 e $t1 sendo armazenado em $t2 (nesse caso, o resultado é zero).

Figura 10 – Instruções do tipo I

```
-- Teste Load e Store
0 => "0010000001000101", -- addi $t0, $zero, 5
1 => "0010000010000001", -- addi $t1, $zero, 1
2 => "0101010001000101", -- sw $t0, 5, $t1 (joga $t0 em 5 + $t1)
3 => "0100010011000101", -- lw $t2, 5, $t1 (carrega 5 + $t1 em $t2)
4 => "0001011000011000", -- add $t2, $t2, $zero
others => "0000000000000000"
```

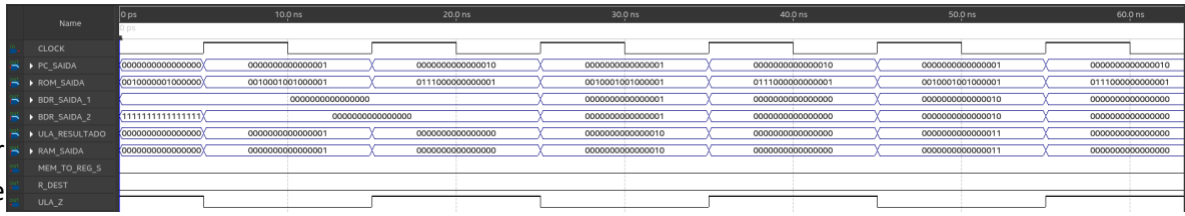


Agora, inicializamos $t0 $t1 com os valores 5 e 1, respectivamente. No endereço 2, armazenamos o valor de $t0 (5) em 5 + o valor de $t1 na memória de dados. Em seguida, carregamos o mesmo valor da memória(5 + $t1) para o registrador $t2. Afinal, realizamos uma soma de $t2 com $zero para visualizar que o valor do load foi carregado com sucesso.

Figura 11 – Instrução tipo J

```
-- Teste do j
0 => "0010000001000000", -- addi $t0, $zero, 0
1 => "0010001001000001", -- addi $t0, $t0, 1
2 => "0111000000000001", -- j 1
others => "0000000000000000"
```

Por fim, inicializamos o registrador $t0 com 0 (para limpar qualquer valor que pudesse ter ali), incrementamos ele em 1, através do addi $t0, $t0, 1, e pulamos de volta para a instrução 1 (o addi), repetindo indefinidamente, mostrando que o j está funcionando.



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

3 Considerações finais

Este trabalho apresentou o projeto e implementação do processador uniciclo de 16 bits referente ao projeto final de disciplina de Arquitetura e Organização de computadores. O processador contém todos os requisitos para que tenha essa alcunha pois contém operações aritméticas, memória de dados, condicionais, controle de dados e afins.

4 Referências bibliográficas

Organização e Arquitetura de Computadores – 4ª ed. David A. Patterson e John L. Hennessy

8 Bit Microprocessor Design Using VHDL. Disponível em: https://www.youtube.com/watch?v=tTlhIDgAGYY. Acesso em: 30 nov. 2023.

FPGA4student.com. VHDL code for MIPS Processor. Disponível em: https://www.fpga4student.com/2017/09/vhdl-code-for-mips-processor.html. Acesso em: 24 nov. 2023.