Relatório de Implementação do Caminho de Dados do MIPS*

Jessé Pires Barbato Rocha, Jhonatan Guilherme de Oliveira Cunha
Coordenação do Curso de Bacharelado em Ciência da Computação - COCIC
Universidade Tecnológica Federal do Paraná - UTFPR
Campus Campo Mourão
Campo Mourão, Paraná, Brasil
jesserocha@alunos.utfpr.edu.br e jhonatancunha@alunos.utfpr.edu.br

Resumo

Este trabalho consiste em um relatório do projeto final da disciplina Arquitetura e Organização de Computadores, ministrada pelo professor Paulo Cesar Gonçalves. O objetivo do projeto, dividido em duas etapas, era implementar o caminho de dados do processador MIPS, bem como o circuito combinacional da unidade de controle principal e o controle da ALU. Por fim, o processador, implementado no software Logisim, deveria suportar um subconjunto de instruções do mesmo. Este relatório, portanto, apresenta o processo para obtenção dos resultados desejados nas duas etapas, além de testes realizados para garantir a corretude funcional do conjunto de instruções implementado.

1 Introdução

O presente trabalho apresenta um caminho de dados de um processador ***MIPS***, ou seja, uma arquitetura onde a CPU somente utiliza registradores para realizar suas operações lógicas e aritméticas [1]. Conforme solicitado, foi implementado um subconjunto de instruções *MIPS*, sendo elas: **(i)** instruções de acesso à memoria: *load word* **(lw)** e *store word* **(sw)**; **(ii)** instruções lógicas e aritméticas: *add*, *sub*, *and*, *or* e *stl*; **(iii)** instruções de desvio: *jump* (**j**) e *branch equal* (***beq***). Vale ressaltar que, para o desenvolvimento do trabalho, foi utilizado o *Software Logisim*.

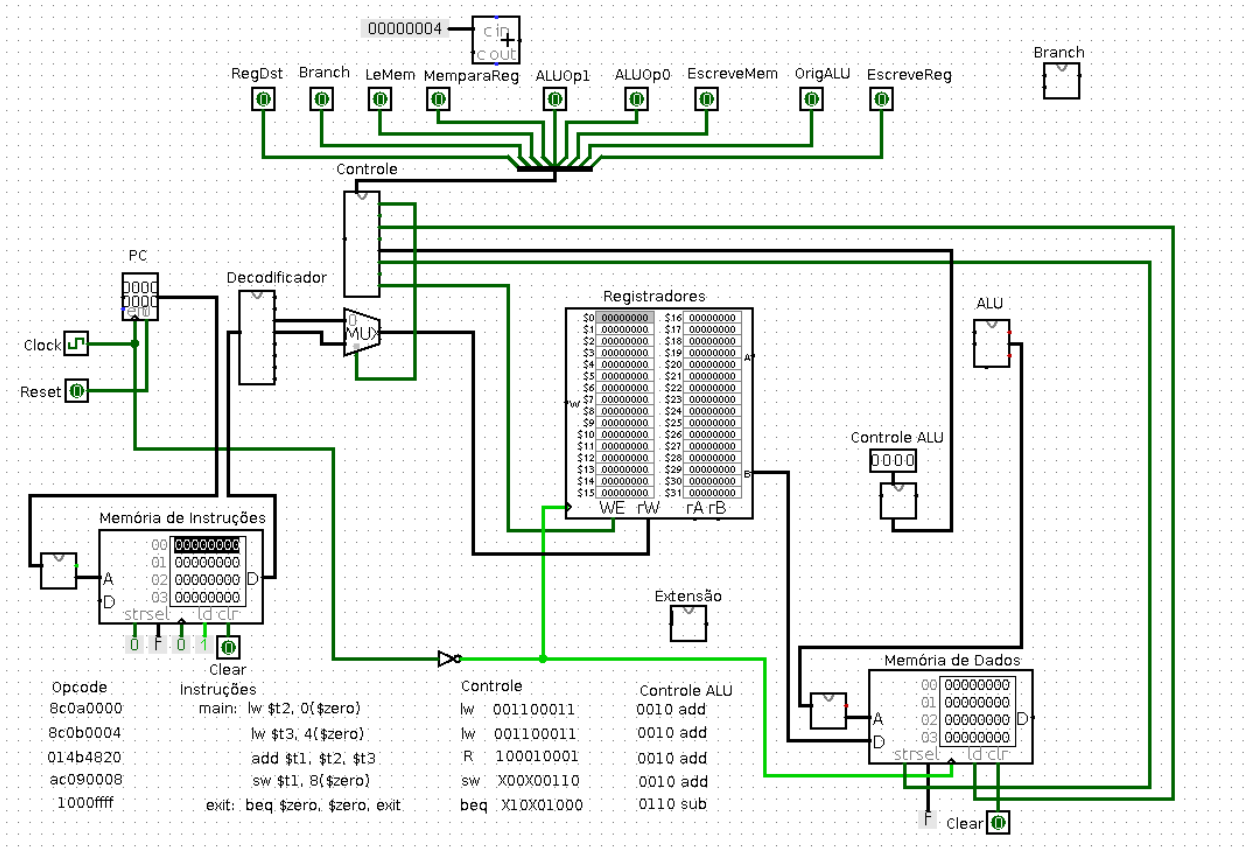
O trabalho está dividido da seguinte maneira: a Seção 2 apresenta o que foi solicitado na primeira etapa do projeto; a Seção 3 apresenta os resultados obtidos do desenvolvimento do que foi solicitado, bem como discussões a respeito do processo e os testes de verificação das instruções; a Seção 4 apresenta o que foi solicitado na segunda etapa do trabalho; a Seção 5 apresenta o resultado obtido, explanação à respeito dos detalhes da implementação e também, testes para verificação da corretude funcional das instruções. Por fim, a Seção 6 apresenta as conclusões chegadas no desenvolvimento do projeto, bem como discussões à respeito do mesmo.

2 Primeira Etapa

A princípio, foi passado pelo professor uma espécie de esqueleto do caminho de dados do processador, veja na Figura 1. A entrega foi dividida em duas etapas. A primeira delas consistia em fazer as conexões faltantes entre os módulos do circuito. Também foi pedido nesta etapa, que se implementasse um controle principal e da *ALU* manualmente ***add***, ***sub***, ***and***, ***or***, ***slt***, ***lw***, ***sw*** e ***beq***. Por fim, era necessário testar o que foi, utilizando os testes disponibilizados pelo professor.

*Trabalho desenvolvido para a disciplina de BCC33B – Arquitetura e Organização de Computadores.

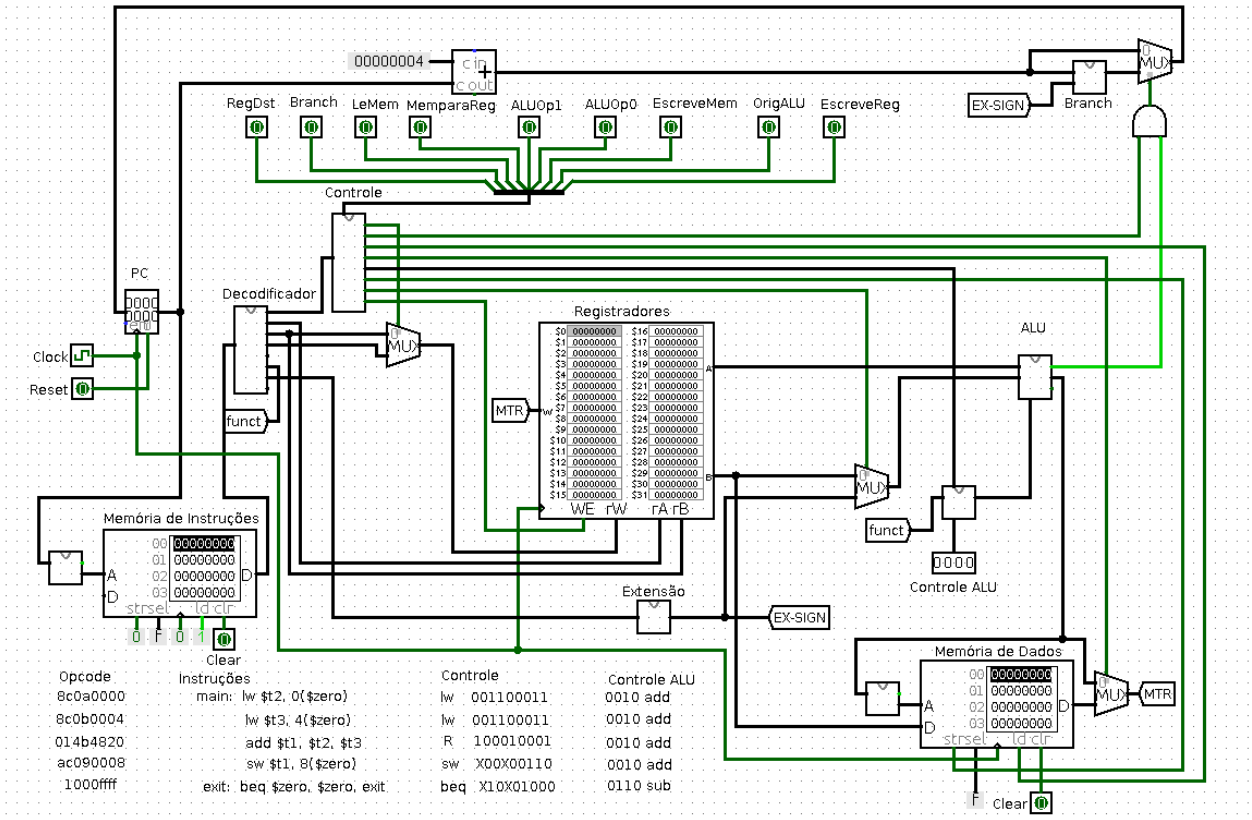
Figura 1: Esqueleto inicial do caminho de dados *MIPS*.



3 Resultado da Primeira Entrega

Seguindo as instruções presentes no arquivo **PDF** disponibilizado, foram identificadas as funções de cada sinal de controle. Doravante, foram realizadas todas as conexões necessárias para que o circuito executasse de maneira correta as instruções retornando as saídas esperadas. Veja na Figura 2, o caminho de dados do processador *MIPS* após as conexões e com o controle principal e da *ALU* manual.

Figura 2: *Datapath* com circuito do controle manual .



Note que foi necessária a inclusão de alguns seletores de dados (***MUXES***), com objetivo de controlar qual *bit* será utilizado para obter os resultados almejados. Veja na subseção 3.1 alguns dos testes realizados para verificar se o circuito funciona corretamente.

3.1 Testes do Caminho de Dados com Controle Manual

Nesta subseção serão apresentados alguns testes de instruções que o circuito da primeira etapa da entrega suporta. Como nesta etapa do projeto ainda não temos o circuito combinacional da unidade de controle, devemos informar todos os *bits* de controle necessários para cada instrução.

3.1.1 Instrução LW

Com objetivo de fazer o circuito funcionar corretamente, foram informadas quais eram as saídas corretas para a unidade de controle, sendo elas: **RegDst:** 0, **Branch:** 0, **LeMem**: 1, **MemparaReg:** 1, **ALUOp1:** 0, **ALUOp0:** 0, **EscreveMem:** 0, **OrigALU**: 1, **EscreveReg**: 1.

Da mesma forma, foi informado ao controle da *ALU*, os 4 *bits* necessários para realizar a operação almejada. Como a *ALU* do circuito apresentado deve realizar uma operação de soma, os *bits* utilizados serão: 0010.

Código 1: Código *Assembly* da instrução **BEQ**.

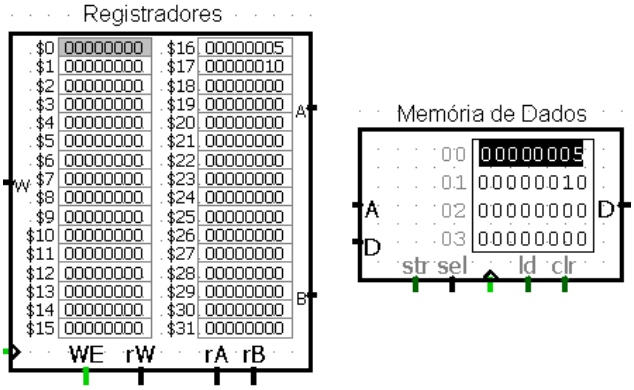
```
1 -- Codigo Assembly feito para testeEscreveMem:s.
2 -- Data: Terca-Feira, 11/05/2021-15:23:15
3 -- Autor 1: Jesse Pires Barbato Rocha
4 -- Autor 2: Jhonatan Guilherme de Oliveira Cunha
5
6         .text
```

```
7           .globl main
8
9   main:   lw $s0, 0($zero)
10          lw $s1, 4($zero)
```

Veja na Figura 3 como ficará o banco de registradores e o de memória após a execução do código *MIPS* no circuito da Figura 2.

ng

Figura 3: Valores dos bancos de registradores e memória após a execução do código.



3.1.2 Instrução ADD

Com objetivo de fazer o circuito funcionar corretamente, foram informadas quais eram as saídas corretas para a unidade de controle, sendo elas: **RegDst:** 1, **Branch:** 0, **LeMem**: 0, **MemparaReg:** 0, **ALUOp1:** 1, **ALUOp0:** 0, **EscreveMem:** 0, **OrigALU**: 0, **EscreveReg**: 1.

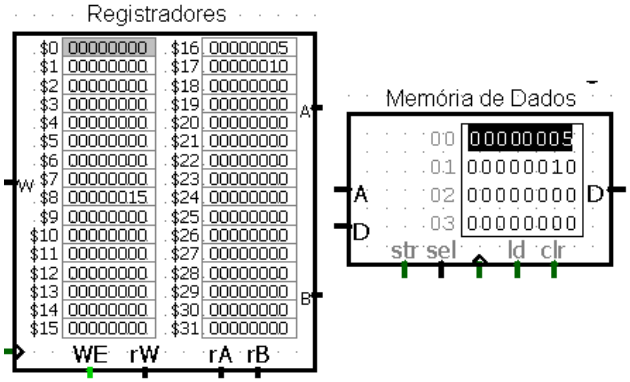
Da mesma forma, foi informado ao controle da *ALU*, os 4 *bits* necessários para realizar a operação almejada. Como a *ALU* do circuito apresentado deve realizar uma operação de soma, os *bits* utilizados serão: 0010.

Código 2: Código *Assembly* da instrução **ADD**.

```
1   -- Codigo Assembly feito para testes.
2   -- Data: Terca-Feira, 11/05/2021-15:23:15
3   -- Autor 1: Jesse Pires Barbato Rocha
4   -- Autor 2: Jhonatan Guilherme de Oliveira Cunha
5
6           .text
7           .globl main
8
9   main:   lw $s0, 0($zero)
10          lw $s1, 4($zero)
11          add $t0, $s0, $s1
```

Veja na Figura 4 como ficará os bancos de registradores e de memória após a execução do código *MIPS* no circuito da Figura 2.

Figura 4: Valores dos bancos de registradores e memória após a execução do código.



3.1.3 Instrução SUB

Com objetivo de fazer o circuito funcionar corretamente, foram informadas quais eram as saídas corretas para a unidade de controle, sendo elas: **RegDst:** 1, **Branch:** 0, **LeMem**: 0, **MemparaReg:** 0, **ALUOp1:** 1, **ALUOp0:** 0, **EscreveMem:** 0, **OrigALU**: 0, **EscreveReg**: 1.

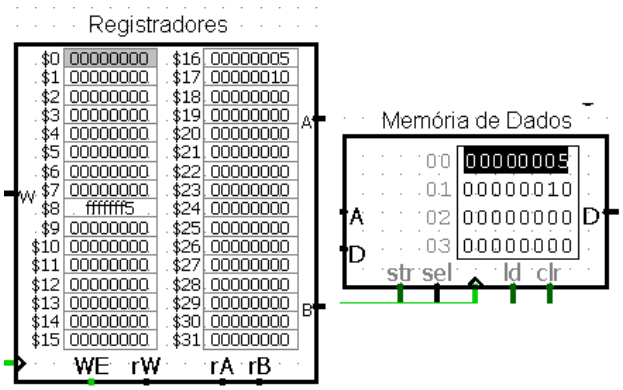
Da mesma forma, foi informado ao controle da *ALU*, os 4 *bits* necessários para realizar a operação almejada. Como a *ALU* do circuito apresentado deve realizar uma operação de subtração, os *bits* utilizados serão: 0110.

Código 3: Código *Assembly* da instrução **AND**.

```
1  -- Codigo Assembly feito para testes.
2  -- Data: Terca-Feira, 11/05/2021-15:23:15
3  -- Autor 1: Jesse Pires Barbato Rocha
4  -- Autor 2: Jhonatan Guilherme de Oliveira Cunha
5
6          .text
7          .globl main
8
9  main:   lw $s0, 0($zero)
10         lw $s1, 4($zero)
11         sub $t0, $s0, $s1
```

Veja na Figura 5 como ficará o banco de registradores e o de memória após a execução do código *MIPS* no circuito da Figura 2.

Figura 5: Valores dos bancos de registradores e memória após a execução do código.



3.1.4 Instrução AND

Com objetivo de fazer o circuito funcionar corretamente, foram informadas quais eram as saídas corretas para a unidade de controle, sendo elas: **RegDst:** 1, **Branch:** 0, **LeMem**: 0, **MemparaReg:** 0, **ALUOp1:** 1, **ALUOp0:** 0, **EscreveMem:** 0, **OrigALU**: 0, **EscreveReg**: 1.

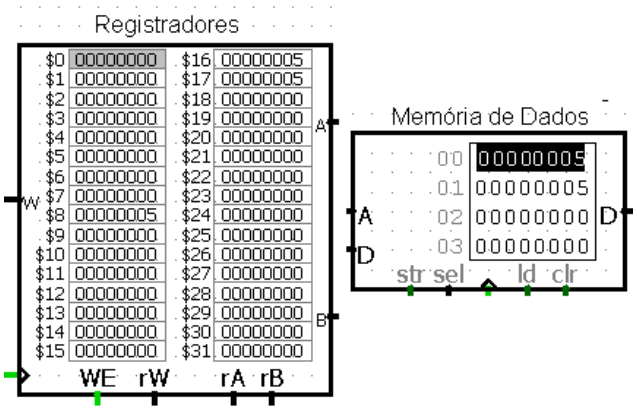
Da mesma forma, foi informado ao controle da *ALU*, os 4 *bits* necessários para realizar a operação almejada. Como a *ALU* do circuito apresentado deve realizar uma operação lógica do tipo **AND**, os *bits* utilizados serão: 0010.

Código 4: Código *Assembly* da instrução **AND**.

```
1  -- Codigo Assembly feito para testes.
2  -- Data: Terca-Feira, 11/05/2021-15:23:15
3  -- Autor 1: Jesse Pires Barbato Rocha
4  -- Autor 2: Jhonatan Guilherme de Oliveira Cunha
5
6          .text
7          .globl main
8
9  main:   lw $s0, 0($zero)
10         lw $s1, 4($zero)
11         and $t0, $s0, $s1
```

Veja na Figura 6 como ficará o banco de registradores e o de memória após a execução do código *MIPS* no circuito da Figura 2.

Figura 6: Valores dos bancos de registradores e memória após a execução do código.



3.1.5 Instrução OR

Com objetivo de fazer o circuito funcionar corretamente, foram informadas quais eram as saídas corretas para a unidade de controle, sendo elas: **RegDst:** 1, **Branch:** 0, **LeMem**: 0, **MemparaReg:** 0, **ALUOp1:** 1, **ALUOp0:** 0, **EscreveMem:** 0, **OrigALU**: 0, **EscreveReg**: 1.

Da mesma forma, foi informado ao controle da *ALU*, os 4 *bits* necessários para realizar a operação almejada. Como a *ALU* do circuito apresentado deve realizar uma operação lógica do tipo **OR**, os *bits* utilizados serão: 0001.

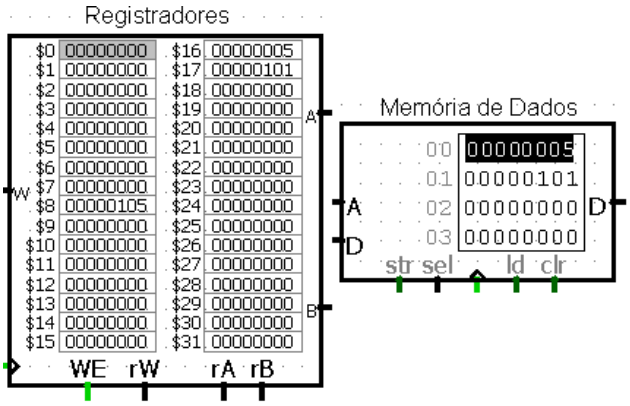
Código 5: Código *Assembly* da instrução **OR**.

```
1  -- Codigo Assembly feito para testes.
2  -- Data: Terca-Feira, 11/05/2021-15:23:15
3  -- Autor 1: Jesse Pires Barbato Rocha
4  -- Autor 2: Jhonatan Guilherme de Oliveira Cunha
5
6          .text
7          .globl main
8
```

```
main:   lw $s0, 0($zero)
        lw $s1, 4($zero)
        or $t0, $s0, $s1
```

Veja na Figura 7 como ficará o banco de registradores e o de memória após a execução do código *MIPS* no circuito da Figura 2.

Figura 7: Valores dos bancos de registradores e memória após a execução do código.



3.1.6 Instrução SLT

Com objetivo de fazer o circuito funcionar corretamente, foram informadas quais eram as saídas corretas para a unidade de controle, sendo elas: **RegDst:** 1, **Branch:** 0, **LeMem**: 0, **MemparaReg:** 0, **ALUOp1:** 1, **ALUOp0:** 0, **EscreveMem:** 0, **OrigALU**: 0, **EscreveReg**: 1.

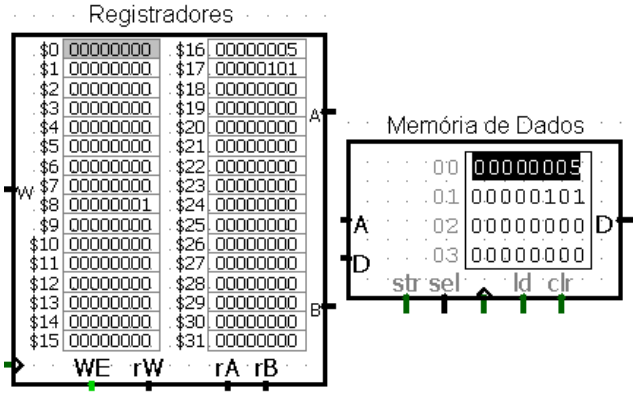
Da mesma forma, foi informado ao controle da *ALU*, que os 4 *bits* necessários para realizar a operação almejada serão: 0111.

Código 6: Código *Assembly* da instrução **SLT**.

```
-- Codigo Assembly feito para testes.
-- Data: Terca-Feira, 11/05/2021-15:23:15
-- Autor 1: Jesse Pires Barbato Rocha
-- Autor 2: Jhonatan Guilherme de Oliveira Cunha

        .text
        .globl main

main:   lw $s0, 0($zero)
        lw $s1, 4($zero)
        slt $t0, $s0, $s1
```

Veja na Figura 8 como ficará o banco de registradores e o de memória após a execução do código *MIPS* no circuito da Figura 2.

Figura 8: Valores dos bancos de registradores e memória após a execução do código.



3.1.7 Instrução SW

Com objetivo de fazer o circuito funcionar corretamente, foram informadas quais eram as saídas corretas para a unidade de controle, sendo elas: **RegDst:** X, **Branch:** 0, **LeMem**: 0, **MemparaReg:** X, **ALUOp1:** 0, **ALUOp0:** 0, **EscreveMem:** 1, **OrigALU**: 1, **EscreveReg**: 0.

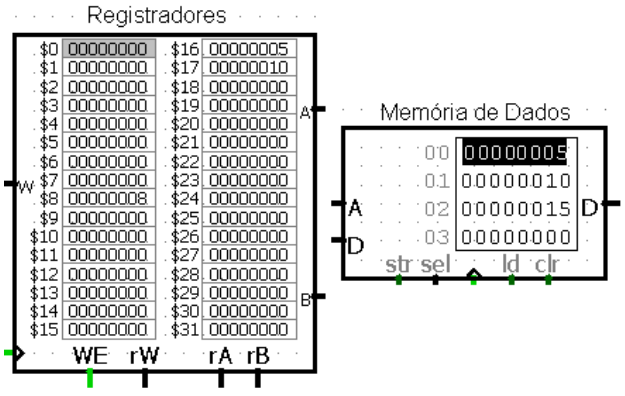
Da mesma forma, foi informado ao controle da *ALU*, os 4 *bits* necessários para realizar a operação almejada. Como a *ALU* do circuito apresentado deve realizar uma operação de soma, os *bits* utilizados será: 0010.

Código 7: Código *Assembly* da instrução **SW**.

```
1  -- Codigo Assembly feito para testes.
2  -- Data: Terca-Feira, 11/05/2021-15:23:15
3  -- Autor 1: Jesse Pires Barbato Rocha
4  -- Autor 2: Jhonatan Guilherme de Oliveira Cunha
5
6          .text
7          .globl main
8
9  main:   lw $s0, 0($zero)
10         lw $s1, 4($zero)
11         add $t0, $s0, $s1
12         sw $t0, 8($zero)
```

Veja na Figura 9 como ficará o banco de registradores e o de memória após a execução do código *MIPS* no circuito da Figura 2.

Figura 9: Valores dos bancos de registradores e memória após a execução do código.



3.1.8 Instrução BEQ

Com objetivo de fazer o circuito funcionar corretamente, foram informadas quais eram as saídas corretas para a unidade de controle, sendo elas: **RegDst:** X, **Branch:** 1, **LeMem**: 0, **MemparaReg:** X, **ALUOp1:** 0, **ALUOp0:** 1, **EscreveMem:** 0, **OrigALU**: 0, **EscreveReg**: 0.

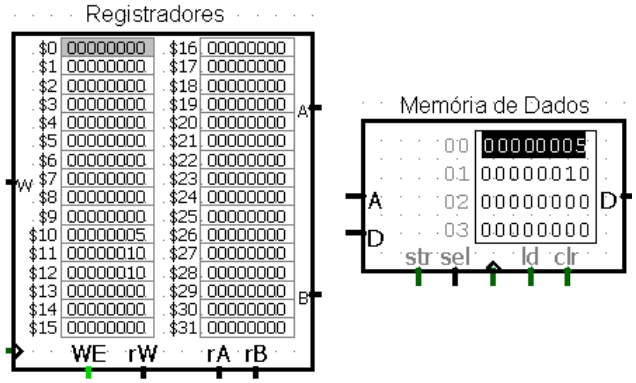
Da mesma forma, foi informado ao controle da *ALU*, que os 4 *bits* necessários para realizar a operação almejada serão: 0110.

Código 8: Código *Assembly* da instrução **BEQ**.

```
1  -- Codigo Assembly feito para testes.
2  -- Data: Terca-Feira, 11/05/2021-15:23:15
3  -- Autor 1: Jesse Pires Barbato Rocha
4  -- Autor 2: Jhonatan Guilherme de Oliveira Cunha
5
6          .text
7          .globl main
8
9  main:   lw $t2, 0($zero)
10         lw $t3, 0($zero)
11         beq $t3, $t2, jumpado
12
13 jumpado: lw $t4, 4($zero)
14          add $t3, $t4, $zero
```

Veja na Figura 10 como ficará o banco de registradores e o de memória após a execução do código *MIPS* no circuito da Figura 2.

Figura 10: Valores dos bancos de registradores e memória após a execução do código.



4 Segunda Etapa

Na segunda etapa do projeto proposto, foi necessário a implementação da unidade de controle principal do circuito e do controle da *ALU*, utilizando lógica combinacional, para as instruções acima citadas seguindo as instruções do professor, bem como o livro base da disciplina [1]. Ou seja, criando um circuito que, dado um determinado valor de entrada, produz um valor de saída com base na combinação de portas lógicas. Também foi pedido que fossem implementadas as instruções *add immediate* **(addi)** e *jump* **(j)**. Assim como na primeira entrega, foi necessário realizar testes, também fornecidos pelo professor. Veja na Seção 5 os resultados obtidos após a conclusão da atividade proposta.

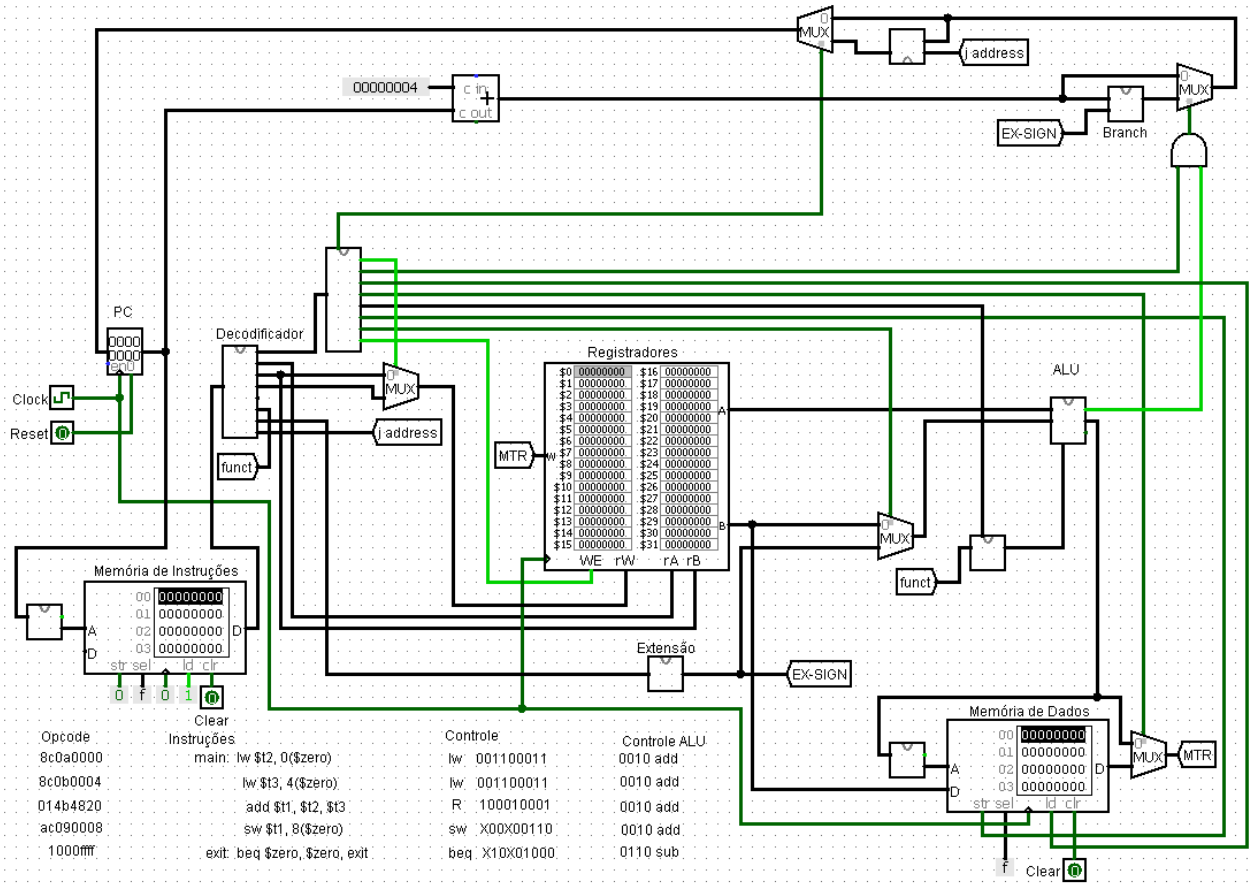
5 Resultado da Segunda Entrega

Partindo dos resultados obtidos na primeira etapa, deu-se início à complementação, exigida na segunda etapa do projeto. Nela, foi solicitada a implementação do controle principal das instruções e do controle da *ALU*. Para tanto, foi

utilizada lógica combinacional para criação do circuito principal solicitado. Além disso, o processador ***MIPS*** deveria passar a suportar também para as instruções *JUMP* ***j*** e *Add Immediate* ***addi***.

A Figura 11 apresenta o resultado das alterações no circuito do *MIPS* solicitados na segunda etapa.

Figura 11: *Datapath* com circuito do controle automático .



Note que foi necessária a inclusão de alguns seletores de dados (***MUXES***), com objetivo de controlar qual *bit* será utilizado para obter os resultados almejados. Veja na subseção 3.1 alguns dos testes que realizamos para verificar se o circuito esta funcionando corretamente.

Além disso, foi criado o circuito combinacional da Unidade de Controle Principal de instruções e o Controle da *ALU* que são apresentados na Figura 12 e Figura 13 respectivamente.

Figura 12: Circuito combinacional da unidade de controle do *MIPS*.

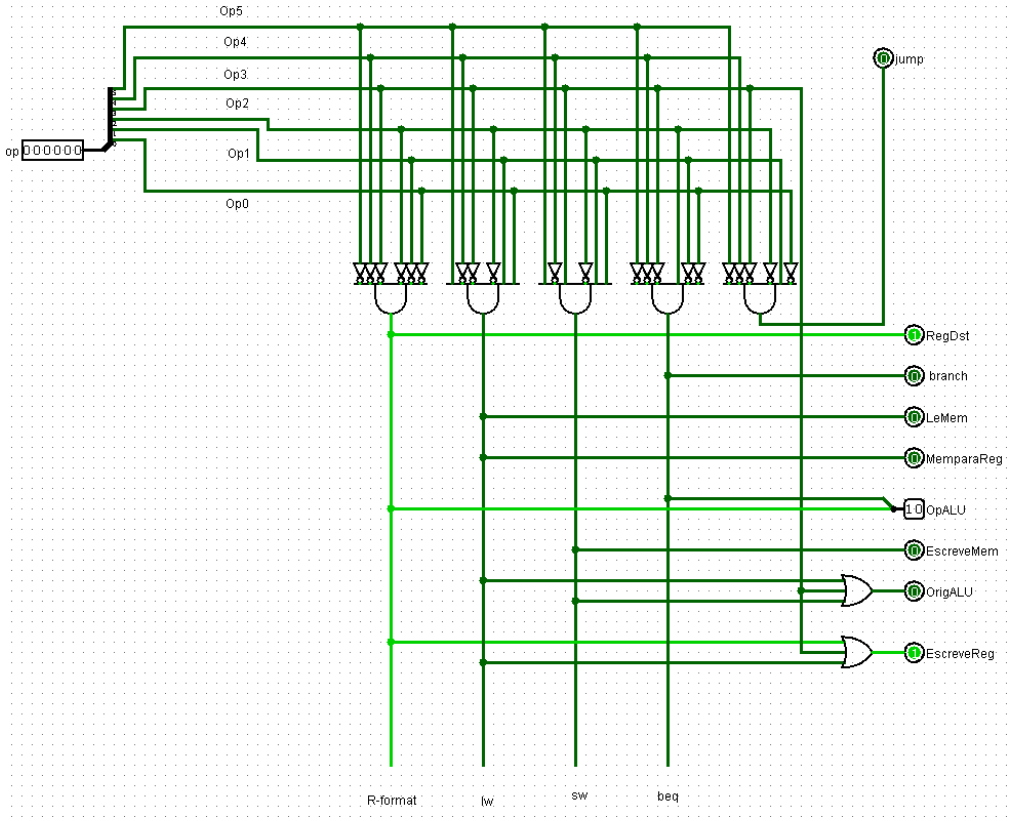
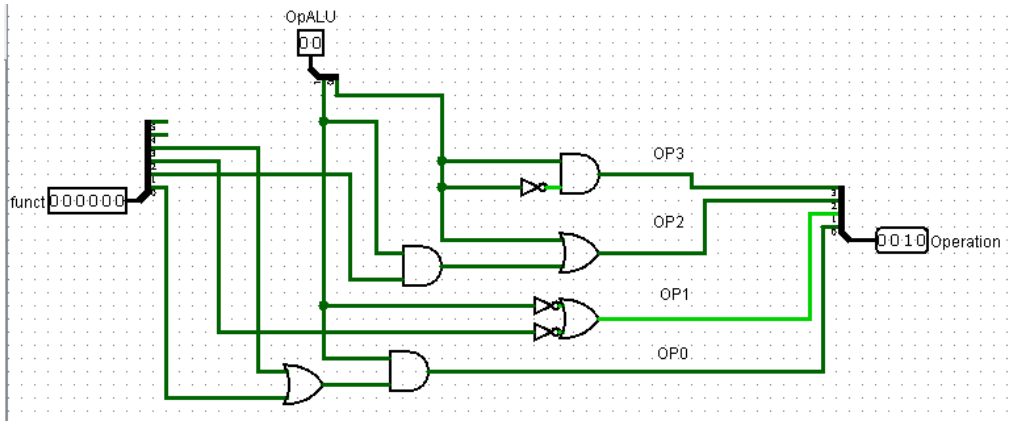


Figura 13: Circuito da unidade de controle da *ULA*.



Após isso, também foi necessário adicionar o componente *JumpAddress* no canto superior de nosso caminho de dados. Este recebe, em uma de suas entradas, o endereço selecionado pelo *Mux* do *Branch* e, na outra entrada, o endereço passado pela instrução do tipo *jump*. Seu resultado será utilizado em um próximo *Mux*, que é responsável por verificar se o endereço a ser utilizado será o oriundo do *Mux* do *Branch* ou o resultado do *JumpAddress*. Veja na subseção 5.1 os testes implementados para verificar o funcionamento das novas instruções.

5.1 Testes do Caminho de Dados com Implementação do Controle Principal e da *ALU*

Nesta subseção, serão apresentados alguns testes para as instruções solicitadas na segunda etapa do trabalho prático.

5.1.1 Instrução JUMP

Com objetivo de fazer o circuito funcionar corretamente, foram informadas quais eram as saídas corretas para a unidade de controle, sendo elas: **RegDst:** X, **Branch:** X, **LeMem**: X, **MemparaReg:** X, **ALUOp1:** X, **ALUOp0:** X, **EscreveMem:** X, **OrigALU**: X, **EscreveReg**: X, **jump**: 1. Note que foi necessária a adição de mais um bit como saída da unidade de controle, com objetivo de utilizar a instrução *jump* corretamente.

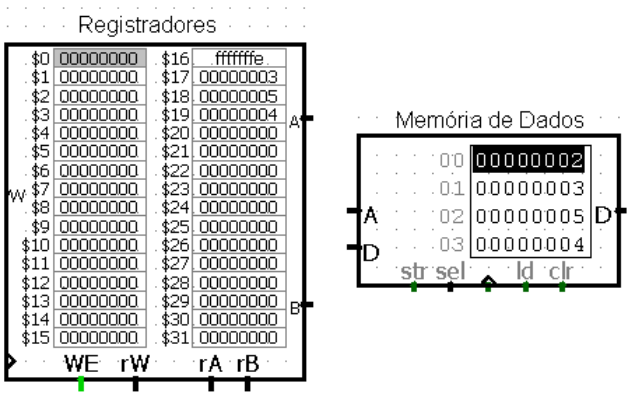
Da mesma forma, foi informado ao controle da *ALU*, que os 4 *bits* necessários para realizar a operação almejada serão: 0110.

Código 9: Código *Assembly* da instrução **JUMP**.

```
-- Codigo Assembly feito para testes.
-- Data: Terca-Feira, 11/05/2021-15:23:15
-- Autor 1: Jesse Pires Barbato Rocha
-- Autor 2: Jhonatan Guilherme de Oliveira Cunha

    .text
    .globl main

main:   lw $s0, 0($zero)
        lw $s1, 4($zero)
        lw $s2, 8($zero)
        lw $s3, 12($zero)

        slt $t0, $s2, $s3
        beq $t0, $zero, Else
        add $s0, $s1, $s2
        j Exit
Else:   sub $s0, $s1, $s2
Exit:
```

Veja na Figura 14 como ficará o banco de registradores e o de memória após a execução do código *MIPS* no circuito da Figura 2.

Figura 14: Valores dos bancos de registradores e memória após a execução do código.



5.1.2 Instrução ADDI

Com objetivo de fazer o circuito funcionar corretamente, foram informadas quais eram as saídas corretas para a unidade de controle, sendo elas: **RegDst:** 1, **Branch:** 0, **LeMem**: 0, **MemparaReg:** 0, **ALUOp1:** 1, **ALUOp0:** 0, **EscreveMem:** 0, **OrigALU**: 0, **EscreveReg**: 1, **jump**: 0.

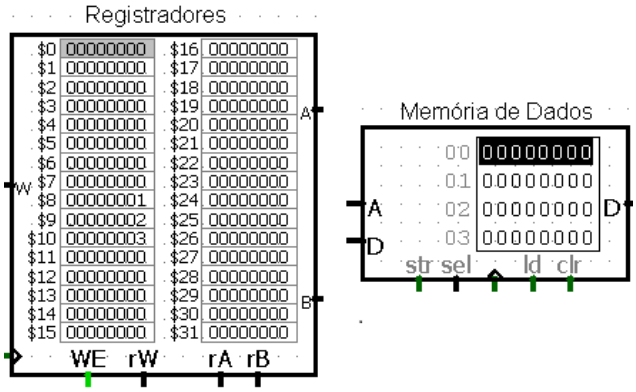
Da mesma forma, foi informado ao controle da *ALU*, que os 4 *bits* necessários para realizar a operação almejada serão: 0010.

Código 10: Código *Assembly* da instrução **ADDI**.

```
-- Codigo Assembly feito para testes.
-- Data: Terca-Feira, 11/05/2021-15:23:15
-- Autor 1: Jesse Pires Barbato Rocha
-- Autor 2: Jhonatan Guilherme de Oliveira Cunha

        .text
        .globl main

main:   addi $t0, $zero, 1
        addi $t1, $zero, 2
        add $t2, $t0, $t1
```

Veja na Figura 15 como ficará o banco de registradores e o de memória após a execução do código *MIPS* no circuito da Figura 2.

Figura 15: Valores dos bancos de registradores e memória após a execução do código.



6 Conclusão

Pela observação dos passos necessários para a construção da tarefa solicitada, foi possível identificar, na prática, como o caminho de dados do processador *MIPS* é implementado. Também foi possível notar a importância de cada *bit* da unidade de controle principal do circuito, bem como a unidade de controle da *ALU*.

Ressaltamos que o conjunto de instruções implementadas representa uma fração do conjunto real do *MIPS*. Porém, já permite que se tenha conhecimento de como o processador as executa, e também, como é feita a utilização do banco de registradores e a memória de dados. Diante do exposto, conclui-se que os objetivos do trabalho foram cumpridos.

Referências

[1] Hennessy, J. Organização e Projeto de Computadores. São Paulo: Grupo GEN, 2017. 9788595152908. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788595152908/. Acesso em: 14 de maio de 2021.