# Geração de Código

Guilherme Augusto de Oliveira

20 de outubro de 2024

## 1 Representação Intermediária

Após a análise semântica, muitos compiladores geram uma representação intermediária do código fonte para facilitar a geração posterior de código. Uma de suas vantagens é que esse tipo de representação é mais fácil de ser manipulada do que o código fonte original. Além disso, por ser mais próximo da linguagem de máquina, é possível otimizar o código gerado para diferentes arquiteturas.

Uma das formas de represenção intermediária é o código de três endereços. Nesse esquema, cada instrução é composta por, no máximo, três endereços. Por exemplo: $x = yopz$, onde $x$, $y$ e $z$ são endereços de memória e $op$ é um operador. Os endereços podem ser representados por um nome símbolico (por conveniência) ou por um código gerado pelo compilador.

Um conceito importante que envolve o código intermediário são as instruções suportadas. As instruções de três endereços mais comuns são: atribuição, operações aritméticas, operações lógicas, operações de desvio condicional e operações de desvio incondicional. Todo o código fonte é traduzido para essas instruções, que são bem parecidas com a representação em código de máquina.

### 1.1 Memória

O tipo de dado e o tamanho de bytes necessários para armazenar um valor em memória são definidos na tabela de símbolos por meio de atributos semânticos. Desse modo, a partir do nome do identificador é possível a quantidade de memória necessária para armazenar o valor associado a ele.

A memória é alocada em sequência, ou seja, o endereço de memória de um identificador é calculado a partir do endereço do identificador anterior. Por exemplo, se o endereço do identificador $x$ é $1000$ e o tamanho de $x$ é $4$ bytes, então o endereço de $y$ é $1004$. Para manter esse controle, o esquema de tradução mantém uma varíavel interna conhecida como *offset*.

### 1.2 Processo de Tradução

Para traduzir as expressões na forma de três endereços, são utilizados atributos semânticos e ações de tradução auxiliares.

- addr: nome, constante ou identificador temporário que indica o endereço de memória de uma expressão.
- top: entrada correspondente ao topo da pilha de operandos.

- gen: gera uma instrução de três endereços.
- newtemp: é uma função auxiliar que retorna um novo identificador temporário.

Fluxos de controle envolvem o uso de rótulos para marcar o início e o fim de um bloco de código. O rótulo é um identificador único que indica o endereço de memória de uma instrução. O esquema de tradução mantém uma variável interna que armazena o rótulo atual. Outra forma de representar endereços é utilizar diretamente o valor.

O principal problema na geração de código é a presença de desvios condicionais. Para resolver esse problema, é utilizada a técnica de remendo (*backpatching*). Essa técnica consiste em preencher os endereços de memória dos desvios condicionais após a geração de todo o código. Para isso, é necessário manter uma lista de endereços de memória que precisam ser preenchidos. Isso é feito utilizando os atributos *truelist* e *falselist*. O primeiro armazena os endereços de memória que precisam ser preenchidos com o endereço de memória da próxima instrução, caso a condição seja verdadeira. O segundo armazena os endereços de memória que precisam ser preenchidos com o endereço de memória da próxima instrução, caso a condição seja falsa.

Para a manutenção das listas são utilizadas as funções auxiliares *makelist*, *merge* e *backpatch*. A função *makelist* cria uma nova lista de endereços de memória. A função *merge* junta duas listas de endereços de memória. A função *backpatch* preenche os endereços de memória com o endereço de memória da próxima instrução.

Exemplo: $if\ x\ relop\ y\ then\ S_1\ else\ S_2$. Nesse caso, a lista *truelist* é preenchida com o endereço de memória da instrução $S_1$ e a lista *falselist* é preenchida com o endereço de memória da instrução $S_2$.

# 2 Conclusão

É possível perceber que a geração de código intermediário facilita a utilização do código fonte original para diferentes arquiteturas. Existem várias técnicas para isso, sendo a geração de código de três endereços uma das mais comuns.

A próxima etapa é a geração de código de máquina, que pode ser executada diretamente pelo processador. Para isso, o intermediário é re-escrito com referências de registradores, endereços de memória (se ainda não estiverem presentes) e instruções de máquina. Sendo que cada arquitetura possui suas particularidades de implementação. O lado bom é que é mais fácil otimizar o código final tendo a representação intermediária.