Trabalho prático 1 - Compiladores
Documentação - Montador

Luiz Felipe Gontijo, Marcos Vinicius, Matheus Pimenta

1. Introdução:

Este trabalho tem como objetivo fixar os conceitos da disciplina de Compiladores, especialmente sobre montagem de um programa. A atividade proposta foi construir um montador de dois passos para uma máquina virtual previamente disponibilizada, onde dadas instruções de um programa em Assembly, deve traduzir e montar o programa em instruções da máquina.

2. Implementação

A implementação do montador segue algumas diretrizes em que foram definidas estruturas e classes que seguem o comportamento do front-end de um compilador. Assim, criamos um Parser e um Lexer para melhor abstrair e modularizar o montador.

As classes implementadas estão descritas abaixo:

2.1 Classe Token

Nessa classe estão definidos todos os tokens da máquina virtual, operações de construção e a sobrescrita do operador '=', para podermos atribuir valor e tipo corretamente. Utilizamos essa classe no *Lexer*.

2.2 Classe Lexer

O *Lexer* é responsável por ler o texto de entrada e quebrar em tokens, que serão consumidos pelo *Parser*. Dessa forma, mantém um controle da linha e caracter do texto em que ele está, bem como posição inicial e o arquivo em questão. Contém funções relacionadas a leitura do texto, como funções para ignorar comentários, espaços e breaklines.

2.3 Classe Parser

A classe *Parser* contém a estrutura *ASTNode*, que vai criar uma *Abstract Syntax Tree* (AST) para o programa lido. Como o *Assembly* é uma linguagem simples, ela acaba por ser como uma lista encadeada, e os tipos possíveis de nós são bem parecidos com os tokens. O *Parser* vai requisitar um token, identificá-lo e adicioná-lo na árvore, e dependendo de qual foi lido, requisitar (ou não) um ou mais tokens complementares. Por exemplo, caso identifique o token

AND, o *Parser* sabe que precisa de dois registradores, e portanto requisita mais dois tokens e verifica se são registradores.

2.4 Classe Montador

Por fim, a classe *Montador* é quem mantém as tabelas dos símbolos identificados, das operações possíveis e das pseudo-instruções, além de ser ela quem controla as duas fases do montador. Ambas as fases são feitas lendo a *Abstract Syntax Tree* do programa. Na primeira, para cada instrução se verifica se ela pertence ou não ao conjunto da máquina virtual, guardando a posição na memória dela caso negativo. Já na segunda fase, se retorna a instrução na linguagem da máquina virtual para cada instrução lida na AST.

3. Testes

Foram executados vários casos de teste. Dentre eles, casos possuindo múltiplas linhas com comentários, código com mais de um breakline entre as linhas e quantidade variável de espaços entre as palavras.

Foi implementado na linguagem Assembly da máquina virtual alguns testes, à exemplo do teste da mediana, que se encontra na pasta "tst". Além disso, para melhor visualização do fluxo do programa, foi criada uma função para imprimir a AST gerada. Segue abaixo a árvore do problema-exemplo passado na especificação do trabalho.

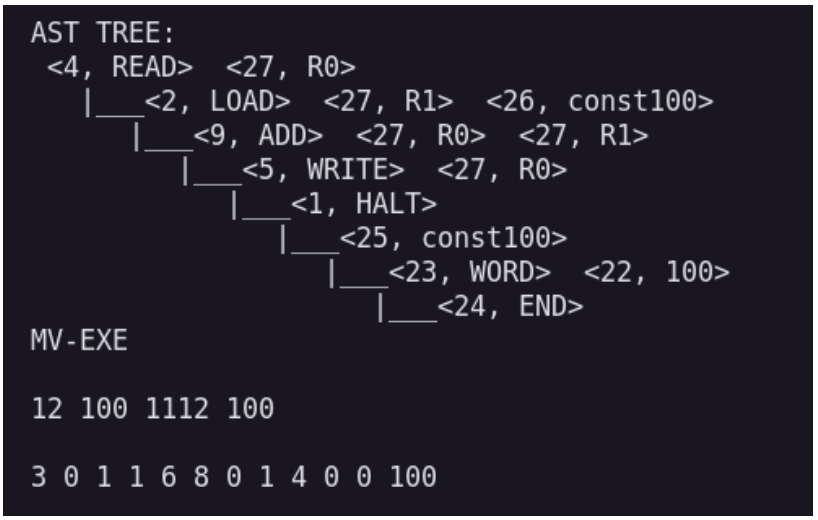

Figura 1. AST do problema-exemplo

4. Conclusão

Neste trabalho foi possível implementar um montador, assim como compreender o funcionamento desse componente do compilador, de forma a aplicar os conceitos aprendidos nas aulas. Também, praticamos conceitos que envolvem o desenvolvimento de estruturas de dados para ler e dar significado aos símbolos de uma linguagem de programação.