Projeto 2 - Um simulador simples do procesador RISC-V

Nesse projeto, cada aluno deve implementar um simulador básico do processador RISC-V. A intenção é praticar conceitos de simulação de processadores, que são mais complexos que os circuitos lógicos do projeto 1, além de ter uma visão da eficiência desses algoritmos.

Objetivos

Para completar com sucesso esse projeto, é necessário:

- Ler uma representação de código de um programa
- Simular o processador
- Gerar um log de execução

Requisitos

O simulador pode ser implementado na sua linguagem de preferência. É importante que ela seja executável no computador do professor. Portanto, você deve fornecer documentação suficiente para a correta execução do código e um dockerfile para resolver toda e qualquer dependência de ambiente. Veja mais detalhes abaixo.

Especificação

Você deve implementar um simulador do processador RISC-V RV32IM, que significa a versão de 32 bits com as instruções básicas e também as instruções de multiplicação e divisão.

Você pode utilizar como referência o simulador Spike para verificar seu código.

As seguintes funcionalidades básicas são necessárias:

- Linha de log a cada instrução executada, veja a especificação do log desejado abaixo.
- Contagem de ciclos de execução: neste primeiro momento, considere que cada instrução gaste um ciclo para executar. É importante que você tenha uma

infraestrutura que permita alterar estes valores pois essa é uma funcionalidade necessária em simuladores de processadores

- Você deve ser capaz de executar um programa saído de um compilador/linker. É opcional emular um sistema operacional. Você pode utilizar o objdump ou objcopy para converter os arquivos executáveis num formato mais direto seu. Neste caso, implemente esta ação através de um script já com as linhas de comando corretas
- Seu simulador deve ser capaz de executar todos os programas de inteiros do ACStone (isto significa que não precisa executar os programas de ponto flutuante).
- Encapsule todos os acessos à memória através de alguma função. Isso permite que você possa inserir temporização na memória alterando apenas essa função

Entrega

Você deverá fornecer um repositório git do seu código, com toda a sua implementação e um arquivo README indicando como executar o programa. Você deve implementar um dockerfile que contenha todas as especificações para gerar seu ambiente de execução e compilar seu código (se utilizar uma linguagem compilada).

Sua árvore de entrega deve ter uma pasta chamada test que contenha todos os códigos do ACStone necessários. Você deve partir do código fonte do ACStone e ter um script que compile e gere os arquivos necessários para entrada do seu simulador.

Você deve gerar um arquivo de log de saída para cada programa e trocar o tipo dele para .log. Por exemplo, o programa 000.main.c deve ter o log chamado 000.main.log. Você deve gerar um arquivo de log para cada programa de teste do ACStone.

Forneça também um arquivo chamado relatorio.pdf que contenha um relatório compacto sobre o seu projeto. O relatório deve conter:

- Descrição geral do seu projeto
- Descrição do seu ambiente de desenvolvimento
- Descrição do seu algoritmo de simulação
- Descrição de como você testou seu projeto
- Considerações gerais sobre seu aprendizado nesse projeto

Colocarei uma atividade no Google Classroom para entrega do link do repositório e do relatório. A entrega deve ser feita até o dia 26/04/2023.

Formato do Log de instruções

O log de instruções deverá ser conforme o padronizado abaixo. Cada instrução executada deve gerar uma linha de log. Cada campo da linha tem um tamanho fixo, exceto o último e deve ter um espaço entre cada campo. A linha possui 6 campos conforme abaixo

1. Endereço do PC da instrução atual, em hexadecimal, com 8 dígitos. Ex: PC=00000100
2. Instrução em hexadecimal. Apenas o valor em hexadecimal dos 32 bits da instrução entre colchetes. Ex.: 012345678
3. Valor do registrador de destino (rd) após a instrução. Ex.: considerando o registrador 15 como destino, x15=000AAA00. Note que nem todas as instruções escrevem num registrador, mas veja que os registradores estão sempre nos mesmos campos em bits nas instruções. Então você deve imprimir sempre o registrador indicado pelos bits 7-11 da instrução após a instrução.
4. Valor do registrador de origem 1 (rs1) antes da instrução. Ex.: considerando o registrador 3 como origem, x03=99988877. Observe o zero entre o x e o 3 para padronizar a formatação. Aqui você deve imprimir o registrador indicado pelos bits 15-19 da instrução, independente da instrução fazer uso desses bits para indexar um registrador.
5. Valor do registrador de origem 2 (rs2) antes da instrução. De forma similar ao item anterior, utilizando os bits 20-24.
6. Dissassembly da instrução. Imprima o mnemônico (sem pseudo instrução) com tamanho de 8 espaços, seguido dos registradores e campos de imediato conforme a sintaxe da instrução. Aqui você deve imprimir tudo em decimal, tomando cuidado para utilizar representação sinalizada onde for necessário. Utilize a nomenclatura conforme a ABI (dica: declare um vetor de nomes de registrador com o nome de cada um corretamente e apenas indexe com o índice do registrador). Aqui você não precisa adicionar zero à esquerda do registrador para escrever com dois dígitos.

Como sugestão de abordagem sobre a implementação, monte a linha de log numa estrutura de dados e passe para uma função fazer a impressão. Assim a saída ficará mais uniforme no seu código. Você pode fazer algo similar com o disassembly indicando o formato em que a instrução deve ser escrita na tela.

Avaliação

Cada arquivo do ACStone será considerado na avaliação. O seu código será avaliado quanto a corretude dos logs de saída do seu simulador.

Sua nota nessa avaliação será dada pela seguinte fórmula:

```
Nota = 8 * (pontos / total de pontos) + 0,2 * nota do relatório
```

Atenção

Caso neessário, o aluno será chamado para apresentar o código e explicar o funcionamento do mesmo, podendo impactar na nota acima.