IMD 0041 – Introdução a Organização e Arquitetura de Computadores

Relatório do projeto final da disciplina – Reordenamento de Instruções

William Lucena dos Santos

Instituto Metrópole Digital

Natal – Brasil

1 Introdução

A motivação deste projeto foi o reordenamento de instruções em assembly mostrado em sala de aula para otimização de ciclos, foi criada uma aplicação que ao receber um grafo de instruções realiza o reordenamento das instruções e a renomeação dos registradores (técnicas vistas em sala de aula) para reduzir a quantidade de ciclos necessários para executar em um pipeline de cinco estágios e considerando que a leitura e escrita podem ocorrer no mesmo ciclo.

2 Solução implementada

O projeto foi realizado em java com o uso de classes e funciona seguindo a seguinte lógica: É passado como entrada um arquivo .txt referente ao grafo com instruções, se o arquivo .txt for válido, cria uma instrução (objeto da classe instrução) para cada linha do arquivo e armazena as instruções em um ArrayList, criando assim uma estrutura ordenada que pode ser manipulada para o reordenamento.

Recebendo um arquivo .txt que tem na primeira linha um cabeçalho igual a

**#inst tipo\_inst dest op1 op2 #inst\_recebe\_resultado** ou **#inst tipo\_inst dest op1 op2 #inst\_recebe\_resultado Ciclo\_Inicial Falsa\_Dependencia**

E cada coluna é separada por um tab ( \t ), a leitura é feita assim:

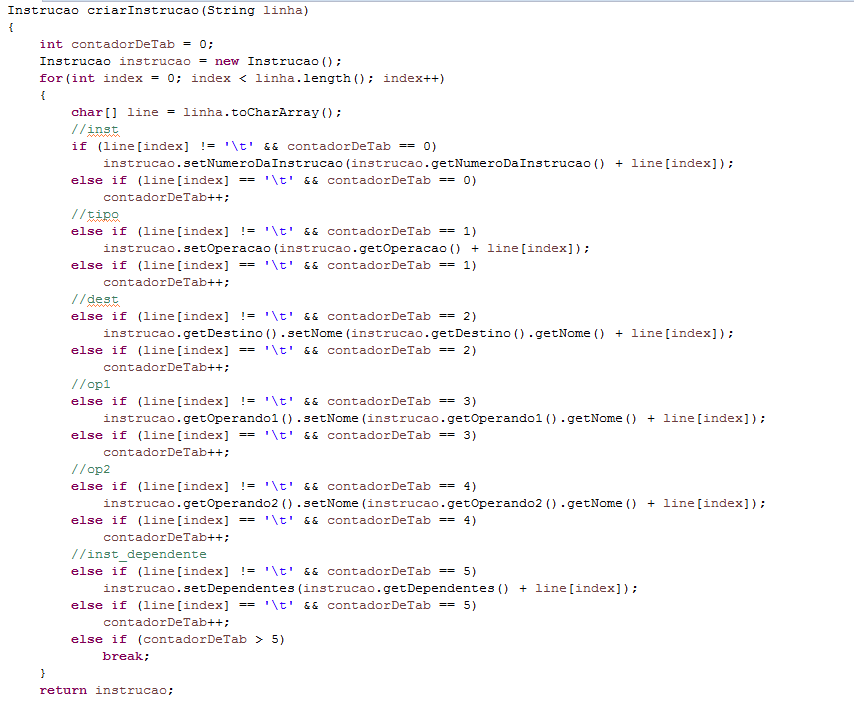


Figura 1: Código da função de criar a instrução

A aplicação tem 3 modos distintos de saída: Somente reordenar as instruções, Somente renomear os registradores e fazer ambas as coisas juntas, o ArrayList é manipulado de formas similares nos 3 modos, os quais são descritos abaixo;

Criado o ArrayList de instruções primeiro busca-se instruções que sejam completamente independentes e as move para o início do ArrayList, essa etapa é feita percorrendo o ArrayList e comparando uma instrução com as dependências de outras, ou seja, cada instrução é comparada com todas as instruções, se houver qualquer dependência a instrução é tida como dependente e não será movida.



Figura 2: Código da função de buscar instruções independentes

Esta etapa é a realizada em ambos os métodos de saída.

Tendo um ArrayList já pre-ordenado com as instruções independentes “isoladas” como primeiras instruções, é feito o reordenamento no ArrayList seguindo os seguintes passos:

Salva-se a lista atual. Percorre a lista do fim até o começo comparando o elemento atual com a lista de dependentes de seu anterior, se o elemento atual não depender do imediatamente anterior a troca é feita, os ciclos são gerados novamente e é começado um laço que enquanto o código se manter correto e sem conflitos essa é feita a troca com o elemento imediatamente anterior e os ciclos gerados novamente, quando houver algum problema com a troca, ela é desfeita e o loop se encerra. Após o loop interno se houver alguma troca incrementa-se o contador de quantas instruções foram reordenadas e o loop externo começa novamente para instrução seguinte. Ao final é comparado se o reordenamento é vantajoso, caso seja retorna o novo ArrayList reordenado, senão retorna a lista que foi salva antes de começar as alterações.



Figura 3: Código de reordenamento parte 1

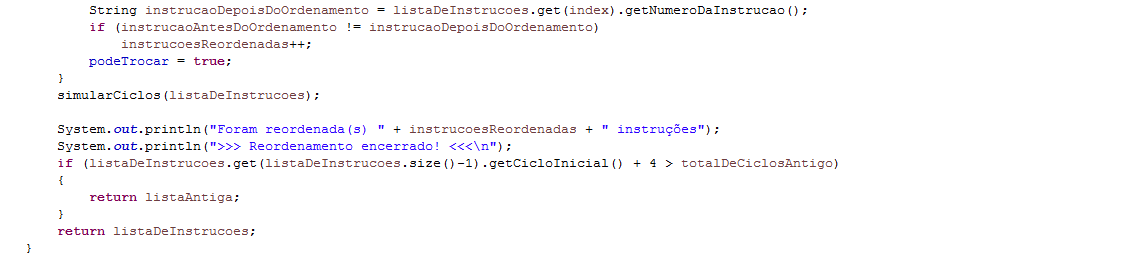


Figura 4: Código de reordenamento parte 2

Esta etapa só é realizada no modo de reordenamento e reordenamento com renomeação de registradores e é independente da função de renomear os registradores.

Dado o ArrayList com as instruções independentes “isoladas” é possível renomear os registradores para corrigir falsas dependências seguindo os seguintes passos:

Inicialmente busca-se as instruções que apresentem falsas dependências e as armazena em um vetor. Com este vetor de “quarentena” percorre-se o banco de registradores para pegar um disponível (nem está sendo lido nem escrito por nenhuma outra instrução) e se encontrar atualiza o registrador dessa instrução em quarentena, senão a função é encerrada por falta de recursos físicos para correção. Com o registrador atualizado (em caso de sucesso da etapa anterior) agora é preciso atualizar os que dependem dessa instrução para o registrador novo, para isso percorre-se a lista buscando essas instruções e compara qual dos registradores é o dependente da instrução atualizada e o renomeia. Ao final atualiza o banco de registradores para não haver conflito em futuras chamadas da função.

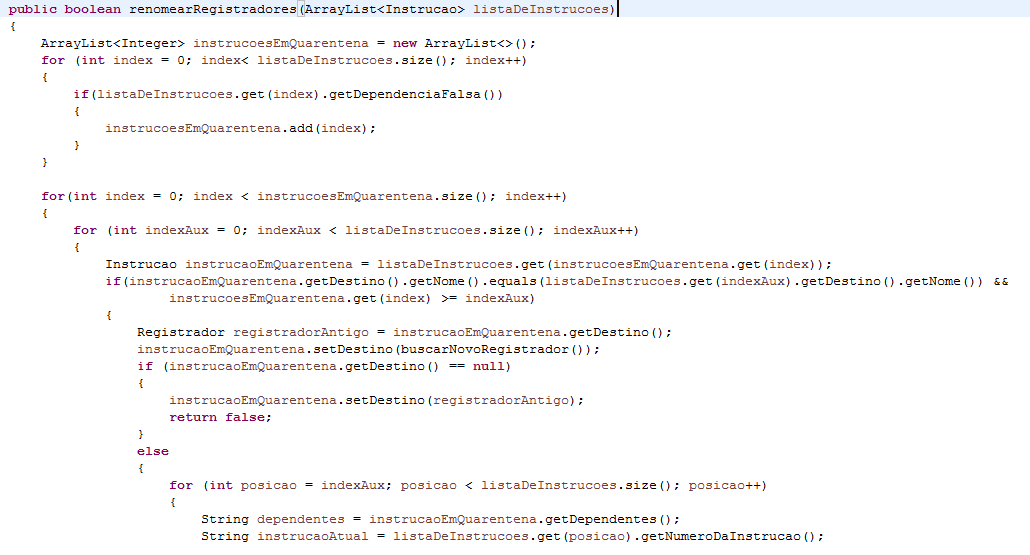


Figura 5: Código de renomeação de registrador parte 1



Figura 6: Código de renomeação de registrador parte 2

Esta etapa é realizada somente no modo de renomeação de registradores e reordenamento com renomeação, sendo totalmente desassociada a função de reordenamento.

Além das funções do projeto, foi implementado uma interface gráfica com JavaFX. Com a interface é possível acessar um menu de arquivo (abrir, salvar (sobrescreve o arquivo aberto) e salvar como (o usuário escolhe onde salvar) e um menu de funções (reordenar, renomear e reordenar com renomeação) e visualizar todas as ações em uma tabela baseada no ArrayList de instruções.

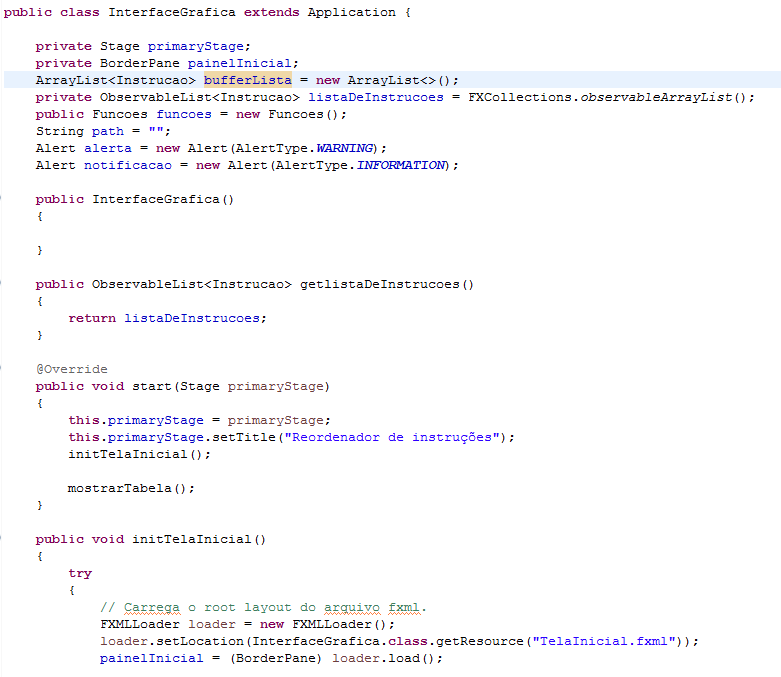


Figura 7: Código da implementação da parte gráfica parte 1



Figura 8: Código da implementação da parte gráfica parte 2

Figura 9: Código de alguns metodos do controlador

4 Conclusão

Ao final deste projeto é claro ver que os conceitos ensinados em sala foram aplicados na implementação e com a análise dos resultados que o reordenamento é vantajoso principalmente quando há muitas instruções independentes e que a renomeação de registradores é bastante eficiente na correção das falsas dependências.

5 Referências

Tutorial usado como base para implementar a interface gráfica: <http://code.makery.ch/library/javafx-8-tutorial/pt/>

Slides de aula do professor Gustavo Girão

Patterson e Hennessy, Organização e projeto de computadores – A interface Hardware/Software

William Stalling, Arquitetura e Organização de Computadores

6 Organização

O projeto está dividido em três pacotes: Estruturas, Funcoes, Interface.

Pacote estruturas contém as classes Instrução que define uma instrução com base nas colunas do grafo e Registrador que define um registrador que contém o nome do registrador e booleanos para saber se ele é lido/escrito.

Pacote funções possui uma classe Funcoes que contém todas as implementações das funções usadas sejam elas de uso interno (funções auxiliares das funções principais) como simularCiclos, analisarCorretude, criarBancoDeRegistradores ou externas (funções principais que podem ser chamadas de fora da classe) como buscarInstrucoesIndependentes, lerGrafo, reordenarInstrucoes.

Pacote Interface possui duas classes java: o main e um controlador que liga os arquivos .fxml ao main; e os arquivos .fxml que possuem o visual da interface gráfica do projeto.

7 Compilação

Baixar o projeto do link <https://github.com/will-lucena/Reordenador-de-instrucoes.git> abrir no eclipse e dar run.