Documentação TP2 Compiladores

Gabriel Lima Canguçu
Giovanna Louzi Bellonia
Thiago Martin Poppe

10 de agosto de 2021

1 Descrição do problema

Foi solicitado a criação de um editor de ligação para uma máquina virtual que foi projetada exclusivamente para a disciplina. O ligador receberá múltiplos arquivos resultantes do montador que é similar ao que foi feito no trabalho 1, porém com algumas modificações.

A saída do editor de ligação será através da **saída padrão** e seguirá um formato previamente especificado, onde a primeira linha sempre será `MV-EXE`, indicando que é um arquivo executável pelo emulador. A segunda linha será composta por 4 números separados por espaço, sendo eles (i) o tamanho do programa, (ii) o endereço de carregamento do programa, (iii) o valor inicial da pilha e (iv) o *entry point* do programa, ou seja, a inicialização do registrador de propósito específico PC (contador de programa). Por fim, a terceira linha será o código em linguagem de máquina, onde teremos a junção dos códigos passados para o ligador com relocação das labels inicialmente indefinidas.

1.1 Observações

- O desenvolvimento do editor de ligação foi feito na linguagem C++ utilizando Linux (versão Ubuntu 20.04) e WSL (por alguns integrantes do grupo).

2 Definições de projeto

2.1 O montador

No caso do trabalho prático 1, foi solicitado uma implementação de um montador de 2 passos. O primeiro passo consistia em inserirmos em uma tabela todos os símbolos desconhecidos que encontrarmos ao longo do programa, isto é as *labels* juntamente com a sua posição de memória dentro do programa. O segundo passo era para realizar o processo de montagem propriamente dito, onde líamos novamente o arquivo de entrada traduzindo as instruções e seus operandos para os seus respectivos códigos em linguagem de máquina.

A modificação feita nesse trabalho em relação ao trabalho anterior se deu basicamente no retorno que é feito pelo montador:

- Não foi retornado mais o cabeçalho MV-EXE por não haver necessidade disso a priori;
- Os 4 números que eram retornados com informações sobre o programa, foram reduzidos para apenas o número que retorna o seu tamanho. Além disso, passou-se a retornar um novo número que informa quantas labels foram definidas por esse programa;
- Retorna-se agora as labels definidas pelo programa atual e a posição em que as definições delas se encontram no programa;
- Há uma mudança no retorno do programa em si, na parte em que o programa referencia labels externas. Nesses pontos, retorna-se o nome da label para que o ligador possa tratá-las depois da forma correta.

2.2 Implementação do Ligador

Para modelarmos o nosso editor de ligação, optamos por criar uma classe chamada `Ligador` que irá lidar com todos os passos e dependências associadas ao processo de ligação. Ao instanciarmos essa classe, o construtor irá receber um `vector` com os nomes dos arquivos fornecidos via linha de comando, que constituem a saída do montador, realizando assim a concatenação dos códigos montados e o armazenamento das tabelas de *labels* definidas por cada um deles.

Para cada *label* contida no programa, devemos nos preocupar em instanciar as posições de memória de modo que não tenhamos conflitos, como por exemplo dois programas ocupando uma mesma região de memória. Tendo isso em mente, guardamos o tamanho total do programa em uma variável acumulativa `program_length`, cujo valor inicial é zero, fazendo com que toda posição de memória x passa a ser definida por $x +$ `program_length`. A variável acumulativa terá seu valor atualizado a cada leitura de um novo arquivo.

O passo seguinte consiste em "resolver" as *labels* que não foram traduzidas pelo montador, isto é, traduzir os tokens que referenciam posições de memória. `solve_labels()` é um método que itera pelo programa, conservando elementos que forem inteiros e tratando elementos que forem *labels*; podendo ser facilmente identificados e traduzidos simplesmente consultando as tabelas de símbolos armazenadas durante a leitura dos arquivos no passo anterior. Em seguida, uma vez que a *label* deve dizer ao contador de programa o número de "passos" que ele deve se deslocar na leitura, a tradução leva em consideração as posições relativas, ou seja, o passo tem tamanho `memory_position - pc_counter`, sendo `memory_position` a posição que define aquela região de memória devidamente "deslocada" para evitar colisões entre os programas.

Por fim, o programa traduzido, bem como seu cabeçalho, são exibidos na saída padrão. O cabeçalho incorpora o tamanho total do programa, que considera todos os arquivos, e também a nova posição inicial, que referencia a posição onde se encontra a *main*.

3 Testes desenvolvidos

Além de testarmos o código base fornecido pela especificação do trabalho prático, desenvolvemos códigos de teste adicionais, que utilizam múltiplos arquivos para testar a corretude do ligador.

Os testes desenvolvidos na verdade foram os mesmos do trabalho prático 01, fragmentando os seus códigos em 3 arquivos, quando necessário. Sendo cada uma dessas partes:

(i) o código principal do programa (*main*);

(ii) funções auxiliares;

(iii) constantes que serão usadas ao longo do programa.

Por exemplo, para o teste do cálculo do n-ésimo número de Fibonacci nós teremos um arquivo `main.amv`, `constants.amv` e `printer.amv`, que terá funções auxiliares para a exibição do resultado calculado.

Além disso, foi desenvolvido um programa simples que possui apenas um arquivo para testarmos a corretude da implementação quando fornecido apenas um arquivo de entrada para o ligador. Também alternamos a ordem dos programas passados quando eram mais de um para garantir que o funcionamento está ok independente dessa mudança.