COMPILADORES
INTRODUÇÃO

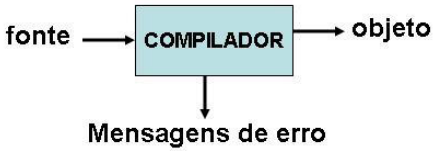
Mariza A S. Bigonha e Roberto S. Bigonha

UFMG

16 de outubro de 2008

Processadores de Linguagens

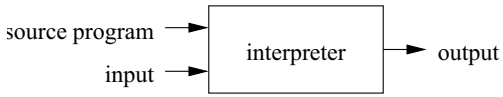
Compilador. Programa que recebe como entrada um programa em uma linguagem de programação, a linguagem fonte, e o traduz para um programa equivalente em outra linguagem, a linguagem objeto.



Primeiro compilador: FORTRAN, 1950: 18 homens-ano

··· Processadores de Linguagens

Interpretador. Em vez de produzir um programa objeto como resultado da tradução, um interpretador executa diretamente as operações especificadas no programa fonte sobre as entradas fornecidas pelo usuário.

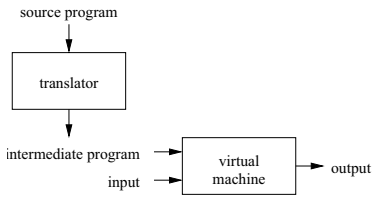


··· Processadores de Linguagens

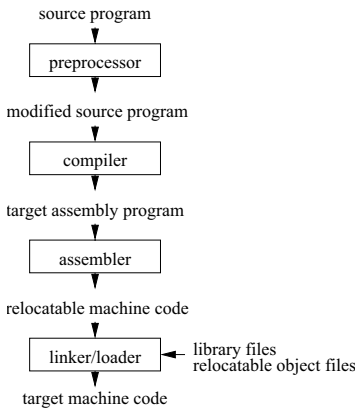
Compiladores Híbridos. Combinam compilação e interpretação. Exemplo: processadores da linguagem JAVA.

- Um programa fonte pode ser primeiro compilado para uma forma intermediária, bytecodes.
- Os bytecodes são então interpretados por uma máquina virtual.

Vantagem. Os bytecodes compilados em uma máquina podem ser interpretados em outra máquina.



· · · Processadores de Linguagens



1. Pre-processadores
   - processamento de macros
   - inclusão de arquivos
   - extensões de linguagens
2. Montadores (assemblers)
3. Carregadores e editores de ligação

Ferramentas Relacionadas

(Utilizam técnicas similares)

- Editores sintáticos ou com estruturas.
- Verificadores de sintaxe.
- Interpretadores.
- Formatadores de Textos.
- Compiladores de Silício.
- Linguagens de acesso a banco de dados.
- Interfaces para comunicação homem/máquina.
- Preprocessadores ou macro-processadores.

Estrutura Geral de um Compilador

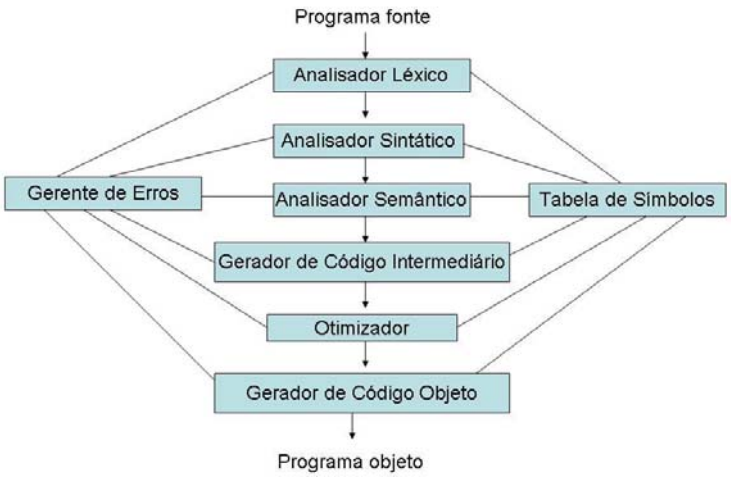
Análise (Front-End). Dependente da linguagem fonte.

- Subdivide o programa fonte em partes constituintes e impõe uma estrutura gramatical sobre elas.
- Usa essa estrutura para criar uma representação intermediária do programa fonte.
- Coleta informações sobre o programa fonte e as armazena em uma estrutura de dados chamada tabela de símbolos, que é passada adiante junto com a representação intermediária para a parte de síntese.

Síntese (back-end). Dependente da linguagem objeto.

- Constrói o programa objeto desejado a partir da representação intermediária e das informações na tabela de símbolos.

Fases de Compilação



Estrutura Geral de um Compilador – Análise

- Análise linear ou léxica.
- Análise hierárquica ou sintática.
- Análise semântica.

··· Estrutura Geral de um Compilador – Análise

Análise Léxica

Analisador léxico: lê o fluxo de caracteres que compõe 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>
$nome - token$: símbolo abstrato usado durante a análise sintática.
$valor - atributo$: 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.

··· Estrutura Geral de um Compilador – Análise

··· Análise Léxica

Exemplo: X := A + B / 5

(id, 'X')
(':=',)
(id, 'A')
(addop, '+')
(id,'B')
(mulop,'/')
(num, '5')

··· Estrutura Geral de um Compilador – Análise

Análise Sintática

Analisador sintático: utiliza os primeiros componentes dos $tokens$ produzidos pelo analisador léxico para criar uma representação intermediária tipo árvore, que mostra a estrutura gramatical da seqüência de tokens.

Uma representação típica é uma árvore de derivação em que cada nó interior representa uma operação, e os filhos do nó representam os argumentos da operação.

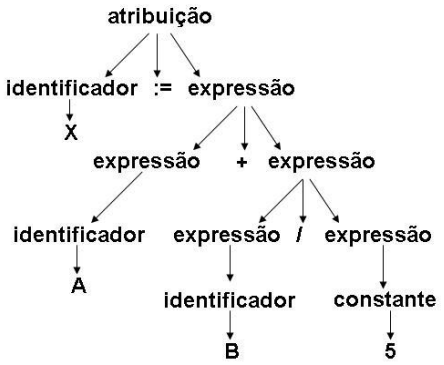
··· Análise Sintática

- Estrutura hierárquica ou gramática:
  1. identificador é expressão
  2. número é expressão
  3. se e_1 e e_2 são expressões, também são expressões:

$e_1 + e_2$
$e_1 - e_2$
$e_1 * e_2$
e_1/e_2
(e_1)

··· Análise Sintática

Exemplo de Árvore de Derivação:: X := A + B / 5



Análise Semântica

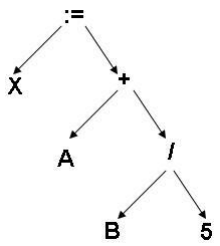
Analisador semântico: utiliza a árvore de derivação e as informações na tabela de símbolos para verificar a consistência semântica do programa fonte com a definição da linguagem.

Analisador semântico: reúne informações sobre os tipos e as salva na árvore de derivação ou na tabela de símbolos, para uso subseqüente durante a geração de código intermediário.

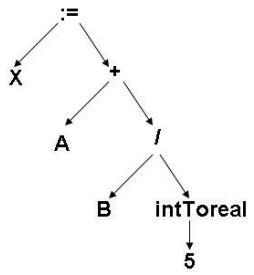
Uma parte importante da análise semântica é a **verificação de tipo**, em que o compilador verifica se cada operador possui operandos compatíveis.

··· Análise Semântica

Exemplo: X := A + B / 5



Exemplo: X := A + B / 5



· · · Estrutura Geral de um Compilador – Análise

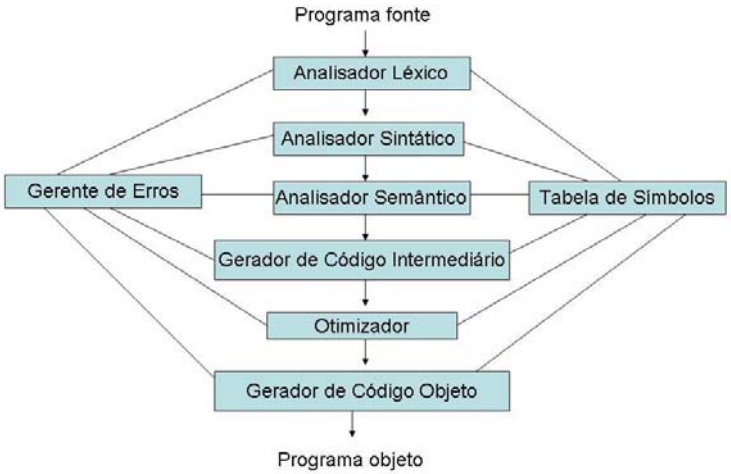
Geração de Código Intermediário

Após a análise sintática e semântica do programa fonte, muitos compiladores geram uma representação intermediária explícita de baixo nível ou do tipo linguagem de máquina, que é possível imaginar como um programa para uma máquina abstrata.

Essa representação intermediária deve ter duas propriedades importantes:

- ser facilmente produzida
- ser facilmente traduzida para a máquina alvo.

· · · Estrutura Geral de um Compilador – Análise



Estrutura Geral de um Compilador – Síntese

Otimização de código

A fase de otimização de código independente das arquiteturas de máquina faz algumas transformações no código intermediário com o objetivo de produzir um código objeto melhor.

Normalmente, melhor significa mais rápido.

Outros objetivos podem ser desejados, como:

- um código menor ou
- um código objeto que consuma menos energia

· · · Estrutura Geral de um Compilador – Síntese

· · · Otimização de código

uma seqüência de instruções ou código de três endereços

t_1 = **inttofloat(60)**
$t_2 = id_3$ * t_1
$t_3 = id_2 + t_2$
$id_1 = t_3$

Uma seqüência de código menor:

$t_1 = id_3$ * **60.0**
$id_1 = id_2 + t_1$

Geração de Código

Gerador de código: recebe como entrada uma representação intermediária do programa fonte e o mapeia em uma linguagem objeto.

Se a linguagem objeto for código de máquina de alguma arquitetura, devem-se selecionar os registradores ou localizações de memória para cada uma das variáveis usadas pelo programa.

Um aspecto crítico da geração de código está relacionado à cuidadosa atribuição dos registradores às variáveis do programa.

Geração de Código

Dada a seqüência de código:

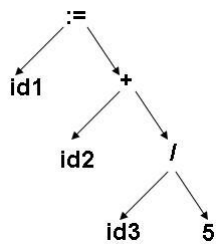
$t_1 = id_3$ * 60.0
$id_1 = id_2 + t_1$

O código intermediário ao lado poderia ser traduzido para o código de máquina:

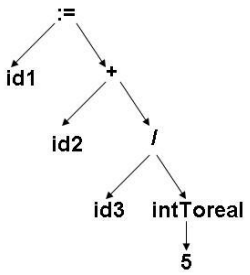
LDF R2, id_3
MULF R2, R2, #60.0
LDF R1, id_2
ADDF R1, R1, R2
STF id_1, R1

Formas internas Fonte **X := A + B / 5**
Analisador Léxico — $id_1 := id_2 + id_3/5$

Analisador Sintático



Analisador Semântico:



Código Intermediário

t_1 := `inttoreal`**(5)**
t_2 := id_3 / t_1
t_3 := id_2 + t_2
id_1 := t_3
-

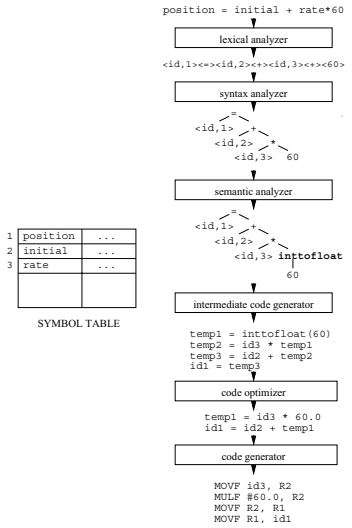
Otimizador

t_1 := id_3 / **5.0**
id_1 := id_2 + t_1

Gerador de Código

MOVF id_3, R_2
DIVF #5.0, R_2
MOVF id_2, R_1
ADDF R_2, R_1
MOVF R_1, id_1

··· Estrutura Geral de um Compilador - Outro Exemplo



Agrupamento de Fases em Passos

- **Fases de um compilador** diz respeito à sua organização lógica.

  As atividades de várias fases podem ser agrupadas em um passo.

- **Passos de um compilador** lê um arquivo de entrada e o escreve em um arquivo de saída.
  - Fases de análise léxica, sintática, semântica e geração de código intermediário podem ser agrupadas em um passo.
  - Fase de otimização do código pode ser um passo opcional.
  - Pode haver um passo para o back-end.

··· Agrupamento de Fases em Passos

Compiladores de diferentes linguagens fonte para 1 máquina alvo:

combina diferentes front-ends com 1 único back-end para essa máquina.

Compiladores para diferentes máquinas alvo:

combina um único front-end com back-ends para diferentes máquinas alvo.

Ferramentas de Construção de Compiladores

Geradores de Analisadores Léxicos produzem analisadores léxicos a partir de uma descrição dos tokens de uma linguagem em forma de expressão regular.

Geradores de Reconhecedores Sintáticos produzem automaticamente reconhecedores sintáticos a partir de uma descrição gramatical de uma linguagem de programação.

Mecanismos de tradução dirigida por sintaxe produzem coleções de rotinas para percorrer uma árvore de derivação e gerar código intermediário.

· · · Ferramentas de Construção de Compiladores

Geradores de Código Automático produzem um gerador de código a partir de uma coleção de regras para traduzir cada operação da linguagem intermediária na linguagem de máquina para uma máquina alvo.

Mecanismos de análise de fluxo de dados facilitam a coleta de informações sobre como os valores são transmitidos de parte de um programa para cada uma das outras partes.

Conjuntos de ferramentas para a construção de compiladores oferecem um conjunto integrado de rotinas para a construção das diversas fases de um compilador.

Evolução das LPs e os Impactos nos Compiladores

Projeto de linguagens de programação e compiladores estão intimamente relacionados

⇓

Avanços nas LPs impõem novas demandas sobre os projetistas de compiladores.

- Compiladores criar algoritmos e representações para traduzir e dar suporte aos novos recursos das LPs.
- Compiladores auxiliam a difundir o uso de LP de alto nível, minimizando o custo adicional da execução dos programas escritos nessas LPs.
- Compiladores são responsáveis por efetivar o uso das arquiteturas de computador de alto desempenho nas aplicações dos usuários.
- O desempenho de um sistema de computação é tão dependente da tecnologia de compilação que os compiladores são usados como uma ferramenta na avaliação dos conceitos arquitetônicos antes que um computador seja montado.

Modelagem no Projeto e Implementação do Compilador

O estudo dos compiladores é principalmente um estudo de como projetar os modelos matemáticos certos e escolher corretamente os algoritmos, mantendo-se o equilíbrio entre a necessidade de generalização e abrangência versus simplicidade e eficiência.

Entre os modelos mais importantes destacam-se:

Máquinas de estado finito e expressões regulares: úteis para descrever as unidades léxicas dos programas como palavras-chave, identificadores etc. e os algoritmos usados pelo compilador para reconhecer essas unidades.

Gramáticas livres de contexto e árvores: usadas para descrever a estrutura sintática das linguagens de programação, como o balanceamento dos parênteses ou construções de controle.

A ciência da Otimização do Código

O termo otimização no projeto de compiladores refere-se às tentativas que um compilador faz para produzir um código que seja mais eficiente do que o código óbvio.

Otimização é um nome errado, pois não é possível garantir que um código produzido por um compilador seja tão ou mais rápido do que qualquer outro código que realiza a mesma tarefa.

A otimização de código vem-se tornando cada vez mais importante e mais complexa no projeto de um compilador.

Mais complexa porque as arquiteturas dos processadores se tornaram mais complexas, gerando mais oportunidades de melhorar a forma como o código é executado.

· · · A ciência da Otimização do Código

Mais importante porque os computadores maciçamente paralelos exigem otimizações substanciais, ou seu desempenho é afetado por algumas ordens de grandeza.

Com a provável predominância de **arquiteturas de máquinas multicore** (computadores com chips possuindo maiores quantidades de processadores), os compiladores enfrentarão mais um desafio para tirar proveito das máquinas multiprocessadoras.

O uso de uma base matemática rigorosa permite mostrar que uma otimização está correta e produz o efeito desejado para todas as entradas possíveis.

· · · A ciência da Otimização do Código

Otimizações precisam atender os seguintes objetivos de projeto:

- Otimização precisa ser correta, ou seja, preservar a semântica do programa compilado.
  Nunca é demais enfatizar a importância da exatidão.

- Otimização precisa melhorar o desempenho de muitos programas.

  Desempenho diz respeito à velocidade de execução do programa. Especialmente em aplicações embutidas, também pode ser necessário diminuir o tamanho do código gerado.

  Para dispositivos móveis, é desejável que o código gerado minimize o consumo de energia.

· · · A ciência da Otimização do Código

- O tempo de compilação precisa continuar razoável.

  É preciso manter o tempo de compilação pequeno para dar suporte a um ciclo rápido de desenvolvimento e depuração.

- O esforço de engenharia empregado precisa ser administrável.

  Compilador é um sistema complexo; é preciso manter o sistema simples para garantir que os custos de engenharia e manutenção do compilador sejam viáveis.

Otimizações para Arquiteturas de Computador

A rápida evolução das arquiteturas de computador gerou uma demanda insaciável por novas técnicas de compilação.

Quase todos os sistemas de alto desempenho tiram proveito de duas técnicas básicas:

- **Paralelismo**
  - de $instrução$: várias operações são executadas simultaneamente.
  - de $processador$ diferentes $threads$ da mesma aplicação são executadas em diferentes processadores.
- **Hierarquias de memória**. As hierarquias de memória são uma resposta à limitação básica de que podemos construir um dispositivo de armazenamento muito rápido ou muito grande, mas não um dispositivo de armazenamento que seja tanto rápido quanto grande.

Paralelismo

Microprocessadores modernos exploram o paralelismo de instrução.

O paralelismo pode não ser visível ao programador.

Programas são escritos como se todas as instruções fossem executadas seqüencialmente.

O hardware verifica dinamicamente se há dependências no fluxo seqüencial das instruções e, quando possível, as emite em paralelo.

Escalonador de instruções: torna mais eficiente o paralelismo de instrução.

· · · Paralelismo

O paralelismo de instrução também pode aparecer explicitamente no conjunto de instruções.

Máquinas VLIW (Very Long Instruction Word) possuem instruções que podem emitir várias operações em paralelo. Exemplo: máquina Intel IA64.

Todos os microprocessadores de alto desempenho de uso geral incluem instruções que podem operar sobre um vetor de dados ao mesmo tempo.

Técnicas de compiladores têm sido desenvolvidas para gerar automaticamente código para essas máquinas a partir de programas seqüenciais.

Hierarquias de Memória

Uma hierarquia de memória consiste em vários níveis de armazenamento com diferentes velocidades e tamanhos, com o nível mais próximo do processador sendo o mais rápido, porém o menor.

O tempo médio de acesso à memória de um programa é reduzido se a maior parte dos seus acessos for satisfeita pelos níveis mais rápidos da hierarquia.

Tanto o paralelismo quanto a existência de uma hierarquia de memória melhoram o desempenho potencial de uma máquina, mas ambos precisam ser utilizados de modo eficaz pelo compilador, a fim de oferecer um desempenho real em uma aplicação.

Projeto de Novas Arquiteturas de Computador

Primeiros projetos de arquiteturas de computadores: os compiladores só eram desenvolvidos após a construção das máquinas.

Isso mudou.

No desenvolvimento de arquiteturas de computadores modernas: os compiladores são desenvolvidos no estágio de projeto do processador, e o código compilado, executando em simuladores, é usado para avaliar os recursos arquitetônicos propostos.

RISC

Exemplo: Arquitetura RISC (Reduced Instruction-Set Computer ? computador com um conjunto reduzido de instruções) - compiladores influenciaram no projeto da arquitetura de computador.

Antes a tendência era desenvolver gradativamente conjuntos de instruções cada vez mais complexos, com o objetivo de tornar a programação assembler mais fácil.

Exemplo: Arquiteturas CISC (Complex Instruction-Set Computer ? computador com um conjunto de instruções complexas).

A maioria das arquiteturas de processadores de uso geral, incluindo PowerPC, SPARC, MIPS, Alpha e PA-RISC, é baseada no conceito de RISC.

A arquitetura x86 - o microprocessador mais popular - possui um conjunto de instruções CISC,

Arquiteturas Especializadas

Durante as três últimas décadas, foram propostos muitos conceitos arquitetônicos. Eles incluem:

- Máquinas de fluxo de dados
- Máquinas de vetor
- Máquinas VLIW (Very Long Instruction Word ? palavra de instrução muito longa)
- Arranjos de processadores SIMD (Single Instruction, Multiple Data - única instrução, múltiplos dados)
- Arranjos sistólicos
- Multiprocessadores com memória compartilhada
- Multiprocessadores com memória distribuída.

··· Arquiteturas Especializadas

O desenvolvimento desses conceitos arquitetônicos foi acompanhado pela pesquisa e desenvolvimento de novas tecnologias de compilação.

Algumas dessas idéias deram origem aos projetos de máquinas embutidas.

Diferente dos processadores de uso geral, nos quais as economias de escala levaram à convergência das arquiteturas de computador, processadores de aplicações específicas apresentam uma diversidade de arquiteturas de computador.

Tecnologia de compiladores: necessária não apenas para dar suporte à programação para essas arquiteturas, mas para avaliar os projetos arquitetônicos propostos.

FIM