Trabalho Prático 1
Compiladores I - 2021/1

Caio Alexandre Campos Maciel (2018054680)
Othávio Ruddá da Cunha Araújo (2018054567)

Departamento de Ciência da Computação (DCC)
Universidade Federal de Minas Gerais (UFMG)

1. Introdução

Este trabalho consiste em construir um montador que permite aos usuários que escrevam programas em um formato semelhante ao assembly, passem para o executável final e obtenham o código de máquina referente a ele. Com este em mãos, pode-se utilizá-lo como entrada para o emulador (máquina virtual básica), disponibilizado pelo professor Renato nos arquivos iniciais.

1.1. Emulador

A máquina virtual utilizada nas simulações possui uma memória de 1000 posições capaz de armazenar apenas números inteiros. Possui também um total de 7 registradores, dos quais 3 são de propósito específico e 4 são de propósito geral. Os registradores de propósito específico são os seguintes:

- PC - Contador de Programa: Contém o endereço da próxima instrução a ser executada.
- AP - Apontador de Pilha: Aponta para o elemento no topo da pilha.
- PEP - Palavra de Estado do Processador: Armazena o estado da última operação lógica/aritmética realizada pelo programa. Consiste em dois bits, um para indicar que operação resultou em 0 e o outro para indicar que resultou em um número negativo.

A máquina virtual é capaz de executar, ao todo, 21 instruções, que estão especificadas no quadro a seguir:

Cód.	Símb.	Oper.	Definição	Ação
0	HALT		Parada	
1	LOAD	R M	Carrega Registrador	Reg[R] ← Mem[M + PC]
2	STORE	R M	Armazena Registrador	Mem[M + PC] ← Reg[R]
3	READ	R	Lê valor para registrador	Reg[R] ← "valor lido"
4	WRITE	R	Escreve conteúdo do registrador	"Imprime" Reg[R]
5	COPY	R1 R2	Copia registrador	Reg[R1] ← Reg[R2] *
6	PUSH	R	Empilha valor do registrador	AP ← AP – 1; Mem[AP] ← Reg[R]
7	POP	R	Desempilha valor no registrador	Reg[R] ← Mem[AP]; AP ← AP + 1
8	ADD	R1 R2	Soma dois registradores	Reg[R1] ← Reg[R1] + Reg[R2] *
9	SUB	R1 R2	Subtrai dois registradores	Reg[R1] ← Reg[R1] – Reg[R2] *
10	MUL	R1 R2	Multiplica dois registradores	Reg[R1] ← Reg[R1] × Reg[R2] *
11	DIV	R1 R2	Dividendo entre dois registradores	Reg[R1] ← dividendo(Reg[R1] ÷ Reg[R2]) *
12	MOD	R1 R2	Resto entre dois registradores	Reg[R1] ← resto(Reg[R1] ÷ Reg[R2]) *
13	AND	R1 R2	AND (bit a bit) de dois registradores	Reg[R1] ← Reg[R1] AND Reg[R2] *
14	OR	R1 R2	OR (bit a bit) de dois registradores	Reg[R1] ← Reg[R1] OR Reg[R2] *
15	NOT	R	NOT (bit a bit) de um registrador	Reg[R] ← NOT Reg[R] *
16	JUMP	M	Desvio incondicional	PC ← PC + M
17	JZ	M	Desvia se zero	Se PEP[zero], PC ← PC + M
18	JN	M	Desvia se negativo	Se PEP[negativo], PC ← PC + M
19	CALL	M	Chamada de subrotina	AP ← AP – 1; Mem[AP] ← PC; PC ← PC + M
20	RET		Retorno de subrotina	PC ← Mem[AP]; AP ← AP + 1

1.2. Montador

O montador codificado é de dois passos e deve receber como entrada um programa em linguagem simbólica com linhas no seguinte formato:

$$[< label >:] < operador >< operando1 >< operando2 > [; comentário]$$

As labels, caso estejam presentes, estão definidas no início da linha e sempre terminam com o caractere ':'. Além disso o montador permite comentários, que são iniciados com o caractere ';'. Os operandos disponíveis são tanto registradores de propósito geral (R0, R1, R2, R3) como posições de memória do programa, identificadas por labels.

Além do conjunto de instruções definido na tabela acima, o montador também oferece as seguintes instruções:

- WORD I : Reserva a posição de memória e a inicializa com o valor I
- END : Indica o final do programa para o montador

2. Solução Proposta

O montador pensado para este projeto, realiza ao todo, duas passagens no arquivo recebido como entrada do usuário.

Na primeira passagem pelo arquivo, o program vai lendo linha por linha e completando a tabela de simbolos, um map $< std :: string, int >$, neste caso. Nessa função, ele começa verificando se a linha é vazia ou se contém algum comentário(';'). Caso o primeiro seja verdadeiro, retorna. Caso dentro da linha lida haja algum label, ele insere um pair (label, pos-atual) dentro da tabela de símbolos. A cada linha vai também incrementando o contador da pos-atual de acordo com a memória necessária por cada símbolo. Ao fim da primeira passada, printa a assinatura ('MV-EXE') e os quatros inteiros: tamanho do programa, endereço de carregamento, valor inicial da pilha e entry point do programa.

Com isso, possui-se agora a tabela de símbolos mapeada de forma correta. Assim, antes da segunda passada, voltamos o apontador para o início do arquivo e recomeça-se a leitura. Faz-se novamente o tratamento das linhas vazias e dos comentários. Mas agora, com os símbolos em mãos, busca-se o número de argumentos requerido por cada um deles. Dessa forma, consegue-se fazer um switch para printar na tela os valores corretos do símbolo de seus argumentos de acordo com a tabela. Vale salientar também que, ao iniciar-se a segunda passagem, o contador do programa também é passado para a função, pois é através dele que os endereços devem se basear para serem calculados de forma correta. Assim, adiciona-se a cada linha lida os códigos na linguagem de máquina para a variável codigo, que ao final é retornada para main e mostrada na tela para o usuário.

3. Modo de Execução

A execução do montador deve ser feita na pasta raiz do projeto onde se encontra o arquivo Makefile. Antes, no entanto, faz-se necessário a execução do comando make, para montar os objetos e fazer o link dos arquivos. Assim, o montador pode ser acionado com o seguinte comando:

./bin/montador $< arquivo - entrada >$

O argumento $arquivo - entrada$ corresponde ao nome do arquivo que contém o programa na linguagem semelhante ao assembly, cuja as instruções são aquelas da tabela já mostrada acima.

4. Testes

Para a realização de testes, foram implementados na linguagem do montador, o programa proposto na especificação e dois outros. Todos eles encontram-se no diretório "./tst/":

- O primeiro lê um número N da entrada e imprime o N-ésimo número da sequência de Fibonacci. Utilizando o valor 5 como entrada, o valor exibido será 3;
- O segundo programa recebe 5 números e calcula a mediana deles. Utilizando os números 5, 7, 4, 9 e 3 como entrada, o resultado foi 5.

5. Conclusão

Desse modo, o montador implementado neste trabalho buscou satisfazer os requisitos da especifação do projeto, de forma clara e coesa. O programa foi desenvolvido em C++, com a modularização das funções em arquivos separados para melhor entendimento.

No desenvolvimento deste primeiro trabalho, algumas dificuldades foram encontradas durante seu percurso. Entre elas, pode-se citar: escolha pela melhor estrutura para armazenar a tabela de símbolos do primeiro passo, tratamento de comentários e, também, sobre a montagem dos programas testes Fibonacci e Mediana, que demandaram um bom tempo de estudo para sua criação.

No entanto, o projeto serviu de grande aprendizado, dada a importância da função do montador na área de Compiladores, que se faz essencial ainda nos dias atuais.