DCC006: Organização de computadores I

Trabalho Prático #2

Integrantes

Arthur Souto Lima - 2018055113

Pablo Correa Costa - 2017014774

Renato Reis Brasil - 2013031127

Problema 1 - ORI - Bitwise or Immediate

ORI é a operação lógica OR bitwise com um registrador e um imediato. A instrução é do tipo I, isto é, [31:20] bits do imediato, [19:15] do registrador rs1,[14:12] do funct3, [11:7] do registrador de destino e [6:0] do opcode. O opcode é 0010011 e o funct3 110.

Para esta instrução alteramos o módulo decode criando um caso para o funct3 110 uma vez que a instrução do addi já existe na especificação e ela possui os mesmo opcode de ORI. A alteração criada é exatamente para implementar a operação lógica OR sobre um imediato.

Problema 2 - SLLI - Shift Left Logical Immediate

A instrução SLLI é do tipo I. O operando que será shiftado está no registrador rs1 e a quantidade que ele será shiftado é igual aos últimos 5 bits do imediato (lower bits). Sendo assim, precisamos identificar os 5 lower bits do immediato e deslocar o registrador com esse número em 0s. A identificação se dá por: ImmGen <= {inst[24:20], 1'b0}. O opcode da operação é 0010011. E o funct 3 = 001.

Problema 3 - Load Upper Immediate

Para essa intrução foi criado um fio que liga o ImmGen diretamente ao banco de registradores. Além disso foi criado um bit de controle para definir se o valor a ser escrito no banco de registradores vai vir da entrada writedata ou da recém-criada ImmGen.

O deslocamento dos bits é feito no módulo decode, da senguinte maneira: ImmGen <= {inst[31:12], 12'b0}. Portanto os últimos 20 bits da instrução são selecionados, e em seguida deslocados.

Como essa instrução já existe no RISC-V, usamos o formato e opcode da especificação. A intrução tem tipo U, ou seja, [31:12] são os bits do imediato, [11:7] são os bits do registrador de destino e [6:0] o opcode. O opcode é 0110111.

Problema 4 - Load With Increment

As operações básicas de soma e leitura na memória já estão implementadas. Portanto, para implementar essa intrução bastou mudar a unidade de controle. Como no caso da instrução Load Word, os bits memtoreg, regwrite e memread são definidos como 1. A diferença é que o alusrc é 0, já que a entrada da ALU vai ser os dois registradores de entrada. O aluop é zero, ou seja, soma.

A intrução tem um formato parecido com o tipo S, com a diferença de que nos bits 11:7, em vez de conter o valor imediato, colocamos o registrador de destino. O opcode escolhido foi 0000111.

Problema 5 - SWAP

A instrução de swap envolve atribuição do valor de um registrador rs1 a um registrador rs2 e vice versa. Existe uma instrução que faz basicamente isso no RISC-V: a instrução de swap da AMO(Atomic Memory Operands), que carrega o dado do endereço de rs1, salva o valor no registrador rd e passa para rs2 e depois salva de volta o valor presente em rs2 em rs1. Essa instrução é do tipo R, logo será definida pelo funct3 na implementação = 001 e funct5 = 00001 (AMOSWAP.W). Para isso, tivemos que acrescentar o caso em que a funct5 define a operação no módulo de control unit.

Problema 6 - Store Sum

A implementação dessa instrução exigiu três novos bits de controle, que são 0 para as demais instruções e 1 para store sum. Eles são descritos a seguir:

1. Definir a entrada 1 da ALU para que seja o imediato e não o conteúdo do registrador 1
2. Definir a entrada do endereço de memória a ser acessado para que seja o conteúdo do registrador 1 e não a saída da ALU
3. Definir o dado a ser escrito para ser a saída da ALU e não o conteúdo do registrador 2

A maior dificuldade encontrada nessa implementação foi encontrar uma forma de mudar quem a memória usaria como endereço.

Para codificação da instrução em si, considerando que ela não existe no RISC-V padrão, pegou-se a formatação da instrução de store, que possui dois campos de registrador de origem e um campo para o imediato. O funct7 (opcode) escolhido

foi 1110011, que é o do ADD com um 1 no primeiro dígito. Esse funct7 não define quaisquer funções do TP ou do datapath então isso facilita a implementação, já que evita conflitos. O funct3 é 000, apesar de ele não ser relevante para a decodificação da instrução.

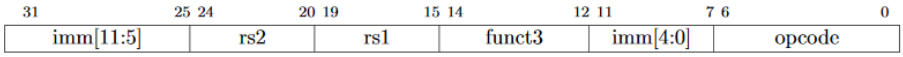

Figure 1: Formatação Instrução

Problema 7 - BLT

A ideia da implementação foi uma adaptação do BEQ que já estava implementado. Criou-se uma flag partindo da ALU para o módulo de fetch que indicava se o conteúdo do primeiro registrador era menor que a do segundo e uma outra flag de controle que discriminava para o fetch que era uma instrução de branch quando menor que e não quando igual. Além disso, adicionamos uma condição "ou" na seção em que o offset seria somado ao PC. Agora, faremos isso quando ou estivéssemos com a flag de BEQ e de zero ativas ou estivéssemos com a flag de BLT e de menor ativas.

Problema 8 - BGE

Com a instrução BLT implementada, a BGE já estava praticamente pronta: bastou criar uma flag para o módulo de fetch avisando que era um branch de "maior que" e adicionar uma condição para incrementar o PC utilizando a negação do flag de resultado negativo da instrução BLT. Para que o igual também fizesse o branch, no módulo de controle, ao identificarmos essa instrução BGE, ativamos também a flag de branch on equal, além da recém criada flag de branch quando maior.

Problema 9 - Jump

Para essa instrução foi preciso criar um bit de controle que fizesse o branch incondicionalmente. Esse bit foi chamado branchnc (branch não-condicional), e liga o módulo decode ao fetch. Dentro do módulo fetch, na parte onde é decidida a próxima instrução, foi adicionada a situação em que o branchnc está ativado.

No restante a instrução é parecida com os outros branchs: os outros bits ou são zero ou não importam, e por isso foram deixados todos iguais a zero.

O formatado da instrução é o tipo UJ, o mesmo da JAL. Os 5 bits correspondentes ao registrador de destino são iguais a zero, apesar de isso não ter importância, já que o bit de controle regwrite está desligado.

simm[20|10:1|11|19:12]	rd	1101111

Figure 2: Formatação Jump

O opcode escolhido foi 1101111, o mesmo do JAL. Isso não é um problema, já que a instrução JAL não está presente nesse processador.