Universidad de Huelva

Universidad de Huelva

Grado en Ingeniería Informática

Tema 7. Generación de Código

Resumen

Autor: Alberto Fernández Merchán
Asignatura: Procesadores del Lenguaje

1. Visión General

El objetivo final de un compilar es traducir el código fuente a código ensamblador de la plataforma de destino. Muchos compiladores tienen una arquitectura dividida en *front-end* y *back-end*.

- **Front-End**: Realiza una traducción de código fuente a intermedio.
- **Back-End**: Realiza una traducción de código intermedio a ensamblador.

Esto permite generar compiladores que comparten el *front-end*.

Las instrucciones en código intermedio corresponden a las instrucciones generales de los procesadores y es independiente de la máquina. Además, no tiene en cuenta características como la arquitectura, número de registros o su funcionalidad.

Otras opciones son los lenguajes pseudointerpretados, donde el compilador genera un código binario de una máquina virtual y se interpreta en tiempo de ejecución, o los traductores **fuente-fuente**, donde se traduce un código fuente a un lenguaje de alto nivel diferente para el que ya existe un compilador eficiente.

2. Código de Tres Direcciones

- Se suele tomar como código intermedio.
- Formado por instrucciones con máximo 3 direcciones (op1, op2 y resultado).
- Se representan por cuartetos (operador, operando1, operando2 y resultado).
- Existen instrucciones con menos direcciones.
- Instrucciones parecidas a las de ensamblador.
- Las direcciones se gestionan simbólicamente.
- No existe ningún estándar. Hay libertad para definir las instrucciones de código intermedio.

2.1. Instrucciones Básicas

- Declaración de Variables
  - No generan código intermedio.
  - Crean una entrada en la tabla de símbolos.
  - Se crean variables locales y variables temporales.
- Asignación
  - Con índice
  - Con punteros
- Bucles
- Condicionales
- Saltos
- Bloques de Instrucciones
- Expresiones Aritméticas
- Condiciones
- Llamadas a Funciones

3. Código Asociado a las Instrucciones Comunes

La sintaxis de las expresiones en el lenguaje fuente es de la siguiente forma:

- Expresion → Expresion **sum** Termino | Termino
- Termino → Término **prod** Factor | Factor
- Factor → **literal** | **id** | **lpar** Expresion **rpar**

El código asociado a una expresión está formado por una lista de instrucciones en código intermedio que permite evaluarla. El resultado se almacena en una variable *temp*.
Para generar ese código se utiliza la función *getNewTemp()*.

- **Código Asociado a una Expresión Unaria**

  1. Se genera el código del operando
  2. Se crea una nueva variable temporal
  3. Se añade la instrucción de asignación: *temp = operador operador.temp*

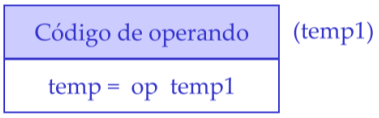

Figura 1: Código Expresión Unaria

- **Código Asociado a una Expresión Binaria**

  1. Se genera el código de los operandos
  2. Se crea una nueva variable temporal
  3. Se añade la instrucción de asignación: *temp = operando1.temp op operando2.temp*

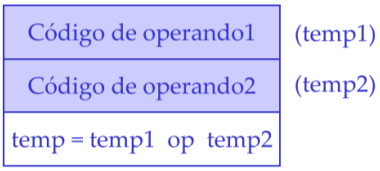

Figura 2: Código Expresión Binaria

- **Código Asociado a Asignaciones**

  1. Generar el código de la expresión
  2. Generar la instrucción *id = Expresión.tmp*

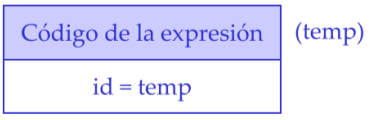

Figura 3: Código Instrucción de Asignación

- **Instrucción if-then(-else)**
  - ***Sintaxis***: InstIf → **if** Condicion **then** Instrucción1 (**else** Instruccion2)

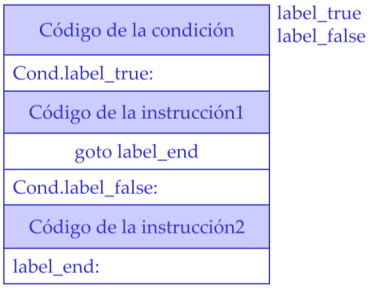

Figura 4: Código Instrucción If-Then-Else

  - **Instrucción switch-case**
    - ***Sintaxis***:
      1. InstSwitch → **switch lparen** Expresión **rparen** BloqueSwitch
      2. BloqueSwitch → **lbrace** (SentenciaCase)* (SentenciaDefault)? **rbrace**
      3. SentenciaCase → **case** Valor **colon** (Instruccion)*
      4. SentenciaDefault → **default colon** (Instruccion)*

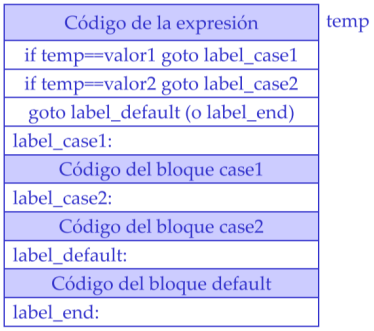

Figura 5: Código Instrucción Switch-Case

  - **Instrucción While**
    - ***Sintaxis***: InstWhile → **while** Condición **do** Instrucción

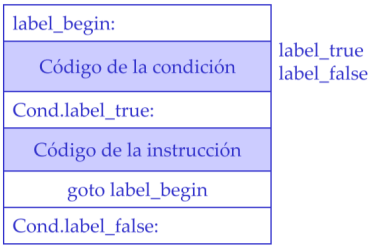

Figura 6: Código Instrucción While

- **Instrucción Do-While**
  - ***Sintaxis***: InstDoWhile → **do** Instrucción **while** Condición

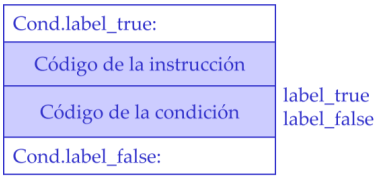

Figura 7: Código Instrucción Do-While

- **Instrucción For**
  - ***Sintaxis***: InstFor → **for lparen** Instrucción1 **semicolon** Condición **semicolon** Instrucción2 **rparen** Instrucción3

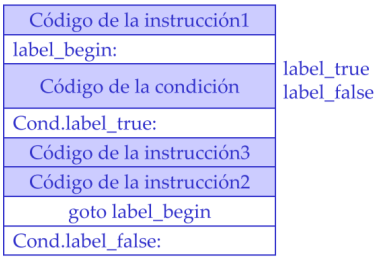

Figura 8: Código Instrucción for

- **Instrucción break**: Representa un salto incondicional (*goto*) hacia la etiqueta final de una instrucción switch, while, do-while o for. Se requiere de una **pila de etiquetas**, de manera que al entrar en una de esas instrucciones se almacena la **etiqueta final** en la pila. Al salir se desapila.
- **Instrucción continue**: Representa un salto incondicional (*goto*) hacia la etiqueta inicial de una instrucción while, do-while o for. Igual que en la anterior, requiere una pila de etiquetas donde se almacene la **etiqueta inicial** de cada instrucción.
- **Llamadas a funciones**
  - ***Sintaxis***:
    1. InstCall → **identificador lparen** ListaParámetros **rparen**
    2. ListaParámetros → (Expresión (**comma** Expresión)*)?

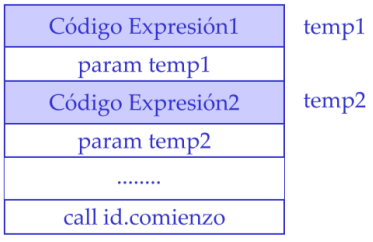

Figura 9: Código Llamada a Función

* Falta un *temp3* que indica el retorno de la función.