Questão 1: ORI

Para realizar a implementação da operação de *Bitwise OR Immediate*, foi necessária uma alteração no controle da ALU. Tanto este quanto a própria ALU já possuem capacidade de obter o valor do imediato e realizar a operação *OR* bit a bit entre o valor armazenado no primeiro registrador (rs1) e o imediato, e portanto precisamos apenas de garantir que o controle da ALU seja capaz de processar o valor do *funct* da operação *ORI*. Como a instrução *ORI* não possui *funct7* definido, pois os bits 31:25 são destinados para o imediato, o valor do *funct* que é passado para o controle da ALU – que é a concatenação do quinto bit do *funct7* ao *funct3* – não tem seu último bit bem definido, ou seja, precisamos colocar um caso específico no controle da ALU para checar se os três primeiros bits do *funct* são iguais ao *funct3* da operação *ORI*, que no caso é 110. Ou seja, o controle da ALU agora informa que a ALU deve realizar a operação de *OR* se os três primeiros bits do *funct* recebido forem iguais a 110 e se o *aluop* for igual a 10. Essa combinação de *funct* e *aluop* permitem identificar unicamente a operação de *ORI*, fazendo com que ela possa ser implementada corretamente. Vale ressaltar que também seria possível resolver o problema em questão de uma forma alternativa: identificar que a operação de *ORI* e concatenar ao *funct3* o número 1, assim o *funct* passado para a alu seria igual a 1110 (14 em base decimal), no entanto optamos por identificar somente o *funct3* em conjunto ao *opcode*. O notebook fornecido contém exemplos da corretude da instrução implementada.

Questão 2: SLLI

Para realizar a implementação da operação de *Shift Left Logical Immediate*, precisamos adicionar um caso específico para essa operação no controle da ALU. Isso se deve ao fato de tanto o *funct3* quanto o *funct7* da operação de *SLLI* serem bem definidos, pois o *shift amount* é reservado apenas para 5 bits da instrução, e portanto o quinto bit do *funct7* é sempre 0. Assim sendo, o *funct* que é passado para o controle da ALU será sempre 0001. Adicionamos esse caso no controle da ALU e criamos um código que é passado para a ALU. A ALU então realiza a operação de *SLLI* no valor armazenado no registrador com o *shift amount* fornecido. O notebook fornecido contém exemplos da corretude da instrução implementada.

Questão 3: BLT

Para realizar a implementação da operação de *Branch on Less Than*, precisamos de realizar três alterações no código fornecido:

1. Primeiramente, precisamos corrigir um bug presente no controle do datapath. O controle não inicializava a variável *brach_lt* como 0 por padrão, ou seja, uma vez que esse sinal fosse igual a um, qualquer operação na ALU com resultado final igual a zero também resultaria em um branch.

2. O segundo passo necessário para implementar o branch foi uma alteração na ALU e no controle da ALU. Primeiramente, a ALU não alterava o valor da variável *zero* em nenhum momento, ou seja, nenhuma operação de branch era possível na CPU. Assim sendo, adicionamos uma checagem condicional para alterar o valor dessa variável para 1 caso o resultado da ALU seja 0. Dessa forma, a operação de *BEQ* agora funciona, pois ela subtrai o valor de dois registradores rs1 e rs2, e caso eles sejam iguais essa subtração é igual a zero, e portanto a variável *zero* será igual a 1 e o branch será de fato realizado.

3. Por fim, precisamos adicionar uma checagem pelo *funct3* do branch no controle da ALU, para podermos diferenciar entre *BEQ* e *BLT*. Caso o *funct3* seja 100, enviamos para ALU o código que identifica a operação de *SLT* (Set Less Than), e caso contrário enviamos o código que identifica a operação de *SUB* para o *BEQ*.O *SLT* é uma das operações padrão da ALU e podemos realizar o *BLT* por meio dela. A *SLT* irá armazenar o valor 1 no bit 0 do resultado da ALU se o valor em rs1 for de fato menor que rs2, ou seja, basta checar se o bit 0 do resultado da ALU é igual a 1 quando a instrução de *BLT* for fornecida. Vale ressaltar que o código *main.v* continha um bug nessa checagem. Em vez de checar pelo bit 0 do resultado da ALU, ele checava pelo bit 31, que é sempre 0 quando se realiza a operação *SLT*, ou seja, o bug fazia com que a operação de *BLT* nunca acontecesse.

Após realizar todos esses passos, foi possível implementar a operação de *BLT* de forma correta. O notebook fornecido contém exemplos da corretude da instrução implementada.