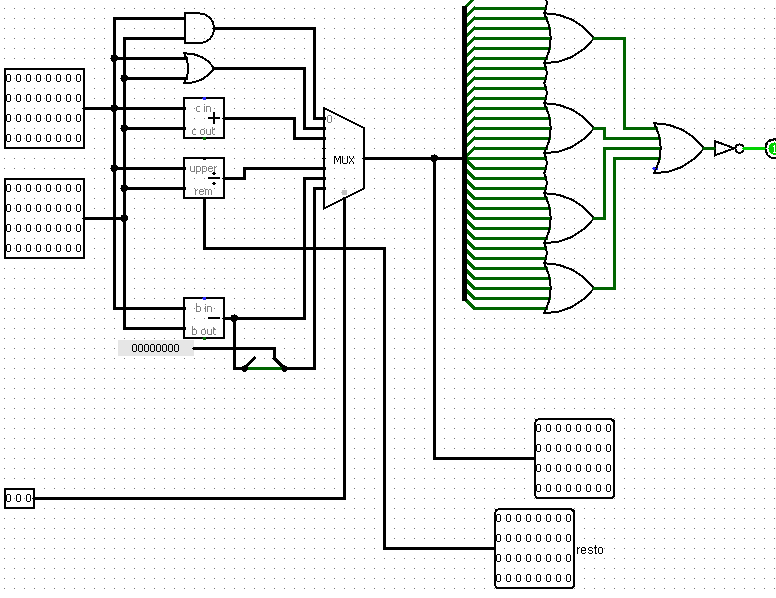
INF01113 – Organização de Computadores B (2023/1)

Trabalho pratico 1 - Grupo 4

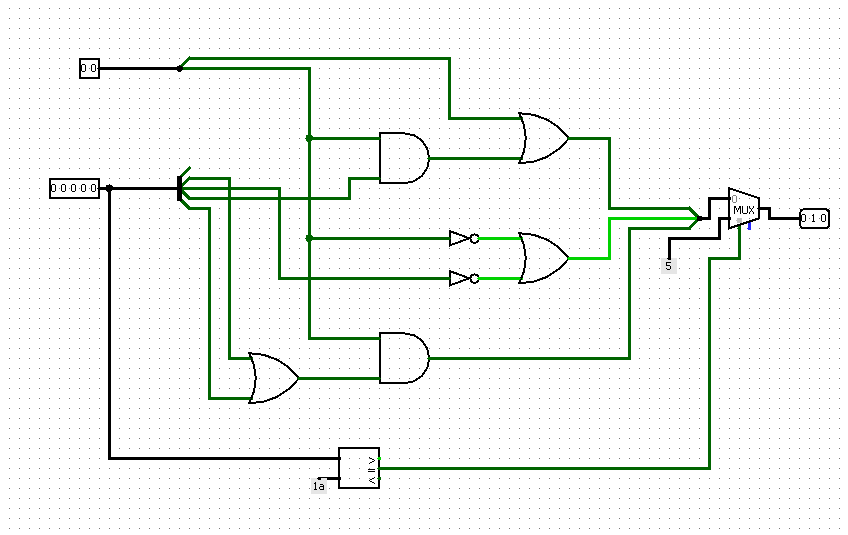
**Monociclo**

**Div**

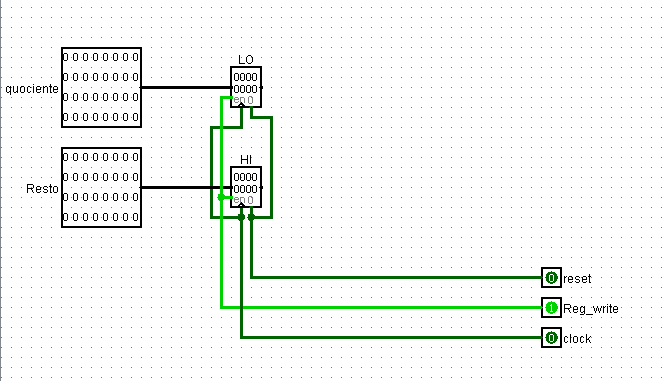
Primeiro adicionamos um divisor na ALU, com o rs sendo dividido por rt, e conectamos o quociente na porta 101 do mux da ALU e o resto num registrador(pino)



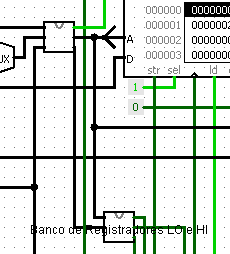
Agora, precisamos implementar uma lógica no ALU\_Control para que saia 101 dado o campo func seja igual a 11010, para isso apenas adicionamos um comparador para que compare a entrada do ALU\_Control com 11010, se for igual, o bit de saída aciona o mux para que a saída seja 101



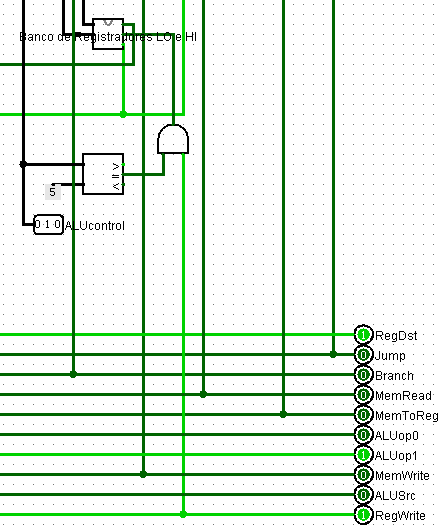
Agora só precisamos escrever o resultado, a operação DIV guarda o quociente e o resto da divisão em dois registradores especiais, LO e HI, respectivamente, logo criamos um banco de registrador auxiliar, BancoReg\_DIV, com esses dois registradores



E no Datapath ligamos a saída da ALU nesse banco de Registradores auxiliar

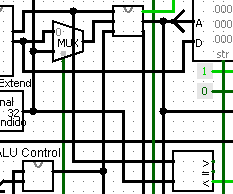


Porém, só queremos escrever nos registradores especiais quando for uma instrução DIV, então adicionamos um comparador entre a saída do ALU\_Control e 101, e o bit resultante do comparador numa porta AND junto com regWrite para saber que precisamos escrever nesses registradores especiais. Com isso temos a instrução de divisão no MIPS monociclo

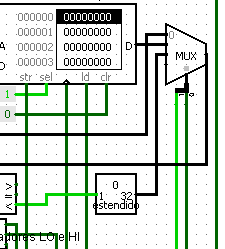


**SLTIU**

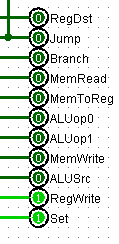
Adicionamos um comparador com o rs sendo comparado com o imediato com sinal estendido.



Se o rs for menor que o valor imediato, o comparador vai retornar 1, senão, retorna 0. Esse valor é estendido com zeros e é inserido no mux que escreve no registrador, usado na instrução lw.

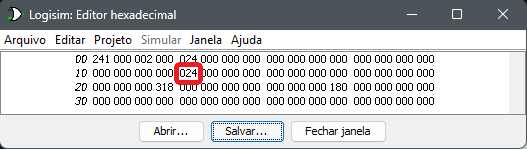


Como foi aumentado o mux, é preciso de outro bit de controle, chamado de Set.  
Por fim, usando o opcode da instrução, o bloco de controle ativa esses bits:



**BLTZ**

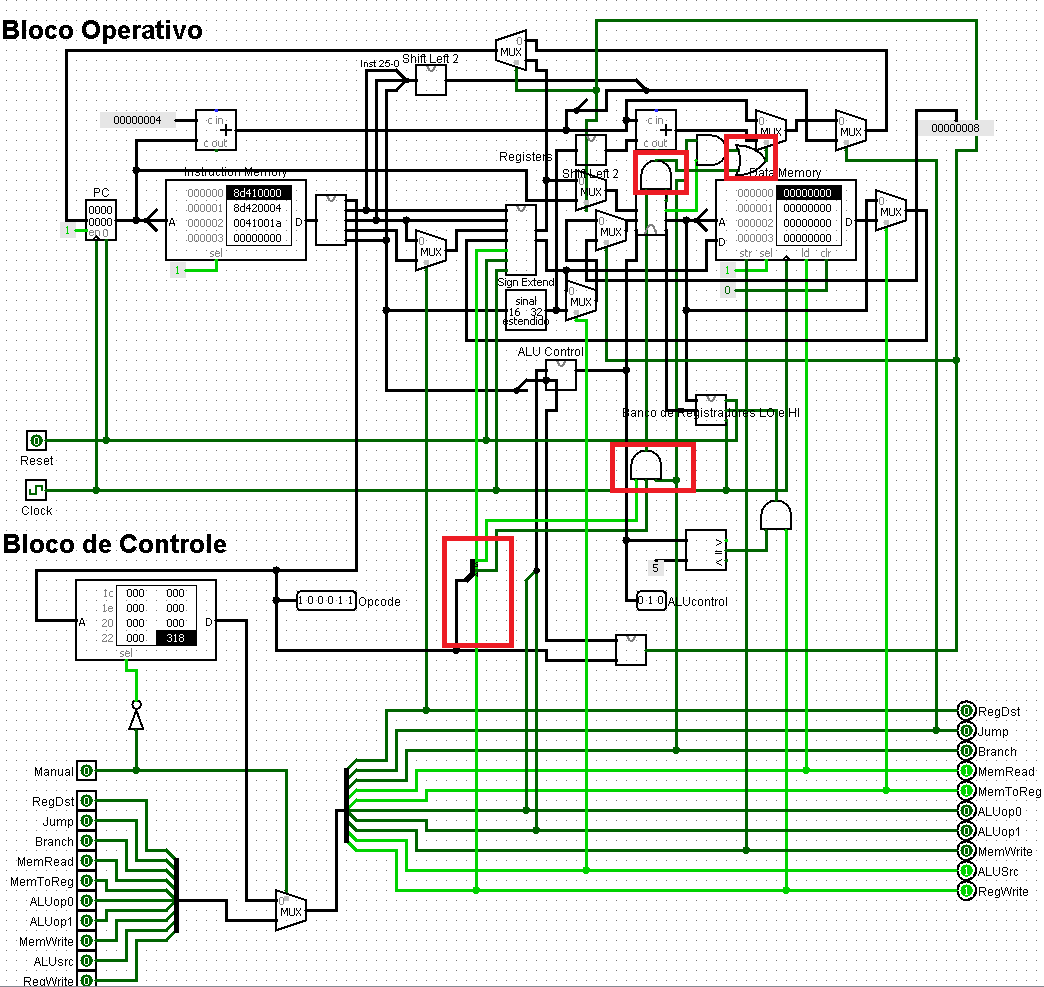
Utilizando Opcode 0x14 (11 0100 em binário). No controle, precisamos do sinal 1 no Branch, e 1 no ALUop0, o resto 0. Isso gera um controle 0x024, igual o do BEQ, porém para diferenciar as instruções, será testado os bits diferentes do Opcode (11 XXXX):



ALU Recebe sinal binário 110 (6a porta do MUX, que faz a subtração dos dois registradores), mandado pelo sinal do ALUop0. No entanto, foi necessário adicionar um fio saindo da ALU que indica que o resultado da operação é negativo (testando o bit mais relevante).

****

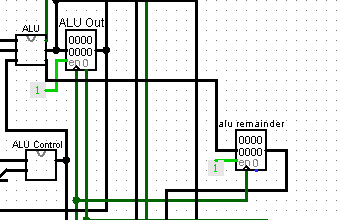
Apos isso, faço um teste AND pra checar se é a instrução BLTZ (achei menos custoso que adicionar um sinal novo ao controle), e se o resultado da ALU for zero, executa o branch:



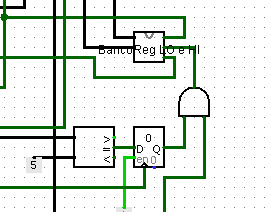
**Multiciclo**

**Div**

Mantemos quase todos os componentes do monociclo para o multiciclo, a diferença é que agora precisamos manter o resultado do resto da ALU entre um ciclo e outro, para isso adicionamos um registrador “alu remainder” na saída do resto da ALU

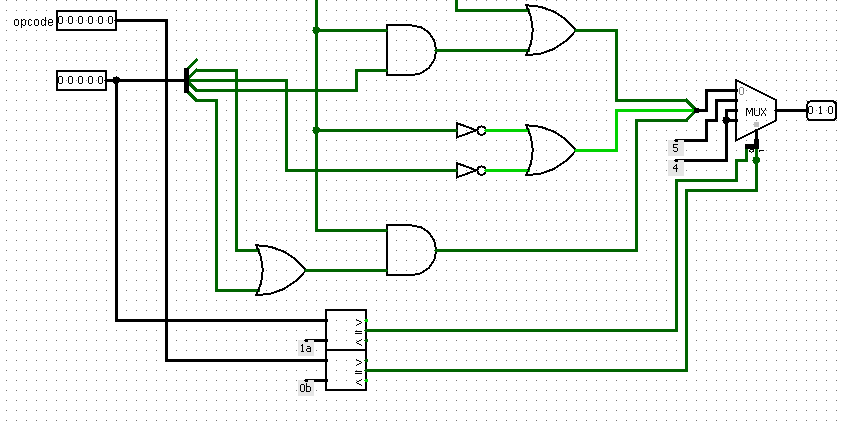


Também é necessário manter a informação que a operação realizada foi um DIV, para gravarmos nos registradores especiais, para isso apenas foi adicionado um registrador na saída do comparador da ALU\_Contro e 101

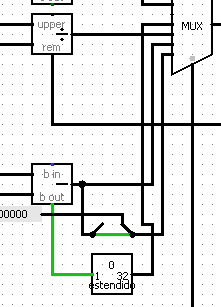


**SLTIU**

Começamos por mudar o ALU control para receber o opcode e testar se o valor é B, se for, retorna o código 4



Na ALU, o código 4 está ligado ao borrow out da subtração estendido para 32 bits



Por fim, resta mudar os estados no controle

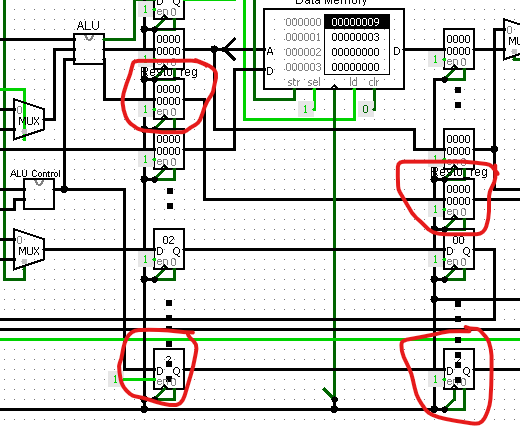


O estado 1 é o decode e o 2 é para usar a ALU com os valores do registrador rs e o valor imediato signext e o estado B é para escrever o resultado da ALU no registrador rt.

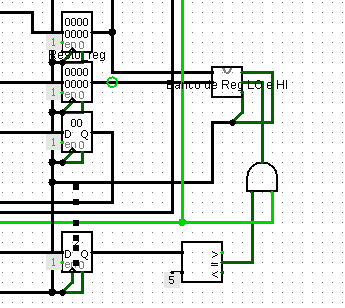
**Pipeline**

**Div**

No pipeline, aproveitamos várias partes dos componentes, as diferenças são que agora precisamos guardar o resto da alu e a saída do ALU\_Control por dois passos. Fazemos isso colocando um registrador intermediário para cada saída, em cada passo, ou seja, 2 regs por passo

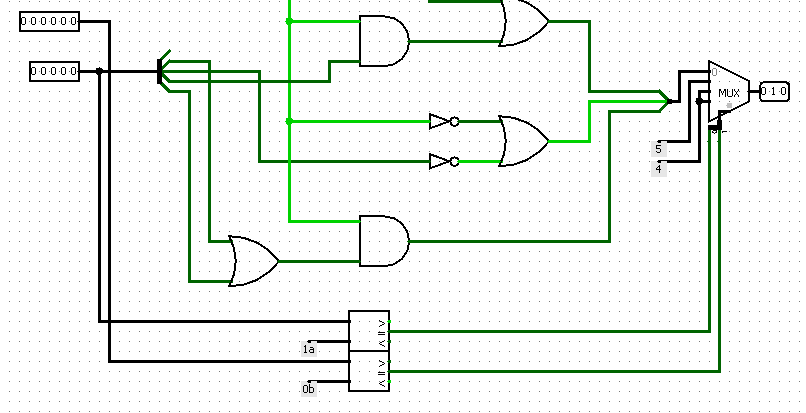
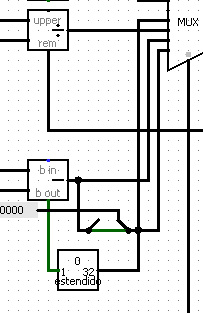


A operação de escrever nos registradores LO e HI voltou a ser como era no monociclo, pois agora temos o dado da ALU\_Control preservado pelo segundo registrador intermediário do ALU\_Control

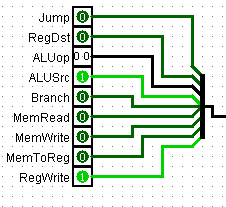


**SLTIU**

Fazendo o mesmo que no multiciclo, adicionando o opcode no controle da ULA e o sinal de borrow na ULA.



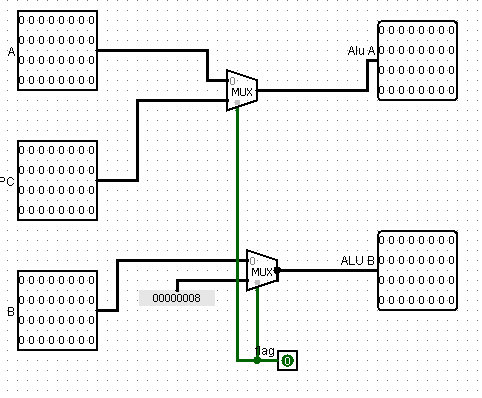
É necessário apenas ligar os bits de controle de escrever no registrador e o bit para o ALU pegar o valor imediato.



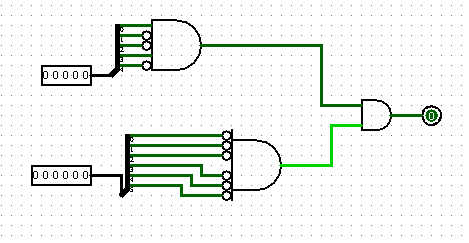
**JALR mono, multi e pipe bem breve:**

JALR é uma função especial que pode ser descrita em como um ADD que faz JUMP. Portanto vamos aproveitar o ADD que já é feito na função especial.

Primeiro paço é certificarmos que as coisas corretas estão indo até a ULA. Prcisamos do PC e uma constante 8 e somalos. O esquema fica assim:



A e B são as entradas originais se fosse qualquer outra função. Flag, como o nome já disse, é uma flag, criada por um outro circuito combinacional que apenas checa se essa é mesmo a função e liga os muxes. Ele se parece assim:



A primeira entrada é a entrada de alu control. Ela deve ser 01001.

A segunda entrada é o opcode e deve ser 000000. Se os dois estiverem corretos sabemos que se trata desta função em questão.

Por fim precisamos conectar os dados do registrador A a entrada de PC, separado por mux que será acionado pela nossa flag.

Para o monociclo apenas essas mudanças bastam.

Para o multiciclo foi necessário adicionar mais um ciclo para o opcode 000000 onde o PCwriteCond será 1, comparando à nossa flag.

Para o pipe cond não existe, mas podemos ligar o flag por OR com PCwrite.