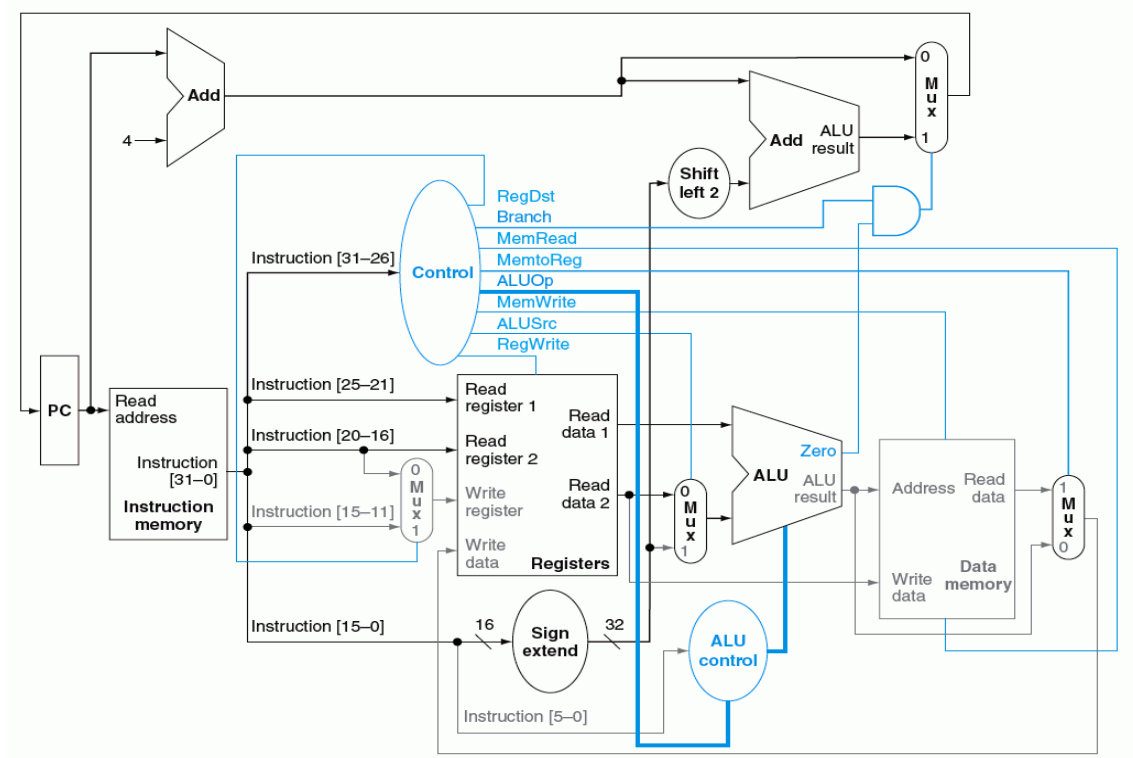
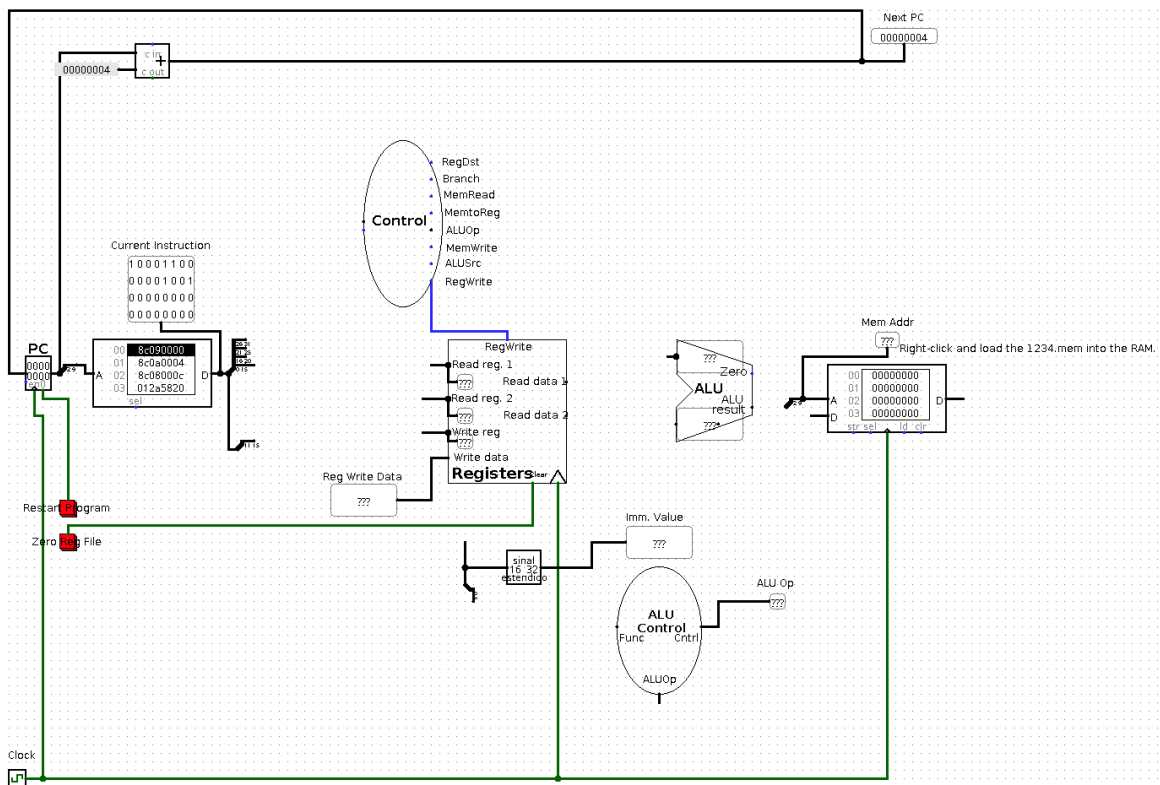


# Criando um Processador MIPS

Processador do nosso livro texto:



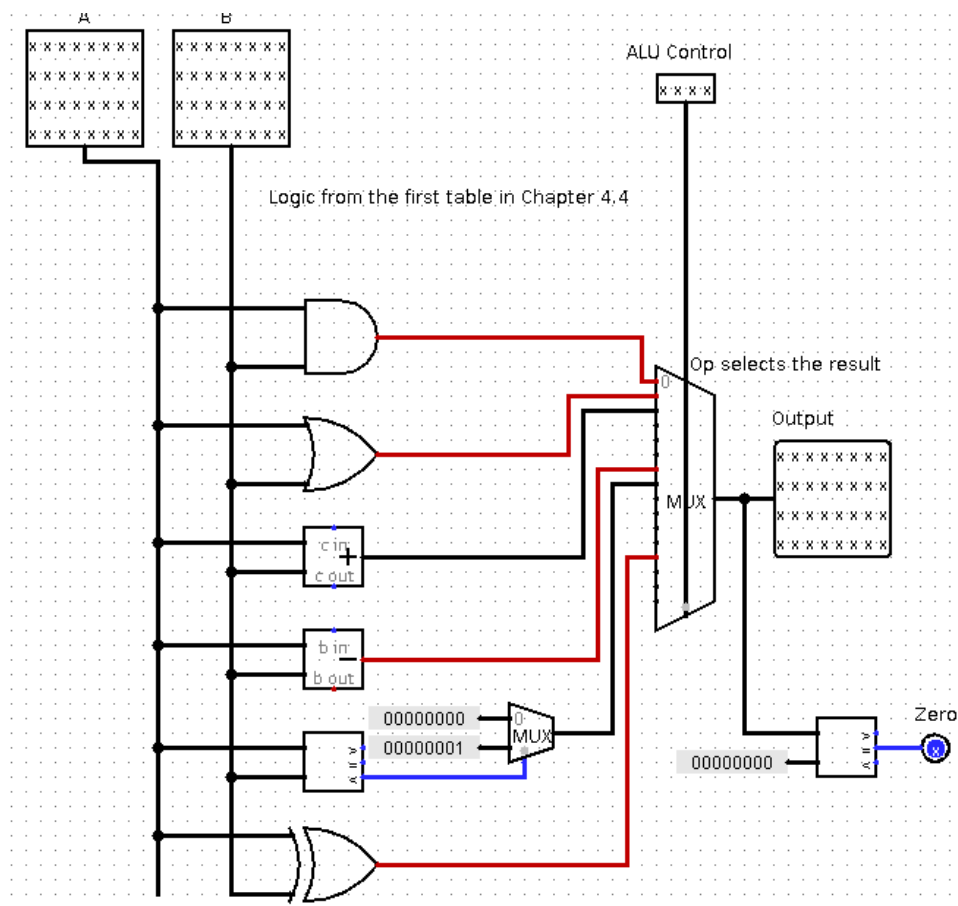
O que temos no logisim:



**Primeiro passo:** Verifique a similaridade entre os caminhos de dados. Estude o caminho de dados do MIPS visto em sala de aula. Sem o seu entendimento, você não conseguirá realizar a criação do seu caminho de dados no LOGISIM. Verifique todas as linhas de controle e o funcionamento das unidades de controle.

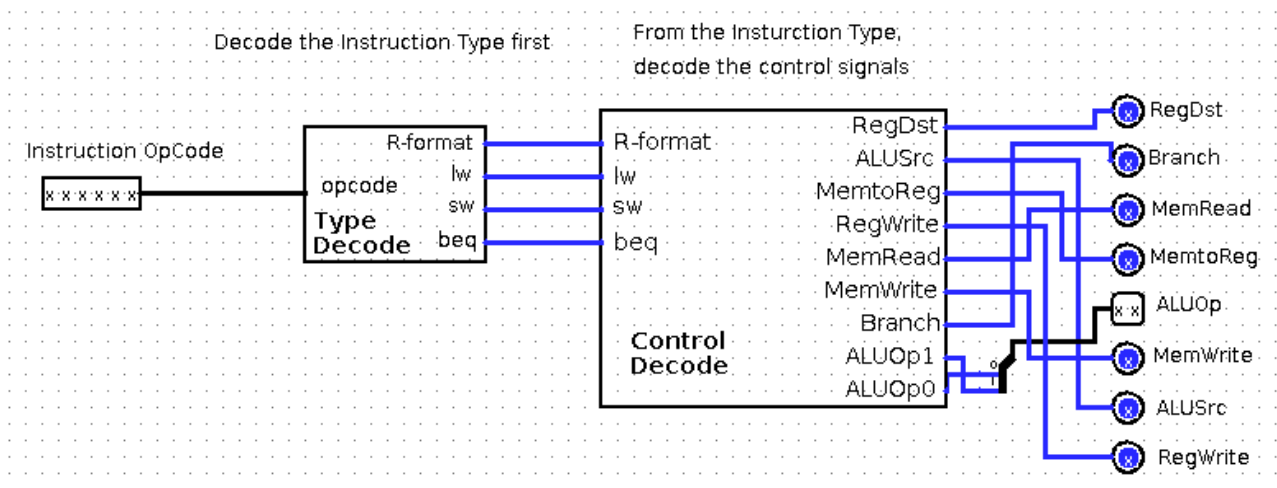
**Segundo passo:** Verifique todos os componentes do projeto inicial no LOGISIM. Inicie com a ALU. Clique com o botão direito em cima do desenho da ALU e depois em “ver alu”. Observe o funcionamento da ALU e como o controle de qual operação a ser realizada é feito. A ALU possui duas entradas (A e B) e calcula uma de 6 funções. Toas as 6 funções são calculadas, mas o MUX seleciona apenas uma para ir para a saída. A seleção é feita pela operação da ALU. Se o controle da ALU for 0010=2, a entrada 2 será selecionada para ir para a saída. A operação 0010 indica a soma entre A + B.

Compare o desenho com a seguinte tabela de controle da ALU:



Controle da ALU	Operação
0000	AND
0001	OR
0010	ADD
0110	SUB
0111	Set on less than
1100	NOR

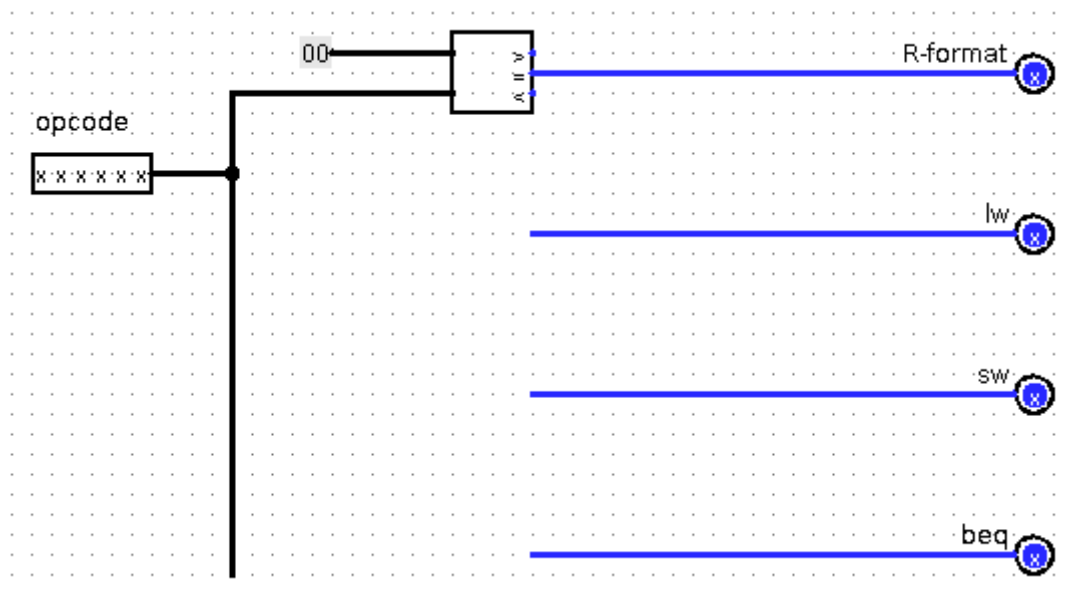
Agora volte para o projeto principal (clique duplo no “mips” no menu do lado esquerdo do Logisim) e verifique a unidade de controle.



A unidade de controle possui várias partes:

- 1) O OpCode como entradas
- 2) “Type Decode” que a partir do OpCode verifica se a instrução é do tipo R-format, lw, sw ou beq.
- 3) “Control Decode” que recebe o tipo da instrução e cria os sinais de controles corretos para executar a referida instrução.

**Tarefa 1:** Finalizar a lógica combinacional para realizar o “Type Decode”.



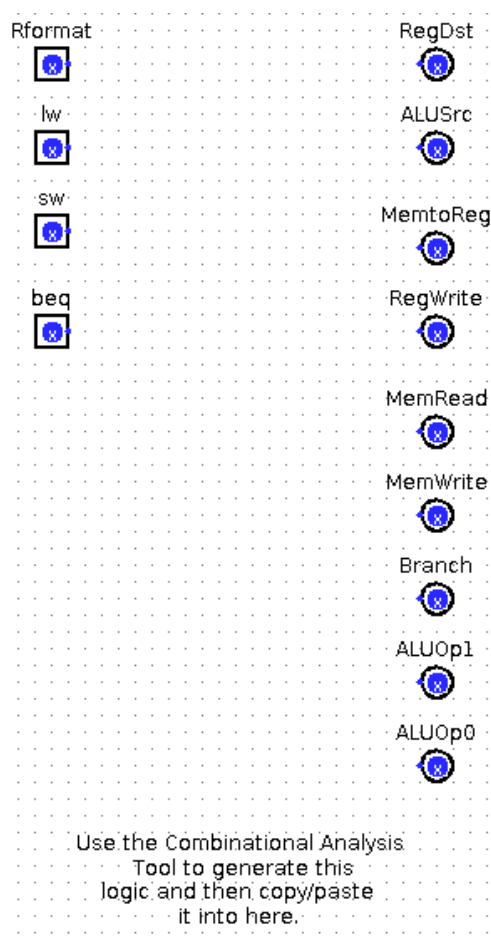
Verifique que foi utilizado um “comparador” e uma constante para verificar se a instrução é do tipo R-format.

Entrada	R-format	Lw	Sw	Beq
Op5	0	1	1	0
Op4	0	0	0	0
Op3	0	0	1	0
Op2	0	0	0	1
Op1	0	1	1	0
Op0	0	1	1	0

Complete a unidade “Type Decoder” adicionando mais comparadores para detectar as instruções corretas.

**Verificação:** Ao finalizar o circuito você precisa testá-lo. Para fazer isso, vá no menu “Simular” do logisim e habilite “Habilitar Simulação”. Utilize o indicador para modificar a entrada do opcode e verifique as saídas. Faça isso com calma para encontrar todos os possíveis errors.

**Tarefa 2:** Finalizar a unidade de controle.



Para criar a lógica combinacional para a unidade de controle, primeiro você deve entender e preencher a tabela abaixo com os seguintes sinais de controle:

Saída	R-format	Lw	Sw	Beq
RegDest	1	0	X	X
ALUSrc				
MemToReg				
RegWrite				
MemRead				
MemWrite				
Branch				
ALUOp1				
ALUOp2				

Após preencher a tabela, utilize a Ferramenta “Combinational Analysys” para gerar o circuito correspondente. Para utilizar essa ferramenta, vá no menu esquerdo e clique com o botão direito no projeto “control-decode” e depois em Analisar Circuito. A seguinte tela irá aparecer:

The screenshot shows the 'Análise Combinacional' window with the 'Tabela' tab selected. It displays a truth table with 16 rows and 13 columns. The columns are labeled: Rformat, lw, sw, beq, RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, ALUOp1, and ALUOp0. The rows represent all possible combinations of the four control bits (0000 to 1111). The values in the table are 'X' for 'write' and '0' for 'no write'.

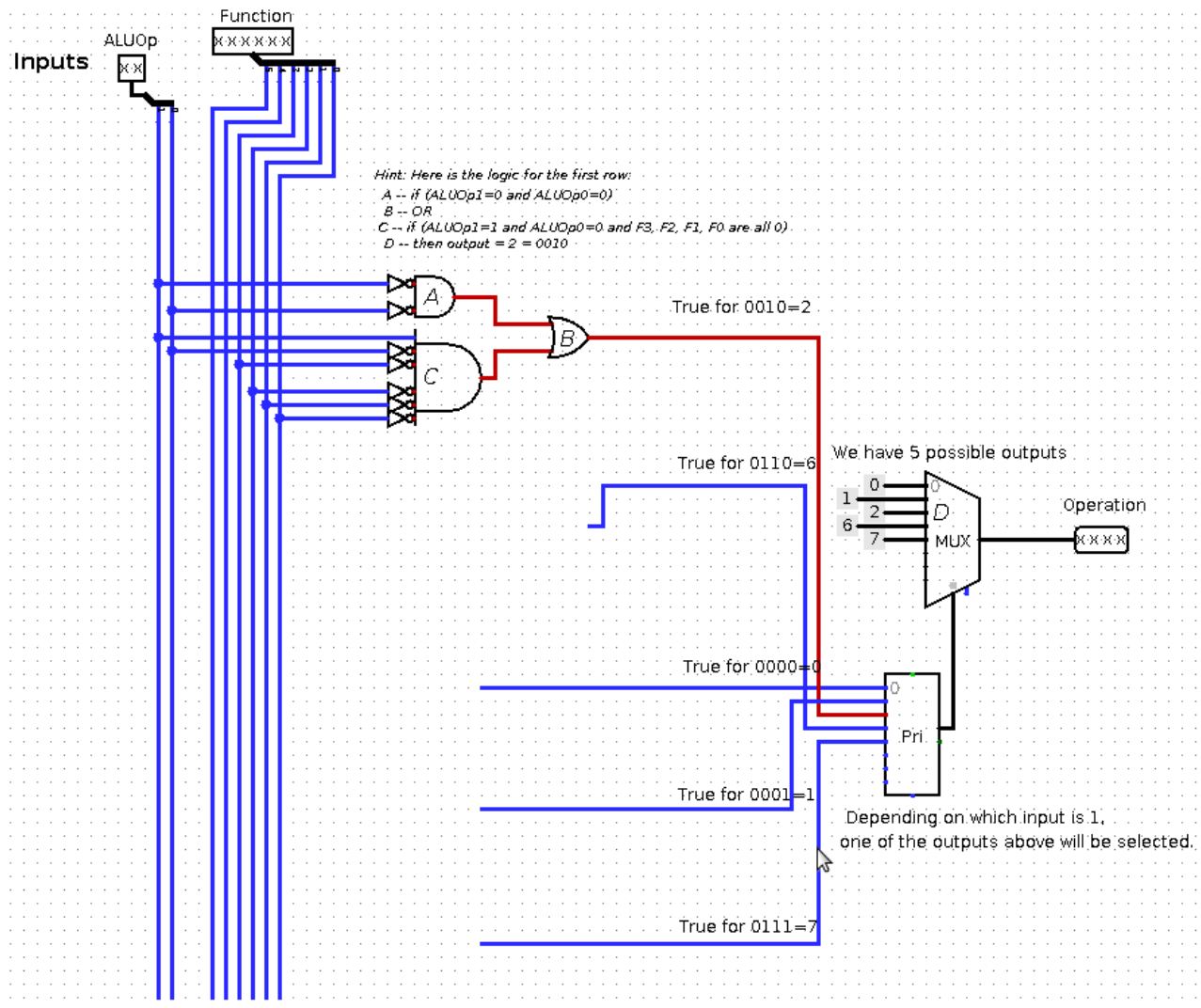
Rformat	lw	sw	beq	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
0	0	0	0	X	X	X	X	X	X	X	X	X
0	0	0	1	X	X	X	X	X	X	X	X	X
0	0	1	0	X	X	X	X	X	X	X	X	X
0	0	1	1	X	X	X	X	X	X	X	X	X
0	1	0	0	X	X	X	X	X	X	X	X	X
0	1	0	1	X	X	X	X	X	X	X	X	X
0	1	1	0	X	X	X	X	X	X	X	X	X
0	1	1	1	X	X	X	X	X	X	X	X	X
1	0	0	0	X	X	X	X	X	X	X	X	X
1	0	0	1	X	X	X	X	X	X	X	X	X
1	0	1	0	X	X	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X	X

Verifique na Aba Entrada e Saída as entradas e saídas do circuito. A aba Tabela, preencha a tabela verdade com os dados da tabela de controle acima preenchida por você. Ao final, clique em construir circuito.

**Verificação:** Uma vez construído o circuito, você deverá testá-lo. Faça da mesma forma como fizemos na aula anterior. Verifique se todos os sinais gerados estão de acordo com o que foi projetado. Não esqueça de conferir se o que você projetou está realmente correto.

### Tarefa 3: Finalizar o Controle da ALU.

Verifique a unidade de controle da ALU. Ela possui 8 entradas divididas em: i) 2 entradas para ALUOp e 6 entradas provenientes do campo “function” da instrução. Verifique as entradas você mesmo.



Tente entender o funcionamento da unidade de controle. Não comece a implementá-la sem antes entender seu funcionamento. Utilize a tabela abaixo para entender o circuito e seu objetivo.

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

Note as constantes (0, 1, 2, 6 e 7) como entrada do MUX e o codificador de prioridade como entrada de seleção do MUX.

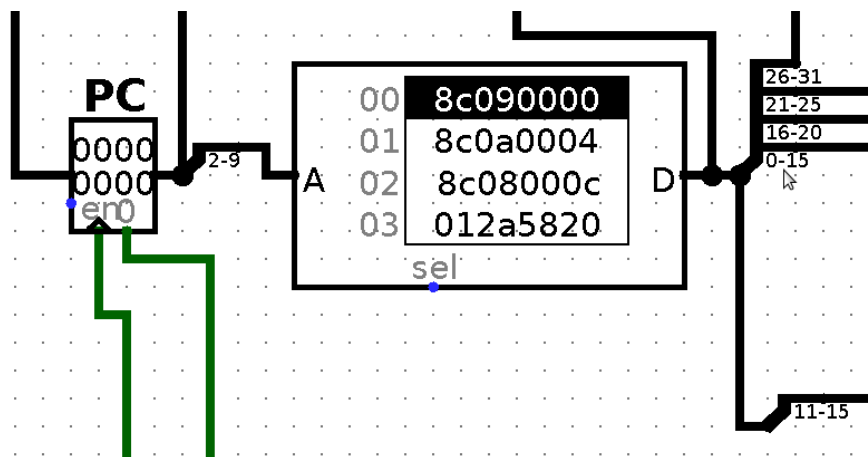
**Verificação:** Teste o funcionamento completo da unidade de controle. Não esqueça de adicionar comentários a sua unidade de controle. Isso também será avaliado.

Conecte a saída ALUOp da Unidade de Controle central com a Unidade de Controle da ALU. Teste novamente o seu circuito, mas agora interagindo com ambas unidades de controle.

**Tarefa 4:** Finalizar o caminho de dados.

Agora que você finalizou (e testou) a lógica de controle do processador, você deverá finalizar o caminho de dados conectando os circuitos adequadamente. A primeira figura deste documento ilustra o caminho de dados visto em sala de aula.

Precisamos prestar atenção em como a instrução é dividida para pegarmos apenas um conjunto de bits. Utilizamos o conector *Splitter* localizado dentro de *Wiring*. Esse conector é utilizado em diferentes lugares. Veja a figura abaixo.



Na saída D da memória de instruções, temos a utilização de dois *Splitters*: um para a divisão dos bits 0-15, 16-20, 21-25, 26-31 e outro para a divisão dos bits 11-15. Clique no botão direito no *Splitter* e depois em mostrar atributos. Tente entender o seu funcionamento.

Para conectar corretamente os circuitos do nosso processador, você deverá utilizar MUXes e modificar a lógica do cálculo do PC, que não funciona para instruções do tipo BEQ. Observe também onde você irá conectar os comandos “MemRead” e “MemWrite” na memória de dados do seu processador. Passe o mouse em cima de cada conector da memória para verificar o seu significado. Observe os bits utilizados no campo de endereço da memória.

**Tarefa 5:** Testar o nosso processador com um código.

Para testar o nosso código, você deverá inicialmente carregar alguns dados para a memória de dados. Para isso, carregue o arquivo dados1.mem para a memória de dados. Clique com o botão direito na memória e clique em *load image*. Você pode modificar os itens diretamente clicando em

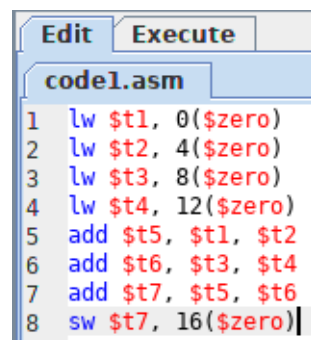
editar conteúdo. Verifique os atributos da memória como o tamanho do endereço e o tamanho dos dados. A figura abaixo ilustra o conteúdo da memória após o carregamento dos dados.

```

00 00000010 00000020 00000030 00000040 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
10 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
20 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
30 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
40 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
50 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
60 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
70 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
80 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
90 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
a0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
b0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
c0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
d0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
e0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
f0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

```

Após, devemos carregar um programa (instruções) para a memória de instruções do processador. Para isso, carregue o arquivo code1.mem para a memória de instruções. A memória de instruções possui seu conteúdo em hexadecimal para facilitar a visualização. O arquivo code1.mem possui as seguintes instruções:



```

Edit Execute
code1.asm
1 lw $t1, 0($zero)
2 lw $t2, 4($zero)
3 lw $t3, 8($zero)
4 lw $t4, 12($zero)
5 add $t5, $t1, $t2
6 add $t6, $t3, $t4
7 add $t7, $t5, $t6
8 sw $t7, 16($zero)

```

Obs: O registrador \$zero sempre possui seu conteúdo igual a 0. Veja como isso é implementado no banco de registradores.

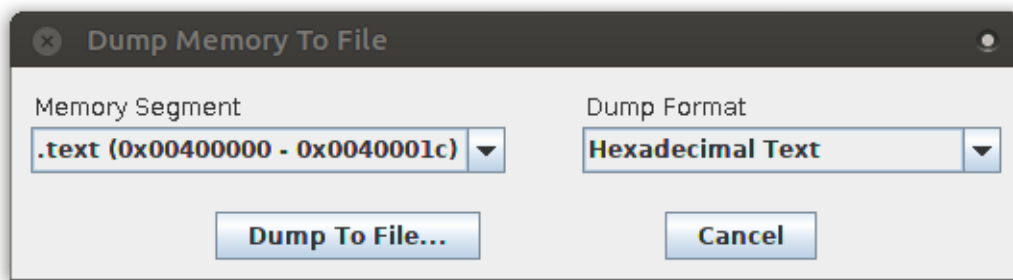
Após carregar as informações para a memória de dados e de instruções, clique em habilitar simulação, depois no botão com o dedo indicador para manipular o circuito e depois no botão vermelho “Restart Program” e “Zero Reg File”. O primeiro botão irá iniciar o registrador PC com zero indicando um reinício do programa. O segundo botão irá zerar todos os registradores do banco de registradores.

**Verificação:** Clique no botão “clock” repetidamente e veja o seu programa executar no processador. Cada instrução é executada em um ciclo de clock. Veja o conteúdo da memória e do banco de registrador após a execução do programa.

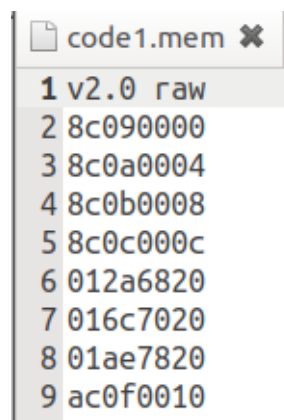
**Tarefa 6:** Criar um programa no MARS e um novo arquivo para a memória de dados. Crie um programa que utilize a instrução BEQ.

Abra o MARS e escreva o seu código diretamente. Crie um simples mas um novo programa. Não é necessário ter o .text ou .data no início do programa como vimos nas aulas anteriores de laboratório. Após digitar o código no MARS, clique em “Run” e depois em “Assemble”. Será gerado o código de máquina para o seu código MIPS. Depois, clique em “File” e “Dump memory”. Em “Dump Format” selecione “Hexadecimal Text”, como ilustrado abaixo. Não é necessário modificar o campo “Memory Segment”.





Clique em “Dump To File” e selecione o nome do arquivo. Após, abra o seu arquivo e adicione na primeira linha “v2.0 raw”, como ilustrado na figura abaixo. Como você usou outras instruções e registradores, as demais linhas serão diferentes.



```
1 v2.0 raw
2 8c090000
3 8c0a0004
4 8c0b0008
5 8c0c000c
6 012a6820
7 016c7020
8 01ae7820
9 ac0f0010
```

Para criar o arquivo da memória de dados, abra o seu editor de texto favorito coloque números separados por espaço. A primeira linha também deve conter v2.0 raw”. Abra o arquivo dados1.mem para verificar como deve ser o arquivo.

**Tarefa 7:** Modificar o caminho de dados e os circuitos necessários para introduzir a execução da instrução *addi \$regDestino, \$regFonte, IMEDIATO*. Para testar o seu circuito, execute um código que possui a instrução *addi*. O OP CODE da instrução é 001000. Você deverá modificar os circuitos das Tarefas 1 e 2 para que seu processador execute a referida instrução.

## **Versões**

Versão 1.0 – 2018/2 – Daniel Ludovico Guidoni