Trabalho Prático 1 – Compiladores I

Vitor Rodarte Ricoy - 2019007112

Vitor Assunção Rabello de Oliveira - 2019007104

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

1. Introdução

O trabalho proposto tem como objetivo treinar e fixar os conceitos aprendidos em aula relativos ao processo de montagem de um programa. Esse trabalho consiste na implementação de um montador de dois passos para uma máquina virtual previamente projetada. Para isso nos foi dada a especificação do conjunto de instruções da máquina virtual utilizada no trabalho, além do emulador para executar os programas gerados pelo montador. Também foram pedidos dois programas de testes a serem desenvolvidos em Assembly com o objetivo de testar o montador desenvolvido.

2. Descrição Geral

O montador implementado consiste em dois passos. No primeiro passo, são identificados os labels e também são tratadas as pseudo-instruções. No segundo passo, as operações são convertidas para o correspondente delas no arquivo executável da máquina virtual, sendo este correspondente um inteiro para o código de operação e os operandos de acordo com a operação. No fim desses passos, o programa para máquina virtual é gerado, usando as informações dos dois passos executados.

Para o primeiro passo, cada linha do arquivo de entrada é lida. Caso o arquivo tenha sido lido até o fim e não tenha sido encontrado o END, o montador se comporta como se existisse um END no final do arquivo. Para cada linha, são desconsiderados os comentários, e a linha é quebrada em palavras, que são substrings separadas por espaços. Após isso, é identificado se a linha contém um label. Caso contenha, é salvo na tabela de símbolos o label e a linha correspondente no programa final, considerando o tamanho de cada instrução.

Depois de tratar o label, é tratada a pseudo-instrução WORD, que adiciona na linha a constante correspondente. Por fim, é tratado o END, que encerra o primeiro passo, e também é tratado o caso de uma linha vazia (ou composta apenas de espaços).

Já no segundo passo, inicialmente são contadas as constantes declaradas no início do programa. Isso é usado para determinar o ponto de início do programa final, já que este deve ser logo após as declarações iniciais constantes. Depois de contadas as constantes iniciais, que são inseridas no programa final como o valor inteiro que elas representam, as linhas seguintes resultantes do primeiro passo são analisadas, convertendo-as para o seu equivalente no código de máquina do emulador.

No caso dos operandos das operações, caso o operando não seja um registrador, é verificado se ele é um label válido. Caso seja, o valor desse label é lido da tabela de símbolos e é corrigido para o endereçamento relativo ao PC, para que a referência ao label funcione corretamente. Caso o operando não encaixe nos dois casos, ele é considerado uma posição da memória, devendo assim, ser um inteiro.

Por fim, após essa etapa de conversão, o montador retorna o programa final, que possui apenas a string “MV-EXE” na primeira linha. A segunda linha indica o tamanho do programa (que é o número de inteiros gerados no segundo passo), o endereço de carregamento (que é fixado em zero no nosso montador), o valor inicial da pilha (fixado no tamanho do programa somado de 1000, para que a pilha tenha 1000 posições) e o entry point do programa (que é definido com base no número de constantes que são declaradas no início do programa). Finalmente, na última linha são escritos os inteiros correspondentes às conversões realizadas no segundo passo.

3. Decisões de Projeto

A primeira decisão de projeto tomada foi relativa ao tratamento de um programa que não tenha a pseudo-instrução END. Decidimos que o nosso montador se comporta como se existisse um END na última linha de todo programa. Em outras palavras, ao ler um final de arquivo sem ter encontrado um END, o primeiro passo do montador é interrompido. Assim, o montador sempre interrompe o primeiro passo, seja por ter encontrado um END ou por ter encontrado o fim do arquivo.

Outra decisão tomada foi com relação ao endereço de carregamento do programa. Decidimos carregar o programa sempre na primeira posição da memória, por questão de simplicidade, já que não foram apresentados motivos contra essa decisão.

Também foi decidido que a pilha terá mil posições disponíveis, como informado pelo monitor no Teams. Ou seja, o registrador AP é inicializado com o tamanho do programa, que já considera a declaração de constantes, somado a mil.

Como instruído pelo monitor, foi considerado que a pseudo-instrução WORD é traduzida como a própria constante, logo a constante correspondente é inserida na própria linha do WORD. Ou seja, a operação WORD é convertida apenas para o inteiro da constante correspondente. Também foi considerado que o uso de WORD no meio do programa deve ser tratado pelo próprio programador, que terá que ter o cuidado de evitar a execução da linha da declaração.

Outra decisão tomada foi que um operando que não é um registrador nem um label válido deve ser uma posição de memória. Como foi pedido que não tratássemos erros no programa de entrada, o erro de um label inexistente não é tratado, gerando comportamentos inesperados do montador ao tentar converter esse label para uma posição de memória. Também vale notar, que é considerado que a posição de memória escrita pelo usuário já endereça relativo ao PC.

Por fim, a última decisão tomada é que foi considerado que todo label deve ter um ou mais espaços após os dois pontos, como informado pelo monitor no Teams. Dessa forma, por questões de simplicidade, apenas esse caso é considerado.

4. Testes

Para os testes do montador foram utilizados três programas. O exemplo fornecido com o template para o trabalho e os dois programas pedidos: do cálculo da mediana e o do cálculo do n-ésimo número de Fibonacci. Todos esses programas foram enviados na pasta tst na entrega do trabalho. Com esses testes escolhidos pudemos testar os aspectos desenvolvidos

no montador, como o tratamento de comentários, linhas vazias, linhas somente com comentários, entre outros. Seguem os resultados dos testes:

4.1. Programa de Exemplo

Segue o resultado gerado pelo montador para esse programa:

```
MV-EXE

12 0 999 0

3 0 1 1 6 8 0 1 4 0 0 100
```

Agora, seguem os resultados do emulador para diferentes entradas:

```
Input: 0
Output: 100

Input: 1
Output: 101

Input: -5
Output: 95

Input: 200
Output: 300

Input: 1000000000
Output: 1000000100
```

4.2. Programa do Cálculo da Mediana

Segue o resultado gerado pelo montador para esse programa:

```
MV-EXE

273 0 1273 0

3 0 2 0 263 3 0 2 0 259 3 0 2 0 255 3 0 2 0 251 3 0 2 0 247 1 0 240
1 1 238 9 0 1 18 8 8 0 1 2 1 227 16 9 8 0 1 2 0 219 5 0 1 1 1 214 9
0 1 18 8 8 0 1 2 1 203 16 9 8 0 1 2 0 195 5 0 1 1 1 190 9 0 1 18 8 8
0 1 2 1 179 16 9 8 0 1 2 0 171 5 0 1 1 1 166 9 0 1 18 8 8 0 1 2 1
155 16 9 8 0 1 2 0 147 5 0 1 2 0 137 1 0 135 1 1 133 9 0 1 18 8 8 0
1 2 1 122 16 9 8 0 1 2 0 114 5 0 1 1 1 109 9 0 1 18 8 8 0 1 2 1 98
16 9 8 0 1 2 0 90 5 0 1 1 1 85 9 0 1 18 8 8 0 1 2 1 74 16 9 8 0 1 2
0 66 5 0 1 2 0 57 1 0 55 1 1 53 9 0 1 18 8 8 0 1 2 1 42 16 9 8 0 1 2
0 34 5 0 1 1 1 29 9 0 1 18 8 8 0 1 2 1 18 16 9 8 0 1 2 0 10 5 0 1 4
0 0 0 0 0 0 0
```

Agora, seguem os resultados do emulador para diferentes entradas:

```
Input: 1 77 25 59 20
Output: 25

Input: 1 2 3 4 5
Output: 3

Input: 5 4 3 2 1
Output: 3
```

```
Input: -5 -3 -2 10 0
Output: -2

Input: -5 -3 -2 -1 -6
Output: -3
```

4.3. Programa do Cálculo do N-ésimo Número de Fibonacci

Segue o resultado gerado pelo montador para esse programa:

```
MV-EXE

38 0 1038 2

1 0 3 0 1 1 -6 1 2 -10 5 3 0 8 3 3 17 17 1 3 -21 6 2 8 2 1 7 1 9 0 3
17 2 16 -14 4 1 0
```

Agora, seguem os resultados do emulador para diferentes entradas, sendo que foi considerado que Fibonacci(0) = 0 e Fibonacci(1) = 1:

```
Input: 0
Output: 0

Input: 1
Output: 1

Input: 2
Output: 1

Input:8
Output: 21

Input: 10
Output: 55
```