UFRGS - INF01147 - Compiladores - todas as edições
Turma - Prof. Marcelo Johann

Trabalho Prático - Especificação da Etapa 5:
Geração de Código Intermediário

Resumo:

Na quinta etapa do trabalho de implementação de um compilador para a linguagem deve ser feita a geração de código intermediário à partir da AST.

Tarefas necessárias:

a. Implementação da estrutura de TACs (Códigos de instrução de três endereços) – Um novo módulo deve prover uma estrutura com código de instrução e três ponteiros para a tabela de símbolos, permitindo que essas instruções sejam encadeadas em listas de instruções. Cada instrução indica a execução de uma operação com zero, um ou dois operandos que serão valores em memória, e cujo resultado também está em memória, sendo os endereços de memória representados pelos símbolos na tabela de símbolos. O módulo deve prover rotinas utilitárias para criar, imprimire unir listas de instruções;
b. Criação de símbolos temporários e *labels* – Devem ser feitas duas rotinas auxiliares que criam novos símbolos na tabela de símbolos (*hash*), uma para variáveis temporárias e outra para *labels*. Elas serão usadas na geração de código para guardar sub-resultados de cada operação e para marcar os pontos de desvio no fluxo de execução;
c. Geração de código – faça uma rotina que percorre a AST recursivamente, retornando um trecho de código intermediário (lista de TACs) para cada nodo visitado. Essa rotina primeiro processa os nodos filhos, armazenas os trechos de código gerados para cada um deles, depois testa o nodo atual e gera o código correspondente para este nodo. A geração em geral consiste na criação de uma ou mais novas instruções (TACs), união dos trechos das sub-árvores e dessas novas instruções, opcionalmente com a criação de novos símbolos intermediários e labels, retornando um trecho de código completo desse novo nodo;

Desenvolvimento

As instruções em código intermediário servem para isolar as tarefas de geração da sequência básica de instruções dos detalhes e formato específicos de uma arquitetura-alvo. Além disso, a geração usada nesse trabalho emprega técnicas genéricas de forma funcional, didática, mas pode ser otimizada de várias formas antes da geração de código *assembly*. Dois exemplos de otimização são a reutilização de símbolos temporários em expressões e o uso de registradores. Entretanto, essas otimizações não fazem parte desta etapa do trabalho.

A geração de código será feita de baixo para cima e da esquerda para a direita, na árvore. O modo mais simples de encadear novas instruções é representar os trechos de código como

listas encadeadas invertidas, isto é, com um ponteiro para a última instrução de um trecho, e cada instrução apontando para a anterior. Ao final da geração, escreva uma rotina que precorre o código completo e inverte a lista de forma que se possa escrever o código na ordem em que deve ser executado.

Para a geração de símbolos intermediários e *labels*, utilize dois números globais incrementais, e imprima um nome especial seguido desse número em um buffer, para gerar os nomes dos novos símbolos, como por exemplo "__temp0", "__temp1", etc... Faça rotinas **makeTemp** e **makeLabel** para isso.

Para a geração de código, além das rotinas utilitárias de TACs e da rotina recursiva principal que percorre a AST, utilize outras rotinas auxiliares. Isto tem dois motivos: primeiro, a semelhança na geração de código em vários nodos da árvore, especialmente nos operadores aritméticos, relacionais, etc.... Em segundo lugar, use rotinas auxiliares para evitar que a rotina recursiva fique grande. Ela deve fazer "switch(node->type)"e chamar uma rotina auxiliar de geração de código para cada tipo (ou grupo) de instrução.

Além das TACs que irão gerar instruções, você deve usar dois tipos de TACs utilitárias, uma TAC_SYMBOL, que somente facilita a recursão, mas não irá gerar nenhum código assembly, e TAC_LABEL, que marca um rótulo mas funciona como instrução separada, facilitando o isolamento das tarefas de geração do código. Ou seja, você não precisa saber para qual instrução deve saltar, mas sim apenas para qual ponto, e a instrução será a próxima encadeada.

Lista de TACS

Para auxiliar na sua tarefa segue aqui uma lista sugerida de TACs, a qual pode ser completada.
TAC_SYMBOL, TAC_MOVE, TAC_ADD, TAC_MUL, ..., TAC_LABEL, TAC_BEGINFUN, TAC_ENDFUN, TAC_IFZ, TAC_JUMP, TAC_CALL, TAC_ARG, TAC_RET, TAC_PRINT, TAC_READ ...

Controle e organização do seu código fonte

Você deve seguir as mesmas regras das etapas anteriores para organizar o código, permitir compilação com **make**, permitir que o código seja rodado com **./etapa5**, e esteja disponível como **etapa5.tgz**.

Porto Alegre, Setembro de 2020