Introduction to Compiler

轉譯程式(Translator)

- 為一系統軟體,其功能是將輸入的原始程式轉換成另一種相對應的程式語言(如組合語言 機器語言)
- 包含下列四種
  - Assembler
  - Compiler
  - Preprocessor
  - Interpreter

2

編譯程式

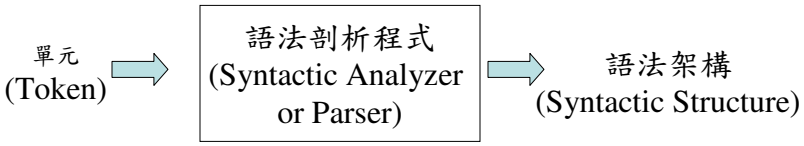
1.語彙分析階段(Lexical Analysis Phase)

- 變數、常數、區分符號、關鍵字、運算元
- 文字表(Literal Table)
- 識別字表(Identifier Table)
- 符號表(Symbol Table)

3

編譯程式

2.語法分析階段(Syntactic Analysis Phase) 或稱Parsing



4

編譯程式

3.解釋階段(Interpretation Phase)

- 在語法分析階段辨認出語句結構時,便呼叫相對應的動作常式(Action Routine)
- 動作常式的功能乃將原始程式轉換成中間形式碼,並且在識別字表中加入必要資訊
- 本階段可合併於語法剖析程式(Parser)處理

5

編譯程式

4.與機器無關之最佳化階段(Machine Independent Optimization Phase)

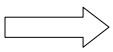
- 將由Parser 所輸出的Matrix 或Syntax Tree進行最佳化,所輸出最佳的Matrix (Reduces Syntax Tree), 以減少儲存空間及執行時間

6

與機器無關之最佳化階段

- 最佳化處理技巧
  - Elimination of Common Sub-expression

X:=a+(b*c);
Y:=D+(b*c);



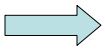
Temp:=(b*c);
X:=a+Temp;
Y:=D+Temp;

7

與機器無關之最佳化階段

- 最佳化處理技巧
  - Compile time Computation

A=(2*3)+B;

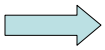


A=6+B;

8

與機器無關之最佳化階段

- 最佳化處理技巧
  - Boolean Expression Optimization

If C1 or C2 Then S1  If C1 then S1
Else If C2 then S1

9

與機器無關之最佳化階段

- 最佳化處理技巧
  - Loop Optimization

```
Bound:=10;
While (I<=Bound-2) do
  While (I<=10) do
  Begin
     X:=1;
     Y:=X+Z;
  End;
```

```
Bound:=10;
t:=Bound-2;
While (I<=t) do
 Begin
   X:=1;
   While (I<=10) do
   Begin
      Y:=X+Z;
   End;
 End;
```

10

編譯程式

5.儲存位置分配階段(Storage Assignment Phase)

- 事先預留記憶體空間以便儲存產生的目的碼
- 可併入Code Generation Phase
- 目的
  - 指定位置給予程式中使用到的變數
  - 預留位置以便儲存某些運算的中間結果
  - 設定位置給程式中所有的文字
  - 給定起始值

11

編譯程式

6.Code generation Phase

- 產生目的碼
- 進行Machine Dependence Optimization
  - 刪除多餘的Store and Load 指令
  - 儘量利用未被使用的Register
  - 以執行速度較快的指令取代執行速度較慢的指令

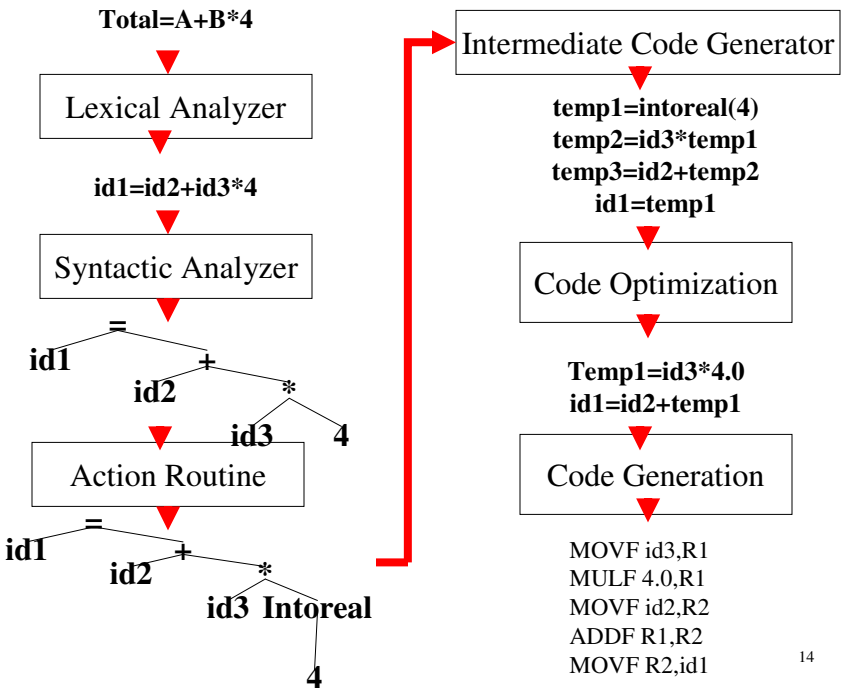
12

編譯程式

7.組合並輸出(Assembly and Output Phase)

- 解決目的碼之間的位址變數
- 輸出可重定位之目的碼(Re-locatable object code)

13



14

YACC,Compiler-Compiler

- YACC (Yet Another Compiler-Compiler)
  - 為一個在UNIX系統上用來產生其它程式語言之Parser的產生程式. 如Pascal APL C 等等
  - 使用YACC時,需提供一個語彙掃瞄程式(LEX).
  - YACC的輸入為程式語言的文法規則
  - 由YACC產生的parser 是一採用Bottom-Up的剖析技術,LALR(1).
  - 由YACC產生的parser具有很好的錯誤偵測能力.

19

Description of compiler phrases

Register Allocation Choose a register to hold each of the variables and temporary values used by the program; variables not live at the same time can share the same register.

Code Emission Replace the temporary names in each machine instruction with machine registers.

18

Description of compiler phrases

Control Flow Analysis Analyze the sequence of instructions into a *control flow graph* that shows all the possible flows of control the program might follow when it executes

Data flow Analysis Gather information about the flow of information through variables of the program; for example, *liveness analysis* calculate the places where each program variable holds a still-needed value (is live).
(static/dynamic binding)

17

Description of compiler phrases

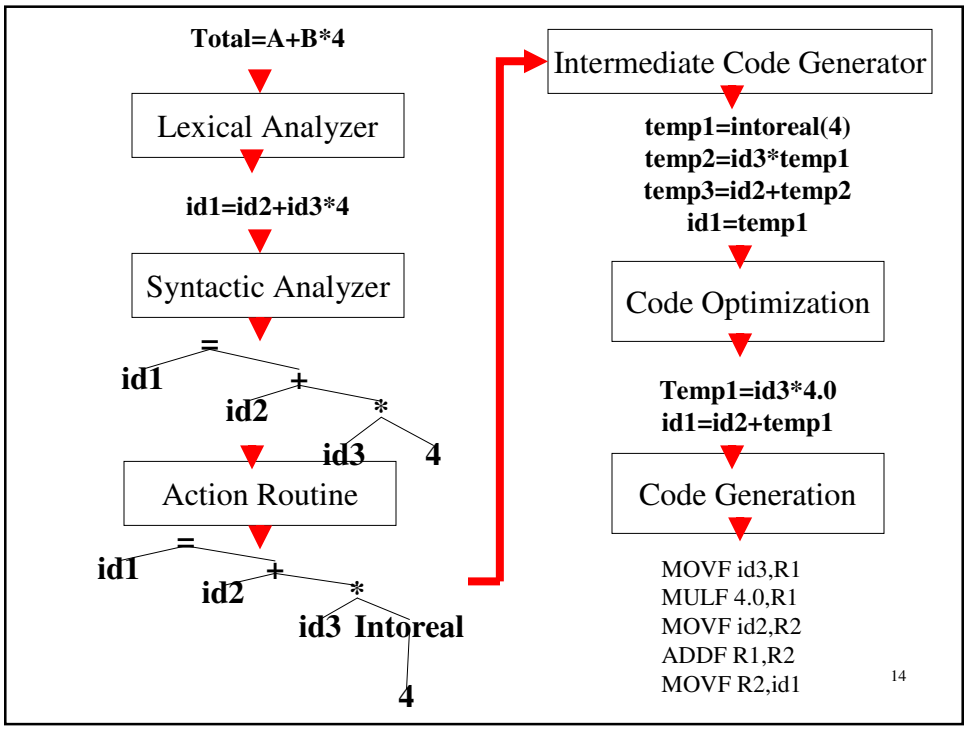
Phase	Description
Translate	Produce intermediate representation (IR) trees, a notation that is not tied to any
particular	source language or target-machine.
Canonicalize	Pick side effects out of expressions, and clean up conditional branches, for the convenience of the next phrase.
Instruction Selection	Group the IR-tree nodes into clusters that correspond to the actions of target-machine instructions.

16

Description of compiler phrases

Phase	Description
Lex	Break the source file into tokens.
Parse	Analyze the phrase structure of the program.
Semantic Actions	Build a piece of abstract syntax tree corresponding to each phrase.
Frame Layout	Place variables, function-parameters, etc. into activation records (stack frame) in a machine-dependent way.

15



14

編譯程式

7.組合並輸出(Assembly and Output Phase)

- 解決目的碼之間的位址變數
- 輸出可重定位之目的碼(Re-locatable object code)

13

編譯程式

6.Code generation Phase

- 產生目的碼
- 進行Machine Dependence Optimization
  - 刪除多餘的Store and Load 指令
  - 儘量利用未被使用的Register
  - 以執行速度較快的指令取代執行速度較慢的指令

12

編譯程式

5.儲存位置分配階段(Storage Assignment Phase)

- 事先預留記憶體空間以便儲存產生的目的碼
- 可併入Code Generation Phase
- 目的
  - 指定位置給予程式中使用到的變數
  - 預留位置以便儲存某些運算的中間結果
  - 設定位置給程式中所有的文字
  - 給定起始值

11

與機器無關之最佳化階段

- 最佳化處理技巧
  - Loop Optimization

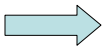
```
Bound:=10;
While (I<=Bound-2) do
  While (I<=10) do
  Begin
     X:=1;
     Y:=X+Z;
  End;
```

```
Bound:=10;
t:=Bound-2;
While (I<=t) do
 Begin
   X:=1;
   While (I<=10) do
   Begin
      Y:=X+Z;
   End;
 End;
```

10

與機器無關之最佳化階段

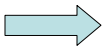
- 最佳化處理技巧
  - Boolean Expression Optimization

If C1 or C2 Then S1  If C1 then S1
Else If C2 then S1

9

與機器無關之最佳化階段

- 最佳化處理技巧
  - Compile time Computation

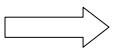
A=(2*3)+B;  A=6+B;

8

與機器無關之最佳化階段

- 最佳化處理技巧
  - Elimination of Common Sub-expression

X:=a+(b*c);
Y:=D+(b*c);



Temp:=(b*c);
X:=a+Temp;
Y:=D+Temp;

7

編譯程式

4.與機器無關之最佳化階段(Machine Independent Optimization Phase)

- 將由Parser 所輸出的Matrix 或Syntax Tree進行最佳化,所輸出最佳的Matrix (Reduces Syntax Tree), 以減少儲存空間及執行時間

6

編譯程式

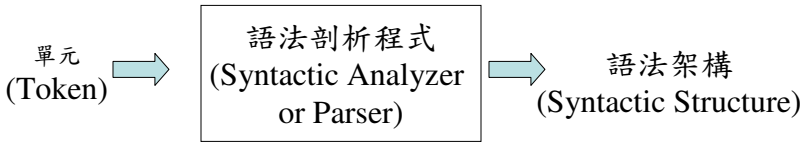
3.解釋階段(Interpretation Phase)

- 在語法分析階段辨認出語句結構時,便呼叫相對應的動作常式(Action Routine)
- 動作常式的功能乃將原始程式轉換成中間形式碼,並且在識別字表中加入必要資訊
- 本階段可合併於語法剖析程式(Parser)處理

5

編譯程式

2.語法分析階段(Syntactic Analysis Phase)
或稱Parsing



4

編譯程式

1.語彙分析階段(Lexical Analysis Phase)

- 變數、常數、區分符號、關鍵字、運算元
- 文字表(Literal Table)
- 識別字表(Identifier Table)
- 符號表(Symbol Table)

3