Relatório - EP Fase 1
Laboratório de Programação 2

Victor Sanches Portella Lucas Dário Ruan Costa

18 de setembro de 2013

1 Integrantes

- Victor Sanches Portella - Nº USP: 7991152
- Lucas Dário - Nº USP: 7990940
- Ruan Costa - Nº USP: 7990929

2 Introdução

Neste relatório daremos uma breve explicação sobre as estruturas e classes criadas nesse EP. Daremos explicações que facilitem a entendimento do leitor quanto ao entendimento do código.

Portanto, primeiramente, faremos uma explanação sobre o funcionamento geral do programa, e após isso faremos uma breve explicação sobre alguns elementos existentes.

3 O programa

A ideia do programa, assim como explicado no enunciado, é receber um programa escrito em uma linguagem similar à linguagem *Assembly*, transformar esse programa em um vetor de instruções e simular a execução desse programa.

Para resolver esse problema, usamos Programação Orientada a Objeto em *Perl*. Assim, para tal, criamos 3 classes: **Comando**, **Programa** e **Máquina**.

A classe **Comando** representa o comando em si. A classe **Programa** representa o Montador, e é responsável por transformar o arquivo de entrada

em um vetor de objetos **Comando**. A classe **Máquina** representa a máquina virtual em si, representando a simulação de um determinado programa. A seguir, uma explicação um pouco mais específica sobre cada classe.

3.1 Classe Comando

Cada objeto dessa classe representa um comando, e é representado por dois atributos:

- **Opcode(Obrigatório):** String que define a função em si. Esse código é o mesmo do que o Mnemonico usado pela definição do comando no código fonte.
- **Valor(Opcional):** Valor do argumento, caso exista. Se não existir, o valor desse atributo é **undef**. O valor, nos comandos implementados, pode ser interpretado como uma *String* no caso do Jump, ou como um número nos outros casos.

Além dissso, usamos essa classe para verificar alguns erros de sintaxe. Ao tentarmos criar um comando, temos uma tabela para verificar se o comando existe e, no caso de existir, se precisa ser acompanhado de um argumento.(**Importante**)Caso o comando não exista, ou precise de um argumento que não foi passado, o programa irá retornar **undef**.

3.2 Classe Programa

Essa classe representa o montador, e possui 3 atributos:

- **Vetor:** Possivelmente o argumento mais importante dessa classe. Esse é o vetor que representa o vetor de instruções, no nosso caso representado por objetos do tipo **Comando**, de todas as linhas já interpretadas pelo próprio objeto.
- **Posi:** Indica a posição da próxima posição vazia no vetor. Esse atributo não seria de fato necessário, existiriam modos de fazer o programa sem ele, mas ele facilita o entendimento.
- **Label:** É uma tabela de hash, onde as chaves são o nome do label, e os valores associados são a posição no vetor para o qual esse label "aponta".[1]

[1] Isso é usado para implementar os comandos de *Jump*.

Um objeto dessa classe possui o método **interpretaLinha()**, que recebe uma string, que seria uma linha de nosso programa, trata a linha, criando uam referencia no hash **Label**, caso tenha um label, e criando um **Comando** e o colocando no vetor, caso o comando exista de fato.

3.3 Classe Máquina

A classe Máquina representa a máquina virtual que irá executar o nosso código. Nessa implementação, fizemos com que cada objeto do tipo **Máquina** esteja atrelado a um objeto do tipo **Programa**. Ou seja, para cada programa que você queira executar, você cria um objeto. Assim, todo objeto dessa classe possui três atributos:

- **Dados:** Vetor que representa a pilha de dados do programa em execução.
- **Mem:** Vetor que representa a memória do programa.
- **Programa:** Objeto do tipo **Programa**, usado como fonte do vetor de instruções.

Não existe implementação da pilha de chamada de função pois não foi usado e/ou implementado chamada de funções na linguagem interpretada.

O principal método desse objeto é a função **executa()**, que irá simular a execução do programa. Caso aconteça algum erro, a execução será interrompida no momento da exceção e uma mensagem de erro será exibida.

4 Modo de Execução

Modo de exeução do programa:

```
meu_pc$>./main.pl arquivo_de_entrada
```

Sendo que no **aquivo_de_entrada** se encontra o código fonte a ser executado. Caso você queira enviar os comandos através da *STDIN*, só omitir o nome do arquivo.