Universidade Federal Rural do Semiárido
Departamento de Computação
Ciência da Computação
Arquitetura e Organização de Computadores
Prof. Sílvio Fernandes

2.4 – Trabalho de Implementação

Trabalho em dupla ou individual
Data de entrega (via SIGAA): 22 de Setembro de 2023
O trabalho deve ser apresentado em sala de aula para o professor

Descrição:
Desenvolver o circuito do controle do processador MIPS básico usando o Logisim para o caminho de dados que executa as instruções **LW, SW, BEQ, ADD, SUB, AND, OR** e **SLT**.

A definição completa do circuito do processador MIPS para o Logisim deve ser salvo com o nome dos componentes do grupo da seguinte forma: NomeA-NomeB-mips-Control.circ.

Para começar:

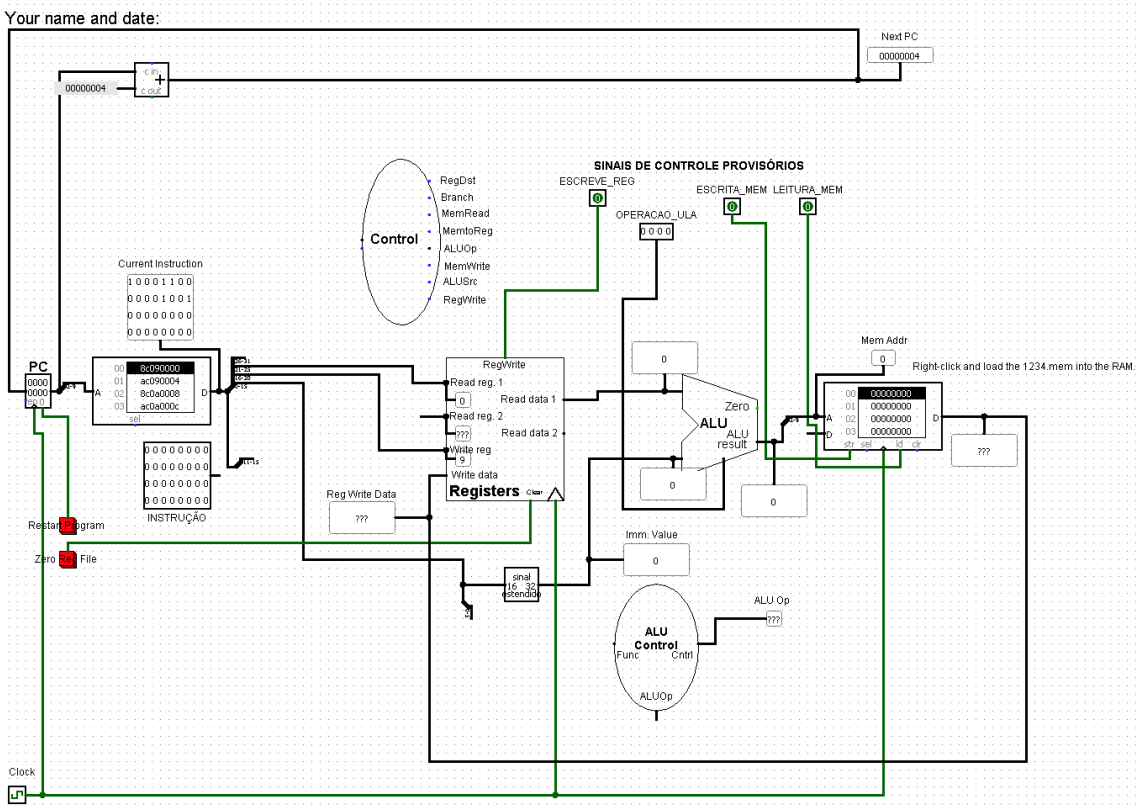
Leia o capítulo 4 do livro principal desta disciplina listado nas referências no final deste texto.

Utilize a versão 2.7.1 do Logisim. Para executar este software é necessário que em seu computador também esteja instalado o *Java Runtime*.

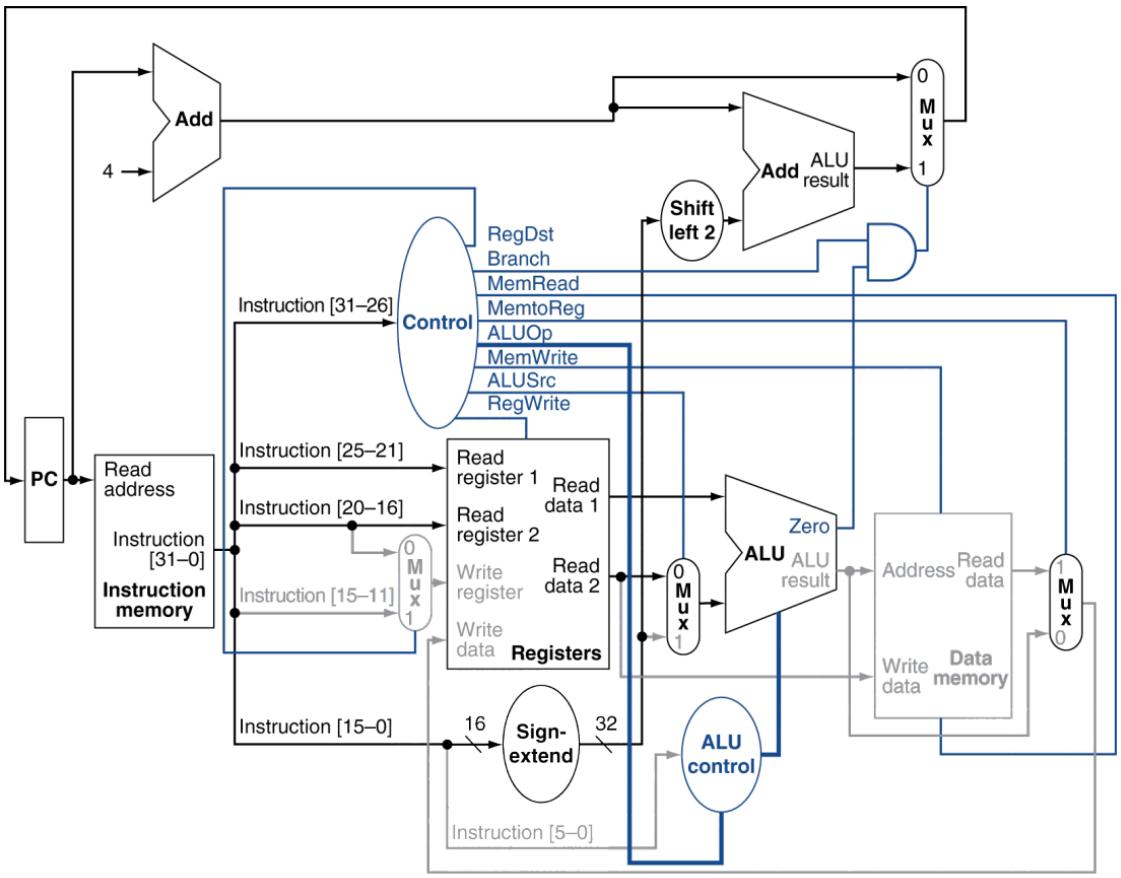
Utilize como ponto de partida o arquivo mips-datapath-lab.zip disponibilizado com esta definição do trabalho no SIGAA. Tal arquivo .zip contém os demais arquivos:

- mips-datapath-lab-R-Saltos.circ: um "esqueleto" do processador MIPS básico com importantes partes faltando propositalmente.
- 1234.mem: um arquivo hexadecimal que contém os valores 1, 2, 3, 4 para inicializar as primeiras 4 palavras da **memória de dados**.
- test-code.mem: um arquivo que contém um programa de teste simples para verificar seu projeto

Abra o arquivo de projeto Logisim ".circ" e verá o seguinte circuito correspondente ao MIPS com algumas partes faltantes:



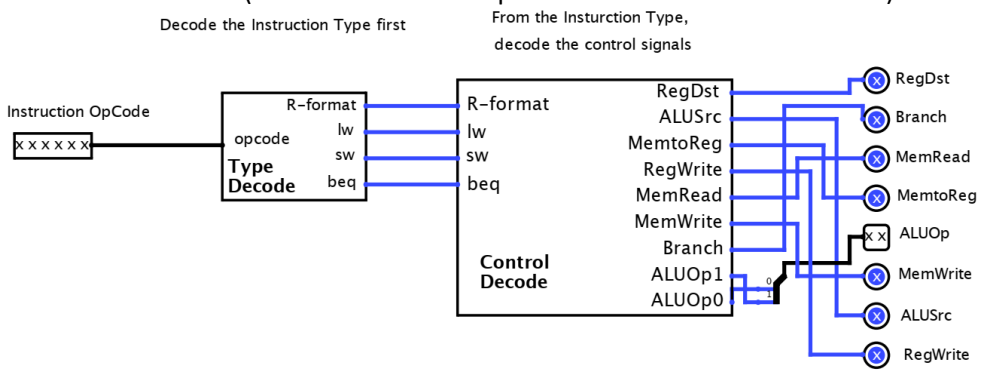
Este circuito após completado deve ser equivalente ao processador apresentado no livro:



Implementação:

Perceba que as linhas em azul representam os sinais que saem do controle e estabelecem o caminho de dados para cada instrução, baseado no código da instrução (codop) e o campo *function* apenas quando a instrução é do tipo R.

Unidade de controle:
Ao dar um duplo clique em "control" você verá a definição do controle, o qual é formado por outros 2 subcircuitos (o decodificador de tipo e o decodificador do controle):



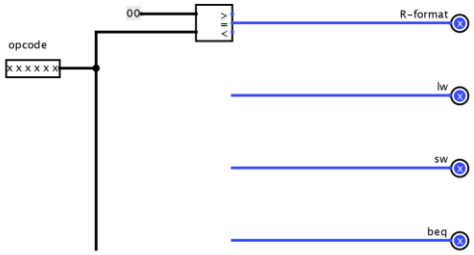
Na figura você as seguintes partes:

1) A entrada: o OpCode dainstrução formada por 6 bits
2) O "Decodificador de tipo" que recebe o OpCode e mostra se a instrução é uma instrução Rformat, lw, sw ou beq.
3) O "Decodificador do controle" que recebe o tipo de instrução e gera os sinais de controle corretos caminho de dados

Parte do seu trabalho será completar o "Decodificador de tipo" e "Decodificador do controle".

Agora vamos chegar ao trabalho real de fazer a lógica de controle funcionar. Existem três partes que você precisará completar: a unidade "Type Decode", a unidade "Control Decode" e a unidade "ALU Control".
Cada um usa uma maneira ligeiramente diferente de gerar sua saída, então fique atento!

Decodificador de tipo:
Comece dando uma olhada dentro da unidade "Type Decode":

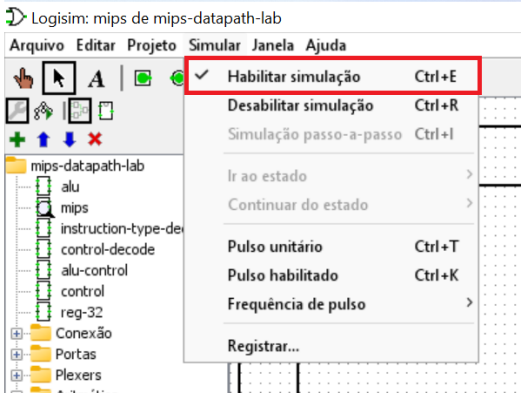


Aqui você vê que esta unidade não está completa. Você precisará completá-lo para concluir o trabalho. No entanto, há uma dica sobre como fazer isso funcionar. Observe como um comparador e um valor constante são usados na parte superior para determinar se a instrução está no formato R. O que essa lógica faz é determinar se o "OpCode" é igual a 00. Se for, então a saída "=" do comparador será verdadeira e o formato R a saída do bloco será definida como 1. Seu trabalho é preencher o restante desta unidade para que ela decodifique corretamente LW, SW e BEQ quando esses tipos de instruções são encontrados.

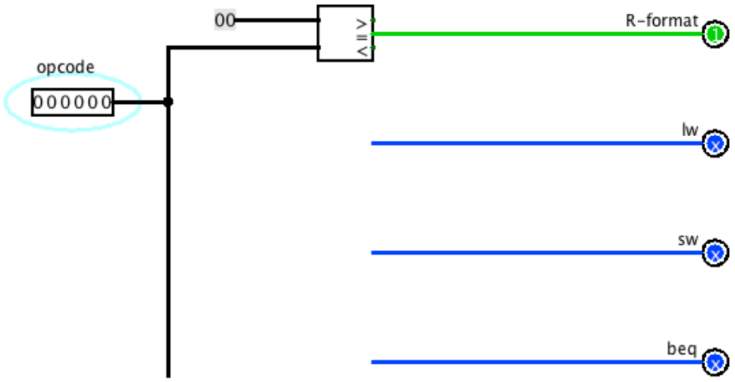
Para determinar o tipo de instrução você precisa seguir as informações do livro, conforme a tabela a seguir que identifica os 6 bits de “OpCode” das instruções a serem implementadas:

Entrada ou saída	Nome do sinal	formato R	lw	sw	beq
Entradas	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

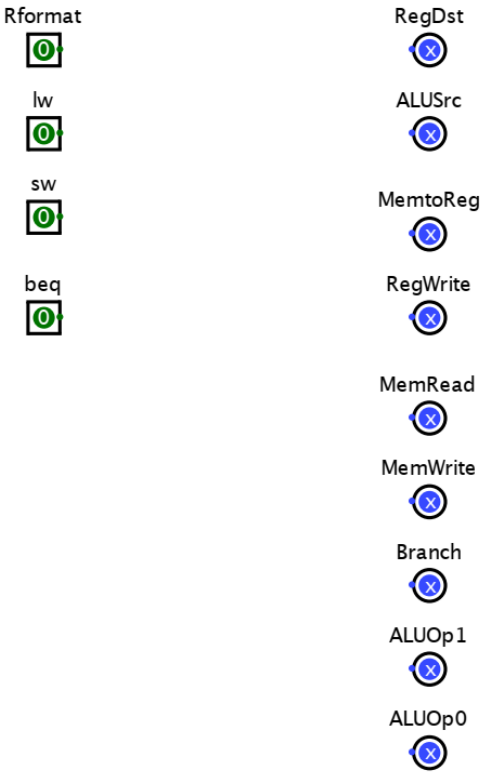
Testando o decodificador de tipo de instrução:
Depois de construir o circuito, você precisa testá-lo. Para fazer isso, tenha certeza que a opção “Habilitar Simulação” no menu “Simular” esteja ativada, conforme a figura:



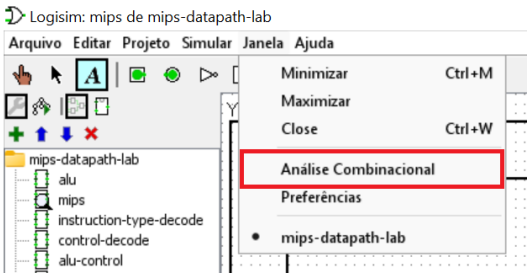
Certifique-se que está com o curso de simulação selecionado (a mãozinha ou Ctrl+2). Em seguida, cliquei em todos os bits “OpCode” para torná-los 0 (não se preocupe com avisos de que o estado é definido de outro lugar; isso está apenas dizendo que este bloco está conectado). Quando todos os bits estiverem definidos como zero, o a saída indicando que é o formato R se torna 1 (verde claro) tal como a figura a seguir. Teste assim para todas as quatro entradas depois que você implementá-las.



Decodificador do controle:
Depois de concluir a unidade “Type Decode”, você terá metade da lógica de controle finalizada. A outra metade está usando o tipo de instrução decodificada no circuito anterior para determinar os sinais de controle a serem enviados. Dê uma olhada dentro a unidade de decodificação de controle (Control Decode) para ver que você precisa preencher toda a lógica, uma vez que ela só tem as entradas e saídas definidas:



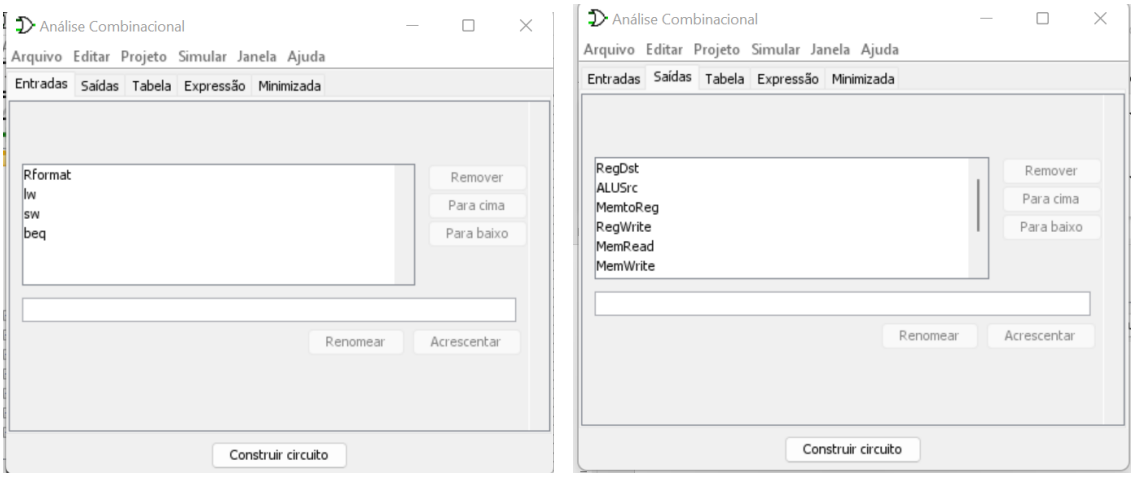
Bem, isso é o que você esperaria: as entradas são o tipo de instrução (à esquerda) e os sinais de controle são as saídas (à direita). Agora, para gerar essa lógica, é recomendado usar a ferramenta “Análise Combinacional” no menu “Janela” no Logisim para construí-lo para você, conforme a figura a seguir.



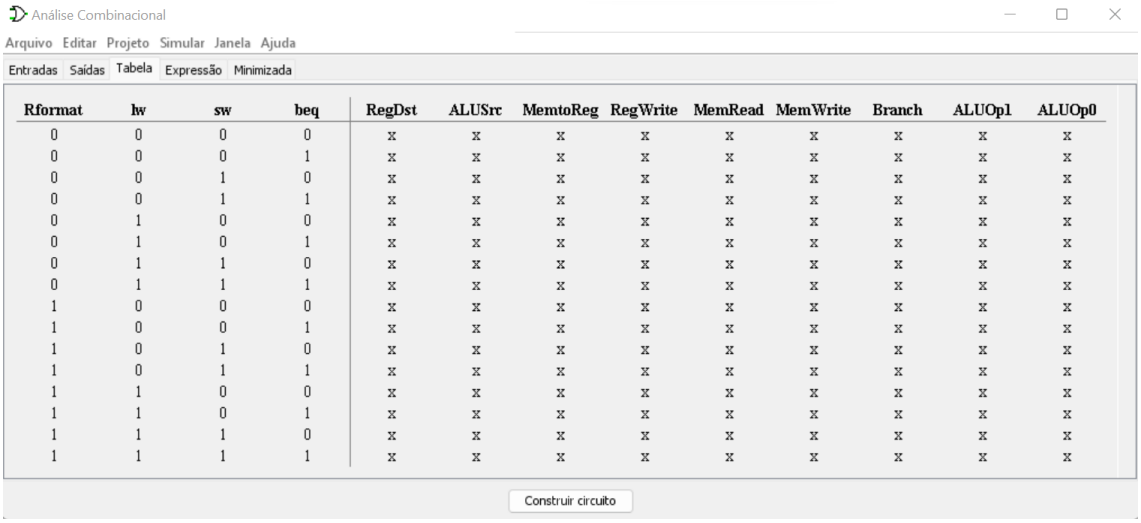
Para fazer isso você precisa saber quais devem ser os valores esperados nas saídas para a combinação dos valores de entrada. Portanto, preencha a tabela-verdade abaixo para cada tipo de instrução. Lembre-se de que você pode (e deve) colocar um X para opções que não lhe interessam. Para fazer isso, você deve pensar sobre o que cada instrução faz. Se você apenas copiar isso do livro ou online, terá muito mais dificuldade para depurar mais tarde, porque não entenderá também. Portanto, entenda o efeito de cada sinal de controle no caminho de dados para cada instrução.

Output	R-format	Lw	Sw	Beq
RegDst	1	0	X	X
ALUSrc				
MemtoReg				
RegWrite				
MemRead				
MemWrite				
Branch				
ALUOp1				
ALUOp0				

Agora que você tem a definição para a Unidade de Decodificação de Controle acima, é hora de construí-la. Ou apenas olhe para ele e descubra a lógica (não é tão complicado) ou use a ferramenta “Análise Combinacional”. Para a Ferramenta de “Análise Combinacional”, primeiro você precisa definir as entradas e saídas do seu circuito. Estes são os cabeçalhos de linha e coluna da sua tabela verdade acima, as quais já estão definidas:



Em seguida, insira os valores na aba “Tabela”. Lembre-se de que se você não se importa com uma saída específica, basta marcá-la como X. Clique nos valores para alterá-los. Você também pode navegar pelos valores da saída da tabela-verdade usando as setas do teclado.



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

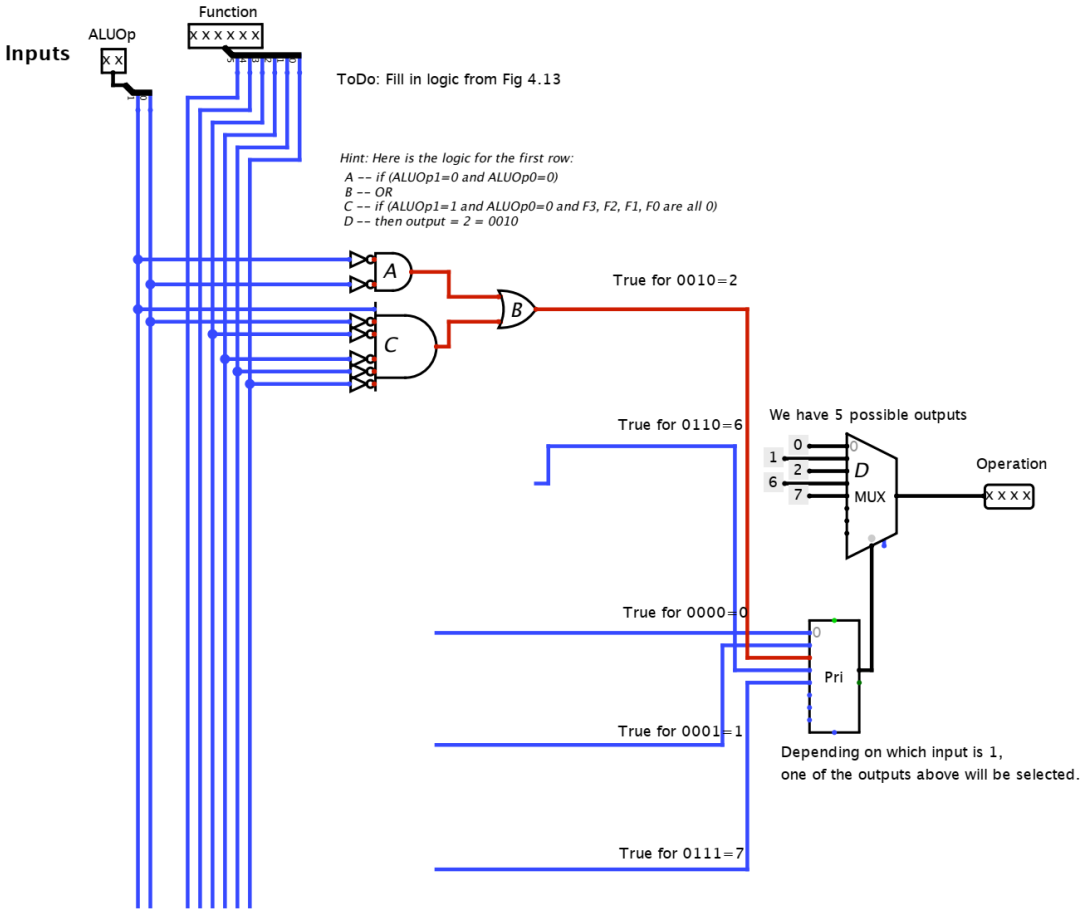
Agora que você fez isso, você pode deixar o LogiSim fazer todo o trabalho duro para você. Em “Expressão” você pode ver as equações lógicas para cada saída, e em “Minimizado” você pode ver os mapas de Karnaugh para cada expressão. Para LogiSim construir o circuito usando

portas lógicas basta clicar em “Construir Circuito” e adicione-o ao projeto atual substituindo o circuito incompleto anterior.

Testando o decodificador de controle:
Depois de construir o circuito, você precisa testá-lo. Faça isso da mesma forma que você fez para o circuito anterior e certifique-se de que os sinais corretos estejam configurados para cada tipo de instrução. Caso algum sinal de saída não seja atribuído com o valor correto, verifique se as conexões nos pinos de entrada e saídas estão feitas corretamente.

Parabéns! Agora você construiu a lógica de controle total e testou cada componente. No entanto, recomendamos fortemente que você teste a coisa completa. Para isso, vá ao circuito de controle (Control) e teste toda a cadeia. Por exemplo, coloque as entradas no “OpCode” da Instrução e verifique se os sinais de controle corretos estão definidos na saída.

Unidade de controle da ULA:
Dê uma olhada dentro da unidade de controle da ALU (alu-control). Esta unidade tem 8 entradas (2 bits “ALUOp” e 6 bits “Function”), portanto, há muitos fios. Na verdade, está muito bem definido. Todas as entradas vão verticalmente para baixo no lado esquerdo e a saída está na extrema direita. A lógica que conecta as entradas às saídas fica no meio. Gaste alguns minutos para descobrir o que a parte à direita está fazendo para gerar a saída antes de ler o próximo parágrafo.



Se você observar a lógica à direita, a saída vem de um MUX (D). O MUX tem valores constantes como entradas (0, 1, 2, 6 e 7). O MUX é acionado por um codificador de prioridade (Pri), que

recebe sinais da lógica no meio. O que acontece é que se uma das entradas do codificador de prioridade for verdadeira, o codificador de prioridade dirá ao MUX para selecionar essa entrada, que produzirá a constante.

Para fazer tudo isso funcionar, você só precisa construir a lógica que determina qual entrada para o codificador de prioridade deve ser verdadeira dadas as entradas “ALUOp” e “Funciton” à esquerda.

Dê uma olhada na tabela verdade para controlar a ULA:

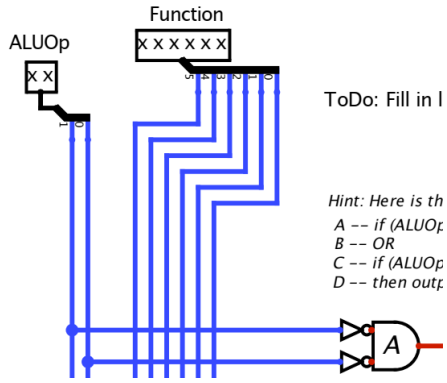
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Operation
0	***0***	***X***	***X***	***X***	***X***	***X***	***X***	***0010***
0	1	X	X	X	X	X	X	0110
1	***0***	***X***	***X***	***0***	***0***	***0***	***0***	***0010***
1	X	X	X	0	0	1	0	0110
1	0	X	X	0	1	0	0	0000
1	0	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

Observe o exemplo a seguir correspondente as linhas sombreadas na tabela (as duas combinações que fornecem 0010 como saída). Tente anotar qual deve ser a lógica para determinar se a operação é 0010 antes de continuar lendo. Olhe para os bits na tabela e descubra quais são as condições.

Vamos dar uma olhada na primeira linha sombreada. Nós temos:

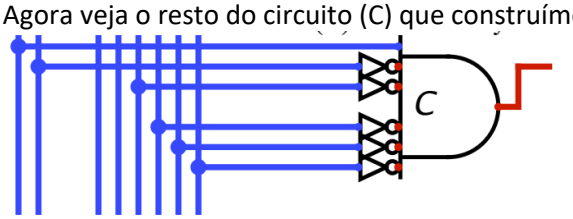
- ALUOp1=0, ALUOp0=0, e todos os Fs são X. Então a lógica aqui é:
- A operação deve ser 0010 sempre que ALUOp1=0 E ALUOp0=0.

(Nós não nos importamos com os Fs porque eles são Xs.)
A partir dessa definição podemos gerar a primeira parte do circuito:

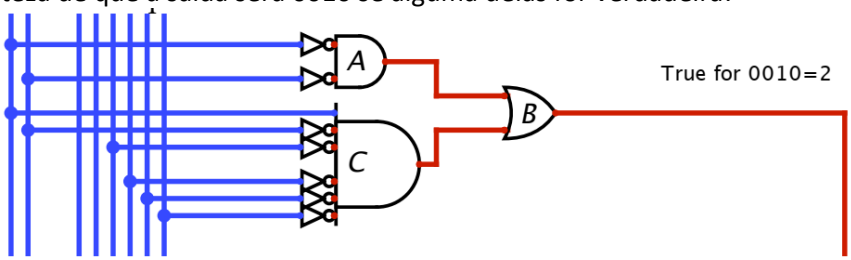


A metade superior (A) é uma porta AND entre (NÃO ALUOp1) e (NÃO ALUOp0). Isso significa que a saída dessa porta AND será verdadeira quando ALUOp1=0 AND ALUOp0=0. (O que é, não surpreendentemente, exatamente o que queremos.)

A segunda linha sombreada é:
• ALUOp1=1 E ALUOp0=0 E F3=0 E F2=0 E F1=0 E F0=0
Agora veja o resto do circuito (C) que construímos a partir disso:



Agora temos dois casos em que a saída deve ser 0010. Precisamos de uma porta OR final (B) para ter certeza de que a saída será 0010 se alguma delas for verdadeira:



O resultado da porta OR é 1 sempre que a saída deve ser 0010. Para fazer a saída ser 0010 esse valor vai para o codificador de prioridade, que seleciona a entrada correta do MUX para enviar os resultados.

Observe que todas as entradas têm Xs para F4 e F5. Isso significa que nunca usamos seus valores. Portanto, se você estiver conectando F4 ou F5 aqui, provavelmente cometeu um erro.

Usando a tabela do acima, complete o circuito para as outras possíveis saídas.

Testando o controle da ULA:
Depois de construir o circuito, você precisa testá-lo. Faça isso da mesma forma que você fez para o circuito anterior e certifique-se de que os sinais corretos estejam configurados para cada tipo de instrução. Use o ícone da mãozinha (Ctrl+1) para o modo de simulação e atribuir valores as entradas.

Testando o processador completo com um código

Para testar seu processador, fornecemos o seguinte programa:

```
lw  $t1,  0($zero)
lw  $t2,  4($zero)
lw  $t0,  12($zero)
loop:
add  $t3,  $t1,  $t2
add  $t4,  $t3,  $t1
sub  $t0,  $t0,  $t2
or   $t5,  $t4,  $t1
slt  $t6,  $t4,  $t5
beq  $t2,  $t0,  loop
```

Este programa aparece em código de máquina, na forma hexadecimal, no arquivo “test-code.mem” fornecido no projeto. Tal arquivo é usado para inicializar a memória de instruções.

E um arquivo de memória que contém os valores 1,2,3 e 4 nas primeiras 4 palavras da memória (1234.mem). Para usar esses arquivos, você precisa clicar com o botão direito do mouse na memória de instruções e nas memórias de dados e carregar os arquivos apropriados. Também é possível criar seus próprios programas e inicializar as memórias com as informações que desejar para executar seus testes.

Depois dos arquivos carregados nas memórias habilite a simulação, clique nos botões “Reset Program” e “Zero Register File” para limpá-los. E, em seguida, escolha “Tick Once” no clock para avançar cada instrução.

Quando você começa a simular, sua primeira instrução estará no processador. Certifique-se de que tudo está funcionando. Verifique as entradas e saídas, os MUXes, os bits de controle. E, finalmente, verifique se o valor está sendo escrito de volta no arquivo de registro e o registro sendo escrito. (Ambos são exibidos nas “pontas de prova” enquanto o processador é executado.)

Quando seu programa ficar sem instruções, você pode verificar novamente o conteúdo do banco de registradores abrindo-o.

Dicas para debugar:

- Não avance além da primeira instrução até que esteja funcionando corretamente.
- Verifique se os valores de todos os registradores (registrador selecionado e seu respectivo valor) estão corretos e se a operação da ULA está correta.
- Use a ferramenta dedo (mãozinha) para ver os valores nos fios do processador.
- Trace os valores de volta ao primeiro lugar onde eles estão errados, e então vá para essa parte para ver o que está acontecendo.
- Você pode entrar na unidade de controle enquanto seu processador está simulando para ver onde estão os erros.
- Você pode visualizar o conteúdo do banco de registradores visualizando dentro dele também.
- Escreva o que você espera que aconteça e verifique. Não adivinhe.
- Quando você precisar reiniciar o programa, clique em Restart Program e clique em Zero Reg File.
- Se você reiniciar a simulação, a memória será zerada e você terá que recarregar os dados da memória.

REFERÊNCIAS:

HENNESSY, John. Organização e Projeto de Computadores. 5 ed. Rio de Janeiro: Elsevier. Grupo GEN, 2017. E-book. ISBN 9788595152908. Disponível em: https://app.minhabiblioteca.com.br/#/books/9788595152908/.

FOYER, Per. Notas de aula de “Computer Architecture - 1DT016”. Disponível em: https://xyx.se/1DT016/labs.php#lab1