Simulador de assembly na arquitetura RISCV
Organização e Arquitetura de Computadores

26/03/2021

Pedro Nogueira
14/0065032
140065032@aluno.unb.br

1. Objetivos

Implementar um simulador de assembly na linguagem C++ na arquitetura RISCV a partir de arquivos binários gerados pelo programa RARS versão 1.5.

2. Introdução

Assembly é uma linguagem de programação que fica a um passo da implementação binária utilizada por um processador para que este possa executar suas funcionalidades implementadas. Executar um arquivo de assembly simplesmente significa quebrar suas instruções em códigos binários devidamente colocados na ordem em que devem ser executados.

Então descendo para este nível de processador, deve-se atentar também ao fato de que cada modelo de processador funciona de uma forma, e inclusive recebe instruções e responde de sua forma designada também. Como exemplo, existem arquiteturas como a MIPS e a RISCV que podem até executar instruções parecidas, como a `add` ou a `jal`, mas essas arquiteturas mexem com seus comandos de 32 bits de formas diferentes, utilizando certos bits para estabelecer certos parâmetros, exatamente como programado em suas arquiteturas.

Com isso em mente, a arquitetura a ser utilizada, não importando quais comandos se quer nela, deve ser muito bem especificada desde o início de sua implementação. Estabelecido então que a arquitetura desejada era a RISCV, o desafio agora era criar funções de um processador RISCV no formato já especificado.

3. Materiais e métodos

3.1. Materiais

As seguintes ferramentas foram utilizadas na execução do trabalho:

- **C++/g++:** Linguagem e compilador do simulador criado. Versão gcc version 9.3.0 (Ubuntu 9.3.0-17ubuntu1 20.04).
- **Windows 10/WSL2:** Sistema operacional, junto da ferramenta que contém o compilador acima. WSL2 contém um Ubuntu versão 20.04.2 LTS.
- **VSCode:** Editor de texto utilizado.
- **RARS:** Compilador de assembly que roda em Java(JRE) usado para gerar os binários. Versão 1.5.

3.2. Métodos

Primeiramente era necessário obter casos de exemplo para o uso do simulador, e para isso era feito um dump de memória de um código assembly exemplo qualquer em binário no RARS. Esse arquivo binário já codificado é o que será usado no simulador do trabalho.

Parte do trabalho tinha sido disponível pelo professor, como a leitura das linhas em binário e suas decodificações, então bastava criar as funções pedidas nas especificações do trabalho, levando em consideração as variáveis e métodos gerados pelo código do professor.

Essas são as instruções requisitadas:

- add
- beq
- bltu
- lbu
- ori
- srli
- addi
- bne
- jal
- lw
- sb
- sub
- and
- bge
- jalr
- lui
- slli
- sw
- andi
- bgeu
- lb
- nop
- slt
- xor
- auipc
- blt
- or
- sltu
- srai
- ecall

`ecall` é uma função que faz diversas coisas com o sistema, como printar bytes ou receber input do io. Para esse trabalho, era simplesmente preciso implementar ECALL com as funções de printar um inteiro, printar uma string e terminar o programa.

4. Execução e Resultados

O trabalho foi executado para cada teste disponibilizado pelo professor. O primeiro teste e seu resultado estão a seguir:

teste0.asm:

```
# testa acesso a memoria
.data
w0:  .word 0x01F203F4
w1:  .word 0
.text
li a7, 1
la s0, w0
lb a0, 0(s0)
ecall # imprime -12
la s1, w1
sb a0, 0(s1)
lw a0, 0(s1)
ecall # imprime 244
lbu a0, 0(s0)
ecall # imprime 244
lb a0, 1(s0)
ecall # imprime 3
lw a0, 0(s0)
sw a0, 0(s1)
lb a0, 3(s1)
ecall # imprime 1
```

Saída:

-1224424431

– program is finished running (dropped off bottom) –

Esse primeiro exemplo testa as funcionalidades `addi`, `auipc`, `lb`, `ecall`, `sb` e `lw`. Por mais que haja exemplos de pseudo-instruções no código, como `li`, o que é usado é o binário gerado pelo RARS, que já traduz elas para comandos nativos da arquitetura.

A saída para esse caso exemplo veio como esperado.

5. Discussões e Conclusões

5.1. Discussões

O trabalho foi finalizado com sucesso, passando em cada um dos casos de exemplos. Foi interessante notar a facilidade com a qual a linguagem C++ teve em manipular os bits de seus argumentos inteiros.

5.2. Conclusões

O trabalho consistia em implementar diversas funções de um processador de modelo RISCV e executar um arquivo contendo instruções binárias, respeitando suas estruturas. Com sucesso, o código consegue agora rodar tais instruções e apresentar seus resultados.