8 – Geração de Código

Linguagens e Programação

Ana Madureira
Engenharia Informática
Ano Letivo: 2021/2022

Fontes:

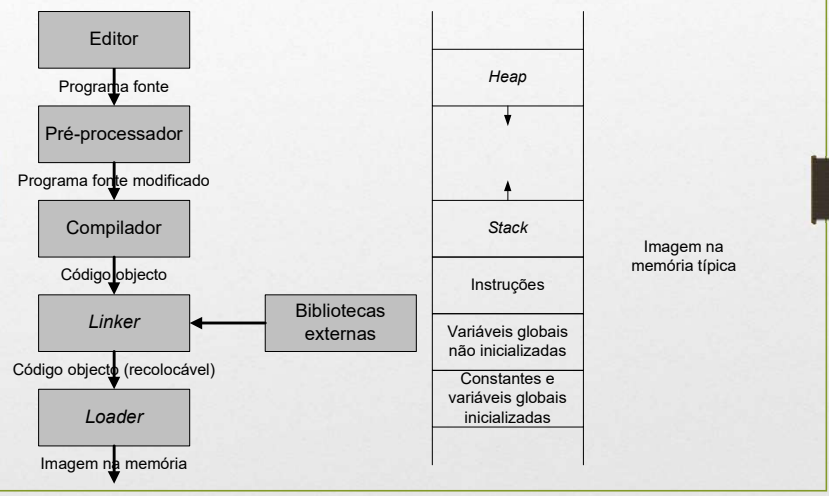
1. Compiladores Princípios e práticas, Keneth C.Louden, Thomson, 2004.
   Cap. 8 Geração de código

2. Processadores de Linguagens – da concepção à implementação, Rui Gustavo Crespo. IST Press.1998.
   Cap. 7 e 8 Geração de Código (Código Intermédio e Código Final)

3. Compiladores Princípios,Técnicas e Ferramentas AlfredV.Aho,R.Sethi e Jeffrey D.Ullman, 2007.
   Cap. 9 Geração de código

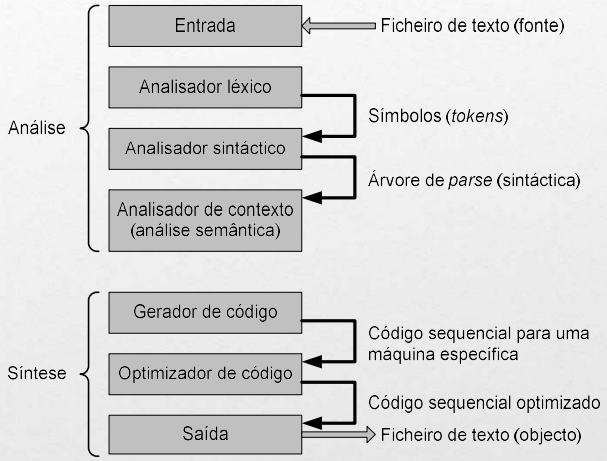
1

Enquadramento do Compilador



2

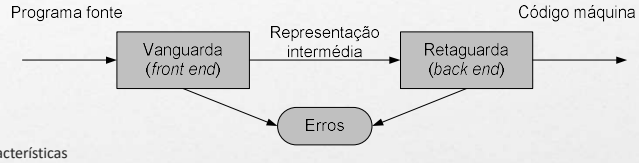
Estrutura Geral de um Compilador (Estrutura conceptual)



3

Estrutura Geral de um Compilador (Estrutura funcional)

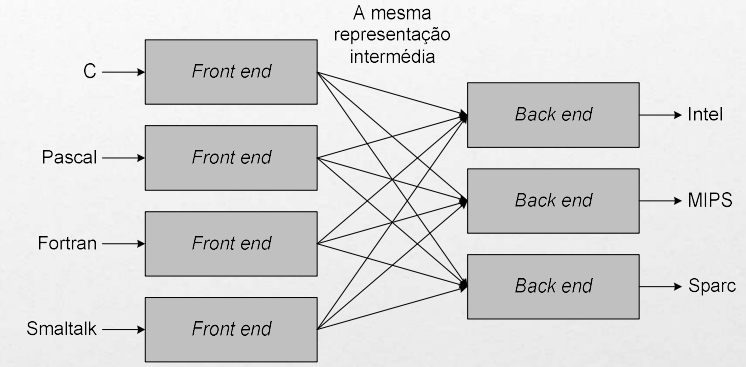
- Divisão do compilador em vanguarda (*front end*) e retaguarda (*back end*)



- Características
  - Existência de uma representação intermédia do programa fonte (máquina abstrata)
  - A vanguarda mapeia o programa fonte numa representação intermédia
  - A retaguarda produz o código máquina (máquina concreta) a partir da representação intermédia
  - Simplifica a produção de compiladores para várias máquinas concretas
  - Simplifica a produção de compiladores para várias linguagens fonte
  - Duas passagens → código mais eficiente que numa única passagem

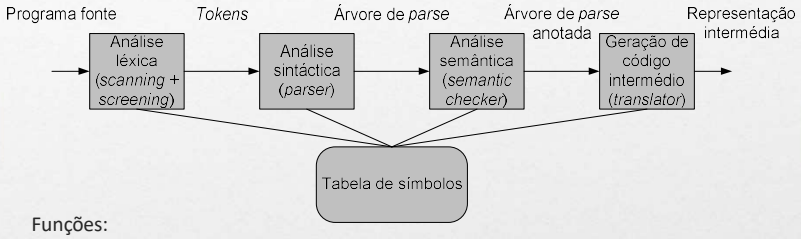
4

Vantagens da representação intermédia



5

Front end

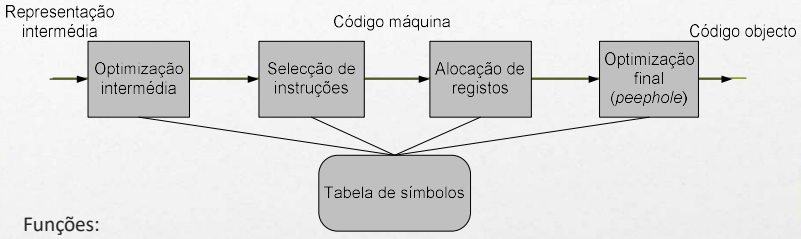


Funções:

- Reconhecer programas válidos
- Produzir mensagens de erro
- Produzir a representação intermédia
- Produzir um mapa de armazenamento preliminar

6

Back end

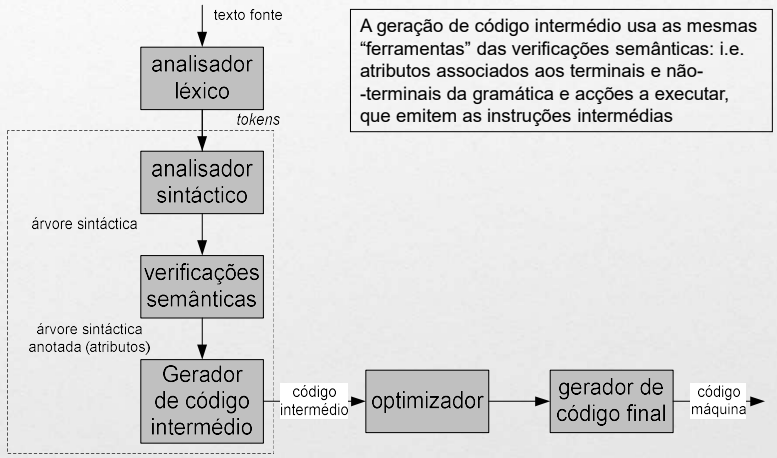


Funções:

- Traduzir a representação intermédia em código alvo
- Escolher as instruções correspondentes a cada operação definida na representação intermédia
- Decidir que informação manter nos registos do processador
- Assegurar a concordância com os formatos usados por outros componentes do sistema de desenvolvimento de *software*

7

Geração de código intermédio



A geração de código intermédio usa as mesmas "ferramentas" das verificações semânticas: i.e. atributos associados aos terminais e não-terminais da gramática e acções a executar, que emitem as instruções intermédias

8

Geração de código intermédio

- O código é produzido pelo percurso na árvore de *parse* construída na fase de análise. A fase de síntese pode decorrer quer em paralelo com a análise quer no seu seguimento; neste último caso, será necessário manter a árvore numa representação apropriada como suporte à segunda fase (que pode constituir uma nova passagem)
  - Nos compiladores de Pascal são comuns duas passagens – a primeira produzindo código intermédio (P_CODE) para uma máquina abstracta de *stack*; a segunda correspondendo à produção de código máquina efectiva para um computador específico
- O método mais comum para realizar a produção de código é através da inserção das acções necessárias no quadro da análise sintáctica

9

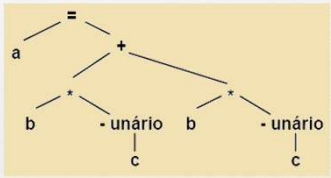
Código intermédio

- O **código intermédio** constitui uma representação do texto fonte, numa sequência de instruções (por vezes uma árvore ou grafo acíclico dirigido), que quando "executadas", têm o mesmo resultado que o especificado no texto fonte
- As instruções do código intermédio são **independentes** do processador alvo
- Vantagens do código intermédio:
  - **Reutilização**: Na construção de um compilador de uma dada linguagem para várias máquinas, só é necessário mudar o gerador final
  - **Optimização**: O módulo de optimização pode ser o mesmo para vários compiladores de linguagens fonte diferentes e para máquinas alvo diferentes

10

Código intermédio (cont.)

- Algumas formas de código intermédio:
  - **Árvores ou grafos sintácticos**, onde os nós interiores representam operações e as folhas operandos. Exemplo para: a = b*-c + b*-c



Máquina de stack:
push b
push c
uminus
mul
pop t1
push t1
push t1
add
pop a

  - **Sequências de instruções** (para a = b*-c + b*-c):

Máquina de 3 endereços:
t1 ← uminus c
t1 ← b * t1
a ← t1 + t1

11

Representação das instruções de 3 endereços

- **Representação em quádruplas** (ou *quads*): sequências de estruturas com 4 campos - código de operação, 1º operando, 2º operando e resultado ou destino

opcode	1º op.	2º op.	result. ou dest.

- **Representação em triplas**: cada instrução só contém o código de operação e os 1º e 2º operandos
  - o resultado não está explicitado na instrução (excepto no caso da instrução de atribuição)
  - assume-se que a própria instrução guarda o valor do resultado que pode servir de operando a outras instruções

Exemplo: Código em *quads* e triplas para "a = b * -c + b * -c"

	opcode	1º op	2º op	result
(0)	uminus	c	-	t1
(1)	*	b	t1	t1
(2)	uminus	c	-	t2
(3)	*	b	t2	t2
(4)	+	t1	t2	t1
(5)	=	t1	-	a

	opcode	1º arg	2º arg
(0)	uminus	c	-
(1)	*	b	(0)
(2)	uminus	c	-
(3)	*	b	(2)
(4)	+	(1)	(3)
(5)	=	a	(4)

12

Código final

- Após a geração do código intermédio e eventual optimização (melhoramento), segue-se, como última fase a geração do código final
- O código intermédio é já próximo do código final da máquina alvo, com a excepção da não utilização de registos (em vez disso usam-se temporários) e da utilização de "instruções" de mais alto nível
- O problema principal da geração do código final é a alocação eficiente dos registos do processador
- O código final gerado toma geralmente duas formas:
  - **Código *assembly***: usa nomes simbólicos para as instruções máquina, registos, variáveis e posições no programa; necessita de um assemblador para gerar o código máquina final
  - **Código máquina recolocável** (*relocatable*): As instruções estão já codificadas, sendo no entanto apenas usados endereços relativos, quer para os dados quer para as posições no próprio código executável; durante o carregamento na memória os endereços absolutos que porventura tenham de existir são corrigidos para o local onde o programa foi carregado

13

Máquina alvo

Como é óbvio o código final a gerar depende fortemente do processador alvo onde se pretende que seja executado

Processadores RISC Reduced Instruction Set Computer:	Processadores CISC Complex Instruction Set Computer:
• Número elevado de registos de uso geral (32 ou mais) • As instruções que envolvem a memória limitam-se geralmente a transferências para os registos (*load* e *store*) • Instruções aritméticas e lógicas de 3 endereços, mas envolvendo apenas registos ou dados imediatos • Instruções com tamanhos e tempos de execução semelhantes (com raras excepções)	• Número limitado de registos e alguns com fins específicos (SP, FP, …) • Geralmente instruções de 2 endereços (o destino é um dos operandos) • Diversos modos de endereçamento (imediato, registo, absoluto, indexado, indirecto) com custos diferentes (espaço e tempo de execução) • Algumas instruções com efeitos colaterais (autoincremento, etc)

14