Compiladores

Geração de Código Intermediário

Geração de Código Intermediário

- Após a fase de análises o front-end de um compilador gera uma representação intermediária, que serve de entrada para o chamado back-end.

- Cada backe-end é construído para gerar código de máquina para uma combinação processador -sistema operacional.

Geração de Código Intermediário

- Representações Intermediárias:
  - Código de Três Endereços : são instruções com no máximo três endereços, geralmente a instrução é formada por um endereço que recebe o resultado de uma operação, um operador e dois operandos, os outros endereços.

Geração de Código Intermediário

- Regras semânticas são utilizadas para gerarem código de três endereços.
- Essas regras podem escrever em uma arquivo de saída, ou pode ser feita de forma incremental atribuindo o código gerado em atributos dos símbolos da gramática.
- Vamos apresentar exemplos de DDS's que geram código para expressões aritméticas, instruções de controle de fluxo de execução e expressões booleanas.

Geração de Código
Expressões Aritméticas

Produções	Regras Semânticas
A → id = E;	escreva(id.lexema '=' E.temp
E → E_1 + T	E.temp=newTemp() escreva(E.temp '=' E_1.temp '+' T.temp)
E → T	E.temp=T.temp
T → T_1*F	T.temp= newTemp() escreva(T.temp '=' T_1.tmp '*' F.temp)
T → F	T.temp=F.temp
F → (E)	F.temp=E.temp
F → id	F.temp=id.lexema
F → num	F.temp=num.val_lex

Geração código
Expressões Aritméticas

- Na DDS anterior utilizamos as funções newTemp() e escreva(), onde:
  - newTemp() gera variáveis temporárias chamadas ti, onde i começa em um e é incrementado a cada chamada da função, gerando t1, t2, t3, ….
  - escreva() tem o objetivo de escrever no arquivo de saída, que no final deve conter todo o código gerado.

Geração de Código
Expressão Aritmética

- Todos os atributos da gramática são sintetizados, logo a DDS é S-Atribuída e a validação das regras deve ser feita no sentido bottom-up da árvore de derivação.
- Assim a string de entrada a=b+c*2 que ao ser submetida ao analisador léxico é passada como id =id + id*num para o sintático que constrói a árvore derivação e então aplica as regras semânticas resultará no código:
  - t1 = c*2
  - t2= b+t1
  - a=t2

Geração de Código
Controle de Fluxo

- As instruções que fazem o controle de fluxo de execução são instruções do tipo if, if-then e while.
- Essa instruções decidem qual bloco de instruções devem ser executadas a partir do resultado de uma expressão booleana, verdadeiro ou falso.
- Um exemplo de como essas instruções têm sua sintaxe definida está colocado na gramática a seguir.

Geração de Código
Controle de Fluxo

S → if B S

S → if B S eles S

S → while B S

S → outra S

S → SS

Geração de Código
Controle de Fluxo

- Ao gerar código de três endereços para essas três instruções é preciso observar :

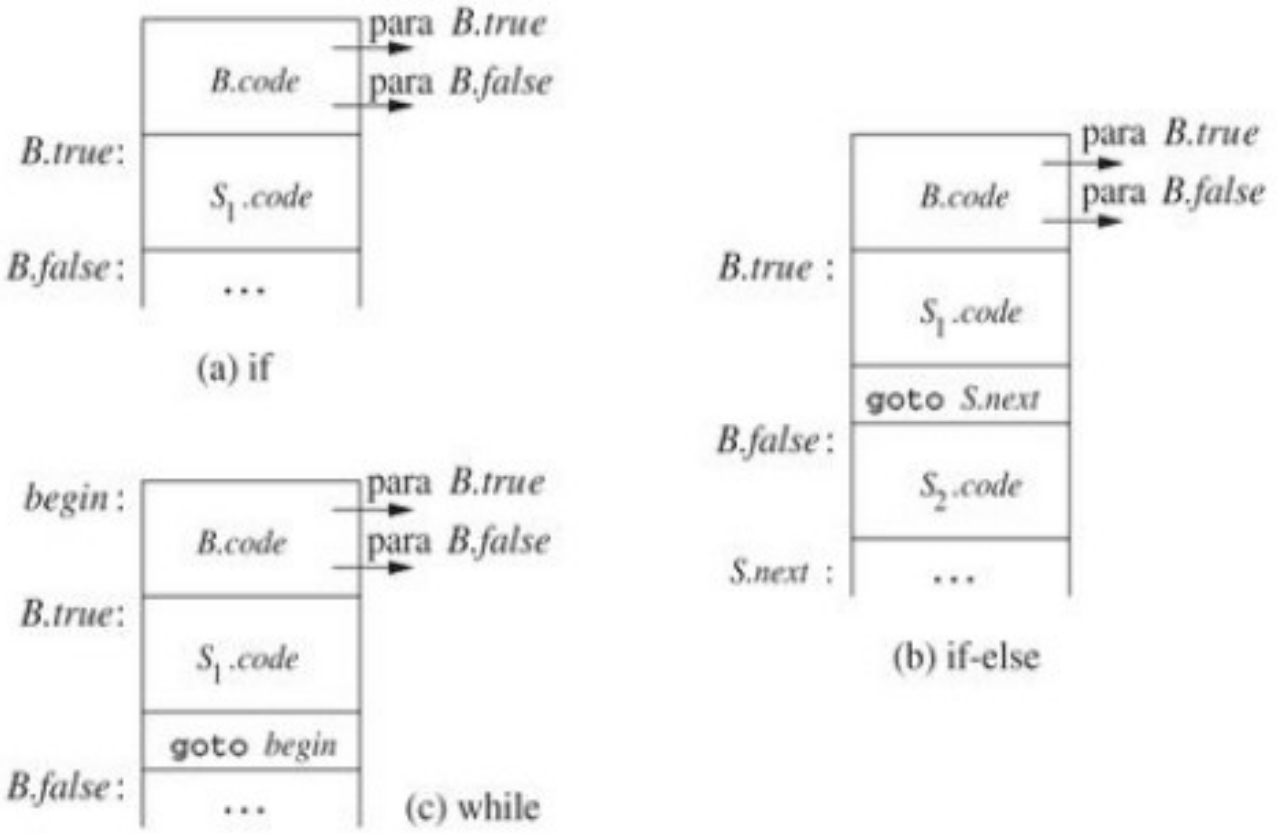

FIGURA 6.35 Código para os comandos if, if-else e while.

Geração de Código
Controle de Fluxo

- Em todas as instruções precisam ter o bloco de código das expressões boleanas antes de tudo.
  - Observe que as expressões boleanas precisam de um ponteiro (um rótulo) indicando para onde o programa deve saltar quando a expressão resultar em verdadeiro, B.true, e quando resultar em falso, B.false.
  - A DDS a seguir exemplifica uma forma de como gerar o código de três endereços para essas sentenças.

PRODUÇÃO	REGRAS SEMÂNTICAS
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \parallel label(S.next)$
$S \rightarrow$ **assign**	$S.code =$ **assign**$.code$
$S \rightarrow$ **if** (B) S_1	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \parallel label(B.true) \parallel S_1.code$
$S \rightarrow$ **if** (B) S_1 **else** S_2	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\parallel label(B.true) \parallel S_1.code$ $\parallel gen(\text{`goto'}\ S.next)$ $\parallel label(B.false) \parallel S_2.code$
$S \rightarrow$ **while** (B) S_1	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \parallel B.code$ $\parallel label(B.true) \parallel S_1.code$ $\parallel gen(\text{`goto'}\ begin)$
$S \rightarrow S_1\ S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \parallel label(S_1.next) \parallel S_2.code$

Geração de Código
Controle de Fluxo

- A função newLabel() é usada para gerar um novo rótulo l_i.
- Enquanto a função label(), gera a string l_i: no código.
- S.next indica a próxima instrução;
- S.code armazena o código gerado de forma incremental.

Geração de Código
Expressão Boleana

- As regras semânticas abaixo determinam como gerar código para expressões boleanas e complementam as instruções de controle de fluxo.

PRODUÇÃO	REGRAS SEMÂNTICAS
$B \rightarrow B_1 \parallel B_2$	$B_1.true = B.true$ $B_1.false = newlabel()$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \parallel label(B_1.false) \parallel B_2.code$
$B \rightarrow B_1 \;\&\&\; B_2$	$B_1.true = newlabel()$ $B_1.false = B.false$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \parallel label(B_1.true) \parallel B_2.code$
$B \rightarrow ! B_1$	$B_1.true = B.false$ $B_1.false = B.true$ $B.code = B_1.code$
$B \rightarrow E_1$ **rel** E_2	$B.code = E_1.code \parallel E_2.code$ $\parallel gen(\texttt{'if'}\ E_1.addr\ \textbf{rel}.op\ E_2.addr\ \texttt{'goto'}\ B.true)$ $\parallel gen(\texttt{'goto'}\ B.false)$
$B \rightarrow$ **true**	$B.code = gen(\texttt{'goto'}\ B.true)$
$B \rightarrow$ **false**	$B.code = gen(\texttt{'goto'}\ B.false)$