Simulador RISC-V

Dérick Daniel Silva de Andrade 23/1003522

Apresentação do trabalho

A implementação do simulador do RISC-V em uma linguagem de alto nível serviu para entendermos como funcionam as principais instruções deste conjunto com mais facilidade, associando os registradores a um array de words e a memória um array de bytes. Além disso, o trabalho estimulou a manipulação bitwise e o entendimento sobre o significado de cada segmento de bits.

Instruções e funções implementadas

As instruções que foram implementadas, além das instruções do RISC-V foram as funções fetch, decode, execute, run, step, sign_extend, load_mem e reset.

Fetch: verifica se o pc é maior que o tamanho da memória. Se for maior, imprime a mensagem dos RARS "-- program is finished running (dropped off the bottom) –" e encerra o programa. Se não, busca na memória a instrução com o endereço do PC e incrementa o PC em 4.

Sign_extend: garante que os imediatos gerados por decode tenham o bit correto de sinal.

Decode: separa cada parte da instrução fornecida por fetch em opcode, funct3, funct7, registrador de destino, rs1, rs2, shamt e o imediato com o número de bits a depender da instrução sendo realizada.

Execute: é uma cadeia de ifs e elses que analisa o opcode, funct3 e funct7 da instrução decodificada. Alguns analisam só o opcode, já que não foi necessário implementar todas as instruções e o opcode era exclusivo daquela instrução nesse caso. No final dela, eu zero o registrador x0.

Run: Chama as funções fetch, decode e execute enquanto não chegar ao fim da memória ou enquanto não acontecer a ecall para finalizar o programa.

Step: chama as funções fetch, decode e execute e imprime o PC e o ri cada vez que enter é pressionado. Usei isso pra debugar e encontrar erros no programa.

Reset: usei para limpar o terminal e resetar os valores dos registradores, da memória e redefinir o pc, sp e gp para seus valores padrão pra executar outro programa.

Load_mem: recebe os valores de dados e as instruções dos arquivos .bin e os armazena no array da memória, as instruções a partir do endereço 0x0000 e os dados a partir do endereço 0x2000.

Main: coloquei um menu pra escolher entre rodar o programa testador.asm ou o primos .asm de uma vez ou passo a passo.

As instruções foram implementadas exatamente como estão descritas no arquivo de referência rápida no aprender, com exceção das instruções que modificam o PC (**branches, auipc, jumps**), pois como o PC é atualizado em fetch, precisei subtrair 4 do PC para as instruções funcionarem corretamente.

Na **ecall**, eu verifico qual o valor armazenado em R[17] que é o a7, se for 1 tem um std::cout pro valor guardado em R[10]. Se a7 tiver 4, então crio um contador, extraio os bytes e vou imprimindo os valores, como caracteres ascii, da memória no endereço que está em R[10] mais o contador, a condição de parada é o caractere a ser impresso ser o caractere nulo. Se a7 for 10, então imprime a mensagem "-- program is finished running (0) –".

Nas funções que usam o número sem sinal, tentei fazer a conversão com uma função própria, mas deu erro então converti explicitamente colocando uint32_t do lado do registrador mesmo.

Resultados

Utilizei os arquivos testador.asm e primos.asm para fazer os testes, em algumas instruções que deram erro, fui analisando o ri do meu programa com o ri do RARS pra entender o que estava acontecendo e fazendo os ajustes até os todos os testes darem OK. No arquivo dos primos, eu tirei a ecall de encerrar o programa pra ver que o dropped off the bottom estava funcionando.