Trabalho final de Sistemas de Programação

Lucas Yuji Harada

May 4, 2020

Contents

1 Objetivo

Para a primeira etapa do projeto, o objetivo é desenvolver três peças essenciais a qualquer computador:

- Loader
- CPU (Von Neumann Machine)
- Assembler

2 Desenvolvimento

2.1 CPU

Baseado em princípios de test-driven-development (TDD), eu decidi primeiro criar os testes de cada função a ser executada pela CPU (fetch, decode, add, load, etc). Como o enunciado não especificava, fiz algumas considerações sobre o tamanho de cada um dos components:

- memória: 12 bits (4096 endereços)
- contador de instruções (PC): 16 bits
- acumulador: 16 bits

O motivo para a escolha da memória com 12 bits é devido ao tamanho do operando. Como cada instrução tem comprimento de 16 bits, sendo os 4 primeiros usados para definir o opcode, foi natural escolher 12 para endereças a memória. A mesma lógica se aplica ao acumulador; como existem instruções que podem carregar valores de até 12 bits nele (LV/load value por exemplo), usei 16 bits. Tive que divergir um pouco da implementação usada nos slides da aula 9, mais especificamente na instrução MV/move to memory, pois ela originalmente apenas poderia guardar 8 bits. Para guardar os 16 bits, decidi então alterar ela para que o endereço indicado pelo operando guarde o byte mais significativo do acumulador, e o endereço seguinte guarde o menos significativo. Para manter compatibilidade com a função LM/load from memory, também a alterei para que ela puxe 16 bits.

2.2 Loader

Comparativamente aos outros componentes, o loader é bem simples por se tratar de um loader absoluto. Caso tivessémos que implementar um linker/relocador, certamente seria muito mais complicado. No momento, estou usando apenas uma função que lê valores binários e guarda dados do tipo Byte na memória virtual.

2.3 Assembler

Assemblers são programas que convertem mnemônicos assembly (ASM) para código de máquina. Podem ser de um ou dois passos. Assemblers de dois passos primeiro precisam gerar uma tabela de símbolos para depois conseguir traduzir à código de máquina, enquanto que assembler de um passo fazem o

processo de coletar referências ainda não resolvidas, símbolos, e a montagem tudo ao mesmo tempo.

3 Uso

Requisitos mínimos:

- git
- python 3.8+

3.1 Instalação

3.1.1 TODO Double check the instalation procedure, maybe add a windows step-by-step as well? or pyinstaller

1. Dentro de um terminal emulator, usar o comando:

   `git clone https://github.com/Adarah/2020-PCS3216-9345853.git`

   - Caso não tenha o git instalado, pode baixar o código fonte manualmente do site da disciplina: `https://sites.google.com/view/2020-pcs3216-9345853`

2. Entrar na pasta que foi baixada: `cd 2020-PCS3216-9345853`
3. Executar `pip install .`

3.2 Testes

Usando a biblioteca pytest e hypothesis, cada um das minhas funções recebem ao menos 200 inputs aleatórios, o que ajuda a garantir que o código funciona como esperado. Para iniciar uma sessão de testes, basta executar o comando `pytest` na raiz do diretório onde o código foi baixado.

3.3 Uso normal

3.3.1 TODO fix this part

Para uso convencional, basta executar o comando `python3 src/vm.py`