Trabalho Prático 2

Implementação de Data Path em um processador Risc-V

Tarcizio Augusto Santos Lafaiete - 2021040105
Gabriel Bifano Freddi - 2021039956

1. Introdução

Neste trabalho prático da disciplina de Organização de Computadores I, foi proposto para os alunos que a partir de uma estrutura base de um processador Risc-V implementado em verilog o caminho de dados fosse incrementado de forma a suportar quatro instruções básicas da linguagem de montagem. As instruções pedidas para este trabalho foram: XORI, AND, Jump e BLTU.

2. Implementações

2.1. XORI - XOR com imediato

Na implementação do XOR com imediato foi necessário realizar alterações na unidade de controle e na unidade de controle da ALU, para que assim o caminho de dados do Risc-V conseguisse suportar esta instrução. Na unidade de controle do processador pode-se aproveitar parte da implementação já existente para a operação ADDI, uma vez que ambas as operações possuem o mesmo opcode e acionam os sinais do barramento de controle, como por exemplo o alusrc que é colocado em alto para realizar as operações utilizando o imediato com fonte da ALU. Além disso, a geração do imediato de 32 bits é igual para estas operações, podendo então ser reaproveitadas. A única alteração feita na unidade de controle foi a troca do aluop de 0 para 3, está mudança tem a haver com as alterações feitas no controlador da ALU.

Falando do controlador da ALU então, nele foram alterados os cases do aluop, na qual quando o aluop está em 3 o aluctl recebe como resultado o valor correspondente ao identificador de alguma das operações realizadas pela ALU. Este identificador por sua vez é dado em relação ao valor de funct3 que no caso do XORI é 100, sendo que o identificador da operação xor na ALU é 13.

2.2. AND - AND Logic

A operação de AND no Risc-V seguiu uma ideia de implementação semelhante à implementação do XORI, sendo necessário alterações na unidade de controle e no controlador da ALU. No módulo de controle já havia uma implementação de ADD que foi utilizada como base, uma vez que ambas as operações na linguagem de montagem tem o mesmo opcode. Da forma que o barramento de controle foi implementado neste módulo, todos os sinais foram mantidos com o padrão, exceto o aluop assume o valor 3.

Com o aluop em 3, a alteração do controlador da ALU, foi semelhante ao do XORI em que através do funct3 do AND que é 111 o aluctl recebe 0 que é o identificador da operação and na ALU.

2.3. J - Jump incondicional

A operação do jump incondicional se diferenciou da implementação das operações anteriormente implementadas. Nesta operação foram necessárias alterações na unidade de controle, no controlador da ALU e no processo de decodificação para se ter esta operação funcionando da forma correta.

Na unidade de controle, já havia uma pseudo-implementação do jump incompleta. Nesta implementação ao ser feita a identificação do opcode o sinal de jump ia para alto. Além desse sinal foi adicionado o acionamento do alusrc foi colocado em zero para evitar o uso do imediato da operação nos cálculos da ALU, o aluop assumiu o valor de 2 e o imediato de 32 bits foi gerado conforme as especificações do Risc-V para o comando de JAL.

No controlador da ALU, ao receber o sinal de aluop como 2 ele envia para o aluctl 3 como o identificador da operação. Este valor de identificador não corresponde a nenhuma operação realizada pela ALU e quando isso acontece por padrão a ALU joga a sua saída para zero, assim é como se a ALU estivesse desativada durante os processos de cálculo do jump e então evitando a realização de operações incorretas no caminho de dados.

Por fim, para a realização do jump, no registrador jaddr_s2 foi realizada a soma entre o valor de pc guardado no registrador do estágio dois do pipeline com o valor do imediato gerado na unidade de controle. A partir deste valor o jaddr vai passando pelos registradores de estágio até chegar no registrador jaddr_s4, presente no estágio 4 do processador. Neste momento como a sinal de jump está em nível alto, então temos o valor de pc sendo alterado pelo valor de jaddr_s4 assim completando o processo do jump incondicional.

Importante ressaltar que não foi necessário para esta instrução qualquer tipo de implementação de stalls e de encadeamento, uma vez que isso já estava presente na implementação base fornecida pelo TP.

2.3 BLTU - Branch less than unsigned

Na implementação da BLTU tinha-se uma facilidade em relação às outras instruções. Esta facilidade era já existir no caminho de dados base do Risc-V um suporte para o comando BLT (branch less than) que em funcionalidade varia em relação ao BLTU apenas pelo fato de que este apenas realiza operações com valores não sinalizados e esta parte deveria ser implementada.

Então, na unidade de controle a alteração realizada foi adicionar uma logica OR para o acionamento do sinal branch_lt, fazendo então ele ir para nível alto caso o funct3 fosse igual a 100 (BLT) ou fosse igual a 110 (BLTU).

Como dito anteriormente, como o processador base implementado já havia suporte para a operação de BLT, não foi necessário a implementação dos processos de stall e caminhamento de dados e nem da passagem do valor do salto para pc.

Seguindo com a implementação no controlador da ALU, o aluop recebido é 1, para este aluop na implementação base era enviado para a ALU o identificador da subtração. Com a adição do BLTU ao conjunto de instruções suportadas pelo processador, foi implementada uma lógica semelhante ao processo realizado no caso em que o aluop é 3, em que o valor do aluctl depende do valor de funct3. Neste caso para os valores de funct3 de BLT, BNE e BEQ foi mantido o endereçamento do identificador da subtração. Agora, para o BLTU foi passado o valor 9, que é um novo identificador que precisou ser incorporado a ALU.

Como dito anteriormente, a ALU recebeu uma nova operação a ser realizada, esta que possui o identificador 9. Nesta operação realizamos a subtração de números não sinalizados. Para isso, pegamos o bit mais significativo dos valores a serem subtraídos e força-se eles a zero, assim sendo tratamos ambos os números como não sinalizados. Logo, caso seja passado para algum dos registrados o valor -1 que em complemento de 2 é representado por ffffffff se torna 7fffffff e é tratado como este valor para a realização da subtração, ou seja, caso tenhamos um caso em que há um BLTU entre 10 e -1, como a ALU irá interpretar -1 como não sinalizado o salto será realizado, pois 10 é menor que 7fffffff e ao ser feita a subtração entre estes dois números ocorrerá um underflow indicando então que rs1 é menor que rs2.

3. Conclusão

No geral, este TP foi desafiador para a dupla, uma vez que necessitava de uma interpretação e conhecimentos bons na organização do Risc-V e também um entendimento considerável sobre verilog. Além de haver dificuldades no entendimento da disposição do trabalho no jupyter notebook, o que acarretou em despender um tempo para o estudo e a familiarização com a disposição dos arquivos, conversões de assembly para o código de máquina e a produção das formas de onda.

Apesar disso, nós ficamos satisfeitos com o resultado do trabalho e sentimos que conseguimos aprofundar nossos conhecimentos um pouco mais no pipeline de 5 estágios do Risc-V que é talvez a parte mais importante da matéria de OC1.