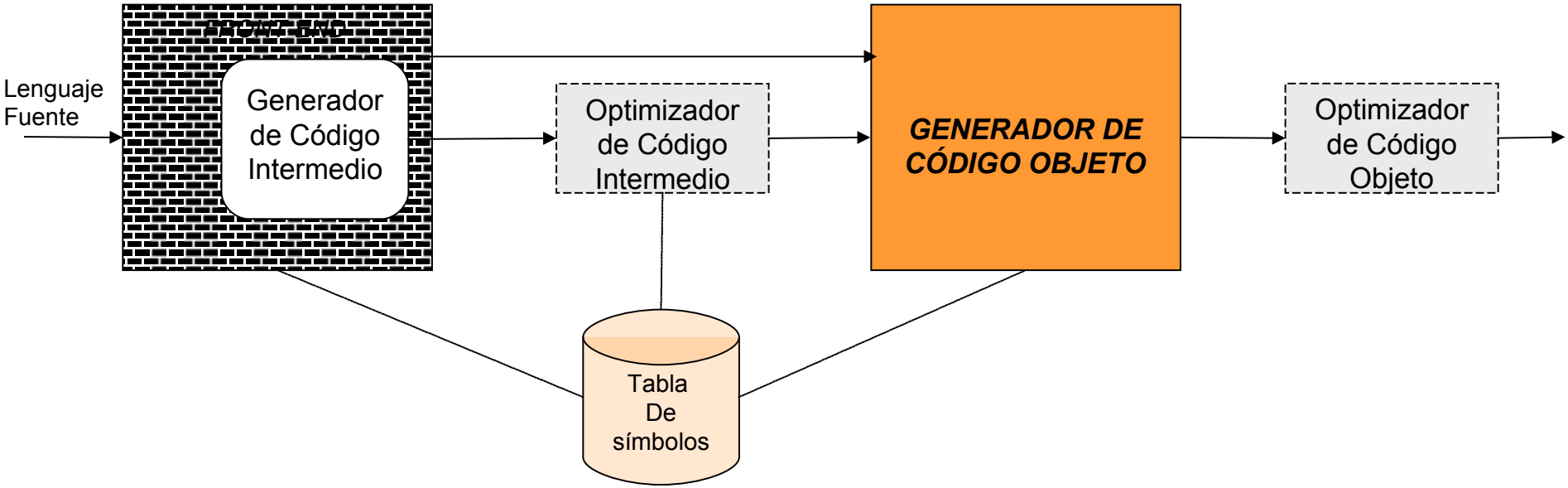
Tema 11 | Generación de Código Objeto

Contenidos

24/09/13

11.1 Aspectos a tener en cuenta en el diseño (1)



Objetivo: Tomar una entrada dada mediante un código intermedio y obtener el equivalente en código objeto.

11.1 Aspectos a tener en cuenta en el diseño (2)

Tópicos a abordar en la fase de generación de código objeto:

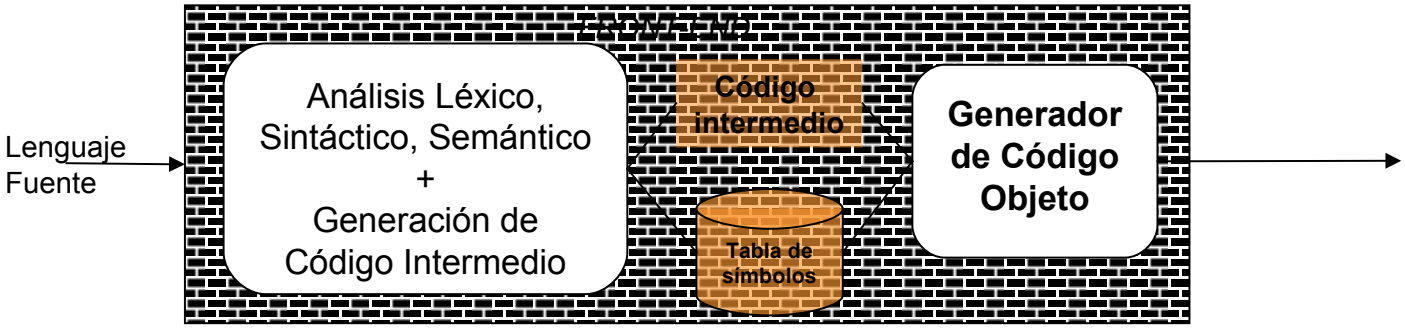
- **Establecer correspondencia** entre *nombres de variables* y *constantes* con direcciones de memoria (Localización de los registros). Establecer un mapa de memoria.
- **Orden de Evaluación**: Optimización del lenguaje objeto (problema NP). Estimación del costo de instrucción.
- **Esquema de Traducción**: Conocido previamente antes de la construcción del traductor.

En resumen, antes de abordar la generación de código objeto es necesario establecer:

- Un mapa de memoria.
- Un código intermedio.

Ello conlleva **completar** la tabla de símbolos y **generar el esquema de traducción** previamente establecido.

11.1 Aspectos a tener en cuenta en el diseño (3)



Entrada al Generador de Código:

- Código intermedio + Tabla de símbolos.

Programa objeto:

- Lenguaje objeto absoluto (posiciones fijas de memoria; ejecución inmediata).
- Lenguaje máquina relocalizable (compilación por separado, cargador-enlazador).
- Lenguaje ensamblador (facilita la generación de código a través del macroensamblador, facilita la lectura del código generado).

11.1 Aspectos a tener en cuenta en el diseño (4)

Administración de la Memoria:

- **Problema**: Se dispone de direcciones simbólicas en la tabla de símbolos (constantes, variables, etiquetas, ...) → Se deben transformar en **direcciones en memoria**.

- Elección del Mapa de Memoria (depende del tipo de lenguaje):
  - Estático.
  - Pila.
  - Montón.

Selección de Instrucciones:

- Desde la eficiencia, las instrucciones que operan con registros son más eficientes que las que operan con memoria.
- Gestión óptima de los registros (problema NP completo): Seleccionar qué variables residirán en registros y en qué registro debe residir cada variable.

11.1 Aspectos a tener en cuenta en el diseño (5)

Elección del Orden de Evaluación:

- La elección del orden en la que se realizan los cálculos depende del número de registros necesitados. Algunos ordenamientos en los cálculos necesitan menos registros que otros para guardar resultados intermedios.

Enfoque en la Generación de Código:

- Generar código correcto.
- El enfoque que debe darse al diseño de la generación de código debe:
  - Facilitar su aplicación, la comprobación y el mantenimiento.
  - Usar de la forma más óptima los recursos de la máquina.

11.1 Aspectos a tener en cuenta en el diseño (6)

Máquina Objeto:

- Los elementos que determinan la máquina objeto son:
  - Nº de registros y su denominación.
  - Tipo de instrucciones (de dos y de tres direcciones).
  - Tamaño de palabra.
  - Direccionamiento a nivel de byte y/o palabra.
  - Costo de cada tipo de direccionamiento (absoluto, registro, indexado, registro indirecto, indexado indirecto).
- Costo de las instrucciones, es decir, compromiso entre uso de registros o memoria y el tamaño de las instrucciones. Ejemplos de costos:

[costo 1] `MOV R0,R1` copia el contenido del registro `R0` en el registro `R1`. Esta instrucción tiene costo **1** porque ocupa sólo una palabra de memoria.

[costo 2] `MOV R5,M` copia el contenido de `R5` en la posición de memoria `M`. Esta instrucción tiene costo **2** porque la dirección de la posición de memoria `M` está en la palabra que sigue a la instrucción.

[costo 3] `SUB 4(R0),*12(R1)` tiene costo **3** porque las constantes `4` y `12` se almacenan en la siguiente palabra tras la instrucción.

11.2 Grafo de Flujo. Bloque Básico (1)

Bloque Básico: Secuencia de proposiciones consecutivas tal que el flujo de control entra por el principio y sale al final sin que existan saltos posibles entre proposiciones.

Proposición Líder: Es la proposición inicial de todo bloque básico.

Algoritmo para particionar un programa en bloques básicos

Entrada: Secuencia de proposiciones de código intermedio.

Salida: Lista de bloques básicos.

Método:

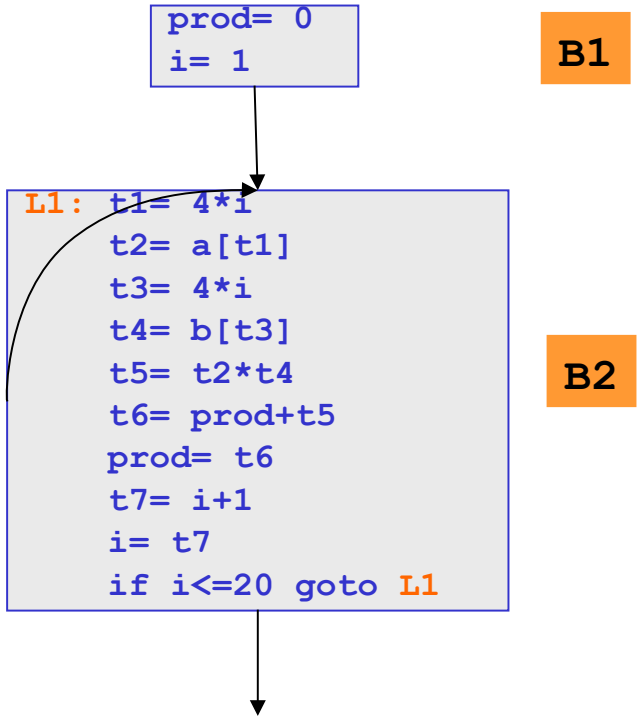
1. Obtención del conjunto de **proposiciones líderes**.
   - 1.1 La primera proposición es proposición líder.
   - 1.2 Cualquier proposición destino de un salto es proposición líder.
   - 1.3 Toda proposición justo después de un salto es proposición líder.
2. Cada **bloque básico** está comprendido entre su **proposición líder** y la **proposición anterior** a la siguiente proposición líder.

11.2 Grafo de Flujo. Bloque Básico (2)

Ejemplo 11.1: Determinación de bloques básicos a partir de proposiciones de tres direcciones.

```
    prod= 0 ;
    i= 1;
    do {
       prod= prod + a[i]+b[j];
       i= i+1 ;
    } while (i<=20) ;
```

```
     prod= 0
     i= 1
L1:  t1= 4*i
     t2= a[t1]
     t3= 4*i
     t4= b[t3]
     t5= t2*t4
     t6= prod+t5
     prod= t6
     t7= i+1
     i= t7
     if i<=20 goto L1
```



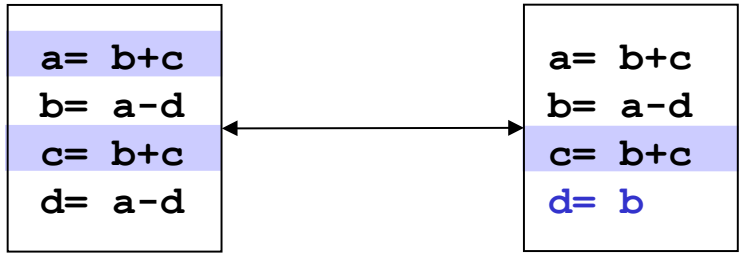
11.2 Grafo de Flujo. Bloque Básico (3)

Acción de un Bloque Básico: Calcula un conjunto de expresiones cuyos resultados aparecen sobre los nombres activos al salir del bloque.

Bloques Básicos Equivalentes: Dos bloques básicos son equivalentes si ambos evalúan las mismas expresiones.

Transformaciones sobre Bloques Básicos que Conservan la Estructura

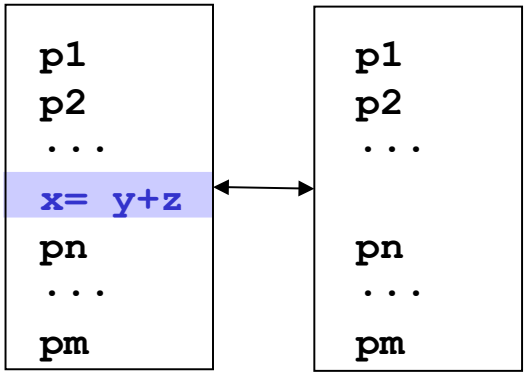
- Eliminación de las sub-expresiones comunes:



11.2 Grafo de Flujo. Bloque Básico (4)

Transformaciones sobre Bloques Básicos que Conservan la Estructura (cont.)

- Eliminación del código inactivo:



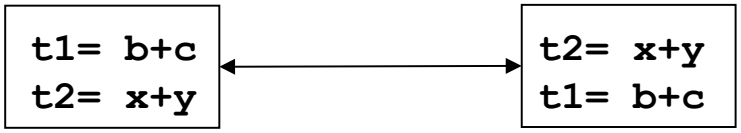
Si en `pn···pm` y en los siguientes bloques básicos no se utiliza `x`, entonces puede eliminarse la evaluación de `x` porque no afecta para la evaluación del bloque básico

11.2 Grafo de Flujo. Bloque Básico (5)

Transformaciones sobre Bloques Básicos que Conservan la Estructura (cont.)

- Renombrar las variables temporales: Dada una proposición `t= b+c`, donde `t` es una variable temporal. Si se cambia esta proposición por `u= b+c`, donde `u` es una variable temporal nueva, así como todos los usos de este ejemplo de `t` por `u`, entonces no se cambia el valor del bloque básico. A ese nuevo bloque se le denomina ***bloque en forma normal***.

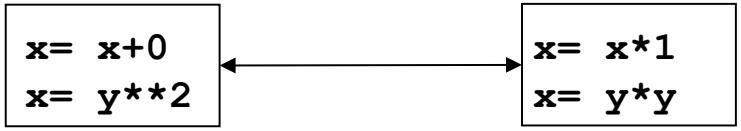
- Intercambio de proposiciones: Dadas dos proposiciones cuyos valores son totalmente independientes, siempre es posible intercambiarlas sin que ello afecte al valor del bloque.



Un bloque básico en forma normal permite intercambios de proposiciones

11.2 Grafo de Flujo. Bloque Básico (6)

Transformaciones Algebraicas:



Grafos de Flujo: Es un grafo dirigido con los nodos etiquetados por bloques básicos donde los arcos expresan el flujo de control de la ejecución de los nodos.

Nodo Inicial: Nodo etiquetado con el bloque básico cuya proposición líder es la proposición inicial del programa.

Desde un bloque básico sale una línea de flujo hacia la proposición líder de un bloque básico, que puede ser el mismo.

Todo bloque básico, excepto el inicial, posee una línea de flujo de entrada hacia su proposición líder.

Un bloque **B1** se dice que es predecesor de un bloque **B2** si de la última proposición de **B1** aparece un salto incondicional o condicional a la proposición líder de **B2** o, por el contrario, **B2** es el bloque inmediatamente siguiente al bloque **B1** (**B2** es sucesor de **B1**).

11.2 Grafo de Flujo. Bloque Básico (7)

Estudio del uso a posteriori de los nombres de variables: Poder establecer una política de conservación en memoria de determinados nombres según sean o no utilizados a posteriori.

Sean `i` y `j` dos proposiciones tal que existe un flujo de control que lleva desde `i` hasta `j` y en `i` se evalúa `x`. Se dice que `j` usa el valor de `x` calculado en `i`.

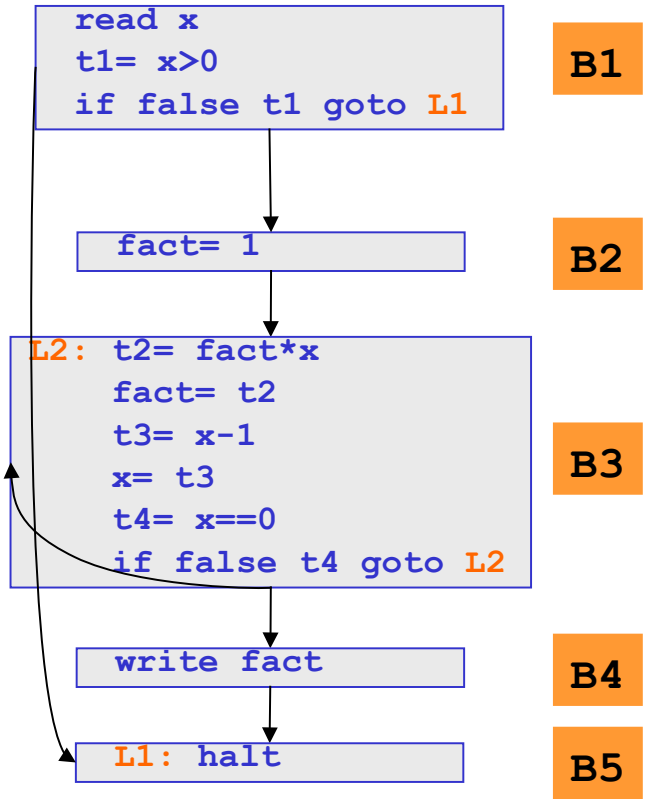
Simplificación del número de temporales utilizados: Cada vez que se requiere un temporal, comprobar si alguno de los ya asignados deja de tener uso.

11.2 Grafo de Flujo. Bloque Básico (8)

Ejemplo 11.2: Determinación de bloques básicos a partir de proposiciones de tres direcciones.

```
    scanf ("%d",&x);
    if (x>0)
    {
       fact= 1;
       do
       {
          fact= fact*x;
          x-- ;
       } while (x==0)
       printf ("%d",fact);
    }
```

```
     read x
     t1= x>0
     if false t1 goto L1
     fact= 1
L2:  t2= fact*x
     fact= t2
     t3= x-1
     x= t3
     t4= x==0
     if false t4 goto L2
     write fact
L1:  halt
```



11.3 Generador de Código Simple (1)

Estrategia: Mantener los valores en los registros siempre que sea posible. En otro caso, se almacenan en memoria en estas situaciones:

- Si el registro es necesario para otro cálculo.
- Si está justo antes de un salto o de un procedimiento.

Descriptor de Registros: Conoce en cada instante lo que almacena cada registro, así como las direcciones para sus nombres.

Será consultado siempre que se necesite un nuevo registro. Al comienzo, todos los registros estarán vacíos.

Descriptor de Direcciones: Conoce la posición de memoria donde puede encontrar el valor en curso de ejecución. La dirección puede ser:

- Un registro.
- Una dirección de la pila.
- Una dirección de memoria.

Toda esta información suele estar incluida en la tabla de símbolos.

11.3 Generador de Código Simple (2)

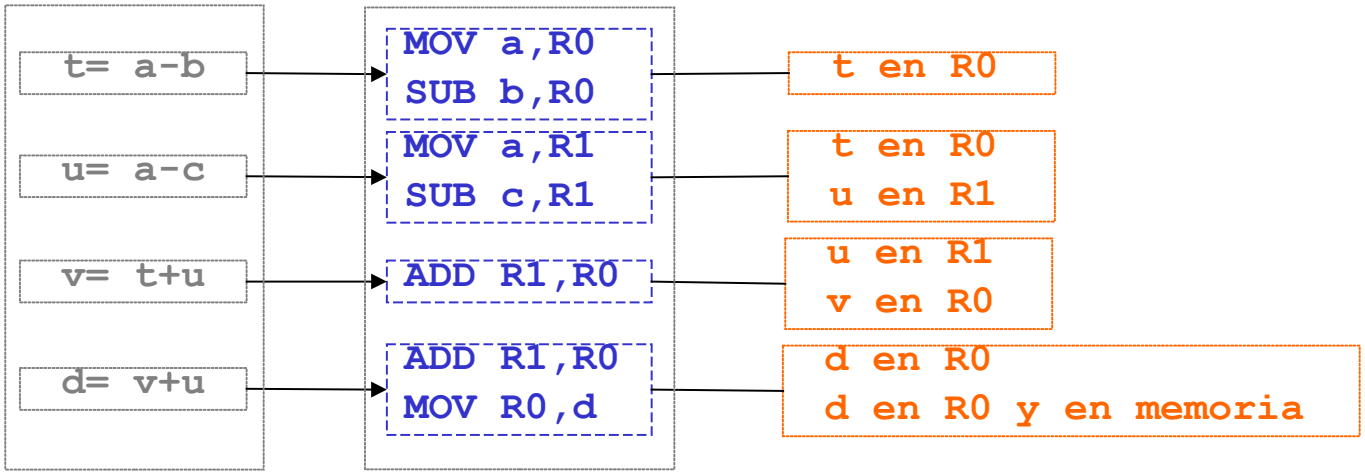
Algoritmo para la Generación de Código Simple

Para cada proposición de tres direcciones del tipo: `X = Y op Z`, hacer:

- Se consulta al **Descriptor de Registros** y se obtiene el registro para guardar el resultado.
- Se consula al **Descriptor de Direcciones** para obtener la posición del operando en curso.
- Se genera **la operación** con la posición de los **operandos** y el **resultado** en el **registro** obtenido en el paso 1.
- Si los **valores en curso** no tienen **uso a posteriori**, se guarda el resultado en **memoria**. En otro caso, se deja como están.

11.3 Generador de Código Simple (2)

Ejemplo 11.3: Generación de Código Simple de varias proposiciones de tres direcciones.



11.4 Estrategia de Asignación de Registros (1)

Gestión de registros empleando una pila de registros libres (se extrae de la pila cuando un registro es requerido).

En general, resulta más óptimo dejar en los registro la dirección base y los punteros a la pila.

Distribución de los Registros: Mantiene en los registros los nombres más usados dentro de un bloque. Al finalizar el bloque se salvan los registros a memoria → **Distribución global**.

Un criterio para determinar los nombres más usados sería tomar el número de referencias de las variables en cada bloque básico, estimando el ahorro que supone mantener las variables en memoria o en los registros.

11.4 Estrategia de Asignación de Registros (2)

Estrategia de asignación de registros mediante la coloración de grafos: En este caso, se distinguen entre registros físicos y registros simbólicos.

Proceso:

Se construye un grafo entre los registros simbólicos, tal que en cada nodo hay un registro simbólico y un arco de otro registro simbólico que necesite de éste para su evaluación. A este grafo se le llama **grafo de inferencia**.

Cada uno de los grafos de inferencia que ligan a un conjunto de registros simbólicos es **coloreado**, es decir, asignado a un determinado registro físico. El número de colores lo determina un valor k que indica el número de registros asignables.

11.5 Representación de los Bloques Básicos por GDA (1)

Un Grafo Dirigido Acíclico (GDA) es una estructura de datos que indica cómo debe ser evaluado un bloque básico (muy similar al árbol sintáctico).

Los nodos hoja representan valores iniciados y los nodos interiores valores calculados. Las etiquetas de los arcos hacia nodos hoja indican nombres o constantes y hacia nodos interiores indican operadores o nombres.

Desde el GDA de un bloque básico es posible determinar:

- Las sub-expresiones comunes.
- Los nombres que se utilizan dentro pero se evalúan fuera de él.
- Qué proposición del bloque utiliza el valor calculado fuera del mismo.

11.5.1 Construcción de los GDAs (1)

Algoritmo para la Construcción de los GDAs

Se parte de la existencia de una función **`nodo(ident)`** que asocia un identificador a un nodo.

Casos:

(i) `X= Y op Z`
(ii) `X= op Y`
(iii) `X= Y`

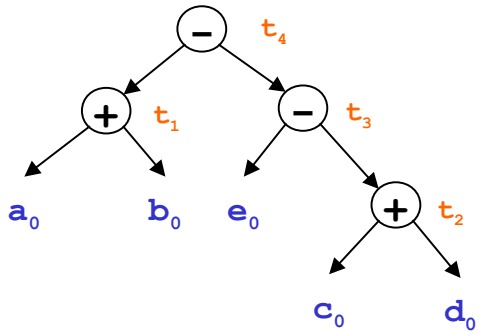
Método:

1. Si **`nodo(Y)`** es indefinido, entonces crear hoja etiquetada por **`Y`**.
2. En (i) buscar si existe un nodo etiquetado con **`op`** y con los hijos a la izquierda **`Y`** y a la derecha **`Z`**. En otro caso, crearlos.
   En (ii) buscar si hay un nodo etiquetado con **`op`** con un solo hijo.
   En (iii) habrá un nodo etiquetado con **`Y`**.
3. Borrar **`X`** de la lista de identificadores asociados al **`nodo(X)`** y añadir **`X`** a la lista de nodos **`n`** del paso (2) y **`nodo(X)=n`**.

11.5.1 Construcción de los GDAs (2)

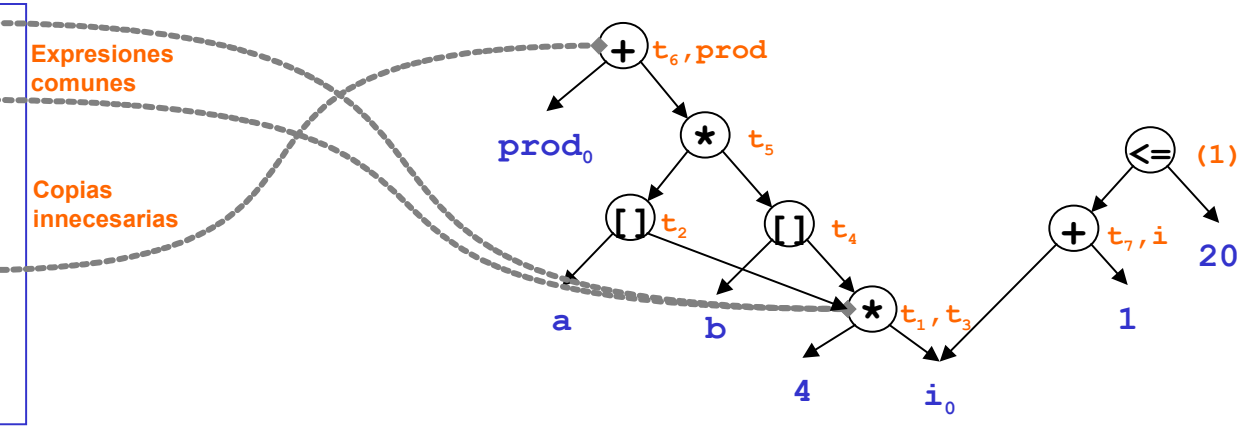
Ejemplo 11.4: Construcción de GDA.

```
t1= a+b
t2= c+d
t3= e-t2
t4= t1-t3
```



Ejemplo 11.5: GDA para el bloque básico B2 del ejemplo 11.1

```
L1: t1= 4*i
    t2= a[t1]
    t3= 4*i
    t4= b[t3]
    t5= t2*t4
    t6= prod+t5
    prod= t6
    t7= i+1
    i= t7
    if i<=20 goto L1
```

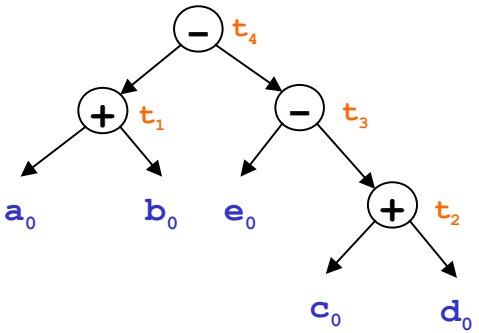


11.5.2 Ventajas en el uso de GDAs (1)

- Detección automática de las sub-expresiones comunes.
- Quedan determinados los valores de los identificadores utilizados en el bloque. (Los que han sido creados como hojas en el paso 1 del algoritmo).
- Determina las proposiciones que calculan valores que podrían ser usados fuera del bloque. (Aquellas proposiciones cuyo nodo ha sido construido en el paso 2 del algoritmo).
- Siempre que es posible, simplifica la asignación por copia.

La evaluación del GDA se puede hacer en cualquier **ordenamiento topológico**, permitiendo cambiar el orden de evaluación de las proposiciones con el fin de generar código óptimo.

Ejemplo 11.6: Cambio de orden de evaluación en proposiciones




```
t1= a+b
t2= c+d
t3= e-t2
t4= t1-t3
```

```
MOV a,R0
ADD b,R0
MOV c,R1
ADD d,R1
MOV R0,t1
MOV e,R0
SUB R1,R0
MOV t1,R1
SUB R0,R1
MOV R1,t4
```

```
t2= c+d
t3= e-t2
t1= a+b
t4= t1-t3
```

```
MOV c,R0
ADD d,R0
MOV e,R1
SUB R0,R1
MOV a,R0
ADD b,R0
SUB R1,R0
MOV R0,t
```

11.5.2 Ventajas en el uso de GDAs (2)

Proceso de Ordenación Óptimo de las Proposiciones desde un GDA: Para una máquina abstracta concreta se propone un método de ordenación en dos fases:

FASE 1 Se recorre el árbol (GDA) en orden ascendente y se etiqueta cada nodo con el número mínimo de registros que se requiere para su evaluación sin almacenamientos en memoria de los resultados intermedios..

FASE 2 Se recorre el árbol (GDA) en orden decreciente con respecto a las etiquetas obtenidas en la fase anterior, generándose el código en dicho orden.

Algoritmo de Etiquetado de Nodos de un GDA

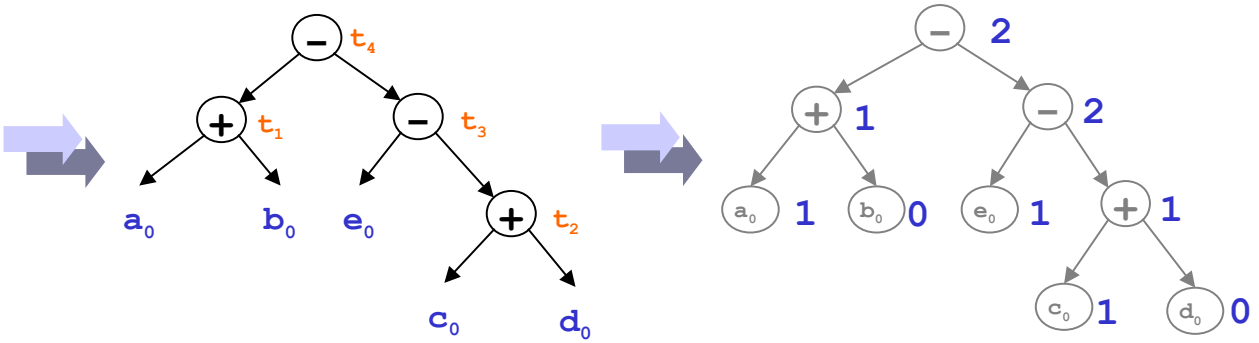
$$\text{etiqueta}(n) = \begin{cases} \text{MAX}(l_1, l_2), & \text{si } l_1 \neq l_2 \\ l_1 + 1, & \text{si } l_1 = l_2 \end{cases}$$

```
if (n es una hoja) then
   if (n es el hijo más a la izquierda) then
      etiqueta(n)= 1;
   else
      etiqueta(n)= 0;
else begin
   sean n1, n2, ..., nk los hijos de n ordenados por etiqueta
   donde etiqueta(n1)>= etiqueta(n2)>= ... >= etiqueta(nk);
   etiqueta(n)= MAX (etiqueta(ni)+i-1)
end
```

11.5.2 Ventajas en el uso de GDAs (3)

Ejemplo 11.7: Árbol etiquetado para el GDA del ejemplo anterior. En azul se muestra el número de registros necesarios para la evaluación de la operación según sus operandos.

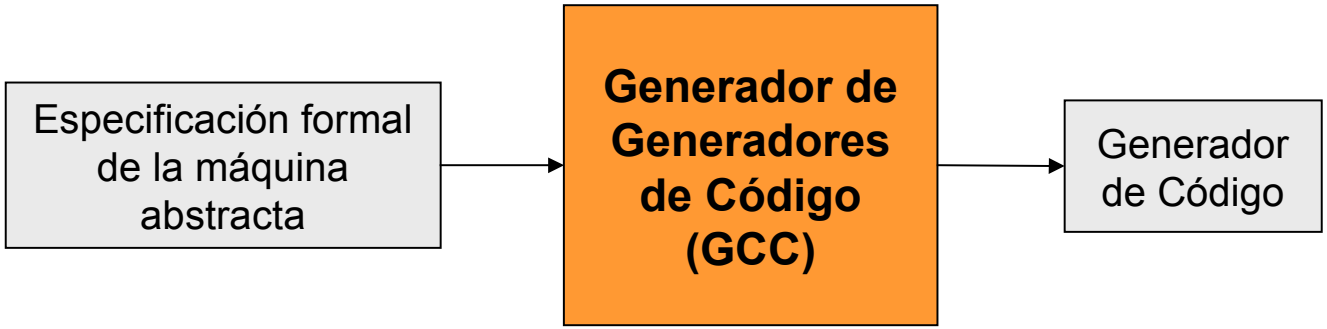
```
t1= a+b
t2= c+d
t3= e-t2
t4= t1-t3
```



Un recorrido en orden ascendente visita los nodos en este orden: `[a b t1 e c d t2 t3 t4]`. El nodo `a` se etiqueta con `1` por ser hoja izquierda. El nodo `b` se etiqueta con `0` por ser hoja derecha. El nodo t_1 se etiqueta con `1` porque son distintos y la etiqueta máxima de sus hijos es `1`. En resumen, para evaluar t_4 se requieren `2` registros.

11.6 Generadores de Generadores de Código (1)

Uno de los problemas a resolver en la fase de generación de código consiste en elegir adecuadamente el subconjunto de instrucción máquina que definen nuestra máquina objeto. El uso de GGC nos ayuda a resolver este problema.



Técnica de implantación de los GCCs: Se basan en sistemas de reescritura aplicados sobre los árboles que representan los GDAs de cada bloque básico.

El sistema de reescritura queda definido mediante el conjunto de reglas de reescritura que representan la especificación formal de la máquina abstracta.

11.6 Generadores de Generadores de Código (2)

Formato de las reglas de reescritura:

```
Sustitución ← Plantilla      { Acción }
```

En el campo **Sustitución** se expresa el nodo por el que se sustituye el árbol plantilla.

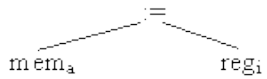
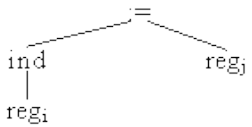
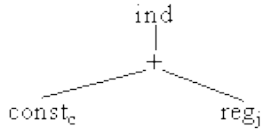
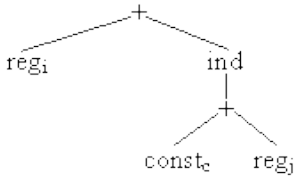
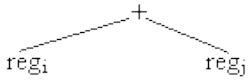
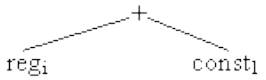
En el campo **Plantilla** se expresa el árbol patrón que hay que buscar en el GDA susceptible de ser traducido.

En **Acción**, de igual modo que en LEX y YACC, se introducen las acciones que consisten en emitir el código máquina correspondiente al patrón en la plantilla.

11.6 Generadores de Generadores de Código (3)

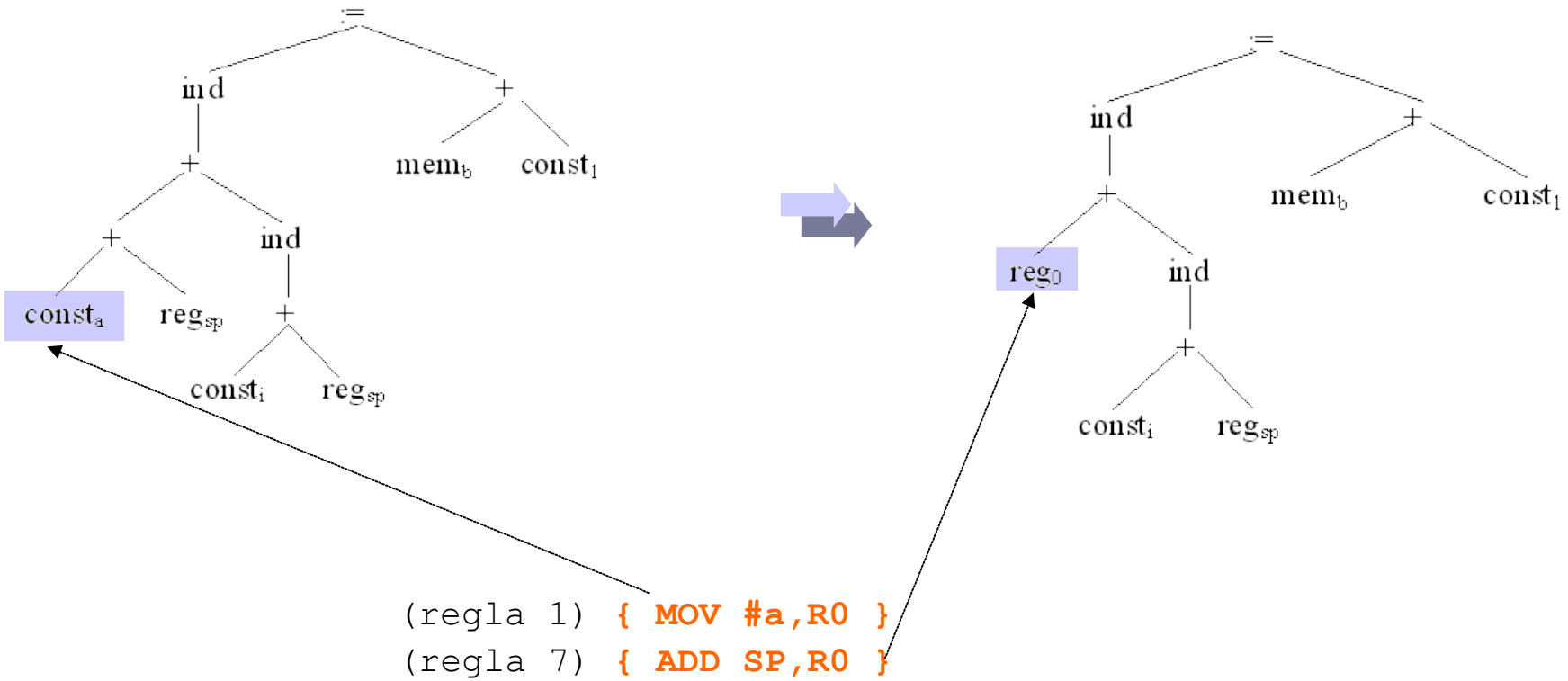
Ejemplo 11.8: Descripción formal de una máquina abstracta.

`ind` indica dirección de memoria.

Nº	Sustitución	Plantilla	Acción
1	reg_i	$const_c$	{MOV #c,Ri}
2	reg_i	mem_a	{MOV a,Ri}
3	mem		{MOV Ri,a}
4	mem		{MOV Rj,*Ri}
5	reg_i		{MOV c(Rj),Ri}
6	reg_i		{ADD c(Rj),Ri}
7	reg_i		{ADD Rj,Ri}
8	reg_i		{INC Ri}

11.6 Generadores de Generadores de Código (4)

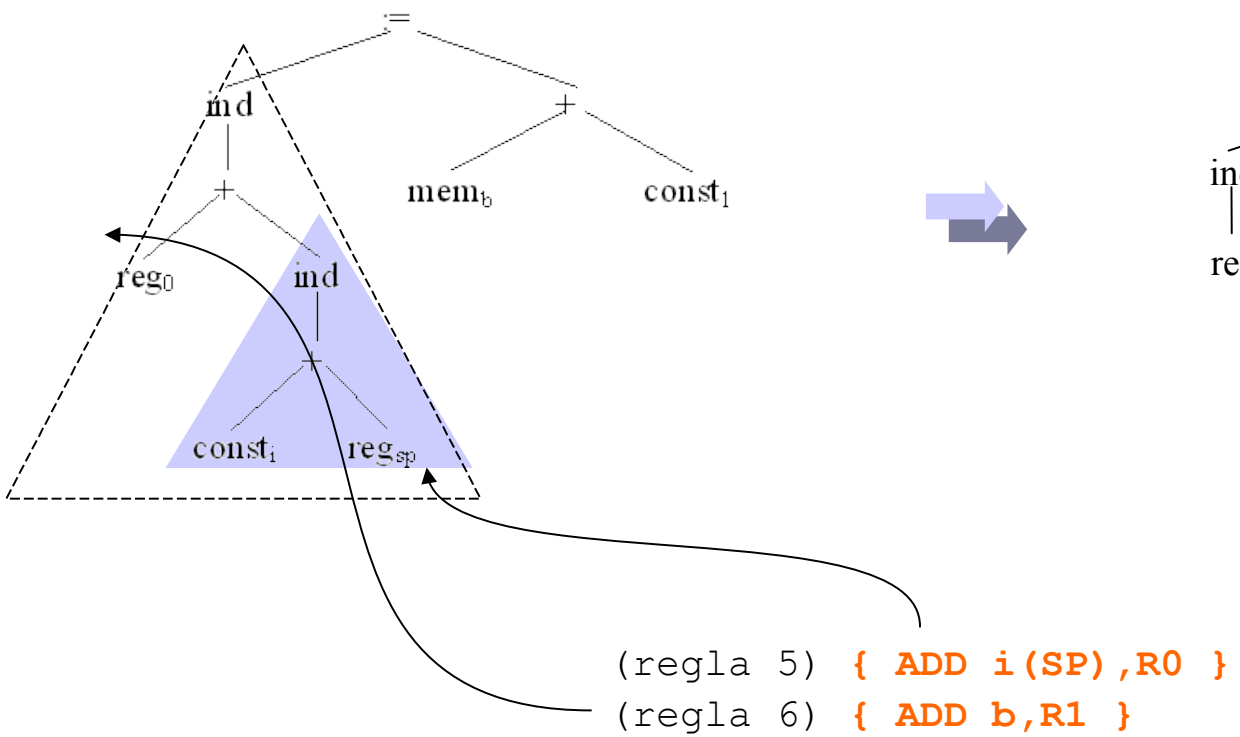
Ejemplo 11.9: Actuación del GGC de la máquina abstracta del ejemplo 11.8 ante el siguiente GDA.

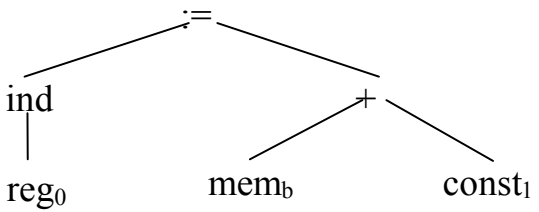



```
(regla 1) { MOV #a,R0 }
(regla 7) { ADD SP,R0 }
```

11.6 Generadores de Generadores de Código (5)

Ejemplo 11.9: (cont.)

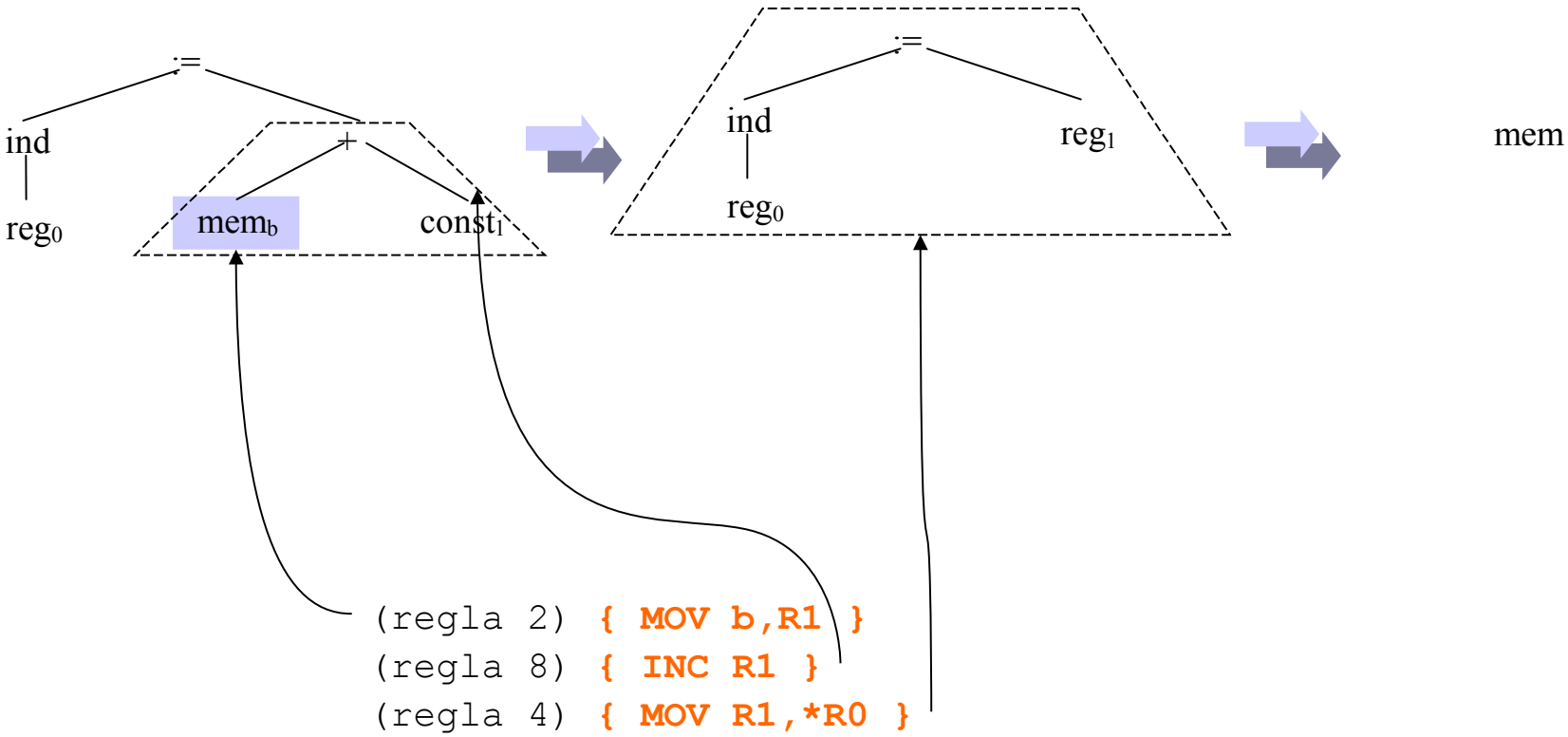





```
(regla 5) { ADD i(SP),R0 }
(regla 6) { ADD b,R1 }
```

11.6 Generadores de Generadores de Código (6)

Ejemplo 11.9: (cont.)



11.6 Generadores de Generadores de Código (7)

Concordancia de patrones mediante análisis sintáctico: Si se opta por una representación en notación prefijo de los GDAs, podemos considerar al conjunto de reglas de reescritura como patrones y poder expresarlos como si fuese una gramática con atributos, permitiendo realizar una generación de código dirigida por sintaxis.

Ejemplo 11.10: Esquema en notación prefijo para la máquina abstracta del Ejemplo 11.8.

1	reg_i : $const_c$	{ MOV #c,Ri }
2	reg_i : mem_a	{ MOV a, Ri }
3	mem : := mem_a reg_i	{ MOV Ri, a }
4	mem : := ind reg_i reg_j	{ MOV Rj, *Ri }
5	reg_i : ind + $const_c$ reg_j	{ MOV c(Rj),Ri }
6	reg_i : + reg_i ind + $const_c$ reg_j	{ ADD c(Rj),Ri }
7	reg_i : + reg_i reg_j	{ ADD Rj, Ri }
8	reg_i : + reg_i $const_1$	{ INC Ri }

El GDA del ejemplo 11.9 resulta de la forma: $:=ind+const_a\ reg_{sp}\ ind+const_i\ reg_{sp}+mem_b\ const_1$