



Unidad VI Generación de Código Intermedio

M.C. Juan Carlos Olivares Rojas

Agenda

6.1 Lenguajes intermedios.

6.2 Notaciones.

6.2.1 Infija.

6.2.2 Postfija.

6.2.3 Prefija.

6.3 Representación de código intermedio.

6.3.1 Notación Polaca.

6.3.2 Código P.

6.3.3 Triplos.

6.3.4 Cuádruplos.



Agenda

6.4 Esquemas de generación.

6.4.1 Expresiones.

6.4.2 Declaración de variables, constantes

6.4.3 Estatuto de asignación.

6.4.4 Estatuto condicional.

6.4.5 Estatuto de ciclos

6.4.6 Arreglos.

6.4.7 Funciones.



Generador de código intermedio

- La administración de la memoria se da en esta etapa.
- Se debe considerar tanto la memoria estática como dinámica, y en esta se utilizan generalmente pilas.



Generador de código intermedio

- Los lenguajes intermedios generalmente tienen árboles de derivación más pequeños que su contraparte original.
- Se puede representar un árbol sintáctico con un Grafo Dirigido Acíclico (GDA).
- La notación postfija es una manera linealizada de representar un árbol sintáctico.



Generador de código intermedio

- $a := b^* - c + b^* - c$
- $abc \ -^* bc \ -^* +=$

- $x := y \text{ op } z$

- $x + y^* z$
- $t1 := y^* z$
- $t2 := x + t1$



6.1 Lenguajes intermedios

- Los lenguajes intermedios nos sirven para representar la producción final de nuestro lenguaje fuente.
- Existen muchos lenguajes intermedios, la mayoría de ellos son una representación más simplificada del código original para facilitar la traducción hacia el código final.



Lenguajes intermedios

- Otros lenguajes intermedios sirven de base o como representación parcial de otros procesos.
- Por ejemplo al compilar un programa en C en Windows o DOS, se produce un código objeto con extensión .obj para que posteriormente el enlazador cree finalmente el código ejecutable .exe



Lenguajes intermedios

- En sistemas basados en Unix, también ocurre algo similar generándose un archivo .o y el ejecutable a.out
- Otros lenguajes intermedios famosos son los generados para la máquina virtual de Java el bytecode; y para la máquina virtual de .NET el MSIL para luego ejecutarse en tiempo de ejecución JIT (Just in Time)



Lenguajes intermedios

- Otros lenguajes intermedios se utilizan en sistemas distribuidos como RPC, CORBA y su IDL, etc.
- En este caso estos lenguajes intermedios se encargan de enmascarar toda la heterogeneidad de las comunicaciones distribuidas en una computadora



6.2 Notaciones

- Las notaciones sirven de base para expresar sentencias bien definidas.
- El uso más extendido de las notaciones sirve para expresar operaciones aritméticas.
- Las expresiones aritméticas se pueden expresar de tres formas distintas: infija, prefija y postfija.



Notaciones

- La diversidad de notaciones corresponde en que para algunos casos es más sencillo un tipo de notación.
- Las notaciones también dependen de cómo se recorrerá el árbol sintáctico, el cual puede ser en inorden, preorden o postorden; teniendo una relación de uno a uno con la notación de los operadores.



6.2.1 Infija

- La notación infija es la más utilizada por los humanos por que es la más comprensible ya que ponen el operador entre los dos operandos. Por ejemplo $a+b-5$.
- No existe una estructura simple para representar este tipo de notación en la computadora por esta razón se utilizan otras notaciones.



6.2.2 Postfija

- La notación postfija pone el operador al final de los dos operandos, por lo que la expresión queda: $ab+5-$
- La notación posftfija utiliza una estructura del tipo LIFO (Last In First Out) pila, la cual es la más utilizada para la implementación.



6.2.3 Prefija

- La notación prefija pone el operador primero que los dos operandos, por lo que la expresión anterior queda: $+ab-5$. Esto se representa con una estructura del tipo FIFO (First In First Out) o cola.
- Las estructuras FIFO son ampliamente utilizadas pero tienen problemas con el anidamiento aritmético.



6.3 Representación de código intermedio

- Existen maneras formales para representar código intermedio.
- Estas notaciones simplifican la traducción de nuestro código fuente a nuestro código objeto ya que ahorran y acotan símbolos de la tabla de símbolos



6.3.1 Notación Polaca



6.3.2 Código P

- El código P hace referencia a máquinas que utilizan o se auxilian de pilas para generar código objeto.
- En muchos caso la P se asociado a código portable el cual garantiza que el código compilado en una máquina se pueda ejecutar en otras.



Código P

- Para garantizar la portabilidad del código se necesita que el lenguaje este estandarizado por algún instituto y que dicho código no tenga extensiones particulares.
- También se recomienda la no utilización de características especiales exclusivas de alguna arquitectura de computadoras en particular.



6.3.3 Triplos

- Las proposiciones de tres direcciones se parece mucho al ensamblador, el cual es un lenguaje intermedio más entendible para la máquina.
- Las estructuras de control (if, switch, while, do-while, for) son realmente etiquetas goto disfrazadas.



Triplos

- El problema de utilizar cuádruplos radica en que se tienen que colocar los valores temporales en la tabla de símbolo.
- Con una estructura de tres campos se pueden omitir los valores temporales, dicha estructura recibe el nombre de triples y tiene los siguientes campos: op, arg1 y arg2



Triplos

- Generalmente el código que generan los triples recibe el nombre de código de dos direcciones, aunque en ocasiones puede variar.
- Cuando se utilizan triples se ocupan punteros a la misma estructura de los triples.
- * b t1 t2 //cuádruplos
- * b (0) //triple



Triplos

- Se debe tener en cuenta el proceso de asignación, de declaración, expresiones booleanas.
- Las expresiones lógicas también pueden pasarse a código de tres direcciones, utilizando para ello expresiones en corto circuito.



Triplos

- La evaluación de expresiones en corto circuito implica que se evalúan condiciones revisando valores anteriores; por ejemplo, para el operador AND con una condición que se detecte como falsa toda la expresión es falsa, en el caso del operador OR si se encuentra una condición verdadera todo será verdadera



Triplos

- La notación de tres direcciones es una forma abstracta de código intermedio.
- Esta notación se puede implementar como registros con campos para el operador y operandos.



6.4.3.1 Intérpretes

- Los intérpretes generalmente utilizan este triplos para generar el código intermedio para ejecutarse una vez considerado la instrucción como válido.
- En este sentido, un compilador es más difícil de implementar ya que tendrá que mantener todas las estructuras generadas que en muchas ocasiones serán cuadrúplos.



6.3.4 Cuádruplos

- Es una estructura tipo registro con cuatros campos que se llaman: op, arg1, arg2 y resultado. OP tiene un código intermedio.
- Los operadores unarios como $x := -y$ no utilizan arg2. Generalmente arg1, arg2 y resultado son valores de tipo puntero y apuntan a una entrada en la tabla de símbolos.



6.4 Esquemas de generación

- Los esquemas de generación son las estrategias o acciones que se deberán realizarse y tomarse en cuenta en el momento de generar código intermedio.
- Los esquemas de generación dependen de cada lenguaje. Tomaremos algunos esquemas de generación del lenguaje C.



6.4.1 Expresiones

- Para generar expresiones estas deben representarse de manera más simple y más literal para que su conversión sea más rápida.
- Por ejemplo la traducción de operaciones aritméticas debe especificarse una por una, de tal forma que una expresión sea lo más mínimo posible.



6.4.2 Declaración de variables, constantes

- Las declaraciones de variables y constantes deben separarse de tal manera que queden las expresiones una por una de manera simple.
- Por ejemplo `int a,b,c;` se descompone a `int a;`
`int b;` `intc;` respectivamente.



6.4.3 Estatuto de asignación

- Las operaciones de asignación deben quedar expresadas por una expresión sencilla, si está es compleja se debe reducir hasta quedar un operador sencillo.
- Por ejemplo: $x = a + b/5$; debe quedar de la forma $y = b/5$; $z = a + y$; $x = z$.



6.4.4 Estatuto condicional

- Las condiciones deben expresarse de manera lo más sencilla posible de tal forma que puedan evaluarse en cortocircuito. Por ejemplo una instrucción como: `if (a == b && f!=5 && f%3==0)` se evalúa primero `x = (a==b && f!=5)` y `y = x && f%3==0`; `if (y)`
- Las instrucciones de decisión compleja como `switch` se reducen a una versión complejas de `if`'s



6.4.5 Estatuto de ciclos

- Los ciclos se descomponen en un ciclo genérico, por lo que ciclos while, for y do-while tienen la misma representación interna. En el caso de C, todo queda en forma de while.
- Las condiciones lógicas también pueden ser evaluadas en cortocircuito y reducidas.



6.4.6 Arreglos

- Los arreglos se descomponen en estructuras básicas de manejo de manera simple, así por ejemplo: `char *a="Hola";` se reduce a:
`a[0]='H'; a[1]='o'; a[2]='l'; a[3]='a'; a[4]='\0';`



6.4.7 Funciones

- Las funciones pueden reducir a en línea, lo que se hace es expandir el código original de la función.
- Las funciones se descomponen simplificando los parámetros de manera individual al igual que el valor de retorno.



¿Preguntas?

