Máquina Virtual (MV)

Cleyson B. de Oliveira*
Pontifícia Universidade Católica do Rio Grande do Sul
PUCRS - Escola Politécnica - Engenharia de Software

15 de setembro de 2020

Resumo

Este artigo abordará a resolução do primeiro problema da disciplina de Sistemas Operacionais - SisOp - ministrado pelo professor Fernando Luis Dotti em 2020/2 para a turma 127. Descrevendo as tecnologias e ferramentas utilizadas para o desenvolvimento do projeto da "Máquina virtual"(MV) com suas soluções e manuais de instrução.

Este desenvolvimento terá como objetivo a compreensão do funcionamento interno da CPU (Central Process Unit), seus registradores e memória, de uma maneira segura e simples, fazendo a sua execução e visualização protegidas pelas ferramentas de segurança já existentes nos sistemas e linguagens de alto nível atuais.

Introdução

A proposta é apresentada como participação da nota de avaliação na cadeira *4647D-04*, onde os alunos, em grupos de até quatro participantes, serão responsáveis por desenvolver, em linguagem de alto nível, um pequeno programa capaz de simular o funcionamento de um processador, mostrando a interação com seus registradores e memória.

O processador em questão terá à disposição um registrador para instruções (IR) e outros oito registradores (R0 a R7), para a execução dos programas em *Assembly*. O mesmo também possuirá o *Program Counter* (PC), este que é responsável por controlar a posição da memória em que a máquina virtual estará interagindo.

A memória será representada através de um *array* contíguo de *n* posições definida nas propriedades da máquina virtual. Cada posição da memória tera capacidade para comportar 4 *bytes*, codificados em *[OPCODE, 1 REG 0..7, 1 REG 0..7, Parâmetro K ou A conforme operação]*.

*Cleyson.Oliveira@edu.pucrs.br

Tecnologias

A seleção de tecnologias foi realizada levando como critério principal a legibilidade e facilidade de execução do código, desconsiderando conceitos como o desempenho e a velocidade de execução ou eficiência em uso de memória. Sendo o principal objetivo a acessibilidade na execução de códigos *Assembly* na máquina virtual, sem que os parâmetros emulados de baixo nível precisem ser levados em consideração.

0.1 Linguagem

O projeto será feito com base na linguagem script Groovy[2], a qual é desenvolvida para plataformas *JAVA* (*JVM*), sendo fortemente utilizada no framework Grails, este que utiliza do framework *Java Spring Boot*. Sua escolha deu-se devido à facilidade de organização de suas funções, onde é permitido chamar os métodos da classe como uma variável *String*, assim permitindo a melhor organização dentro dos pacotes de desenvolvimento.

0.2 Gerenciador de pacotes

Os pacotes serão gerenciados pelo *Gradle*, o qual é um sistema de automação de compilação *open source* que se baseia nos conceitos de *Apache Ant* e *Apache Maven*, porém, utiliza tecnologias mais recentes, como o Groovy, para substituir a declaração de configuração *XML*, conforme utilizado no *Maven*.

Esta ferramenta foi escolhida como gerenciador pois há um maior suporte para a linguagem escolhida; outro ponto é que a sua declaração de configuração é muito mais limpa e legível comparado com o *Maven*, possibilitando, desta forma, a fácil criação de *tasks* para a automação de outras partes do processo, como por exemplo, a criação do artefato de entrega.

0.3 Dependências

1. Codehaus Groovy (v3.0.5) [JVM Languages]: Referência ao compilador do projeto;
2. Junit jupiter api (v5.6.2) [Testing Frameworks]: Usado nas escritas dos códigos e extensões;
3. Junit jupiter engine (v5.6.2) [Testing Frameworks]: [*Runtime*] implementação da *engine*.

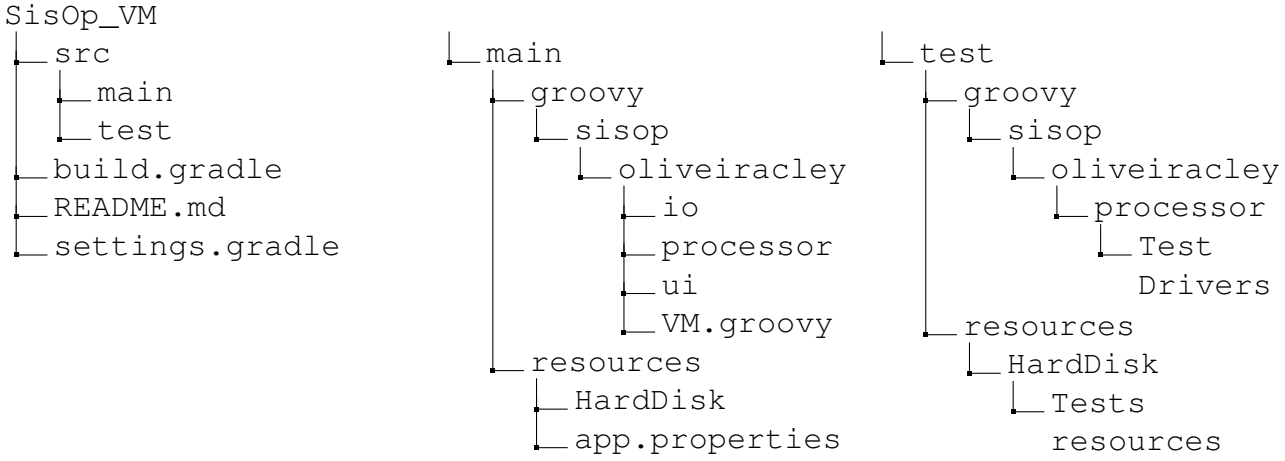
0.4 Manual de utilização

Para executar qualquer comando no projeto é necessário abrir um terminal (*shell*) dentro da pasta raiz. Os comandos a seguir são todos os necessários para utilizar o programa.

1. *gradle run* - Executa o projeto
2. *gradle test* - Executa os testes no projeto
3. *gradle clean* - Exclui os arquivos de compilação
4. *gradle artifact* - Gera o artefato de entrega
5. *gradle run - -args="FileName"* - Executa o projeto passando o arquivo por parâmetro

0.5 Estruturação do projeto

O *Gradle* se responsabilizará por toda a estruturação, compilação e organização dentro da pasta raiz do projeto (*SisOp_VM*). Conforme a seguinte árvore:

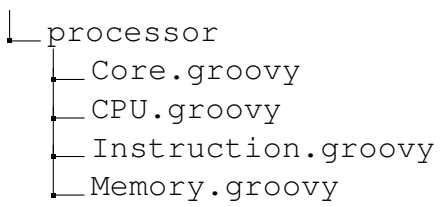


Desenvolvimento

No pacote raiz do desenvolvimento (*sisop.oliveiracley*), temos apenas a classe VM.groovy, qual contem o método *main*, nos permitindo indicar o início do programa. Assim como um projeto em gradle, sendo ele *Spring Boot* ou *Grails*, a classe *main* é exclusiva e usada unicamente para chamar a classe que inicializará todo o programa.

Diferente dos projetos padrões, esta classe conta com uma variável *String* pública e final, qual indica o caminho para o *application.properties*, que é o arquivo de configuração da máquina virtual.

Dentro do pacote *sisop.oliveiracley.processor*, é onde toda a parte importante do emulador acontece. A proposta foi realizar um desenvolvimento que demonstrasse separadamente cada função da máquina virtual, diferenciando então o *OpCode* da memória e da *CPU*.



Core, representa a descrição e execução de todos os *OpCodes*. Esta classe é como uma extensão da *CPU*, seu intuito é tornar os *OpCodes* legíveis e organizados.

O *CPU* é responsável pelo *FETCH* com a instrução da memória e também a parte da repetição até o final do programa *Assembly* que estiver em execução.

A classe Instruction reserva-se à estrutura de como cada posição da memória será organizada e alocada, definindo assim o tamanho e informações que trafegam pelo processador.

Por último, a classe Memory, é responsável por armazenar todos os códigos *Assembly* a serem executados e também os dados gerados pelos programas. A mesma também tem a função de gerenciar a alocação dos programas em todas as suas posições.

Gerenciamento de memória

O sistema de gerenciamento de memória é feito através de paginas e frames. Durante o momento de alocação da memória, a máquina virtual passa de página em página pegando todos os frames livres e alocando em um *map* formado por posição virtual contígual (de 0 até o tamanho do programa que está sendo alocado) como chave e fazendo link como valor a posição relativa da memória real.

Portanto, durante a execução, a *CPU* terá visão de memória contígua começando do 0 e o gerente de memória se responsabilizará por endereçar ao endereço correspondente.

Referências

[1] Maurice Bach. ”**Design of the Unix operating System**”.
http://www.scielo.org.mx/pdf/cys/v19n2/v19n2a13.pdf

[2] open source. ”**Apache Groovy**”.
https://groovy-lang.org

[3] Alfred V. Aho; Jeffrey D. Ullman: “**Fundamentals of algorithmics**”.
Computer Science Press, Madison Avenue, New York, 1992.

Memorial
”χαριτωμενο πόνυ και μαγικός καπνός”