Compiladores

Conceitos Básicos

Processadores de Linguagem

- De forma simples, um compilador é um programa que recebe como entrada um programa em uma linguagem de programação – a linguagem fonte – e o traduz para um programa equivalente em outra linguagem – a linguagem objeto.

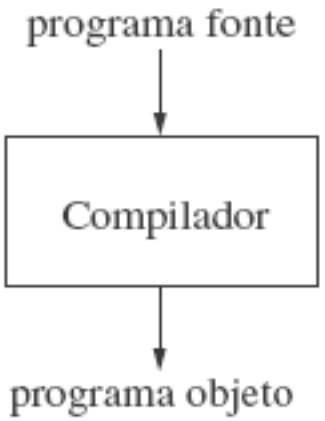


FIGURA 1.1 Um compilador.

Processadores de Linguagem

- Se o programa for um programa em uma linguagem de máquina executável, poderá ser chamado pelo usuário para processar entrada e produzir saída.

FIGURA 1.2 Executando o programa objeto.

Processadores de Linguagem

- Um interpretador é outro tipo comum de processador de linguagem. Um interpretador executa diretamente as operações especificadas no programa fonte sobre as entradas fornecidas pelo usuário.

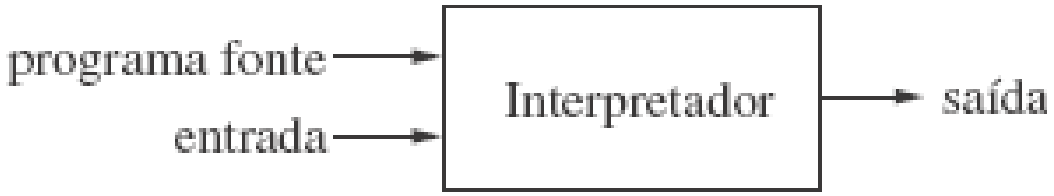


FIGURA 1.3 Um interpretador.

Sistema de Processamento de Linguagem

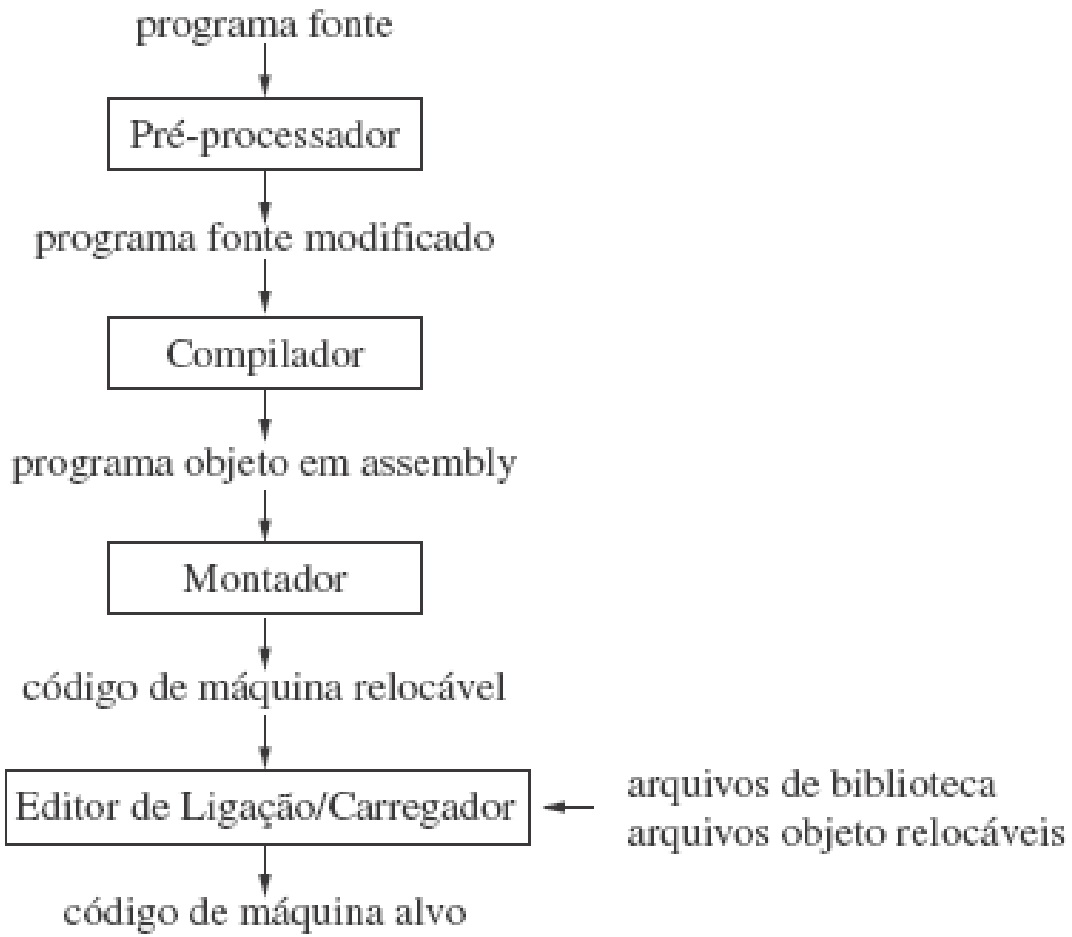


FIGURA 1.5 Um sistema de processamento de linguagem.

- Pré-processador: possui a tarefa de coletar o programa fonte e também pode expandir macros em comandos na linguagem fonte.

Sistema de Processamento de Linguagem

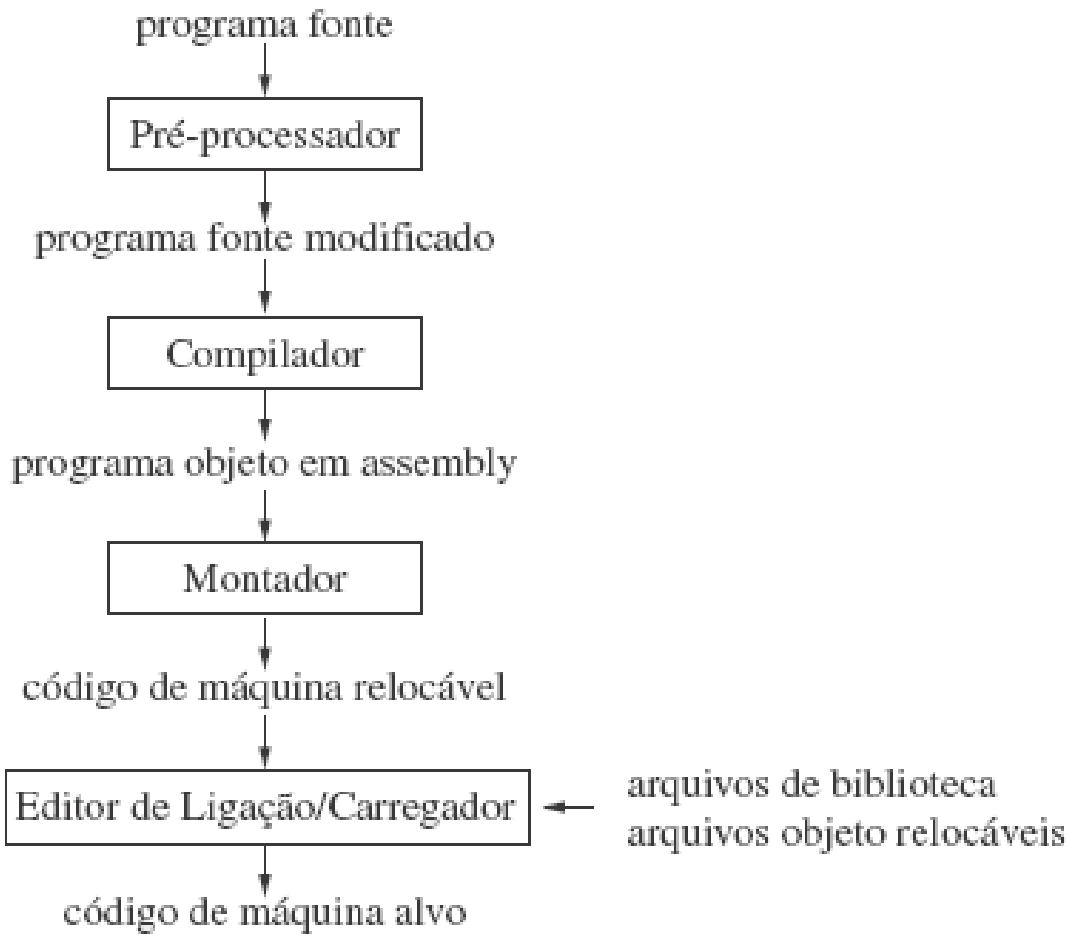


FIGURA 1.5 Um sistema de processamento de linguagem.

- Editor: resolve os endereços de memória externas, onde o código em um arquivo pode referir-se a uma localização em outro arquivo.

Sistema de Processamento de Linguagem

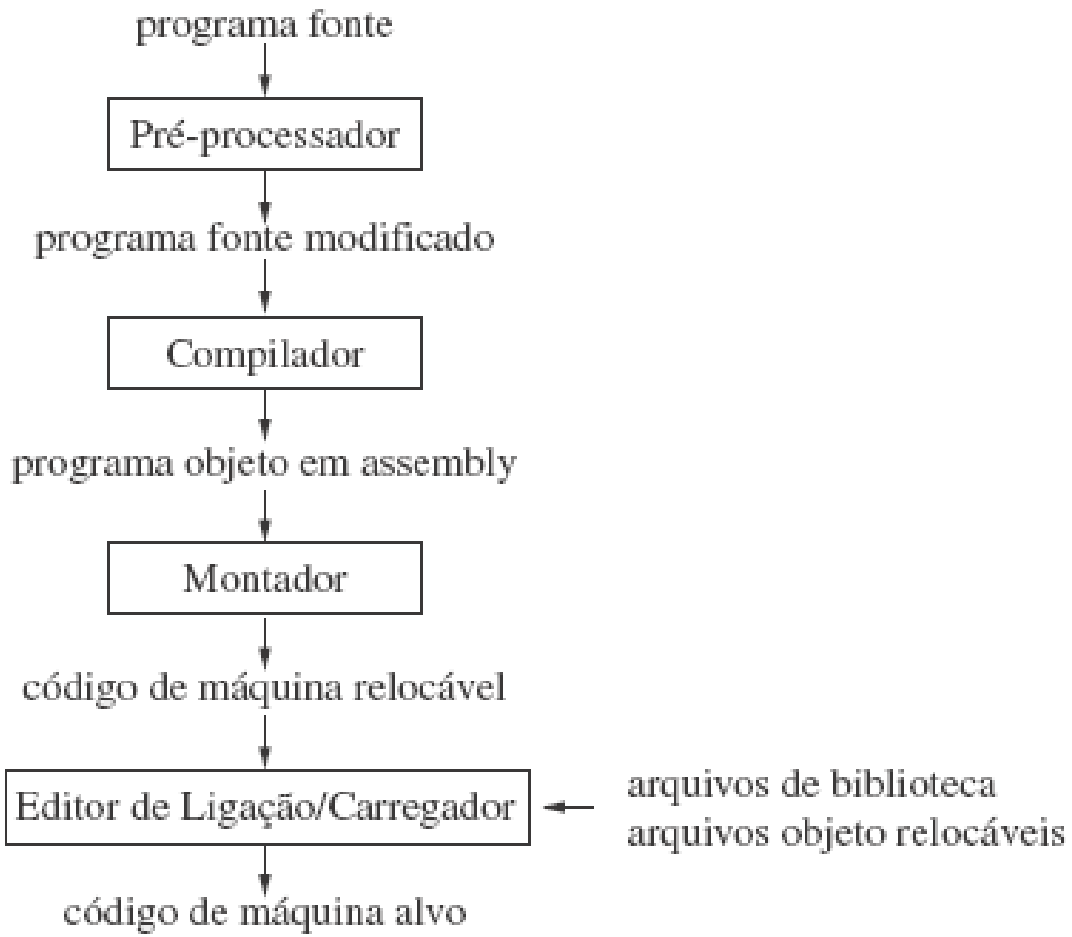


FIGURA 1.5 Um sistema de processamento de linguagem.

- Carregador: reúne todos os arquivos objeto executáveis na memória para execução.

Estrutura de um Compilador

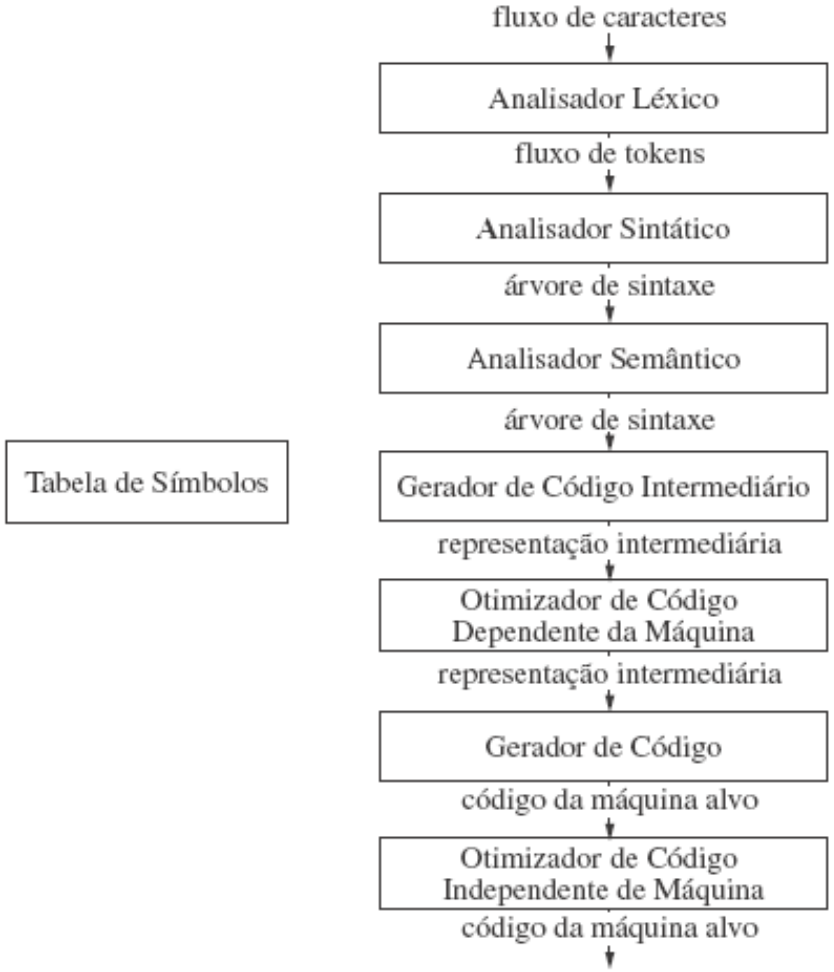


FIGURA 1.6 Fases de um compilador.

Análise Léxica

- O analisador léxico lê um fluxo de caracteres que compõem o programa fonte e os agrupa em seqüências significativas, chamadas lexemas.
- Para cada lexema, o analisador léxico produz como saída um token no formato:
  - *<nome-token, valor-atributo>*

Análise Léxica

- O *nome-token* é um símbolo abstrato que é usado durante a análise sintática, e o segundo componente, *valor-atributo*, aponta para uma entrada na tabela de símbolos referente a esse token.
  - A informação da entrada da tabela de símbolos é necessária para a análise semântica e para a geração de código.

Análise Léxica

- Exemplo: suponha o seguinte comando de atribuição:
  - `position = initial + rate * 60`
- Veja os lexemas criados e passado ao analisador sintático:
  1. `position` é um lexema mapeado em um token <**id**, 1>, onde **id** é um símbolo abstrato que significa *identificador* e 1 aponta para a entrada da tabela de símbolos onde se encontra `position`. A entrada da tabela de símbolos para um identificador mantém informações sobre o identificador, como seu nome e tipo.

Análise Léxica

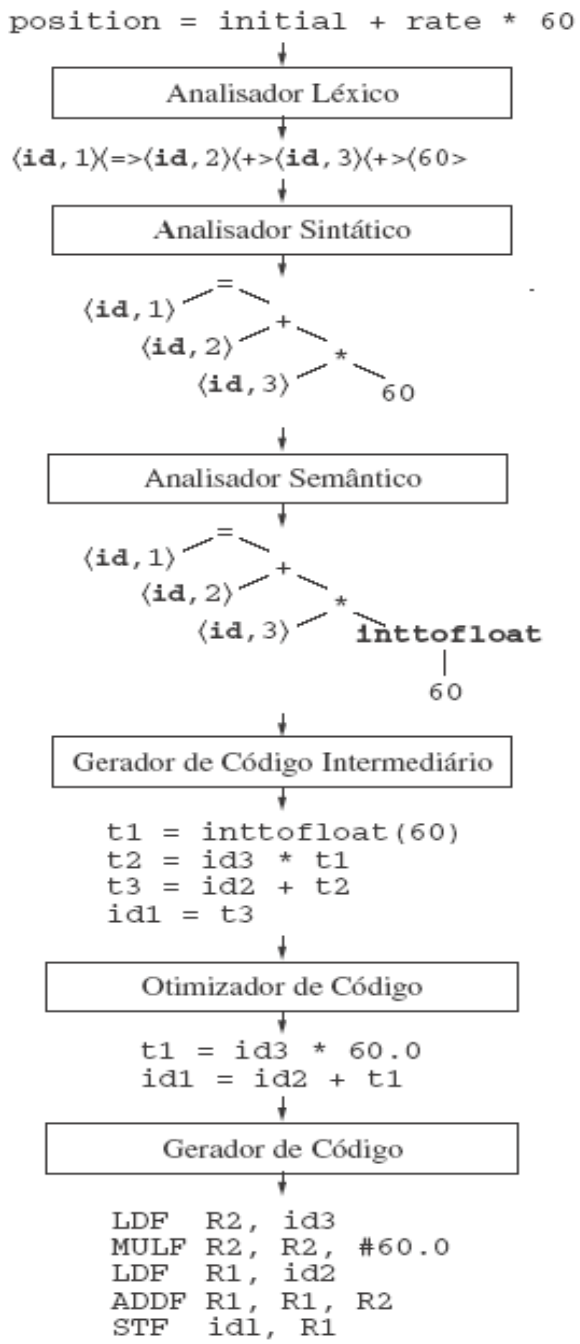
2. O símbolo de atribuição = é um lexema mapeado para o token < **=** >. Como esse token não precisa de um valor de atributo, omitimos o segundo componente.
3. `initial` é um lexema mapeado para o token <**id**, 2>, onde 2 aponta para a entrada da tabela de símbolos onde se encontra `initial`.
4. `+` é um lexema para o token < **+** >.

Análise Léxica

5. `Rate` é um lexema mapeado para o token <**id**, 3> onde o valor 3 aponta para a entrada da tabela de símbolos onde se encontra `rate`.
6. * é um lexema mapeado para o token < * >.
7. `60` é um lexema mapeado para o token <`60`>

Análise Sintática

- O analisador sintático utiliza os primeiros componentes dos tokens produzidos pelo analisador léxico para criar uma representação intermediária tipo árvore.
- Árvore de sintaxe em que cada nó interior representa uma operação, e os filhos do nó representam os argumentos da operação.



position	...
initial	...
rate	...

TABELA DE SÍMBOLOS

Análise Semântica

- O analisador semântico utiliza a árvore de sintaxe e as informações na tabela de símbolos para verificar a consistência semântica do programa fonte com a definição da linguagem.
  - Ele também reúne informações gerais sobre os tipos e as salva na árvore de sintaxe ou na tabela de símbolos, para uso na geração de código intermediário.

Análise Semântica

- Exemplo: verificação de tipo
  - Exigência que um índice de vetor seja um inteiro.
  - Coerções: conversões de tipos.
    - Exemplo: suponha que `position`, `initial` e `rate` tenham sido declarados como números de ponto flutuante, e que o lexema `60` tenha a forma de um inteiro.

Geração de Código Intermediário

- Reproduzir o código-fonte em um código intermediário

```
t1 = inttofloat (60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

Otimização de Código

- É uma fase independente das arquiteturas de máquina e realiza algumas transformações no código intermediário com o objetivo de produzir um código objeto melhor.

```
t1 = id3 * 60.0
id1 = id2 + t1
```

Geração de Código

- Mapeia a representação intermediária do código-fonte em uma linguagem objeto.

```
LDF   R2, id3
MULF  R2, R2, #60.0
LDF   R1, id2
ADDF  R1, R1, R2
STF  id1, R1
```