UFRR

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

ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

RELATÓRIO DO PROJETO: Dynasty

ALUNOS:

Marcus Vinicius Maia dos Santos - 2022007001
Filipe Gabriel Tomaz Brito - 2022010099
Eduardo Coutinho de Souza - 2022007610

Boa Vista - RR
2023

RESUMO

Este trabalho aborda o projeto e implementação de um processador 16bits, utilizando do software Quartus Prime, da linguagem Assembly MIPS e da linguagem VHDL (Very High-Speed Integration Circuit HDL). O processador em questão é um RISC (Reduced Instruction Set Computer) de 16 bits.

O relatório então servirá de instrumento de avaliação dos alunos para a disciplina de AOC (Arquitetura e Organização de Computadores), ministrada pelo professor Herbert Oliveira Rocha.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1. ESPECIFICAÇÃO

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

1.1. Plataforma de Desenvolvimento

Para a implementação do processador - foi utilizada a IDE Quartus Prime:

Figura 1 - Especificações no Quartus Prime

Flow Status	Successful - Tue Nov 28 21:44:40 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	ProjetoAOC
Top-level Entity Name	CPU_DYNASTY
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	9 / 56,480 (< 1 %)
Total registers	40
Total pins	130 / 268 (49 %)
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 %)
Total HSSI PMA TX Serializers	0 / 6 (0 %)
Total PLLs	0 / 13 (0 %)
Total DLLs	0 / 4 (0 %)

Fonte: Elaborada pelos autores.

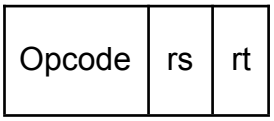
1.2. Conjunto de Instruções

O processador possui 2 registradores: $s0 e $s1. Assim como 10 formatos de instruções de 16 bits cada, instruções do tipo R, I e J. Seguem algumas considerações sobre as estruturas contidas nas instruções:

- Opcode: indica ao processador qual a instrução a ser executada;
- 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.

<u>Tipos de Instruções</u>:

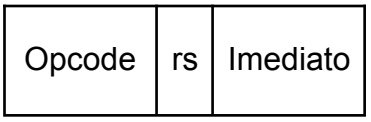
- Tipo R: Este tipo de instrução trata de operações aritméticas.
    - Formato para escrita de código na linguagem MIPS:



    - Formato para escrita em código binário:

Instrução do tipo R		
Opcode	rs	rt
4bits	6bits	6bits
15-12	11-6	5-0

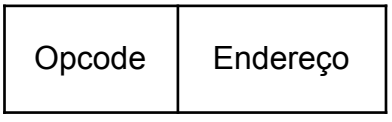
- Tipo I: Este tipo de instrução aborda carregamentos diretos na memória.
    - Formato para escrita de código na linguagem MIPS:



    - Formato para escrita em código binário:

Instrução do tipo I		
Opcode	rs	Imediato
4bits	6bits	6bits
15-12	11-6	5-0

- Tipo J: Este tipo de instrução é responsável por desvios condicionais e incondicionais.
    - Formato para escrita de código na linguagem MIPS:



    - Formato para escrita em código binário:

Instrução do tipo J	
Opcode	Endereço
4bits	12bits
15-12	11-0

Visão geral das instruções do Processador Dynasty:

O número de bits do campo Opcode das instruções é igual a quatro, sendo assim obtemos um total de 16 Opcodes que são distribuídos entre as instruções, assim como é apresentado na Tabela 1.

Tabela 1 – Opcodes suportados pelo Processador Dynasty

Opcode	Sintaxe	Formato	Significado	Exemplo
0000	**ADD**	**R**	**Soma**	**add $s0, $s1**
0001	**ADDI**	**I**	**Soma imediata**	***addi* $s0, 1**
0010	**SUB**	**R**	**Subtração**	***sub* $s0, $s1**
0011	**SUBI**	**I**	**Subtração imediata**	***sub*i $s0, 3**
0100	**LW**	**I**	**Load**	***lw* $s0 ram (00)**
0101	**SW**	**I**	**Store**	***sw* $s0 ram (00)**
0110	**LI**	**I**	**Load imediato**	***li* $s0 2**
0111	**BEQ**	**J**	**Branch Equal**	***beq* endereço**
1000	**IF**	**J**	**If Equal**	***If* $s0 $s1**
1001	**JUMP**	**J**	**Jump**	***j* endereço (0000)**

1.3. Descrição do Hardware

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

1.3.1. ALU

O componente ALU (Unidade Lógica Aritmética) tem como principal objetivo efetuar as principais operações aritméticas (considerando apenas resultados inteiros), dentre elas: soma, subtração e adicionalmente operações de comparação de valor como o BEQ.

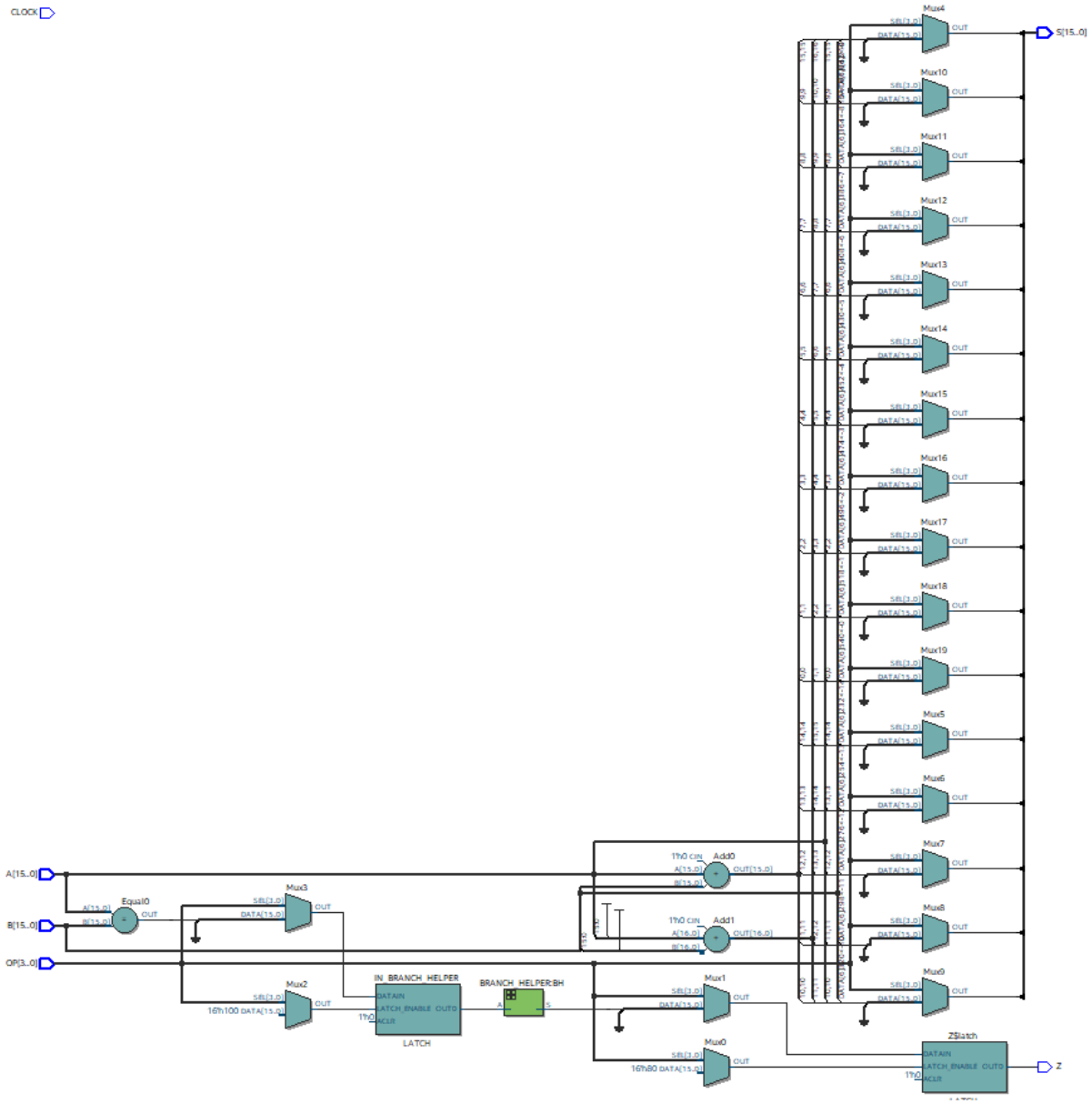
A ALU possui 4 valores como entrada:

- A: dado de 2 bytes;
- B: dado de 2 bytes;
- CLOCK: dado de 1 bit;
- OP: opcode de 4 bits.

A ALU possui 3 valores como saída:

- S: resultado de 2 bytes;
- Z: resultado de 1 bit, utilizado para verificar se o valor retornado é zero;
- OVERFLOW: resultado de 1 bit utilizado para verificar se a operação resulta num overflow.

Figura 2 - RTL Viewer da ALU



Fonte: Elaborada pelos autores.

1.3.2. BANCO_REGISTRADORES

O componente BANCO_REGISTRADORES tem como principal objetivo escrever, ler e armazenar valores nos registradores.

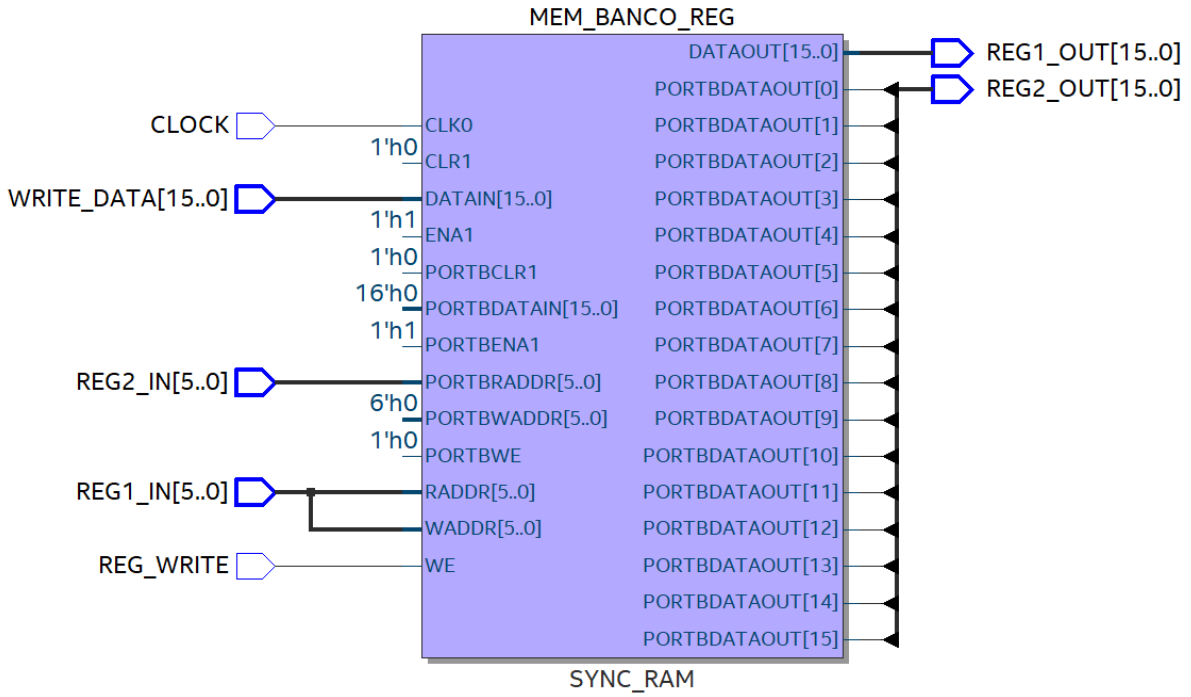
O BANCO_REGISTRADORES possui 5 valores como entrada:

- CLOCK: dado de 1 bit;
- REG_WRITE: dado de 1 bit que indica se tem escrita em registradores;
- REG1_IN e REG2_IN: dados de 4 bits que indicam sobre quais registradores devem ser executados as operações.

O BANCO_REGISTRADORES possui 2 valores como saída:

- REG1_OUT e REG2_OUT: dados de 2 byte armazenados nos registradores.

Figura 3 - RTL Viewer do BANCO_REGISTRADORES



Fonte: Elaborada pelos autores.

1.3.3. CONTADOR_SINCRONO

O componente CONTADOR_SINCRONO tem como objetivo somar 1 ao PC, avançando assim para a próxima linha de código do programa na memória ROM.

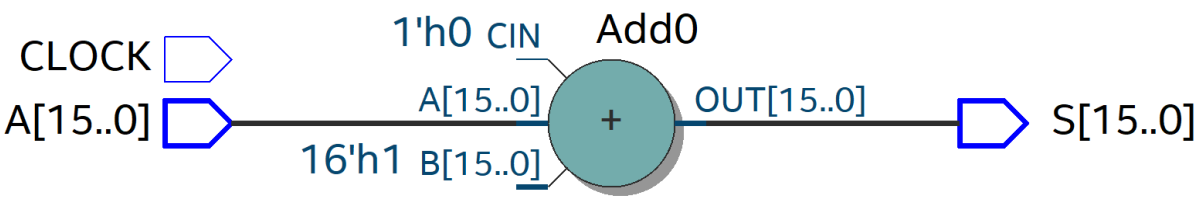
O CONTADOR_SINCRONO possui 2 valores como entrada:

- A: dado de 2 bytes equivalente ao endereço no PC;
- CLOCK: dado de 1 bit.

O CONTADOR_SINCRONO possui uma saída como valor:

- S: PC + 1.

Figura 4 - RTL Viewer do CONTADOR_SINCRONO



Fonte: Elaborada pelos autores.

1.3.4. DIVISOR

O componente DIV_INSTRUCAO divide a instrução recebida em 4 trilhas, que são utilizadas para definir qual o tipo de instrução a ser executada e quais seus operandos.

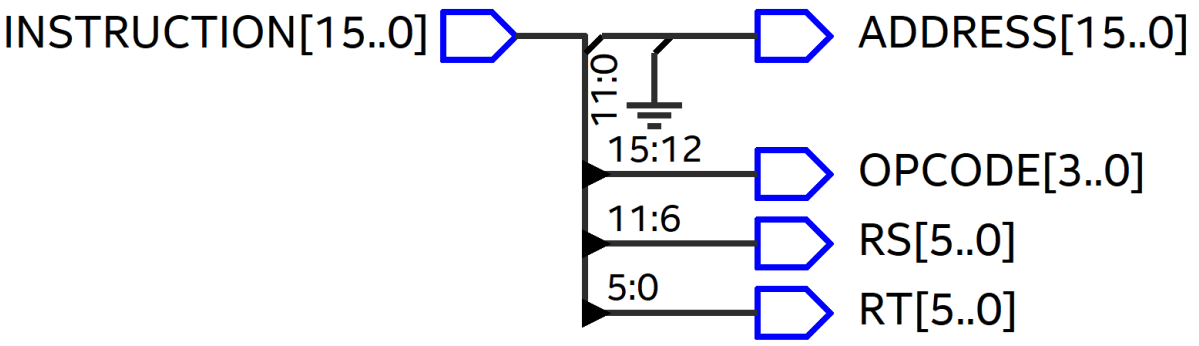
O componente DIV_INSTRUCAO recebe como entrada:

- INSTRUCTION: dado de 2 bytes com a instrução.

O componente DIV_INSTRUCAO tem como saída:

- ADDRESS: dado de 4 bits com endereço;
- OPCODE: dado de 4 bits com o opcode;
- RS: dado de 4 bits com o primeiro operando;
- RT: dado de 4 bits com o segundo operando.

Figura 5 - RTL Viewer do DIVISOR



Fonte: Elaborada pelos autores.

1.3.5. EXTENSOR_6X16

O componente EXTENSOR_6X16 extende um sinal de 6 bits para um de 2 bytes.

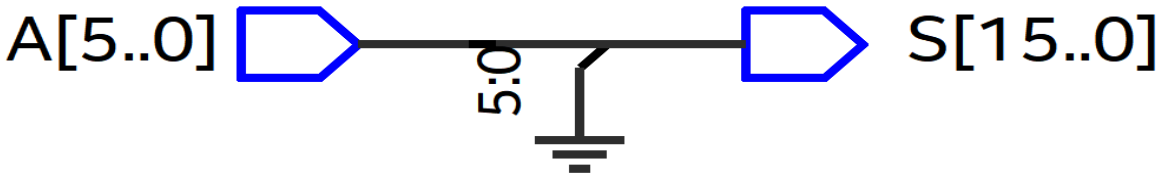
O componente EXTENSOR_6X16 recebe como entrada:

- A: dado de 6 bits a ser extendido.

O componente EXTENSOR_6X16 tem como saída:

- S: dado de 2 bytes extendido.

Figura 6 - RTL Viewer do EXTENSOR_6X16



Fonte: Elaborada pelos autores.

1.3.6. MUX_2X1

O componente MUX_2X1 seleciona um entre dois dados de 1 byte com base num seletor.

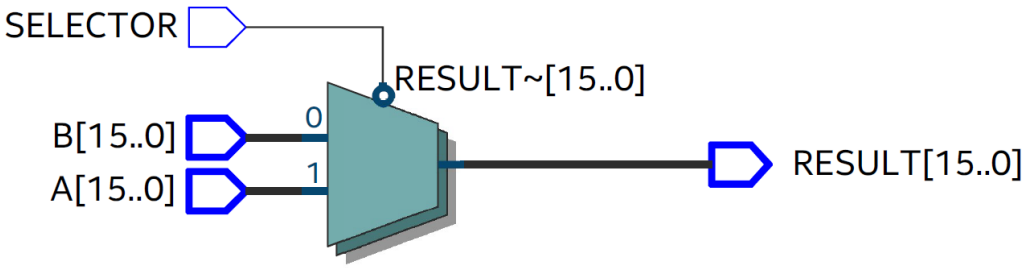
O componente MUX_2X1 recebe como entrada:

- A e B: dados de 2 bytes;
- SELECTOR: dado de 1 bit.

O componente MUX_2X1 tem como saída:

- RESULT: dado de 2 bytes que foi selecionado.

Figura 8 - RTL Viewer do MUX_2X1



Fonte: Elaborada pelos autores

1.3.7. PC

O componente PC(PROGRAM_COUNTER) é responsável por passar a linha de código do programa que deve ser executada.

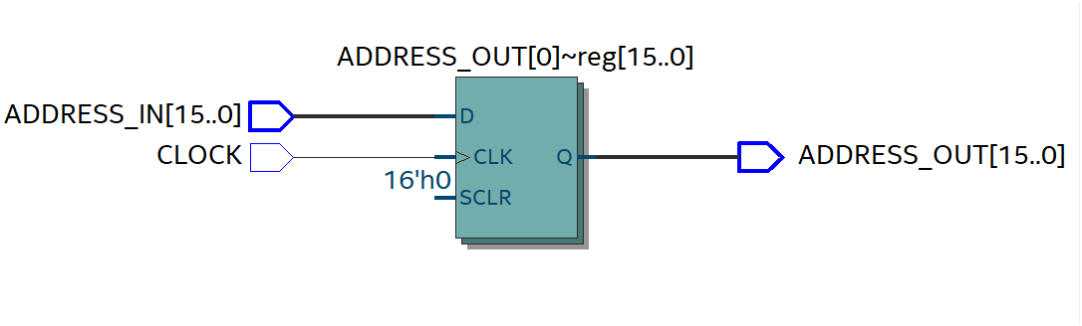
O componente PC recebe como entrada:

- CLOCK: dado de 1 bit;
- ADDRESS_IN: dado de 2 bytes com a linha atualizada.

O componente PC tem como saída:

- ADDRESS_OUT: dado de 2 bytes com a linha atual.

Figura 9 - RTL Viewer do PC



Fonte: Elaborada pelos autores

1.3.8. RAM

O componente RAM é responsável por armazenar e ler até 8 posições de 16 bits por meio de instruções load e store.

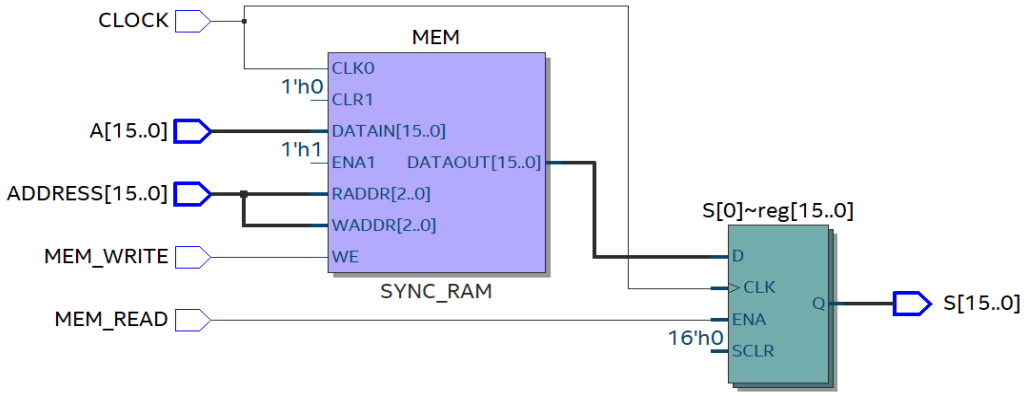
O componente RAM recebe como entrada:

- CLOCK: dado de 1 bit;
- A: dado de 2 bytes a ser escrito;
- ADDRESS: dado de 2 bytes com endereço;
- MEM_WRITE: dado de 1 bit que serve como flag;
- MEM_READ: dado de 1 bit que serve como flag.

O componente RAM tem como saída:

- S: dado de 2 bytes com o resultado.

Figura 10 - RTL Viewer da RAM



Fonte: Elaborada pelos autores

1.3.9. ROM

O componente ROM é responsável por armazenar o programa, com até 256 linhas de código.

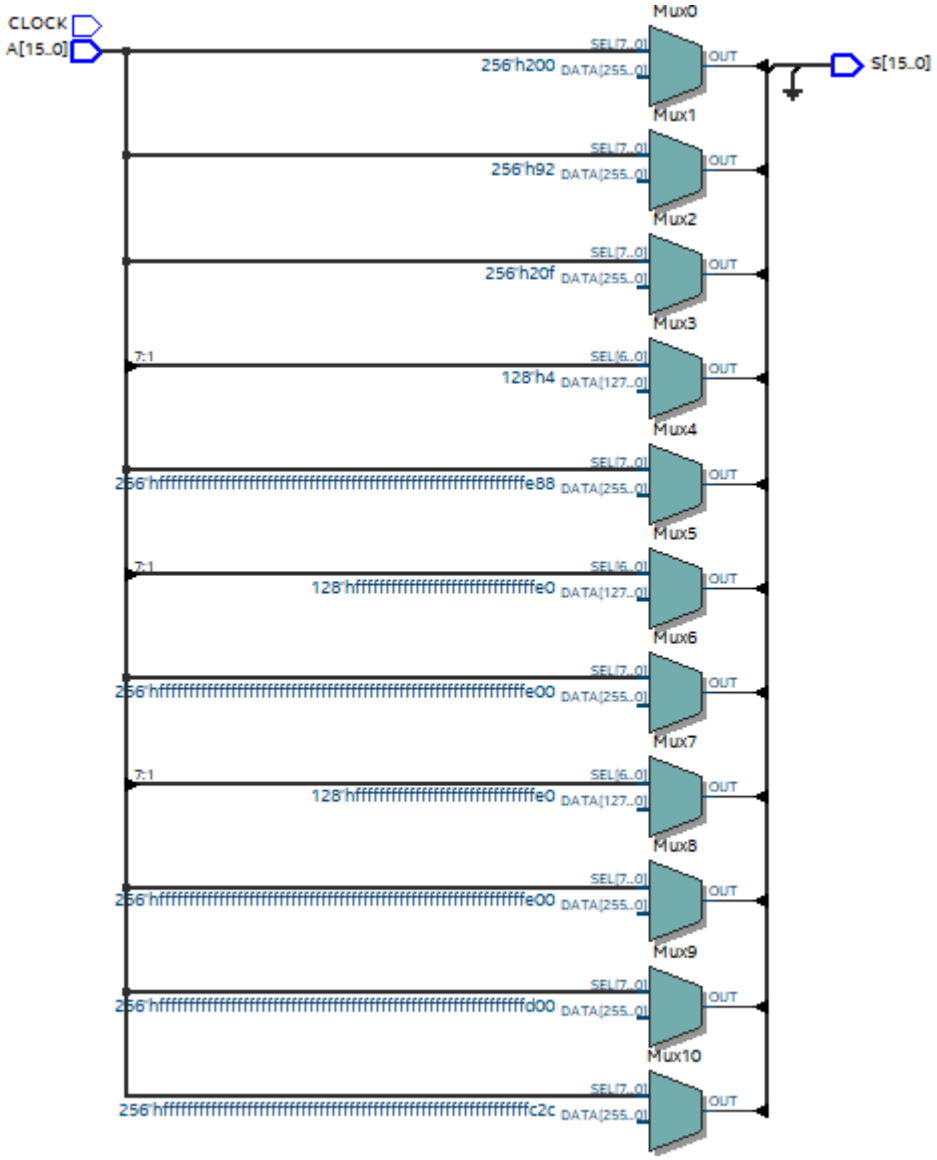
O componente ROM recebe como entrada:

- CLOCK: dado de 1 bit;
- A: dado de 2 bytes com o endereço da linha a ser lida.

O componente ROM tem como saída:

- S: dado de 2 bytes armazenado naquela linha.

Figura 11 - RTL Viewer da ROM



Fonte: Elaborada pelos autores

1.3.10. UNIDADE_DE_CONTROLE

O componente UNIDADE_DE_CONTROLE tem como principal objetivo administrar as flags necessárias para cada tipo de instrução.

O componente UNIDADE_DE_CONTROLE recebe como entrada:

- CLOCK: dado de 1 bit;
- OPCODE: dado de 4 bits, indicando a operação a ser realizada;

O componente UNIDADE_DE_CONTROLE tem como saída:

- JUMP: dado de 1 bit que serve de flag se há uma operação do tipo jump;
- BRANCH: dado de 1 bit que serve de flag se há uma operação característica de instrução tipo J;
- MEMREAD: dado de 1 bit que serve de flag se há leitura de memória;

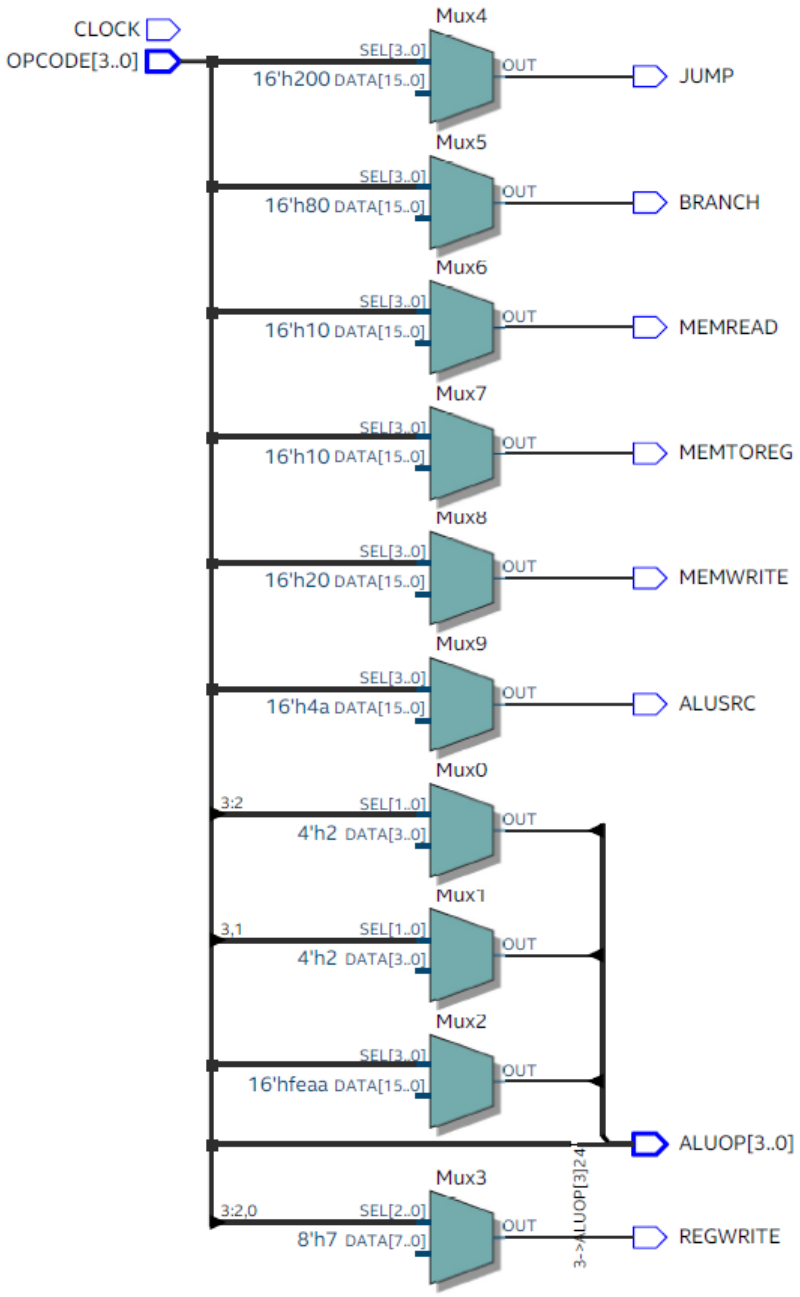
- MEMTOREG: dado de 1 bit que serve de flag se há escrita de um dado da memória num registrador;
- ALUOP: dado de 4 bits que indica a operação a ser realizada na ALU;
- MEMWRITE: dado de 1 bit que serve de flag se há escrita na memória;
- ALUSRC: dado de 1 bit que serve de flag se há necessidade de executar operação com registrador ou imediato;
- REGWRITE: dado de 1 bit que serve de flag se há escrita de dados no banco de registradores.

Tabela 2 – Relações entre OPCODES e flags na UNIDADE_DE_CONTROLE

OPCODE	JUMP	BRANCH	MEMREAD	MEMTOREG	ALUOP	MEMWRITE	ALUSRC	REGWRITE
0000	0	0	0	0	0	0	0	1
0001	0	0	0	0	0	0	1	1
0010	0	0	0	0	0	0	0	1
0011	0	0	0	0	0	0	1	1
0100	0	0	1	1	0	0	0	1
0101	0	0	0	0	0	1	0	0
0110	0	0	0	0	0	0	1	1
0111	0	1	0	0	0	0	0	0
1000	0	0	0	0	0	0	0	0
1001	1	0	0	0	0	0	0	0

Fonte: Elaborada pelos autores.

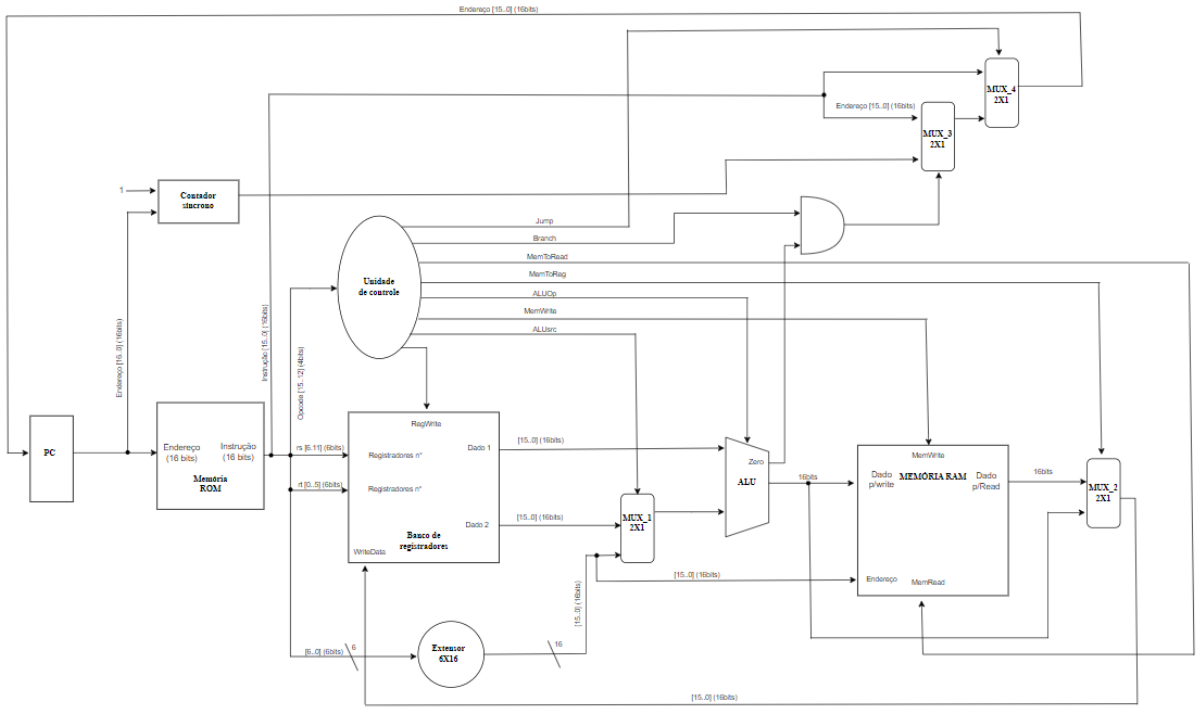
Figura 14 - RTL Viewer da UNIDADE_DE_CONTROLE



Fonte: Elaborada pelos autores

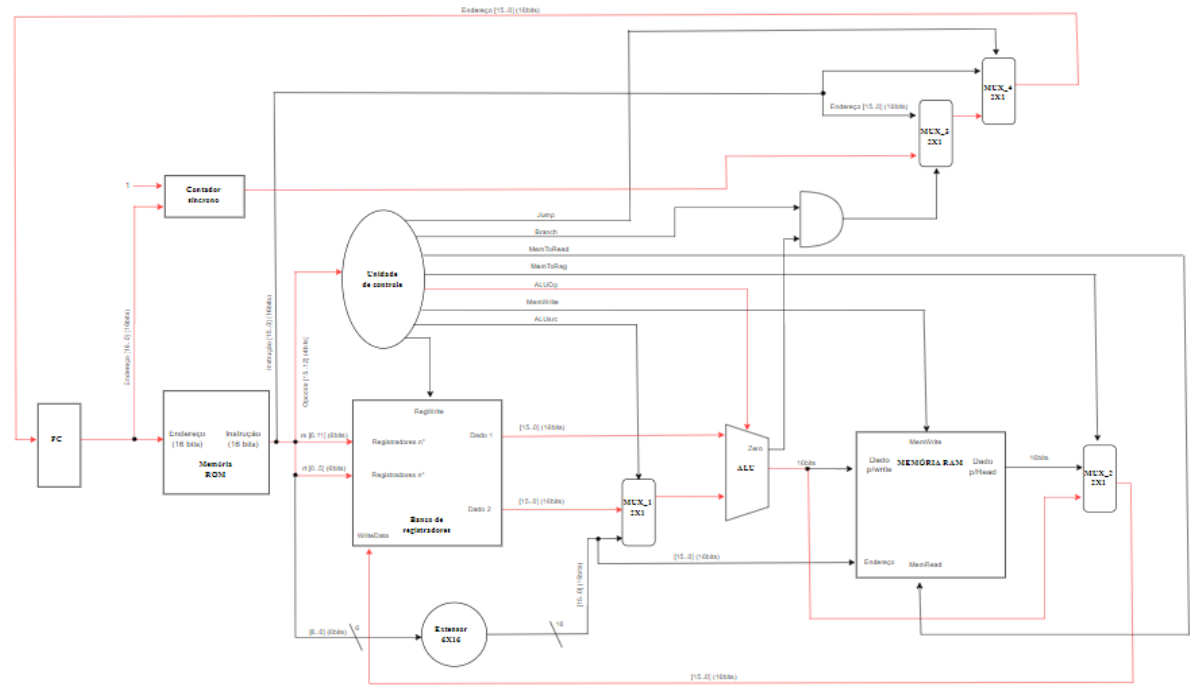
1.4. Datapath

Figura 15 - Datapath



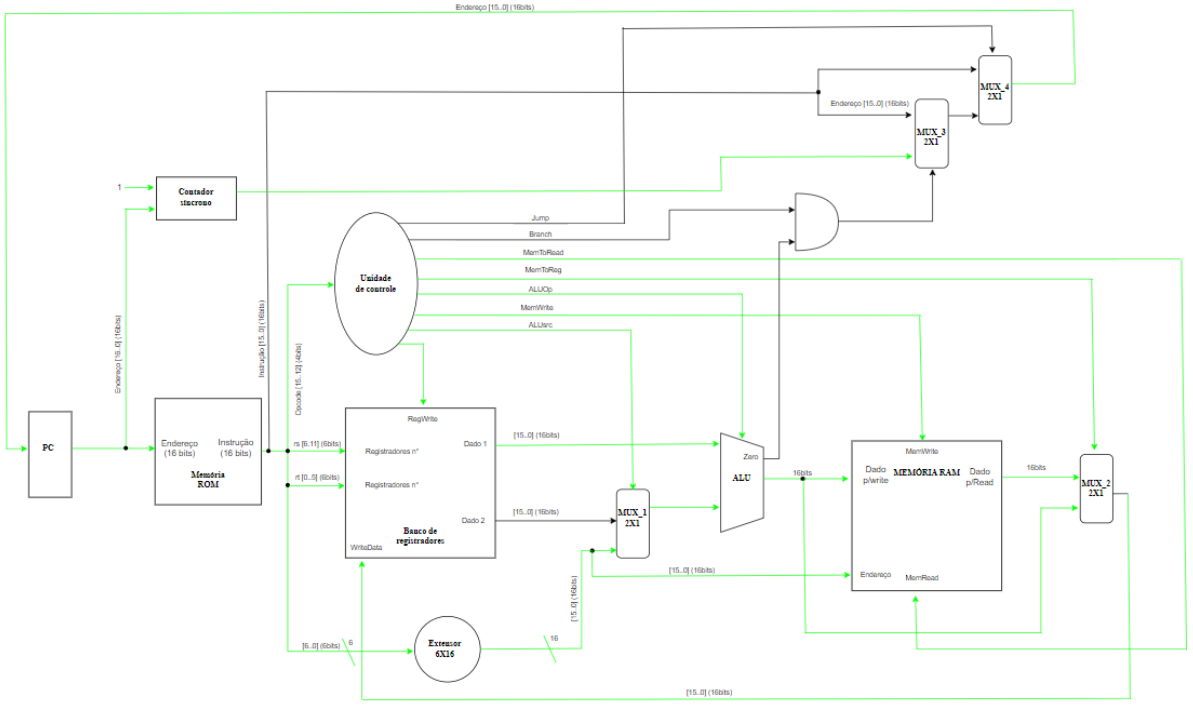
Fonte: Elaborada pelos autores

Figura 16 - Datapath para instrução tipo R



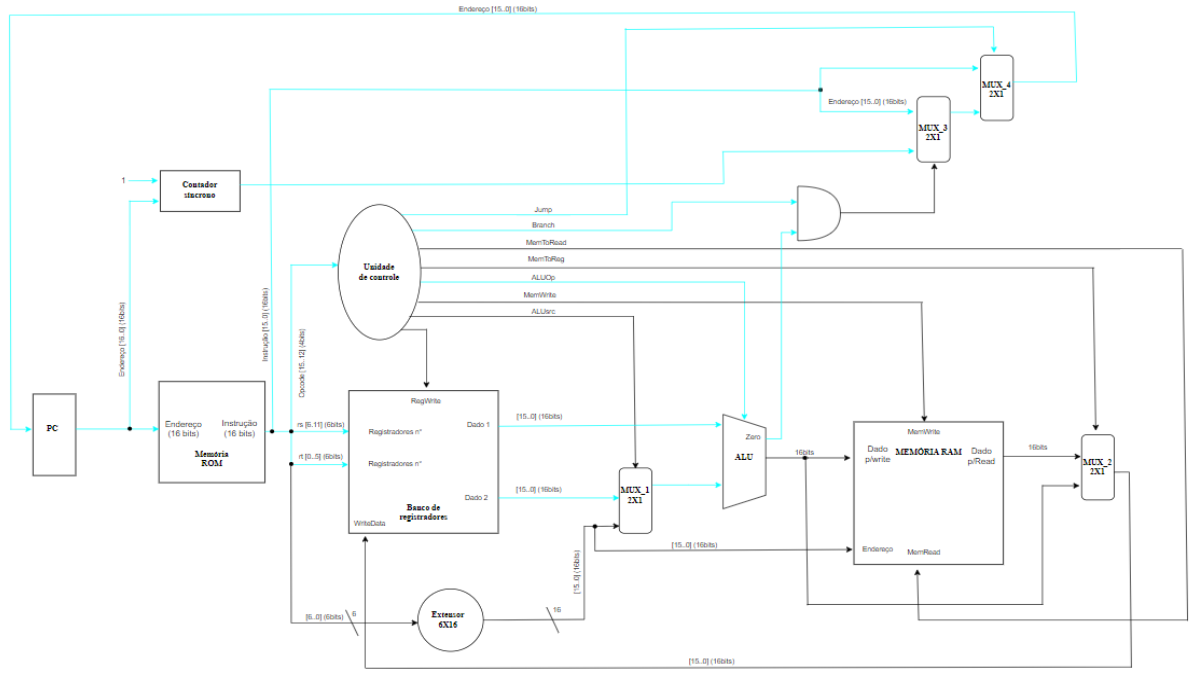
Fonte: Elaborada pelos autores

Figura 17 - Datapath para instrução tipo I



Fonte: Elaborada pelos autores

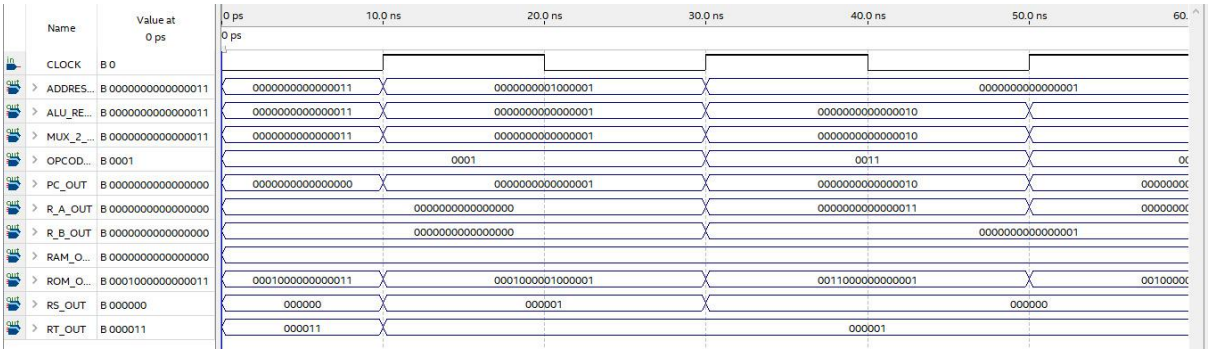
Figura 18 - Datapath para instrução tipo J

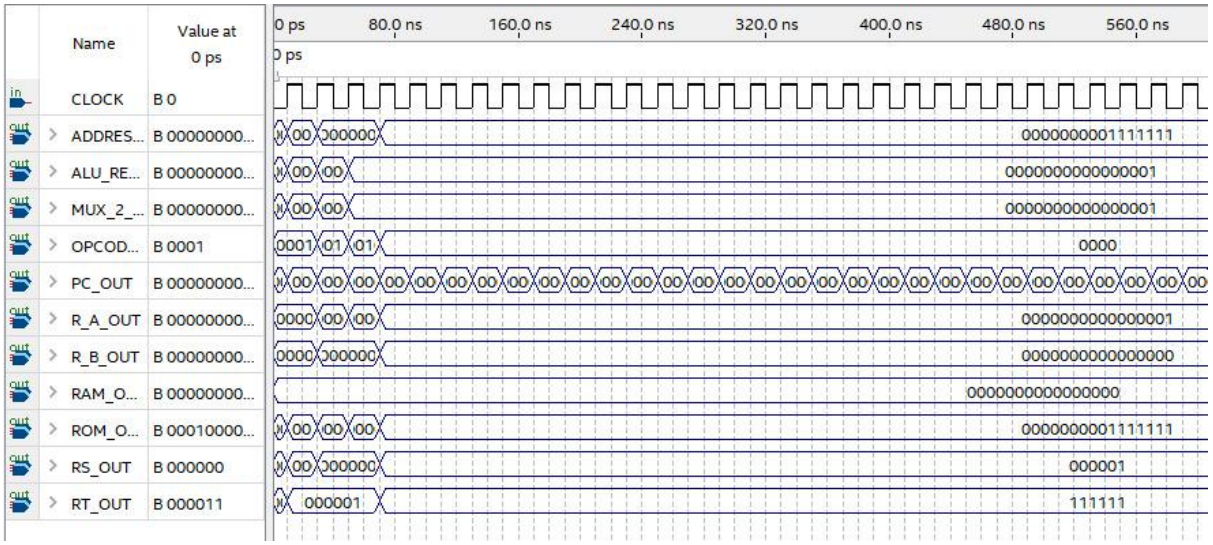


Fonte: Elaborada pelos autores

2. Simulações e Testes

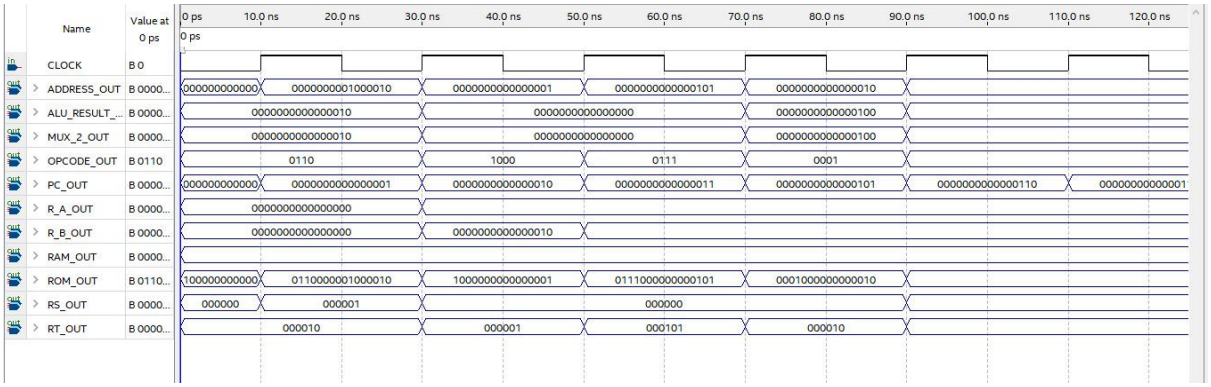
2.1. Teste ADDI, SUB E SUBI



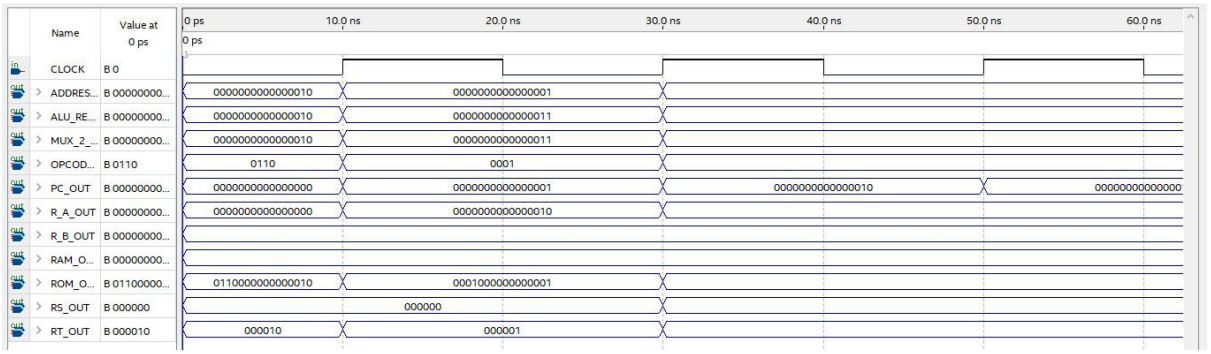


2.2. Teste ADD e ADDI

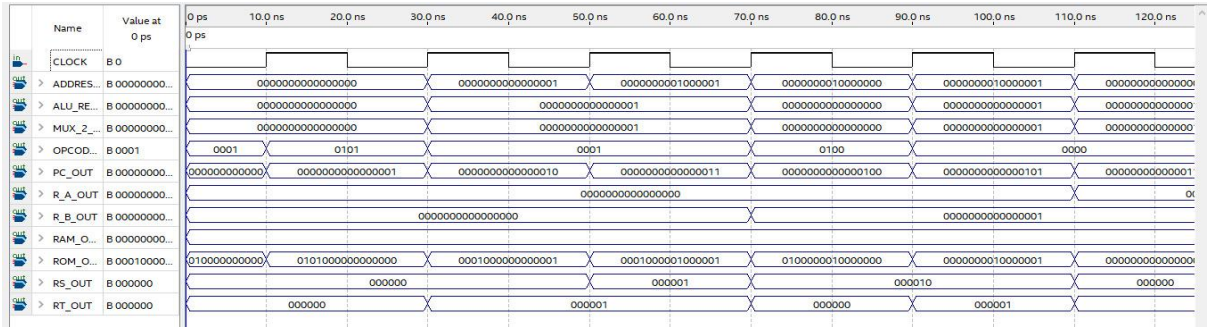
2.3. Teste BEQ

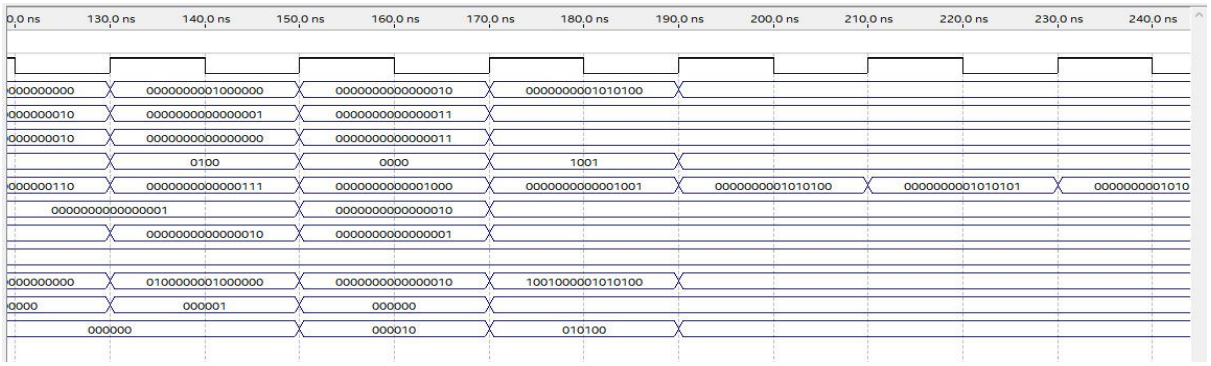


2.4. Teste LI



2.5. Teste FIBONACCI





3. Considerações Finais

Este trabalho apresentou o projeto e implementação do processador de 16 bits denominado de Dynasty, que foi uma rica oportunidade para pôr em prática o que nos foi ensinado na disciplina de AOC, e esclarecer diversos pontos que antes eram difíceis de se entender. “Uma das maiores dificuldades encontradas foi justamente a divisão bits”.

4. Repositório

https://github.com/marcusv0/AOC_EduardoFilipeMarcus_UFRR_2023