Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
Compiladores

Trabalho Prático 1
Montador de Dois Passos

Alunos: Jackson Nunes Silva
Júlia Fonseca de Sena

Professor: Renato Antônio Celso Ferreira

Julho
2021

Conteúdo

1 Introdução **1**

2 Implementação **1**

2.1 Passo 1 1

2.2 Passo 2 1

3 Testes **2**

4 Conclusão **2**

1 Introdução

O objetivo do presente trabalho foi implementar um montador de dois passos para uma Máquina Virtual predefinida, cuja entrada é um arquivo contendo o programa em Assembly e cuja saída padrão é um outro arquivo executável no formato aceito pela máquina. É uma forma de fixação e concretização de conceitos vistos em sala de aula de modo mais abstrato, observando o processo de montagem em prática.

2 Implementação

O montador foi implementado na linguagem C e para rodá-lo corretamente, deve-se utilizar inicialmente o comando *make*, que gera um executável *montador.exe* na pasta **bin**. Portanto, o comando de execução é: ⟨bin/montador arquivo_de_entrada⟩.

2.1 Passo 1

Conforme visto em aula, a função do primeiro passo é ler o arquivo de entrada com o código fonte (neste trabalho, em Assembly) e criar a tabela de símbolos, identificando os *labels* declarados para calcular a posição de memória a que eles se referem.

Para isso, o programa lê o arquivo linha a linha e separa os tokens, buscando aqueles que referenciam um *label*, seja definido antes de um ":", seja referenciado por uma instrução. Paralelamente, mantém um contador da posição de memória, que é incrementado conforme o tamanho da instrução, como por exemplo **LOAD** ou **STORE**, que incrementam em 3, enquanto **READ** e **WRITE** aumentam em 2. Caso um *label* seja referenciado por uma instrução, seu símbolo é adicionado à tabela com posição de memória -1, que será atualizada com o valor do contador assim que a definição do label for encontrada.

Quanto ao endereço de carregamento e ao valor inicial da pilha, decidiu-se, sob instruções do monitor, fixar uma pilha de 1000 posições que sempre ocupa os endereços 0 a 999, colocando o endereço de carregamento logo após a pilha, na posição 1000. O valor inicial de PC será 1000+N, sendo N o número de linhas com a pseudoinstrução **WORD** no começo do programa.

A tabela de símbolos foi implementada como uma lista encadeada simples, na qual cada célula possui um símbolo, um endereço de memória e um apontador para a próxima. Possuindo, assim, funções para construção (adição de símbolo, de endereço ou ambos) e para retornar o endereço de memória de um símbolo específico.

2.2 Passo 2

O passo 2 passa novamente por todas as linhas do arquivo, ignorando comentários e linhas vazias, e imprimindo na tela o código necessário para que o emulador execute o programa com sucesso. Primeiro, verifica-se se a palavra lida termina em :, ou seja, se é a declaração de um *label* (nesse caso, nada é feito sobre a palavra nesse passo). Caso contrário, procura-se pela palavra na tabela de símbolos e entre as instruções. Se estiver na tabela de símbolos, imprime-se o endereço correspondente.

Se não estiver, identifica o código da instrução e lê-se os dados necessários de acordo com o tipo de instrução (por exemplo, **HALT** não precisa de nenhum dado adicional, mas **LOAD** lê um registrador e um endereço na memória). Imprime-se o código da instrução, e, quando necessário, o número do registrador (de 0 a 3) ou endereço da memória conforme a tabela de símbolos. Caso a instrução seja **WORD** ou **END**, seu código não é impresso, já que trata-se de pseudoinstruções, e, no **WORD**, lê e imprime o inteiro que será colocado na posição da memória. Por fim, com o **END** ou com uma instrução ou *label* desconhecidos, o programa retorna, interrompendo a tradução.

3 Testes

Para testar a codificação do montador foram implementados dois programas no Assembly da máquina virtual dada, que se encontram na pasta **tst**. Tais testes foram exatamente os sugeridos na especificação do trabalho, ou seja, o de retornar o n-ésimo número da seqûencia de Fibonacci de um n dado na entrada (cujo nome é *fibo.amv*) e outro para retornar a mediana entre cinco números fornecidos (de nome *mediana.amv*). Em suma, a lógica de cada um é a seguinte:

Fibonacci: Lê um inteiro da entrada e caso seja menor que 3, escreve 1 na saída. Caso contrário, inicia um *loop* subtraindo n a cada iteração e, utilizando dois registradores e a pilha, realiza a soma de fibonacci. Os registradores R1 e R2 são iniciados com 0 e 1 inicialmente e o valor de R2 é empilhado, em seguida é feita a soma, armazenada no registrador R2. Assim, R1 desempilha o valor armazenado (correspondente ao antigo valor de R2) e o novo valor de R2 é empilhado (contendo a soma). Isso se repete até que n seja 0 e quando isso ocorre, valor de R2 é escrito na saída.

Mediana: Lê os cinco inteiros da entrada, colocando-os ordenadamente em cinco posições da memória. Primeiro compara o primeiro e o segundo número, colocando-os em ordem, e, após ler cada um dos outros, checa se os lidos anteriormente são maiores ou menores. Se for maior, o desloca para a posição de cima, se for menor, coloca o novo na posição acima dele e passa para leitura do próximo número. Por fim, imprime o número que se encontra na terceira posição, o qual é a mediana.

Além disso, foram utilizados o teste fornecido pelo professor/monitor e um teste com instruções WORD no começo do arquivo para observar a computação do quarto inteiro.

4 Conclusão

A realização deste trabalho prático foi uma ótima oportunidade para entender o funcionamento de um montador de dois passos na prática, o que até então estava somente na teoria. Em relação à dificuldade de implementação, não houve empecilhos significativos, visto que os conceitos já estavam bem solidificados. Portanto, a experiência no geral foi bastante positiva.