

**Universidade Federal de São João del-Rei**  
**Arquitetura e Organização de Computadores I**  
**Trabalho Prático 2**

**Felipe Francisco Rios de Melo**

## 1. INTRODUÇÃO

Este trabalho prático tem por objetivo desenvolver o caminho de dados e a unidade de controle para um processador MIPS.

## 2. IMPLEMENTAÇÃO

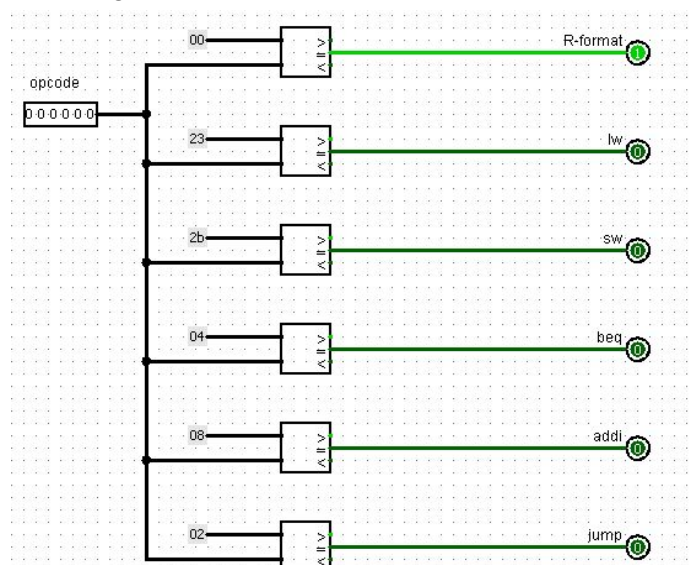
O desenvolvimento do processador se deu por meio de etapas, onde primeiramente foi implementado a unidade de controle e o controle da ALU, em seguida foi completado todo o caminho de dados adicionando novas instruções (addi e jump) e por fim testado através de códigos programados em MIPS.

### 2.1. Unidade de controle

A primeira tarefa foi implementar a unidade de controle, ela é composta pelo *Instruction type-decode* e pelo *control decode*:

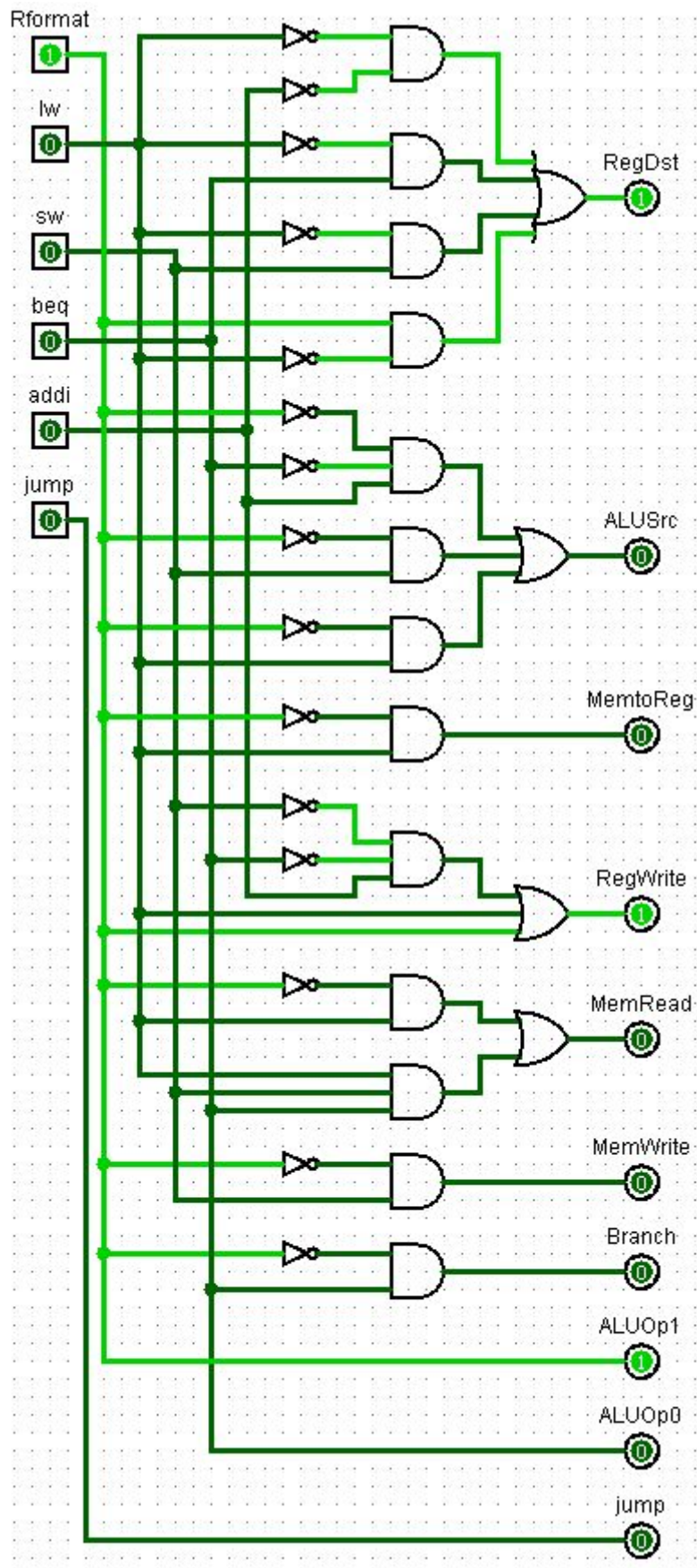
#### 2.1.1. Instruction type-decode

Este circuito recebe o OpCode e a partir dele verifica se a instrução é do tipo R-format, lw, sw, beq, addi ou jump. As comparações são realizadas por meio de Comparadores Aritméticos, já disponibilizados pelo Logisim, onde que há uma entrada que recebe o OpCode e a outra recebe a constante em hexadecimal referente àquela instrução, a saída é 1 caso, as entradas forem iguais e 0 caso contrário.



### 2.1.2. Control Decode

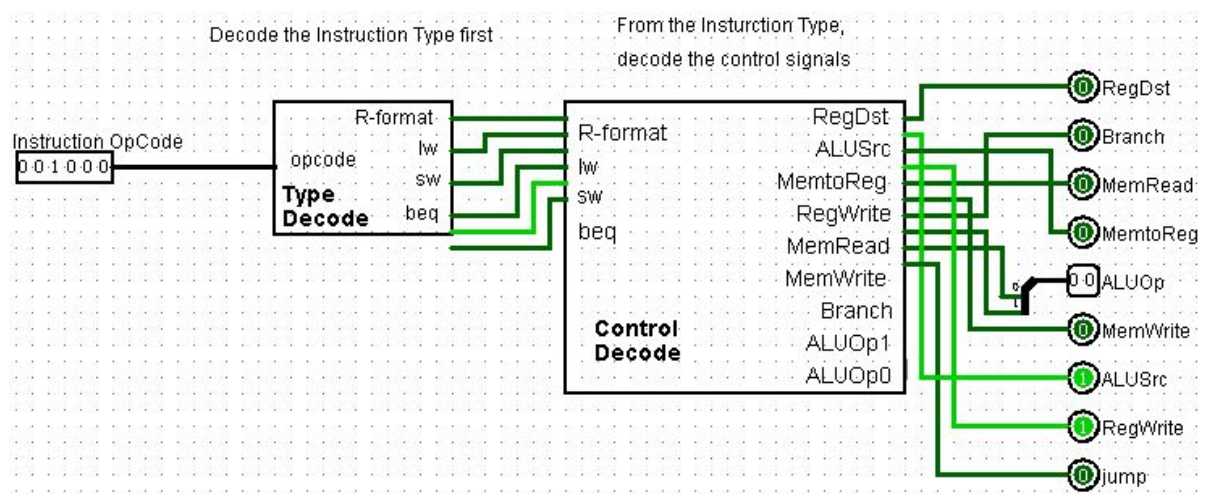
O Control Decode recebe o tipo da instrução e cria os sinais de controles corretos para executar a referida instrução.



O circuito foi gerado por meio da tabela verdade das instruções.

Instruction	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0	Jump
R-format	1	0	0	1	0	0	0	1	0	0
lw	0	1	1	1	1	0	0	0	0	0
sw	X	1	X	0	0	1	0	0	0	0
beq	X	0	X	0	0	0	1	0	1	0
addi	0	1	0	1	0	0	0	0	0	0
jump	0	0	0	0	0	0	0	0	0	1

Por fim, o *instruction type-decode* e o *control decode* são integrados no mesmo circuito, formando a unidade de controle.



## 2.2. Controle da ALU

A unidade de controle da ALU possui 8 bits de entrada, 2 bits referentes a ALUOp e o restante proveniente do campo "function" da instrução.

De acordo com a instrução, uma das operações abaixo deverá ser executada:

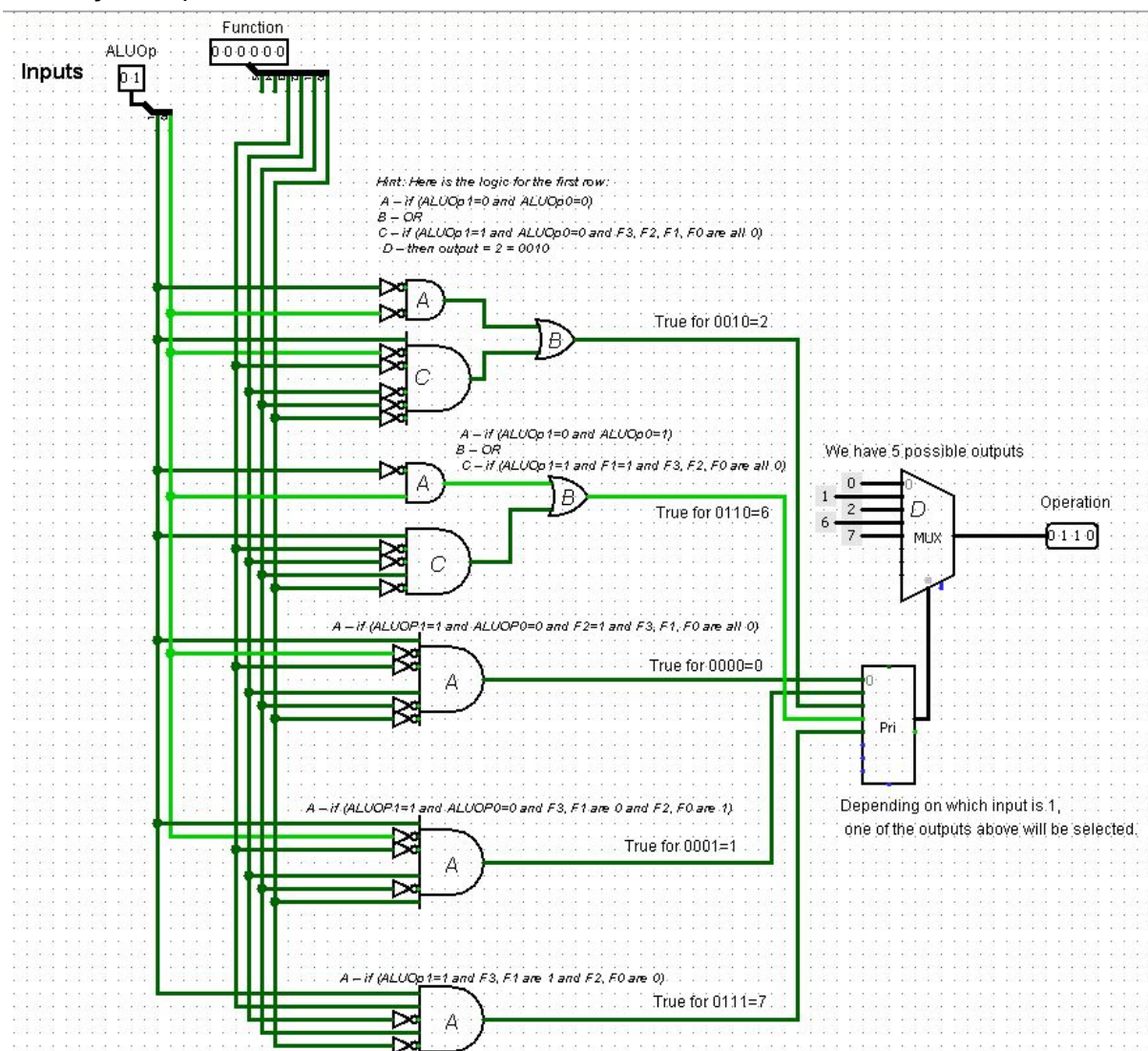
- Aritméticas e lógicas (and, or, sub, add, slt)
- Load/store (add para cálculo do endereço)
- Beq (subtração)

Por meio da seguinte tabela, foi implementado manualmente o circuito combinacional por meio de *AND* e *OR*:

ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Operation
0	0	X	X	X	X	X	X	0010
0	1	X	X	X	X	X	X	0110
1	0	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	0	X	X	0	1	0	0	0000
1	0	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

Dependendo de qual entrada é inserida, uma das 5 possibilidades de saída é ativada para a respectiva entrada, através de um codificador de prioridade que conduz o endereço do valor 1 mais significativo na entrada como seleção do multiplexador, o qual escolhe qual será a operação a ser realizada.

No exemplo abaixo, de acordo com a entrada, é ativado a operação 0110, que corresponde a instrução *beq*.





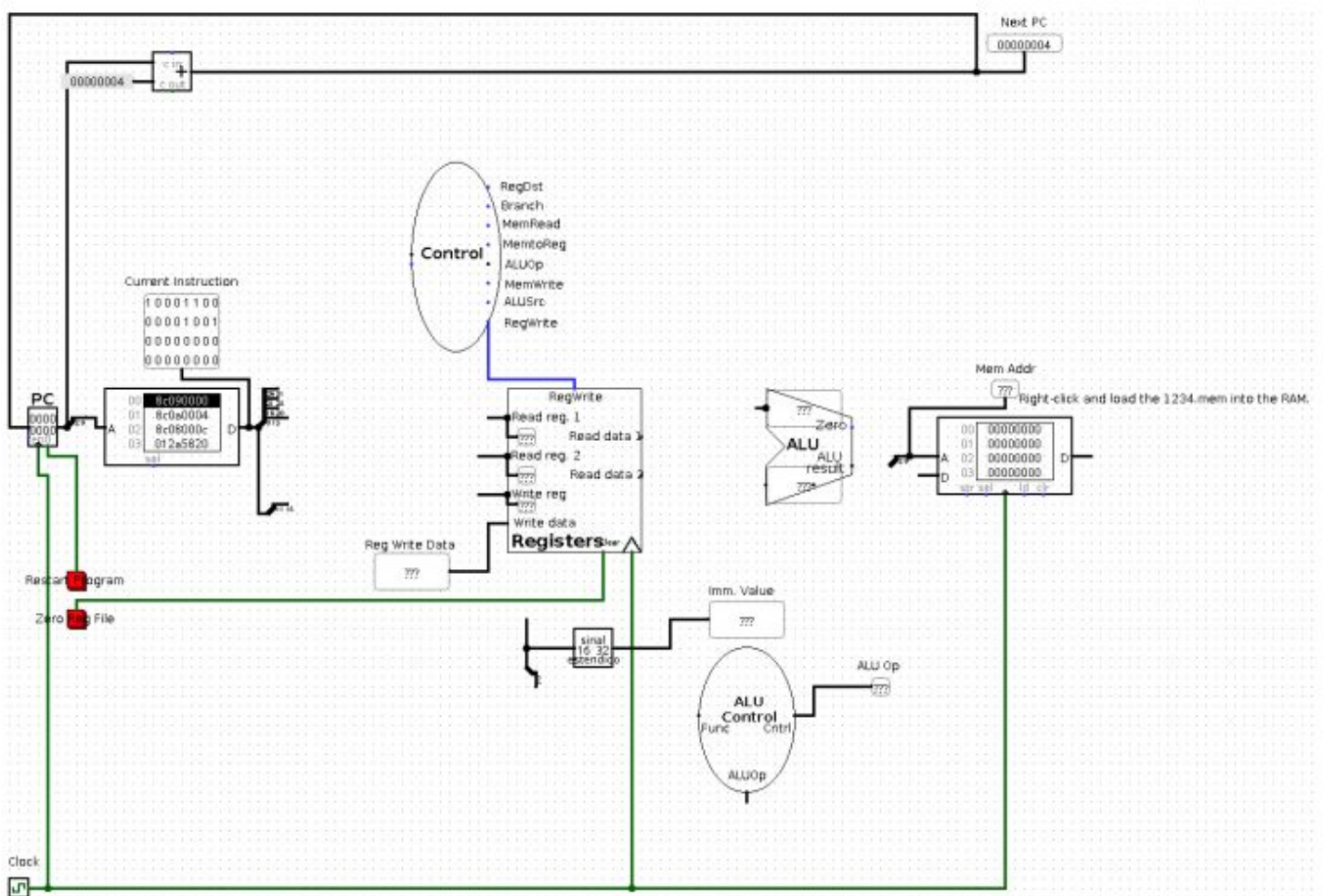
### 2.3. Implementação do addi e jump

O *addi* e o *jump* foi implementado junto às outras instruções no *instruction-type-decode*, com o *OpCode* 001000 e 000010 respectivamente. Quando a instrução chega até o *control decode*, se chegar a instrução *addi*, é ativado o *ALUSrc* e o *RegWrite*, por outro lado se chegar a instrução *jump* é ativado uma instrução também rotulada como *jump*. (o circuito pode ser visto na seção 2.1.1 e 2.1.2)

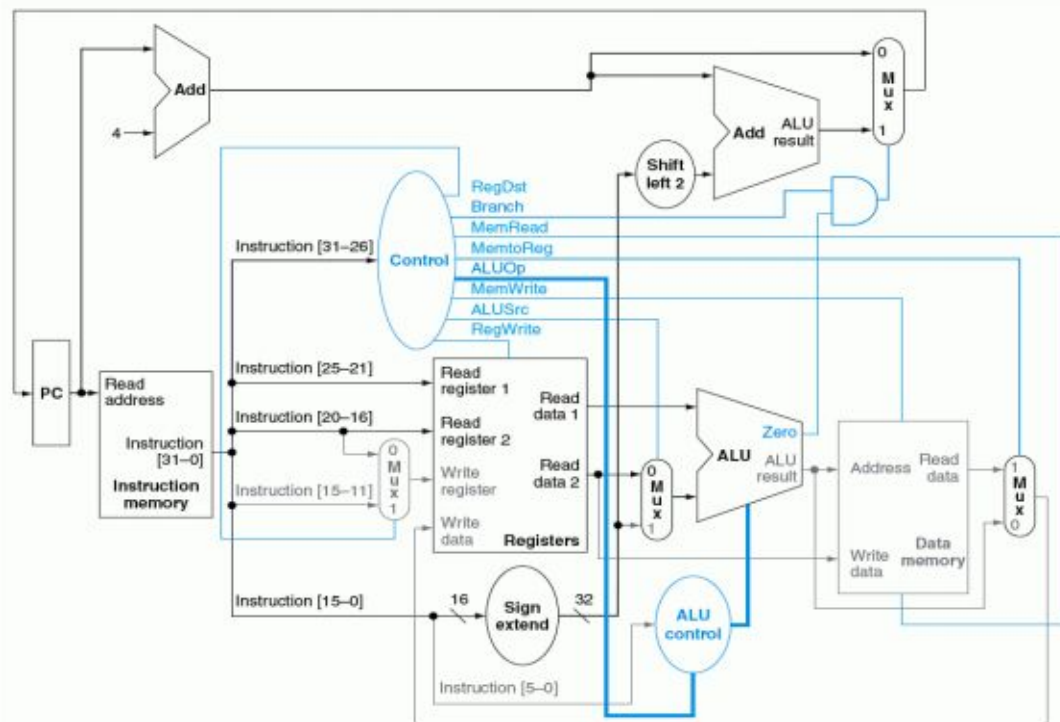
Após isto está terminada a implementação do *addi*, porém para funcionar a função *jump* é necessário, ainda, adaptar a lógica do PC (já havia sido adaptada para o *beq*) para realizar o salto de instruções, onde que PC passa a ser formado pelos 4 bits mais significativos + 26 bits do imediato + 00 (deslocamento de 2).

### 2.4. Caminho de dados (finalizado)

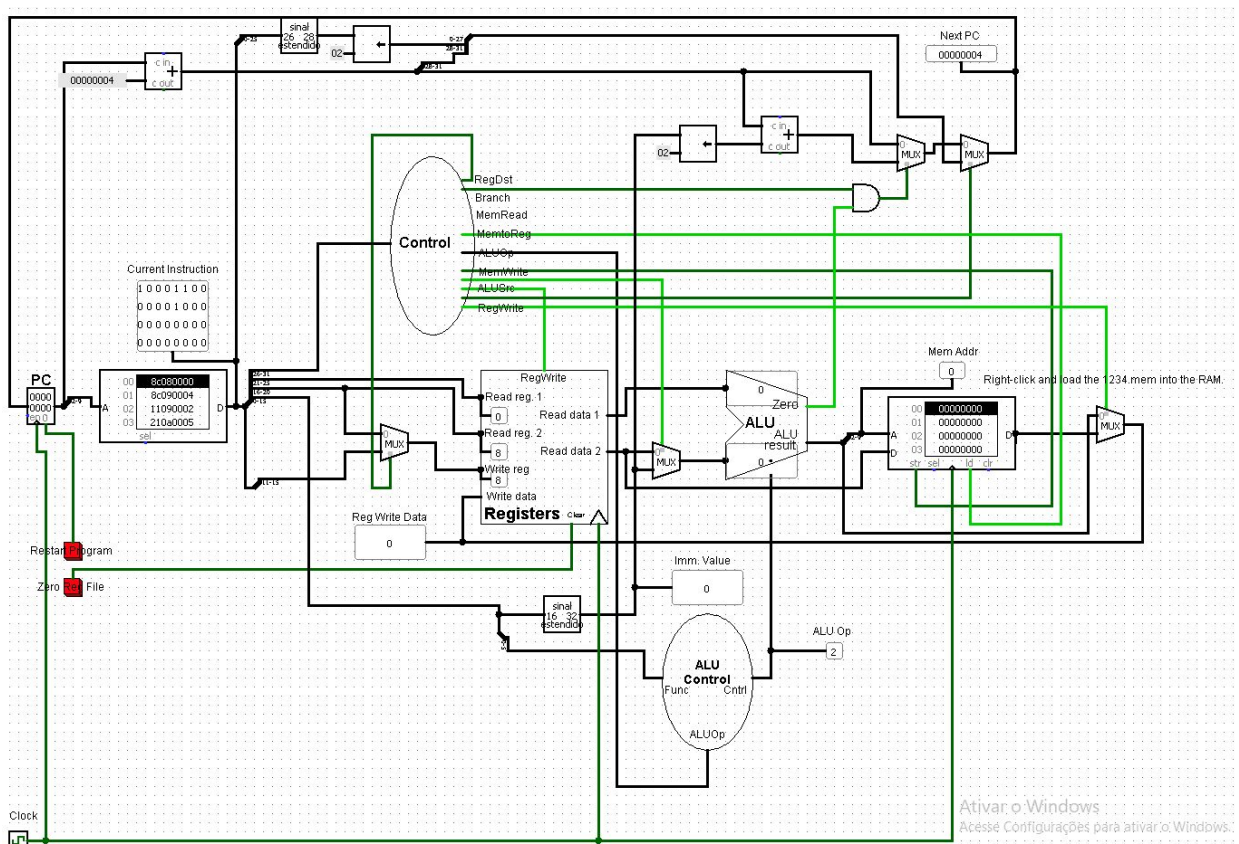
A partir deste ponto:



E dado o modelo de caminho de dados apresentado:



Com o auxílio de distribuidores de bits, multiplexadores, deslocadores de bits, extensores de sinais entre outros componentes e alterando a lógica do cálculo do PC (program counter) para ser compatível às instruções *beq* e *jump* foi completado o caminho de dados da seguinte forma:



### 3. TESTES

Para testar funcionamento do processador foi implementado alguns códigos em MIPS, utilizando as instruções suportadas pelo caminho de dados. Para tal, é carregado os dados (valores dos registradores) na memória de dados e carregado o algoritmo em MIPS (instruções) para a memória de instruções do processador.

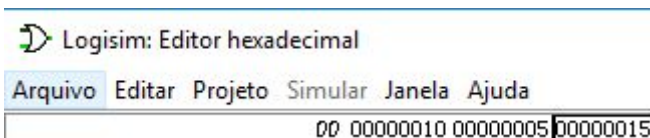
Será mostrado aqui, duas execuções do processador para o seguinte algoritmo:

```
1  lw $t0, 0($zero)
2  lw $t1, 4($zero)
3
4  beq $t0, $t1, else
5  sub $t2, $t0, $t1
6  j  exit
7
8  else:
9  add $t2, $t0, $t1
10
11 exit:
12 addi $t2, $t2, 10
13 sw $t2, 8($zero)
```

Com esse programa é possível observar o funcionamento de todas as instruções oferecidas pelo processador (lw, sw, beq, sub, jump, add, addi).

v2.0 raw  
10 5


Para \$t0 igual a 10 e \$t1 igual 5, temos em \$t2 o valor 15. O que era de se esperar, pois no *beq* temos que 10 é diferente de 5, então o programa segue para linha seguinte (sem desvio), onde é subtraído \$t1 de \$t0 ( $10 - 5 = 5$ ) e armazenado em \$t2, após isso é realizado um jump que desvia para a label *exit*, na qual é realizado um *addi* que adiciona 10 ao valor de \$t2 ( $5 + 10 = 15$ ), e por fim, é guardado na memória o valor final de \$t2 (15). O que pode ser visto nos dados de registradores no logisim:



v2.0 raw  
5 5

Para \$t0 e \$t1 igual a 5, temos em \$t2 o valor 14. O que também era de se esperar, logo que no *beq* temos que \$t0 é igual a \$t1, então o programa é desviado para a label *else*, onde é somado \$t0 e \$t1 ( $5 + 5 = 10$ ) e armazenado em \$t2, em seguida entra na label *exit* que realiza um *addi* adicionando 10 ao valor de \$t2 ( $10 + 10 = 20$ ), e por fim, é guardado na memória o valor final de \$t2 (20). O logisim representa os valores dos registradores na base

hexadecimal, isso implica que, convertendo 20 da base decimal para a hexadecimal, temos o valor armazenado em \$t2 igual a 14. Como pode ser observado nos dados de registradores no logisim:

 Logisim: Editor hexadecimal

Arquivo Editar Projeto Simular Janela Ajuda

00 00000005 00000005 00000014